



UNIVERSIDAD DE LA REPÚBLICA
FACULTAD DE INGENIERÍA



EDA: Experimentación con Drones Autónomos

MEMORIA DE PROYECTO PRESENTADA A LA FACULTAD DE
INGENIERÍA DE LA UNIVERSIDAD DE LA REPÚBLICA POR

Leandro Barros, Bruno Da Costa, Mauro García, Tobias
Rodríguez

EN CUMPLIMIENTO PARCIAL DE LOS REQUERIMIENTOS
PARA LA FINALIZACIÓN DE LA CARRERA DE
INGENIERÍA ELÉCTRICA.

TUTORES

Mariana del Castillo Universidad de la República
Romina García University of Pennsylvania
Federico La Rocca Universidad de la República

TRIBUNAL

Leopoldo Agorio Universidad de la República
Facundo Benavides Universidad de la República
Gabriel Gómez Universidad de la República

Montevideo
miércoles 31 diciembre, 2025

Esta página ha sido intencionalmente dejada en blanco.

La investigación crea nuevos conocimientos.

NEIL ARMSTRONG

Esta página ha sido intencionalmente dejada en blanco.

Agradecimientos

Queremos expresar nuestro profundo agradecimiento a quienes hicieron posible este proyecto y nos acompañaron en cada etapa del camino. A nuestros tutores (Mariana, Romina y Federico) por su dedicación, por guiarnos con paciencia y claridad, y por brindarnos siempre el apoyo necesario para avanzar incluso en los momentos más complejos. Su acompañamiento académico y humano marcó una diferencia fundamental en el desarrollo de este trabajo. A la Facultad de Ingeniería y a la Universidad, por darnos el espacio, la formación y las herramientas para crecer como profesionales y como personas. A Pablo, por introducirnos en este mundo y por ser un referente en el país en esta área; sus conocimientos y su entusiasmo contribuyeron profundamente a nuestra formación y a la orientación de este proyecto. A Horacio, por su constante disponibilidad en el laboratorio y por salvarnos más de una vez cuando nos olvidábamos las llaves. Su apoyo cotidiano fue clave para que pudiéramos trabajar con continuidad. A Sebastián por su asistencia cuando quisimos imprimir modelos en 3D. A Hector, por prestarnos su campo para realizar las pruebas siempre que lo necesitamos y sin ningún compromiso. Su generosidad fue indispensable: sin ese espacio, este proyecto en su forma actual hubiera sido simplemente imposible de llevar a cabo. A Francisco y Gaston por sus compartir su experiencia consejos y en diversos ámbitos relacionados con este proyecto. Finalmente, a nuestras familias y amigos, que nos sostuvieron en el día a día, nos alentaron cuando las cosas se complicaban y celebraron cada pequeño avance. Su paciencia, comprensión y apoyo emocional fueron, sin duda, el motor silencioso detrás de todo este recorrido. A todos ustedes, nuestro sincero agradecimiento.

Esta página ha sido intencionalmente dejada en blanco.

A nuestras familias y amigos, por su apoyo incondicional a lo largo de este camino. A quienes con su ejemplo y sus enseñanzas nos inspiraron a seguir aprendiendo. A quienes estuvieron con nosotros en algún tramo del camino y dejaron una huella que aún nos acompaña. A todas esas personas que, de una forma u otra, hicieron posible este recorrido. A todos ellos, dedicamos este trabajo.

Esta página ha sido intencionalmente dejada en blanco.

Resumen

Este proyecto desarrolla una plataforma experimental para el vuelo coordinado de un enjambre de drones orientado al estudio de algoritmos de posicionamiento que garanticen conectividad inalámbrica entre agentes distribuidos. El problema abordado es la coordinación y el posicionamiento de múltiples vehículos aéreos no tripulados que operan de forma autónoma, distinguiendo entre drones de tarea, responsables de realizar rutinas predefinidas dentro un área de vuelo asignada, y drones de red, dedicados a soportar la comunicación sin intervención de un piloto, todos ellos integrados en una única red ad-hoc durante la operación. La solución integra una plataforma de hardware basada en cuadricópteros con controladora de vuelo PX4, computadora a bordo y sistema de propulsión eléctrico, una red inalámbrica mallada sobre batman-adv, una arquitectura de software distribuida en ROS 2 y un entorno de simulación específico que permite replicar los experimentos mediante PX4 SITL, Docker y contenedores dedicados por dron. Sobre esta infraestructura se implementaron e integraron dos algoritmos centralizados de posicionamiento: el algoritmo diseñado por el equipo C-DFLP (Clustering - Dynamic Facility Location Problem), para ubicar drones de red a partir de clusters generados mediante las posiciones de los drones de tarea, y el algoritmo GIOD Centralized (Graph Infrastructure On Demand Centralized), basado en aprendizaje automático, que realiza una optimización continua de los agentes de red. La plataforma se validó mediante experimentos individuales y en enjambre, tanto en simulación como en campo, incluyendo vuelos autónomos de figuras geométricas, pruebas estáticas de evaluación de enlaces y experimentos dinámicos con hasta cinco drones operando de forma coordinada. Durante estas pruebas, los drones registran datos de telemetría de los vuelos y métricas de enlaces en una base de datos utilizada para analizar los mismos. Los resultados muestran que la plataforma es robusta para ejecutar misiones reproducibles, permite extender el alcance efectivo de comunicación a través de drones de red y ofrece una base sólida para la experimentación futura con algoritmos de posicionamiento más avanzados.

Esta página ha sido intencionalmente dejada en blanco.

Prefacio

Que la tecnología y la ingeniería sigan siendo un puente y no un muro; herramientas para ampliar oportunidades y acercar a las personas. Que la investigación nos recuerde que el conocimiento no busca imponer, sino comprender, y que el futuro que construyamos sea uno donde la innovación esté al servicio de la paz y del progreso.

EDA.

Esta página ha sido intencionalmente dejada en blanco.

Tabla de contenidos

Agradecimientos	III
Resumen	VII
Prefacio	IX
Glosario de siglas	1
1. Introducción	3
1.1. Definición del problema	5
1.2. Objetivo General	6
1.3. Objetivos Específicos	6
1.4. Alcance	7
1.5. Antecedentes	8
2. Arquitectura del Vehículo	9
2.1. Mecánica y Estructura	9
2.2. Sistema de Control, Sensado y Cómputo	10
2.2.1. Unidad de Control de Vuelo	11
2.2.2. Computadora a Bordo y Comunicaciones	13
2.2.3. Periféricos de Detección y Navegación	14
2.3. Sistema de Propulsión y Alimentación Eléctrica	14
2.4. Integración y Flujo del Sistema	16
3. Red Inalámbrica	21
3.1. Modos de operación de redes WLAN: Infraestructura vs Ad-Hoc	22
3.1.1. Modo Infraestructura	22
3.1.2. Modo Independiente o Ad-Hoc	23
3.2. Protocolos de Ruteo	23
3.2.1. Clasificación de los Protocolos de Ruteo	24
3.2.2. Problemas en Protocolos de Ruteo MANET	24
3.3. B.A.T.M.A.N.-Advanced	25
3.3.1. Integración en el Kernel de Linux	26
3.3.2. Asignación de IPs e Interfaz	26

Tabla de contenidos

3.3.3.	Encapsulamiento de los mensajes	27
3.3.4.	Mensajes de Control	28
3.3.5.	Métrica del Canal	28
4.	Operaciones con un Dron	31
4.1.	Herramientas de Software	31
4.2.	Experimentos	33
4.3.	Validación en Entorno Simulado	36
4.4.	Pruebas de Campo y Resultados	39
5.	Operaciones en Enjambre	43
5.1.	Configuración de las NUCs y la Red	43
5.2.	ROS 2	44
5.2.1.	Elementos Fundamentales de ROS 2	46
5.2.2.	Middleware y QoS	48
5.2.3.	Paquetes y Estructura de Trabajo	48
5.3.	Arquitectura, Diagrama y Flujo de la Plataforma Diseñada	49
5.3.1.	Arquitectura de Software e Interfaces	50
5.4.	Monitoreo en tiempo real	54
5.5.	Adaptación de la Plataforma a Nuevos Algoritmos	56
6.	Algoritmos de Posicionamiento	59
6.1.	Implementación Propuesta	59
6.2.	C-DFLP: Clustering Dinamic Facility Location Problem	60
6.2.1.	Flujo de ejecución	61
6.3.	Algoritmo GIOD Centralized	65
6.3.1.	Optimización y Gradientes	67
7.	Entorno de Simulación Desarrollado	69
7.1.	Desafíos Técnicos y Abordajes Propuestos	69
7.1.1.	Simulación y Control de Múltiples Instancias de SITL	69
7.1.2.	Simulación de la Comunicación Entre Drones y Virtualización con Docker	71
7.1.3.	Integración de ROS 2 y Ejecución de Múltiples Procesos en Cada Contenedor	72
7.2.	Solución Propuesta	73
7.3.	Implementación del Algoritmo GIOD Centralized	73
7.4.	Visualización de Trayectorias	74
8.	Experimentos en Enjambre y Resultados	77
8.1.	Diseño y Resultados de los Experimentos	77
8.1.1.	Prueba de Validación Estática	78
8.1.2.	Prueba de Validación Completa	82

8.1.3. Prueba de Validación Dinámica	88
8.1.4. Prueba de Validación con Nuevos Algoritmos	99
9. Conclusiones	103
A. Pixhawk 2.4.8	107
A.1. Microcontroladores	107
A.2. Sensores	109
B. Conceptos Útiles	111
B.1. Sistemas de coordenadas	116
B.1.1. Sistema de coordenadas corporales:	116
C. Alternativas de comunicación NUC-Pixhawk	119
D. Modulos de Software en Drones, Ground Station y entorno de Simulación	121
D.0.1. Drones	121
D.0.2. Ground Station	124
D.1. Entorno de simulación	126
D.1.1. Conexión de MAVSDK a múltiples instancias SITL . .	126
E. Tablas de los experimentos realizados	127
F. Preparación Inicial del Entorno Real	129
F.1. Gestión y Atenuación de la Potencia de Transmisión	129
F.2. Protocolo de Calibración y Verificación del Control	131
F.3. Sincronización Temporal Distribuida (NTP)	132
F.4. Inicialización y Validación de Sensores	132
F.5. Registros de Datos Obtenidos	133
F.5.1. Registros del Pixhawk	133
F.5.2. Registros de Datos en SQL	134
F.5.3. Reproducción de Vuelos	135
Índice de tablas	137
Índice de figuras	138
Bibliografía	152

Esta página ha sido intencionalmente dejada en blanco.

Glosario de Siglas

AP - Access Point

B.A.T.M.A.N - Better Approach To Mobile Adhoc Networking

C-DFLP - Clustering Dinamic Facility Location Problem

DDS - Data Distribution Service

E/S - Entrada/Salida

EDA - Experimentación con Drones Autónomos

EGNN - Equivariant Graph Neural Network

FCU - Flight Control Unit

FMU - Flight Managment Unit

GIOD - Graph Infrastructure On Demand

GPU - Graphics Processing Unit

GS - Ground Station

IMU - Inertial Measurement Unit

LiDAR - Light Detection and Ranging

LiPo - Lithium-Polymer

MANET - Mobile Ad Hoc Networks

MAVLINK - Micro Air Vehicle Link

MAVSDK - Micro Air Vehicle Link Software Development Kit

MCU - Microcontroller Unit

MEMS - Micro-Electro-Mechanical Systems

NAT - Network Address Translation

NED - North-East-Down

NTP - Network Time Protocol

NUC - Next Unit of Computing

OBC - On-Board Computer

Tabla de contenidos

OGMs - Originator Messages

PDB - Power Distribution Board

PWM - Pulse Width modulation

QoS - Quality of Service

RC - Radio Control

RMW - ROS Middleware Interface

ROS - Robot Operating System

RPAS - Remotely Piloted Aircraft System

SITL - Software In The Loop

SSD - Solid State Drive

TQ - Transmission Quality

UAV - Unmanned Aerial Vehicle

UPenn - Universidad de Pennsylvania

VANT - Vehículo Aéreo No Tripulado

VANET - Vehicular Ad-hoc NETwork,

WANET - Wireless Ad Hoc Network

WLAN - Wireless Local Area Network

WSN - Wireless Sensor Networks

Capítulo 1

Introducción

Los drones de comunicación encuentran un sinnúmero de aplicaciones que mejoran el bienestar de la sociedad. Estos sistemas permiten la ejecución de tareas distribuidas de sensado y monitoreo, como la captura coordinada de imágenes para mapeo y fotogrametría, la medición de variables ambientales (temperatura, humedad, concentración de gases o contaminantes) y la inspección de infraestructuras extensas, entre otras. Un caso de especial relevancia es el de los escenarios de emergencia y desastre. Eventos como terremotos, inundaciones o incendios forestales pueden dañar o destruir la infraestructura de comunicaciones de una región, dejando a comunidades enteras incomunicadas en momentos críticos. En este contexto, una flota de drones autónomos orientados a la provisión de conectividad (agentes de red) puede desplegarse rápidamente para establecer una red inalámbrica temporal. Esta malla aérea permite asistir a los equipos de rescate y a los drones dedicados a tareas específicas de búsqueda, relevamiento o monitoreo (agentes de tarea), facilitando el intercambio de información y la coordinación en tiempo real. La cobertura brindada por esta red aérea autoorganizada, en la cual los drones de comunicación se reposicionan dinámicamente mediante algoritmos de control y posicionamiento, resulta fundamental para extender el alcance de la red, mantener la conectividad y sostener los servicios de comunicación de emergencia mientras se restablece la infraestructura permanente.

En esta solución juega un rol clave el dron, o vehículo aéreo no tripulado (VANT), una aeronave capaz de operar sin tripulación a bordo. En la literatura técnica se utilizan comúnmente las siglas UAV (Unmanned Aerial Vehicle) o RPAS (Remotely Piloted Aircraft System). Desde el punto de vista funcional, un dron puede mantener un vuelo controlado tanto mediante operación remota (piloteado desde tierra) como mediante sistemas autónomos a bordo [1] [2].

La consolidación tecnológica de estos sistemas ha impulsado una amplia gama de aplicaciones en sectores civiles, industriales y militares. Estos vehículos se utilizan actualmente en actividades recreativas y producción audiovisual; en agricultura de precisión; en logística, facilitando entregas de

Capítulo 1. Introducción

última milla; en la inspección de infraestructuras y el monitoreo ambiental; así como en tareas de seguridad, vigilancia, búsqueda y rescate. En el ámbito científico, los drones permiten la recolección de datos en zonas remotas o de difícil acceso. Por otra parte, en el entorno militar han evolucionado notablemente. Esta diversidad de usos refleja su valor como plataformas aéreas versátiles para la adquisición de información y el soporte operativo [3] [2] [4].

A pesar de sus beneficios, la utilización de drones enfrenta diversos desafíos. En el plano técnico, las interferencias electromagnéticas, las condiciones meteorológicas adversas, los ciberataques o las limitaciones de autonomía (derivadas del tipo de fuente de energía y de los sistemas de navegación) pueden comprometer la operación de un dron. En el plano ético, su uso plantea preocupaciones relacionadas con la privacidad, la vigilancia no autorizada y el riesgo de delegar decisiones críticas a sistemas autónomos. Económicamente, los costos de adquisición, operación y mantenimiento de plataformas avanzadas dificultan su adopción masiva. En conjunto, los drones representan una tecnología de alto potencial, pero su integración sostenible requiere superar estas limitaciones, definir marcos regulatorios adecuados y evaluar cuidadosamente sus implicaciones sociales y operativas [1] [2] [3].

Para superar algunas de estas limitaciones y afrontar tareas de mayor complejidad surge el concepto de enjambre de drones. Un enjambre de drones es un conjunto de vehículos aéreos no tripulados que trabajan de forma coordinada, autónoma y sincronizada para alcanzar objetivos comunes. Esta modalidad colaborativa se aleja del modelo de control individual: los drones del enjambre actúan como una “mente colmena”, compartiendo datos en tiempo real y tomando decisiones colectivas mediante algoritmos particulares y arquitecturas de hardware y software avanzadas. Gracias a esta inteligencia distribuida, el conjunto de drones puede mantener formaciones, evitar colisiones y adaptarse a cambios del entorno sin necesidad de un control humano constante [5] [6].

Existen dos enfoques principales para coordinar un enjambre de drones: control centralizado y control descentralizado. Cada paradigma presenta fundamentos teóricos distintos, con ventajas particulares y aplicaciones óptimas según el contexto de la misión. En un sistema de control centralizado, un nodo maestro o estación base recopila la información de todos los drones, toma todas las decisiones y envía instrucciones específicas a cada unidad. Este esquema resulta eficaz cuando se requiere una coordinación muy precisa y la misión está claramente definida. Por el contrario, en un enfoque descentralizado no existe un controlador único; cada dron toma decisiones de forma autónoma basándose en reglas locales, en su propia percepción del entorno y en la comunicación con sus vecinos. Inspirada en la inteligencia colectiva y en los sistemas multiagente observados en la naturaleza, esta aproximación

1.1. Definición del problema

distribuida es preferible en entornos dinámicos o impredecibles, especialmente cuando involucra un número elevado de drones [7] [8].

Desde la perspectiva académica, los enjambres de drones autónomos representan una avanzada línea de investigación en robótica, comunicaciones inalámbricas y sistemas distribuidos. Esta área se destaca por su carácter multidisciplinario, integrando disciplinas como el control automático, la computación a bordo, las redes de comunicación e incluso la visión por computadora. Desde la perspectiva estratégica, el fortalecimiento del dominio local en estas tecnologías emergentes (de interés global), especialmente en lo referente a vehículos aéreos autónomos, posiciona a Uruguay como referente regional en investigación aplicada sobre enjambres de drones. Además, impulsar este campo contribuye a la formación de profesionales especializados y promueve la innovación tecnológica en el país.

En síntesis, la motivación de este trabajo se fundamenta en un problema concreto: la necesidad de redes de comunicación resilientes mediante enjambres aéreos. Abordar este desafío no solo ofrece una solución efectiva ante situaciones críticas como la descrita, sino que también promueve el avance del conocimiento y el fortalecimiento de capacidades locales en un área de alta proyección científica y tecnológica. Con esta motivación delineada, en la siguiente sección se presenta la definición precisa del problema que este proyecto busca resolver.

1.1. Definición del problema

El desarrollo práctico de algoritmos de coordinación para enjambres de drones está limitado por la falta de plataformas experimentales que sean versátiles, accesibles y reproducibles. No existe una infraestructura modular que permita implementar y validar algoritmos avanzados de posicionamiento autónomo, tanto en simulación como en vuelos reales con drones de pequeña escala. Aunque los drones comerciales se han masificado, suelen carecer de la capacidad de cómputo necesaria para ejecutar tareas distribuidas, y en caso de poseerla, suelen estar privatizadas por el fabricante, limitando su configuración y uso por parte del usuario. A su vez, muchos algoritmos de enjambre, especialmente los orientados a optimizar la conectividad de red, se mantienen en simuladores debido a la dificultad de llevarlos a plataformas físicas. En este escenario aparecen dos brechas principales:

1. **Brecha tecnológica:** dotar a drones económicos de capacidades avanzadas de cómputo y comunicación para que operen como un enjambre inteligente. Esto implica incorporar hardware adicional y un entorno de software adecuado.
2. **Brecha experimental:** falta de una plataforma sistemática para evaluar estos algoritmos en condiciones reales y medir su impacto en

Capítulo 1. Introducción

métricas como el throughput.

El proyecto aborda el diseño, implementación y validación tanto de un algoritmo de posicionamiento como de una plataforma física y simulada que permita operar un enjambre de drones orientados a maximizar la eficiencia en la comunicación, considerando las limitaciones del hardware y las condiciones reales de operación. Este desafío requiere integrar hardware, software y experimentación, tal como se desarrolla en los capítulos posteriores.

1.2. Objetivo General

El objetivo general del proyecto es diseñar y desarrollar una plataforma experimental compuesta por un enjambre de drones autónomos (al menos cinco unidades) y un entorno de simulación asociado. Esta plataforma permitirá ejecutar y validar algoritmos de posicionamiento estratégico de drones. En esencia, se busca sentar la infraestructura necesaria para desplegar una red autónoma de drones, denominada también agentes de red, que sirva como medio de comunicación auxiliar en situaciones donde la red convencional esté fuera de servicio. A su vez se implementarán rutinas de movimiento simples para drones que actúen como agentes de tarea a los que se les brindará la conectividad. Adicionalmente, la plataforma tendrá la flexibilidad de ser utilizada para futuras investigaciones en el campo de los enjambres de drones, extendiendo su relevancia más allá de los objetivos inmediatos del presente proyecto.

1.3. Objetivos Específicos

Uno de los objetivos específicos consiste en lograr una configuración adecuada del *hardware* y *firmware* de control de cada dron, junto con la calibración precisa de los sensores integrados. Para ello, en una primera instancia se realizarán pruebas de vuelo controlado de forma remota que permitan verificar la estabilidad básica de cada vehículo. Posteriormente, y como parte de una segunda etapa, se llevará a cabo la adaptación de la interfaz de comunicación entre la controladora de vuelo y la NUC (*Next Unit of Computing*) que actúa como computadora a bordo (OBC: *On-Board Computer*). Esta fase incluye la instalación de un sistema operativo basado en Linux, la incorporación de ROS 2 y la integración de los paquetes de *software* necesarios para que cada dron quede en condiciones de ejecutar algoritmos de manera autónoma en coordinación con su controladora de vuelo.

Un segundo objetivo está orientado al desarrollo del *software* que permite implementar rutinas básicas de vuelo y posicionamiento autónomo. En este caso, la propuesta abarca desde despegues y aterrizajes automáticos hasta

1.4. Alcance

movimientos simples, como traslados en línea recta, que sirven para comprobar la capacidad de los drones de ejecutar acciones coordinadas. Con el fin de asegurar un funcionamiento robusto, estas rutinas se evalúan inicialmente en el entorno de simulación y posteriormente en los drones físicos, lo que posibilita verificar que el algoritmo de posicionamiento subyacente mantenga un comportamiento consistente entre ambos escenarios. Esta estrategia facilita la validación de aspectos esenciales relacionados con la estabilidad, la ejecución de comandos y el mantenimiento de posición, antes de avanzar hacia comportamientos colaborativos de mayor complejidad.

Con el fin de probar tanto la plataforma de simulación como la plataforma física en sus etapas iniciales, será necesario la creación de un algoritmo propio de posicionamiento de drones. Este debe ser lo suficientemente robusto para soportar las distintas pruebas que serán realizadas, buscando que los drones se posicionen de manera que se mantenga la cobertura de comunicación entre los nodos de tarea durante todo el experimento, incorporando mecanismos básicos de control que preserven la seguridad e integridad de la operación. Un aspecto central de su diseño es que su funcionamiento interno sea completamente conocido y comprensible, pudiendo así anticipar y analizar su comportamiento de forma teórica, facilitando la interpretación de los resultados experimentales.

Finalmente, se integrará el algoritmo de posicionamiento propuesto por la tutora del proyecto, con el propósito de demostrar la flexibilidad de la plataforma y su capacidad para evaluar distintos enfoques de optimización de ubicación dentro de la red de drones.

1.4. Alcance

El proyecto se desarrolla con un enjambre de cinco drones, lo que permite evaluar comportamientos colectivos básicos. Además, se utilizan los algoritmos únicamente como casos de estudio destinados a validar la plataforma, sin que su optimización constituya un objetivo principal. En cuanto al entorno virtual, la simulación se lleva a cabo mediante un modelo tridimensional que represente el comportamiento físico de los drones bajo condiciones ideales y sin incluir factores ambientales adversos.

Por otra parte, el hardware empleado corresponde exclusivamente a los componentes provistos por el Instituto de Ingeniería Eléctrica de la Universidad de la República, de modo que las capacidades del sistema quedan limitadas a este equipamiento. Asimismo, la comunicación entre drones se realiza mediante una red ad-hoc basada en el protocolo de ruteo B.A.T.M.A.N. [9]. Sobre Wi-Fi, mientras que la interacción entre la computadora a bordo y la controladora de vuelo se maneja a través de MAVSDK (*Micro Air Vehicle Link Software Development Kit*) [10]. Sin desarrollar nuevos protocolos ni modificar el firmware existente. Finalmente, todas las pruebas se efectúan en

Capítulo 1. Introducción

entornos controlados, por lo que no se incluyen funcionalidades adicionales como evasión de obstáculos, operación en condiciones climáticas adversas o mecanismos avanzados de resiliencia.

1.5. Antecedentes

Este proyecto, “EDA: Experimentación con Drones Autónomos” se sustenta en antecedentes técnicos, institucionales y académicos que acompañan el camino recorrido. Proyectos como “*Proyecto TERO*” [11] que es el punto de partida para utilizar la flota de drones disponible, “*ENDRA: Enjambre de Drones Autónomos*” [12] que introduce el vuelo autónomo de un enjambre de drones, “*Exploración robótica colaborativa de interiores*” [13] utiliza ROS 2 para la movilidad de agentes, o también papers más específicos como “*A Networked Multiagent System for Mobile Wireless Infrastructure on Demand*” [14] que presenta un sistema multiagente de UAVs que establece una red inalámbrica ad-hoc que se reconfigura de manera dinámica para satisfacer requisitos de calidad y “*EGNN-based topology control in wireless mobile infrastructure on demand with shared access restrictions*” [15] donde se presenta la optimización de agentes de red en una infraestructura móvil para mejorar la comunicación entre agentes de tarea.

Este proyecto se para sobre los hombros de estos cinco gigantes e incorpora características propias para poder montar y construir la plataforma de experimentación con drones autónomos y algoritmos de posicionamiento enfocada en la conectividad entre agentes.

Se puede encontrar el repositorio del proyecto aquí

Capítulo 2

Arquitectura del Vehículo

El cumplimiento de los objetivos de experimentación con enjambres autónomos, detallados en la introducción de este proyecto, depende de la disponibilidad de una plataforma de *hardware* robusta, modular y con capacidad de cómputo suficiente. Este capítulo describe la arquitectura de *hardware* disponible y ensamblada para cada dron de la flota.

El diseño del sistema no se concibe como un fin en sí mismo, sino como una herramienta de experimentación diseñada a medida. La filosofía de diseño se centra en la integración de componentes comerciales probados para las funciones de vuelo esenciales, complementados con una computadora a bordo de alto rendimiento para la ejecución de la lógica de autonomía y coordinación.

Esta arquitectura híbrida permite desacoplar las tareas críticas de control de vuelo en tiempo real de los procesos de alto nivel. Una controladora de vuelo (FCU: *Flight Control Unit*) dedicada gestiona la estabilización y la navegación de bajo nivel, mientras que la OBC ejecuta el entorno de software de alto nivel y gestiona la comunicación en red del enjambre.

El análisis del *hardware* se organiza en tres secciones principales. La Sección 2.1 describe la mecánica y la estructura, donde se presenta el armazón físico del dron y los elementos que le otorgan soporte y rigidez. La Sección 2.2 se centra en el sistema de control, sensado y cómputo, abarcando los componentes electrónicos que funcionan como el “cerebro” y los “sentidos” del vehículo, entre ellos la controladora de vuelo, la OBC y los sensores externos. Finalmente, la Sección 2.3 aborda el sistema de propulsión y la alimentación eléctrica, detallando la cadena de potencia que va desde la batería hasta los motores y que permite generar el empuje necesario para el vuelo.

2.1. Mecánica y Estructura

La estructura comprende el armazón físico que soporta y protege a todos los demás componentes del vehículo, asegurando la rigidez necesaria para

Capítulo 2. Arquitectura del Vehículo

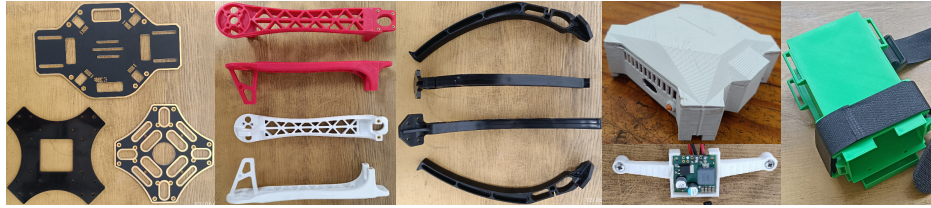


Figura 2.1: En orden de derecha a izquierda se observa el chasis inferior y superior junto con la placa de montaje de la OBC, los brazos, el tren de aterrizaje, la carcasa protectora del OBC junto con el soporte de un regulado de voltaje, y finalmente, el soporte de la batería junto con un velcro.

un vuelo estable y resistiendo las vibraciones y los impactos operativos. Se desarrollan a continuación todos los componentes que conforman la mecánica y estructura del dron utilizado en el proyecto, los mismos se ilustran en la Figura 2.1

La plataforma seleccionada es un cuadricóptero basado en el chasis (*frame*) DJI F450. Esta se justifica por su probada robustez, diseño modular que facilita las reparaciones y una amplia disponibilidad de repuestos. El chasis central consiste en dos placas de fibra de vidrio (FR4), una inferior y una superior. La placa inferior posee la característica de integrar pistas de cobre, funcionando como una Placa de Distribución de Potencia (PDB: *Power Distribution Board*) básica, lo que simplifica el cableado de la batería a los variadores de velocidad (ESC: *Electronic Speed Control*) [16].

A este chasis base se acoplan cuatro brazos de plástico resistente, también del modelo DJI F450, diseñados para absorber vibraciones. El tren de aterrizaje modelo MallofUSA F450, proporciona la altura necesaria para proteger los componentes vulnerables montados en la parte inferior del chasis, como el sensor LiDAR (*Light Detection and Ranging*) y la batería.

Además de los componentes comerciales, la integración requirió el diseño y la fabricación de piezas a medida mediante impresión 3D. Estas piezas incluyen una base de montaje y una carcasa protectora para la OBC, un soporte específico para la batería de polímero de litio (LiPo: *Lithium-Polymer*) y soportes para los reguladores de voltaje auxiliares. Los componentes clave de la estructura se resumen en la Tabla 2.1.

2.2. Sistema de Control, Sensado y Cómputo

Este subsistema representa el núcleo “inteligente” del dron. Se presenta el mismo en la Figura 2.2. Compuesto por una arquitectura modular que incluye la controladora de vuelo para la estabilización en tiempo real, la OBC para la autonomía de alto nivel y un conjunto de sensores internos y externos para la percepción del estado y del entorno.

2.2. Sistema de Control, Sensado y Cómputo

Tabla 2.1: Componentes principales de la estructura mecánica.

Componente	Modelo/Material	Función Principal
Chasis (Inferior y Superior)	DJI F450 (Fibra de Vidrio, FR4)	Placa central, PDB integrada.
Brazos (x4)	DJI F450 (Plástico resistente)	Soporte de motores.
Tren de Aterrizaje	MallofUSA F450 (Plástico)	Soporte en tierra, protección.
Base OBC	Diseño a medida (Impreso en 3D)	Interfaz entre chasis y OBC.
Carcasa OBC	Diseño a medida (Impreso en 3D)	Protección de la computadora.
Soporte Pololu	Diseño a medida (Impreso en 3D)	Soporte para regulador a OBC.
Soporte Batería	Diseño a medida (Impreso en 3D)	Fijación de la batería.

2.2.1. Unidad de Control de Vuelo

El núcleo de la estabilización y el control de bajo nivel es la FCU 3DR Pixhawk 2.4.8. Se trata de un sistema de piloto automático de código abierto, utilizado en la comunidad de investigación. Su arquitectura de *hardware* se basa en dos microcontroladores (MCU: *Microcontroller Unit*) principales para garantizar la redundancia y la fiabilidad. En la Figura 2.3 se presentan tanto la apariencia externa como la interna de la controladora donde se pueden ver todos los componentes que mencionamos a continuación, una vista mas detallada se observa en la Figura A.1.

El procesador principal es la Unidad de Gestión de Vuelo (FMU: *Flight Managment Unit*), basada en un MCU ARM Cortex-M4 (STM32F427VIT6) de 168 MHz. Esta unidad ejecuta el *firmware* de control de vuelo (PX4) y los algoritmos de estimación de estado del dron como, por ejemplo, los filtros de Kalman. La FMU se complementa con un co-procesador de Entrada/Salida (E/S) STM32F100C8T8 dedicado a la generación de las señales de Modulación por Ancho de Pulsos (PWM) para los motores, y a la gestión de las entradas del receptor de radio. Esta división de tareas permite que el co-procesador actúe como un sistema de seguridad (*failsafe*), manteniendo el control básico en caso de fallo del procesador principal.

Para la estimación de su estado (actitud y altitud), la Pixhawk integra un conjunto de sensores inerciales de tipo MEMS (Sistemas Microelectromecánicos o por su sigla en inglés *Micro-Electro-Mechanical Systems*). Este conjunto incluye una Unidad de Medición Inercial (IMU: *Inertial Measurement Unit*) primaria de 6 ejes, modelo MPU-6000, que combina un acelerómetro y un

Capítulo 2. Arquitectura del Vehículo

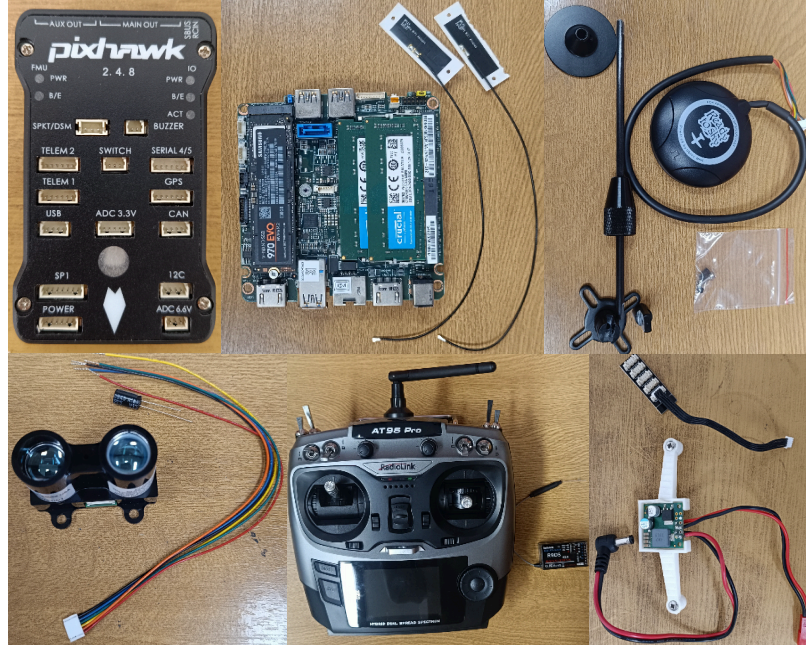


Figura 2.2: Comenzando de arriba a la izquierda, los componentes que conforman el control y sensado del Dron son: La controladora de vuelo Pixhawk, la computadora a bordo con sus antenas, el gps, el Lidar para medir distancias, el radio control con su receptor, y el regulador de voltaje Pololu.

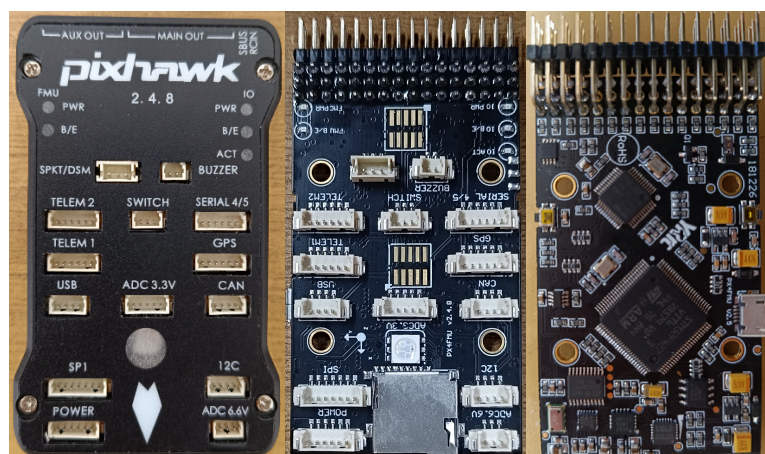


Figura 2.3: Pixhawk 2.4.8 por dentro y por fuera.

2.2. Sistema de Control, Sensado y Cómputo

giroscopio triaxial. Esta IMU se complementa con sensores redundantes: un giroscopio triaxial secundario (L3GD20) y un combo de acelerómetro-magnetómetro (LSM303D). Además, un barómetro de alta resolución (MS5611) proporciona mediciones de la presión atmosférica, lo que permite una estimación estable de la altitud sobre todo cuando se combina con el LIDAR descrito a continuación en esta sección 2.2.3.

Si bien la FCU Pixhawk 2.4.8 presenta limitaciones frente al *software* contemporáneo y puede generar dificultades durante los procesos de calibración, su arquitectura sigue siendo sólida y está diseñada para asegurar un control de vuelo fiable y estable. Esto se logra mediante la combinación de dos microcontroladores que separan las funciones críticas de control y seguridad, junto con sensores inerciales redundantes que permiten una estimación precisa y robusta del estado del dron. En conjunto, estos elementos garantizan que la FCU pueda mantener la estabilidad del vehículo, incluso ante fallos parciales o perturbaciones durante el vuelo. A pesar de su antigüedad, esta controladora es un buen punto de partida para controladoras más modernas.

2.2.2. Computadora a Bordo y Comunicaciones

Para ejecutar las tareas de alto nivel que requiere el vuelo autónomo en enjambre junto y la gestión de la red de comunicación, cada dron está equipado con una computadora a bordo. El componente utilizado es una placa Intel NUC, modelo NUC7i3DNBE.

Esta plataforma logra un equilibrio entre un formato compacto (4x4 pulgadas), una capacidad de procesamiento considerable (procesador Intel Core i3 de 7^a generación) y un consumo energético moderado. Para optimizar su rendimiento en esta aplicación, el sistema fue ensamblado con los componentes brindados: se instalaron dos módulos Crucial 8 GB obteniendo una memoria total de 16 GB de RAM DDR4, lo que proporciona recursos suficientes para la ejecución del entorno de *software*. El almacenamiento se destinó a una unidad de estado sólido (SSD: *Solid State Drive*) Samsung V-NAND 970 EVO con interfaz NVMe, garantizando un arranque rápido del sistema operativo Ubuntu Linux y un registro de datos *logging* de alta velocidad durante los experimentos.

Un componente importante para este proyecto es el módulo de conectividad inalámbrica, para el mismo se integró una tarjeta Intel AX200 en la ranura M.2 de la NUC. Este módulo proporciona conectividad *Wi-Fi 5 802.11ax*. Su elección es fundamental, ya que esta interfaz de red es la capa física sobre la cual opera el protocolo de enrutamiento descrito en el Capítulo 3, permitiendo la comunicación ad-hoc entre los drones del enjambre. La conexión se asegura mediante dos antenas externas Molex, configuradas para operar en la banda de 2.4 GHz.

Capítulo 2. Arquitectura del Vehículo

2.2.3. Periféricos de Detección y Navegación

El sistema de cómputo y control se complementa con un conjunto de sensores externos que proporcionan información crucial del entorno para la navegación autónoma.

El posicionamiento global lo brinda el receptor GPS (*Global Positioning System*) **ReadytoSky Ublox M8N**. Este módulo no solo proporciona la localización geoespacial (latitud, longitud), sino que también integra el magnetómetro (brújula) primario del sistema. Esta configuración, que ubica el magnetómetro en un mástil elevado junto a la antena GPS, es una práctica de diseño que aleja el sensor de la interferencia electromagnética (EMI: *Electromagnetic Interference*) generada por los cables de alta corriente de los ESC y la batería.

Para la navegación a baja altura y las maniobras de aterrizaje de precisión, se requiere una medición de altitud sobre el nivel del suelo más precisa que la proporcionada por el barómetro. Para esta función, se tiene el sensor **Garmin LidarLite V3** que, basado en tecnología LiDAR, ofrece mediciones de distancia de alta precisión en un rango de 0 a 40 metros. Este sensor opera correctamente entre 4,75 y 5 V CC, tensiones superiores a 5,5 V pueden dañarlo y por debajo de 4,5 V sus mediciones se vuelven poco fiables. Por este motivo, se conecta al puerto USB de la **Intel NUC** que le proporciona una salida estable de 5 V.

Tanto el módulo GPS como el LiDAR se comunican con la **Pixhawk** a través del bus serie I2C. Dado que la FCU posee un único puerto I2C, se hizo necesaria la inclusión de un HUB I2C. Este componente actúa como un expansor (*splitter*) del bus, permitiendo la conexión simultánea de ambos periféricos a la controladora de vuelo.

Finalmente, para el control manual y como sistema de seguridad para el piloto, el sistema incluye un enlace de radio control. Este consta de un transmisor en tierra **RadioLink AT900** y un receptor a bordo **RadioLink R9DS** conectado directamente a la **Pixhawk**. Un resumen de estos componentes clave de control y sensado se presenta en la Tabla 2.2.

2.3. Sistema de Propulsión y Alimentación Eléctrica

El sistema de propulsión y alimentación es responsable de proporcionar la energía eléctrica y convertirla en el empuje mecánico necesario para el vuelo. Este define la autonomía, la agilidad y la capacidad de carga del vehículo. Los componentes que conforman la propulsión y alimentación eléctrica del dron se presentan en la Figura 2.4

La fuente de energía principal es una batería de polímero de litio. Aunque el modelo exacto no fue un componente fijo del proyecto, el sistema se diseñó para operar con flexibilidad. Basado en los motores 2212/920 KV y los ESC

2.3. Sistema de Propulsión y Alimentación Eléctrica

Tabla 2.2: Componentes del sistema de control, detección y cómputo.

Componente	Especificación	Función Principal
Controladora de Vuelo	FCU 3DR Pixhawk 2.4.8	Estabilización y control de bajo nivel.
Computadora	Intel NUC NUC7i3DNBE	Cómputo de alto nivel (ROS 2).
RAM (NUC)	16 GB (2x8GB) Crucial DDR4	Memoria para procesamiento.
SSD (NUC)	Samsung V-NAND 970 EVO	Almacenamiento S.O. y logs.
Wi-Fi (NUC)	Intel AX200 Wi-Fi 6	Comunicación de red (MANET).
GPS	ReadytoSky Ublox M8N	Posicionamiento global (GNSS) y brújula.
LIDAR	Garmin LidarLite V3	Medición de altitud (AGL) 0-40m.
Antenas Wi-Fi (x2)	Molex (2,4/5 GHz)	Interfaz de RF para el módulo AX200.
Receptor de Radio	RadioLink R9DS	Enlace de control manual (failsafe).
Transmisión de Radio	RadioLink AT900	Control remoto para piloto.
HUB I2C	Genérico	Expansor de bus I2C.



Figura 2.4: Componentes que conforman la alimentación y propulsión del Dron.

Capítulo 2. Arquitectura del Vehículo

de 30 A, la plataforma es compatible con configuraciones de 4 celdas 14.8 V y 15.2 V. El modelo usado en el proyecto es LiHV Turnigy Power Systems 4s2p 15.2 V. La capacidad de las baterías utilizadas en los experimentos se sitúa en el rango de los 5200 mAh, permitiendo hasta seis minutos y medio de tiempo de vuelo sin comprometer los límites de operación.

La energía de la batería se distribuye a través del módulo de potencia principal, modelo CRIUS 28V/90A. Este componente cumple dos funciones: primero, actúa como un sensor de voltaje y corriente hasta 90 A, enviando esta información a la Pixhawk para el monitoreo en tiempo real de la autonomía restante; segundo, integra un Circuito Regulador de Batería (BEC: *Battery Eliminator Circuit*) que proporciona una salida de voltaje limpia y estable de 5.3 V con hasta 3 A para alimentar la controladora de vuelo y sus periféricos directos (como el receptor RC y el GPS). Este módulo se conecta a su vez a las pistas de cobre en el chasis, alimentando los ESCs. Esta conexión es directa a la batería, no entra en acción el BEC.

La salida de 5.3 V del BEC principal es insuficiente para alimentar la OBC. Para resolver esto, se utiliza un regulador de voltaje *step-down* (reductor) auxiliar de la marca Pololu. Este regulador toma el voltaje de entre 13.3 V a 50 V (ej. 15.2 V con nuestras baterías) y lo reduce de manera eficiente a 12 V para alimentar de forma independiente la OBC. Desde esta última se alimenta el sensor LiDAR, evitando caídas de tensión en el sistema de control de alto nivel y protegiendo la OBC ante descargas de corriente como un cortocircuito.

La cadena de propulsión que genera el empuje consta de tres componentes por cada brazo. Los Variadores de Velocidad (ESC) HobbyPower Simonk 30A, reciben la señal de control PWM (*Pulse Width modulation*) de la Pixhawk y el voltaje de la batería, y entregan la potencia para excitar los motores. Los motores *brushless* (sin escobillas) son ReadytoSky 2212 920 KV con un índice de 920 RPM por Voltio. Finalmente, los motores accionan las hélices GEMFAN 1045 de 10 pulgadas de diámetro por 4.5 pulgadas de paso, que convierten la rotación en empuje aerodinámico. Los componentes de este subsistema se detallan en la Tabla 2.3.

La Figura 2.5 muestra como quedan los drones armados, con el sistema de propulsión y alimentación, listos para volar.

2.4. Integración y Flujo del Sistema

El diseño final del vehículo integra los tres subsistemas descritos (estructura, control y propulsión) en una plataforma de experimentación coherente y funcional. La disposición física de los componentes fue diseñada para optimizar el centro de gravedad, minimizar la interferencia electromagnética y facilitar el mantenimiento. Una guía visual de los componentes existentes en el dron

2.4. Integración y Flujo del Sistema

Tabla 2.3: Componentes del sistema de propulsión y alimentación eléctrica.

Componente	Especificación	Función Principal
Batería	LiPo 4S, 5200mAh	Fuente de energía principal.
Módulo de Potencia	CRIUS 28V/90A	Sensor V/I y BEC 5.3V/3A para Pixhawk.
Regulador Aux.	Pololu (genérico)	Regulador Step-Down (BEC) para OBC.
Variadores (ESC) (x4)	HobbyPower Simonk 30A	Control de velocidad de motores.
Motores (x4)	ReadytoSky 2212 (920 kv)	Generación de empuje (brushless).
Hélices (x4)	GEMFAN 1045 (10 × 4,5)	Conversión de rotación en empuje.



Figura 2.5: Drones ensamblados listos para volar.

ensamblado se presenta en la Figura 2.6, la misma reúne los componentes ya mostrados e incorpora el transformador de la OBC que permite alimentarla en tierra (esté o no montada en el dron) para trabajar antes o después de los vuelos sin depender de las baterías, el cable USB-microUSB utilizado para la conexión entre la Pixhawk y la OBC, y el botón de seguridad, que impide el “armado” del dron mientras no se lo presione. En el contexto de los UAVs, armar el dron (o activar los motores) se refiere al estado en el que los motores están girando a baja velocidad, sin que el vehículo despegue aún.

El funcionamiento del sistema integrado se puede comprender analizando los flujos de control, potencia y datos.

El flujo de control es modular: la OBC, operando como el cerebro de

Capítulo 2. Arquitectura del Vehículo



Figura 2.6: Guía visual de la totalidad de las piezas del dron.

2.4. Integración y Flujo del Sistema

alto nivel, ejecuta la lógica de misión y toma decisiones (ej. “ir al punto X”). Estos comandos de alto nivel se envían, a través de la conexión USB/Serie utilizando el protocolo descrito en la Sección 4.1 a la FCU (**Pixhawk**). La **Pixhawk**, actuando como el controlador de bajo nivel en tiempo real, recibe esta consigna y la traduce en acciones de estabilización, calculando y enviando las señales PWM precisas a cada uno de los cuatro ESC para ejecutar el movimiento deseado.

El flujo de potencia se origina en la batería de 4 celdas. Se divide en tres ramas principales: (1) la rama de alta potencia, que alimenta directamente a los cuatro ESC; (2) la rama de control, que pasa por el módulo **CRIUS** para alimentar la **Pixhawk** y sus periféricos; y (3) la rama de cómputo, que pasa por los reguladores **Pololu** para alimentar la OBC y el LiDAR.

Finalmente, el flujo de datos del enjambre es habilitado por la OBC. La OBC (NUC), mediante su modulo Wi-Fi y la red (descrita en la Sección 3.3) permite a cada dron comunicarse directamente con las OBCs de los otros drones y con la estación de control en tierra, intercambiando datos y enviando órdenes de coordinación.

En conjunto, esta configuración, que combina un *frame* robusto (F450), un piloto fiable (**Pixhawk**) y una computadora de alto rendimiento (NUC), entre otros, proporciona una plataforma de hardware experimental escalable, capaz de ejecutar los algoritmos de vuelo en enjambre autónomo que se detallan en el Capítulo 6 de esta tesis.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 3

Red Inalámbrica

Tal como se expuso previamente, uno de los elementos fundamentales en un sistema de enjambre de drones (tanto en arquitecturas centralizadas como descentralizadas) es la capacidad de establecer una conectividad efectiva y continua entre los nodos que lo componen. Esta conectividad no solo permite el intercambio de información esencial para el control y la cooperación, sino que habilita dinámicas colectivas más complejas, como la toma de decisiones distribuida o la adaptación a entornos cambiantes, todos ellos aspectos centrales en un enjambre autónomo.

En este contexto, una red inalámbrica se presenta como el medio natural de comunicación para un sistema compuesto por unidades móviles y autónomas. Aun así, resulta pertinente mencionar las tecnologías desarrolladas para el vuelo de drones mediante conexión cableada donde el dron se vincula al operador a través de una fibra óptica. Este enfoque, utilizado principalmente en entornos militares, ofrece transmisión extremadamente estable, vídeo de alta calidad y la ventaja táctica de no emitir señales de radio. Sin embargo, su uso es limitado debido al alto costo, el alcance reducido, el peso adicional del cable y el riesgo de enganches, especialmente en operaciones con múltiples drones. Además, estos sistemas están diseñados para control directo por un operador, por lo que no se adaptan a modelos de cooperación y autonomía requeridos por un enjambre de drones [17] [18].

Nos enfocaremos entonces en el uso de WLAN (*Wireless Local Area Network*) debido a que ofrece el mejor equilibrio entre alcance, velocidad, flexibilidad y autonomía operativa para entornos dinámicos y distribuidos. Se podría considerar inicialmente por su bajo consumo y simplicidad las WPAN (*Wireless Personal Area Network*), en donde existen tecnologías como Bluetooth (IEEE 802.15.1), Zigbee (IEEE 802.15.4), infrarrojo o RFID. Bluetooth, por ejemplo, permite estructuras como piconets y scatternets que habilitan cierta organización multi-dispositivo. Sin embargo, estas tecnologías no están diseñadas para escenarios con movilidad rápida, tasas de transmisión elevadas ni mantenimiento de rutas estables en redes multi-salto. Su alcance

Capítulo 3. Red Inalámbrica

reducido, su limitada capacidad de escalado y su comportamiento poco robusto ante cambios topológicos las vuelven inadecuadas para un enjambre de drones.

Por otro lado, alternativas como WMAN (*Wireless Metropolitan Area Network*) o WWAN (*Wireless Wide Area Network*) proporcionan amplias coberturas, pero dependen de infraestructura externa (torres celulares, estaciones base o enlaces licenciados) lo cual contradice el requisito de operar de manera autónoma y en entornos sin conectividad preexistente.

En contraste, WLAN permite desplegar una red ad-hoc autónoma y descentralizada, capaz de operar sin puntos de acceso, sin servicios externos y sin infraestructura fija. Esta característica la convierte en la opción más adecuada para escenarios donde los drones deben coordinarse y cooperar mientras se desplazan, manteniendo conectividad estable en un entorno dinámico y sin requerir equipamiento adicional ni dependencia de redes públicas [19] [20] [21].

3.1. Modos de operación de redes WLAN: Infraestructura vs Ad-Hoc

Las redes inalámbricas Wi-Fi pueden configurarse para operar en distintos modos, pudiendo ser el modo infraestructura o el modo ad-hoc. Se optó por implementar una red ad-hoc, dado que permite la comunicación directa entre nodos sin depender de un punto de acceso central, una característica esencial para entornos móviles y sin infraestructura preexistente, como el de un enjambre de drones. No obstante, para comprender adecuadamente esta elección, resulta útil describir ambos modos, sus características y principales desventajas, enfocado particularmente a nuestro proyecto.

3.1.1. Modo Infraestructura

Es el modo más común en hogares, oficinas y espacios públicos. En este esquema, los dispositivos inalámbricos (estaciones o clientes) se comunican a través de un punto de acceso (AP: *Access Point*), el cual actúa como intermediario entre ellos y la red cableada o Internet. La comunicación sigue una topología en estrella, donde todos los paquetes pasan primero por el AP. Este modo facilita la gestión y seguridad mediante autenticación centralizada (por ejemplo, WPA2-Enterprise y RADIUS), asignación de ancho de banda y control de interferencias. Además, permite ampliar cobertura mediante múltiples puntos de acceso con roaming entre celdas.

Sus limitaciones principales son la dependencia de infraestructura y el punto único de fallo: si el AP deja de funcionar, los clientes pierden conectividad incluso entre sí. Por ello, este modo es ideal para redes permanentes, seguras y administradas centralmente [22].

3.1.2. Modo Independiente o Ad-Hoc

El modo ad-hoc elimina la necesidad de un punto de acceso: los dispositivos se conectan directamente entre sí, formando una red descentralizada donde cada nodo puede no sólo enviar y recibir, si no que también reenviar información. Dentro del conjunto de tipos de redes ad-hoc, se va a estar haciendo foco a las MANET (*Mobile Ad-Hoc Networks*), relevantes para escenarios con nodos móviles.

Entre sus principales ventajas, destacan la baja latencia en comunicaciones de corto alcance (al no depender de que el tráfico pase por un enrutador central), favoreciendo aplicaciones que requieren respuestas rápidas. A su vez, también destacan en la capacidad de establecer enlaces inmediatamente en entornos sin infraestructura disponible, como misiones de rescate, zonas rurales o despliegues temporales. Al no requerir puntos de acceso ni cableado, representan una solución económica y flexible, capaz de extender la cobertura de un nodo conectado a Internet, ofreciendo comunicación inmediata. Su naturaleza descentralizada elimina el punto único de fallo, permitiendo que los nodos ajusten de manera autónoma sus enlaces y rutas, lo que otorga resiliencia frente a fallos individuales.

Sin embargo, a medida que aumenta el número de dispositivos conectados, se incrementa la complejidad en la gestión del tráfico, la latencia de las comunicaciones y la posibilidad de interferencias. Este efecto se presenta en ambos modos de operación, aunque se vuelve especialmente crítico en ad-hoc debido a la ausencia de un punto central que gestione el canal.

Las redes ad-hoc presentan vulnerabilidades de seguridad al carecer de un mecanismo centralizado de autenticación y, además, no todos los dispositivos soportan este modo de operación. La estabilidad general de la red queda fuertemente condicionada por el *hardware* de cada nodo, de modo que cualquier dispositivo puede transformarse en un eslabón crítico según la topología física. En particular, un nodo con baja potencia o con una antena deficiente puede degradar el rendimiento global, por lo que, para alcanzar un desempeño consistente, suele ser necesario actualizar el *hardware* de todos los nodos de manera homogénea. En contraste, cuando el tráfico se centraliza en un único punto de acceso, este suele saturarse primero y convertirse en el principal responsable de los cuellos de botella de la red. En estos casos, actualizar únicamente el punto de acceso suele ser suficiente para mejorar de forma significativa el desempeño general [23] [24] [25] [26].

3.2. Protocolos de Ruteo

En una red ad-hoc, como ya se mencionó anteriormente, los dispositivos se conectan directamente entre sí sin infraestructura fija. Sin embargo, no

Capítulo 3. Red Inalámbrica

todos los nodos están dentro del alcance inalámbrico directo unos de otros, por lo que si un nodo quiere enviar datos a otro que está fuera de su alcance, necesita usar otros nodos como “puentes”. Esos nodos intermedios reciben el paquete y lo reenvían, paso a paso, hasta que llegue al destino.

El modo ad-hoc, por sí solo, no gestiona automáticamente este salto de nodo en nodo (sólo es un modo de enlace directo entre dispositivos Wi-Fi). Para lograrlo, es necesario implementar lo que se conoce como un protocolo de ruteo, responsable de descubrir, seleccionar y mantener la información de encaminamiento entre nodos que no están directamente conectados. Sin esta capa de ruteo, muchos dispositivos no podrían comunicarse dentro de la red.

3.2.1. Clasificación de los Protocolos de Ruteo

La clasificación de los protocolos de ruteo en redes MANET permite comprender las distintas estrategias adoptadas para gestionar el ruteo de paquetes en entornos dinámicos, donde la topología cambia constantemente. Una de las divisiones más aceptadas distingue tres enfoques principales según la forma en que los nodos adquieren y mantienen la información de rutas.

Los protocolos proactivos mantienen tablas de ruteo constantemente actualizadas hacia todos los destinos, garantizando baja latencia en la transmisión a costa de una sobrecarga continua en la red. En contraste, los protocolos reactivos establecen rutas solo cuando es necesario enviar datos, reduciendo el consumo de recursos pero introduciendo demoras iniciales en la comunicación. Finalmente, los protocolos híbridos combinan ambos enfoques, empleando estrategias proactivas dentro de zonas locales y reactivas entre zonas, buscando equilibrar latencia y consumo de recursos.

Además de esta clasificación principal, existen enfoques complementarios que incorporan características adicionales según el contexto de aplicación. Algunos protocolos adoptan estructuras jerárquicas o geográficas para mejorar la escalabilidad y reducir la sobrecarga, mientras que otros priorizan la eficiencia energética, la tolerancia a fallos o la transmisión a múltiples destinos. Esta variedad de enfoques refleja la complejidad del ruteo en redes MANET, donde la elección del protocolo depende de un balance entre robustez, movilidad y consumo de recursos [27].

3.2.2. Problemas en Protocolos de Ruteo MANET

El diseño de protocolos de ruteo para redes MANET enfrenta varios desafíos derivados de su naturaleza móvil y descentralizada. Uno de los principales problemas es la gran cantidad de mensajes que los nodos deben intercambiar para descubrir y mantener rutas. Cuando demasiados nodos transmiten al mismo tiempo, el canal inalámbrico se congestiona y aumenta el consumo de energía, algo especialmente crítico en dispositivos alimentados

3.3. B.A.T.M.A.N.-Advanced

por batería. Otro inconveniente aparece cuando un nodo intenta reenviar datos hacia el vecino que considera “más prometedor”, por ejemplo, por estar más cerca del destino. Si en ese momento no tiene vecinos adecuados, la comunicación puede quedar bloqueada o perderse.

La ausencia de una estructura jerárquica para organizar direcciones también dificulta que la red escale. Como cada nodo mantiene su propia parte de la información de ruteo, todos deben actualizarse constantemente para reflejar los cambios de topología, lo que genera más tráfico de control y mayor procesamiento.

A esto se suma que mantener rutas activas y enviar mensajes de actualización consume energía de forma continua, lo que limita el tiempo operativo de los dispositivos móviles. Además, factores como interferencias entre transmisiones cercanas o la concentración excesiva de tráfico en ciertos nodos pueden reducir la estabilidad general y el rendimiento de la red.

3.3. B.A.T.M.A.N.-Advanced

El protocolo B.A.T.M.A.N. (*Better Approach To Mobile Adhoc Networking*) es un protocolo de ruteo proactivo diseñado para redes inalámbricas ad-hoc con topología Mesh (configuración de red donde cada dispositivo está interconectado a la mayoría o a todos los demás nodos) especializado en redes MANETs. A diferencia de muchos otros protocolos que buscan construir una visión completa de la topología de la red, B.A.T.M.A.N. adopta un enfoque distribuido y simplificado: en lugar de saber toda la ruta hacia un destino, cada nodo sólo necesita saber a qué vecino enviar un paquete para que éste continúe su camino.

La idea central es que cada nodo mantenga información sobre el mejor próximo salto (“*best next hop*”) hacia un destino, sin almacenar ni calcular la ruta completa. Este diseño reduce la complejidad y el volumen de información que cada nodo debe manejar, facilitando la escalabilidad y la adaptación a cambios dinámicos en la red.

El protocolo ha evolucionado a lo largo de distintas versiones, entre ellas B.A.T.M.A.N. IV y B.A.T.M.A.N. V, las cuales introducen mejoras tanto en los criterios de selección del mejor próximo salto como en los mecanismos de estimación de la calidad de los enlaces. En particular, B.A.T.M.A.N. V incorpora métricas más refinadas basadas en el rendimiento efectivo del enlace observado, superando enfoques puramente estadísticos. En sus implementaciones más recientes, se encuentra B.A.T.M.A.N. Advanced (*batman-adv*), la cual fue la versión escogida, integrada como un módulo del kernel de Linux (ejecutando internamente el protocolo B.A.T.M.A.N. V internamente), trabajando a nivel 2 (capa de enlace) en lugar de nivel 3 (IP), a diferencias de su versión anterior (*batmand*), lo que le permite funcionar de forma transparente,

Capítulo 3. Red Inalámbrica

como si todos los nodos estuvieran en la misma red local. Su diseño busca abstraer la complejidad del ruteo IP al trabajar directamente con tramas Ethernet, permitiendo una mayor transparencia en el reenvío de paquetes y facilitando la integración con protocolos y servicios que asumen una red local, como ARP o DHCP. La información presentada sobre el protocolo fue tomada íntegramente de la documentación oficial [9].

3.3.1. Integración en el Kernel de Linux

B.A.T.M.A.N. reconoce el coste computacional de encargarse del enrutamiento del tráfico de datos. Para ello ofrece como solución integrarse en el kernel de Linux (sin pasar por el espacio de usuario).

A modo de contexto, en Linux se distinguen dos espacios de ejecución, el *kernel space*, el cuál controla el *hardware* y las comunicaciones, y el *user space*, que es donde se alojan las aplicaciones, sin acceso directo al *hardware*.

Al operar en el kernel, batman-adv actúa como un driver de red interno, interceptando las tramas directamente desde la interfaz física y reenviándolas sin salir del núcleo del sistema. Esto elimina llamadas al sistema y copias innecesarias, reduciendo la carga de CPU y mejorando la eficiencia en dispositivos con recursos limitados. Si batman-adv no estuviera en el kernel, no podría operar realmente en capa 2, porque desde el espacio de usuario no se tiene acceso directo ni eficiente a las tramas Ethernet.

3.3.2. Asignación de IPs e Interfaz

El protocolo batman-adv crea una interfaz virtual llamada `bat0`, que actúa en la capa de enlace (capa 2). Usar esta interfaz virtual para la comunicación permite que todo el tráfico que circula, sea examinado por el módulo de batman-adv para determinar, en base a la MAC objetivo, la ruta a usar, vinculándola con una de las interfaces físicas (por ejemplo `wlan0`, `eth0`), para que la use como medio de transmisión.

Para su funcionamiento, basta con asignar una dirección IP a la interfaz `bat0`, la cual va a ser usada para la comunicación entre los nodos. La interfaz física no necesita IP, y si la tiene, solo sirve para comunicación directa entre vecinos (en la implementación realizada, se optó por configurar una red `wlan0` para pruebas directas entre vecinos, pero como ya se mencionó, se puede quitar sin comprometer el funcionamiento de la red). El protocolo encapsula y reenviar todo el tráfico IP entre nodos según la topología y la calidad de los enlaces a través de `wlan0`, usándolo como medio físico para enviar las tramas por la red.

Éste maneja principalmente dos tipos de tramas: las tramas de datos encapsuladas, que transportan la información generada por las aplicaciones, y los mensajes de control, encargados de mantener actualizada la topología

de la red y calcular las rutas óptimas. A continuación, se detallan ambos mecanismos.

3.3.3. Encapsulamiento de los mensajes

Cuando un paquete se origina en una aplicación, atraviesa las capas del modelo OSI [28] de la forma habitual. Sin embargo, al utilizar batman-adv, la trama Ethernet resultante no se envía directamente por la interfaz física, sino que pasa primero por la interfaz virtual `bat0`, donde batman-adv la intercepta y la encapsula dentro de una nueva trama.

Este agrega su propio encabezado, que incluye información sobre el siguiente salto, métricas de enlace y otros campos de control. Luego, la trama completa es reencapsulada dentro de una nueva trama Ethernet física, utilizada para transmitirla hacia el siguiente nodo del mallado mediante la interfaz inalámbrica subyacente, que representa la capa física del sistema. Se representa el encapsulado en la siguiente Figura 3.1.

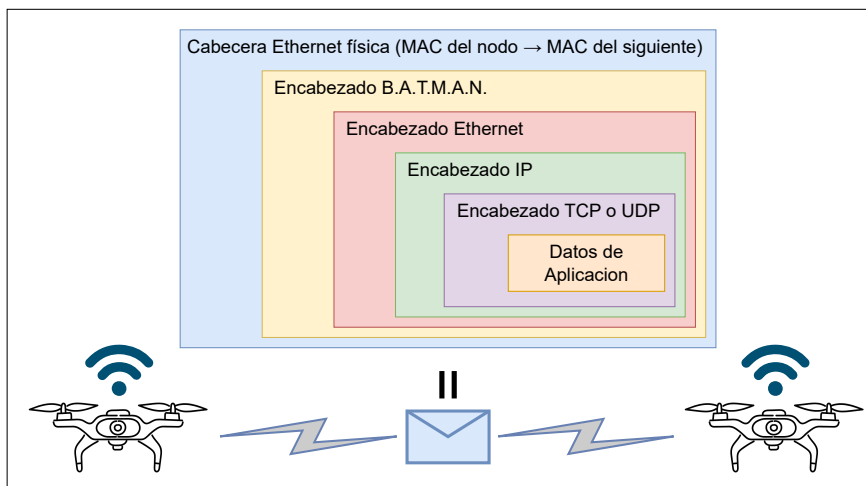


Figura 3.1: Mensajes encapsulados con la trama de batman-adv.

Cuando otro nodo del mesh recibe la trama, elimina el encabezado Ethernet física del enlace, analiza el encabezado de batman-adv para determinar el siguiente salto, y vuelve a encapsular la trama para continuar el reenvío. Finalmente, cuando la trama llega al nodo destino, batman-adv elimina su encabezado y entrega la trama Ethernet original al sistema operativo, como si hubiera llegado directamente de un enlace local.

De esta manera, se logra que todos los nodos crean estar conectados a una misma red local, ocultando la topología real y gestionando el ruteo de forma transparente a nivel de capa de enlace. El encabezado de batman-adv no reemplaza el encabezado Ethernet tradicional, sino que añade una capa

Capítulo 3. Red Inalámbrica

adicional de encapsulamiento entre la capa de enlace lógica y la capa de enlace física.

3.3.4. Mensajes de Control

En batman-adv, los nodos de la red mallada mantienen una visión dinámica de la conectividad mediante el intercambio periódico de mensajes de control. Estas tramas especiales, definidas por el propio protocolo, no contienen datos de usuario: su único propósito es descubrir vecinos, evaluar la calidad de los enlaces y coordinar el funcionamiento interno de la malla.

El mensaje fundamental es el OGMv2 (*Originator Message* versión 2), que cada nodo emite regularmente para anunciar su presencia. Su propagación a través de la red permite que los nodos conozcan qué caminos existen hacia cada originador y qué vecinos son responsables de reenviar esta información.

También se emplea ELP (*Echo Location Protocol*) para medir la calidad del enlace entre vecinos directos. A diferencia de los OGMv2, los paquetes ELP no se propagan: viajan solo entre nodos adyacentes. Esto evita sobrecarga innecesaria y permite obtener mediciones más frecuentes y precisas sobre el enlace local.

Además, el protocolo incluye las extensiones TVLV (*Type-Version-Length-Value*), un mecanismo flexible que permite añadir información adicional sin alterar el formato principal. Entre estas extensiones se encuentran los anuncios de *gateway*, VLANs y las actualizaciones de la *Translation Table* (TT). Esta tabla mantiene, de forma distribuida, la asociación entre cada cliente no-mesh conectado a la red y el nodo mesh que lo aloja. Gracias a un sistema de versionado, cada nodo puede solicitar únicamente los cambios de la TT en lugar de recibir la tabla completa, reduciendo significativamente la carga de control.

En conjunto, estos mecanismos permiten que batman-adv mantenga una red coherente y adaptable sin requerir una tabla de enrutamiento global, distribuyendo la información de forma eficiente y evitando la sobrecarga típica de los protocolos de capa 3.

3.3.5. Métrica del Canal

La versión V del protocolo B.A.T.M.A.N. introdujo una métrica de canal basada en la estimación del *throughput* efectivo entre nodos, reemplazando el enfoque previo que se limitaba a contar paquetes recibidos. Si bien herramientas de usuario conservan por compatibilidad la denominación histórica “tq” (Transmit Quality), el valor reportado corresponde al valor calculado a partir de B.A.T.M.A.N. V. Este nuevo enfoque busca representar de forma más realista el rendimiento de un enlace inalámbrico, incorporando no sólo la

recepción de paquetes sino también la capacidad real del canal para transportar datos útiles.

La estimación del canal se obtiene mediante ELP, mecanismo que intercambia pequeños paquetes entre vecinos directos con el fin de medir localmente el *throughput* de cada enlace. A partir de esos valores, cada nodo origina un OGMv2 por cada vecino, de modo que cada mensaje parte con un valor inicial distinto, correspondiente exclusivamente al enlace directo hacia ese vecino. Antes de incorporar esa medición en el OGMv2, batman-adv aplica una normalización destinada a suavizar fluctuaciones, incluida una penalización del 50 % para estimaciones superiores a 1 Mbit/s, con el fin de evitar que mediciones puntualmente altas generen rutas inestables.

Cuando un nodo recibe un OGMv2, compara el valor del *throughput* acumulada que trae el mensaje con la estimación local del enlace por el cual fue recibido. Si ésta última es menor, ese valor reemplaza al contenido en el OGMv2, reflejando que el enlace actual constituye un cuello de botella potencial. A continuación se aplica una penalización por salto, de modo que la métrica nunca aumenta durante la propagación. El resultado es una métrica que converge naturalmente hacia el rendimiento del peor enlace del camino.

En síntesis, el sistema implementado en B.A.T.M.A.N. V se basa esencialmente en medir localmente el *throughput* de cada enlace, y luego permitir que esa métrica se degrade progresivamente a medida que los OGMv2 atraviesan múltiples saltos. El resultado final es una aproximación realista del rendimiento del camino, guiada por el enlace más restrictivo y ajustada dinámicamente al comportamiento inalámbrico real. Esta estrategia hace que la selección de rutas no dependa de valores idealizados o teóricos, sino de mediciones realizadas en tiempo real, y asegura que la red se adapte rápidamente a cambios en la calidad de los enlaces, movilidad de los nodos o fluctuaciones de interferencia.

Por lo tanto los desafíos que afronta batman-adv en cuanto al ruteo en redes MANET se alinean con los requisitos particulares de nuestro proyecto, especialmente en lo referente a movilidad, resiliencia, compatibilidad y escalabilidad. Aunque no es el protocolo más eficiente energéticamente, su balance entre simplicidad, robustez y autonomía lo hace muy adecuado para un proyecto real con enjambres de drones móviles.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 4

Operaciones con un Dron

Habiendo analizado la plataforma de *hardware* en el Capítulo 2, este capítulo abordará el desarrollo del *software* requerido para habilitar el control del dron, tanto en modo radiocontrolado como en modalidad autónoma. Se denomina modalidad autónoma a la capacidad del dron de seguir una ruta por puntos objetivo sin requerir de la intervención del radiocontrol. Para ello, se presentan las herramientas de *software* involucradas y los fundamentos necesarios para su integración. Primero se introducen los componentes indispensables para el vuelo por radiocontrol y, posteriormente, aquellos necesarios para la ejecución de vuelos autónomos, considerando tanto la validación inicial en entornos de simulación como su posterior implementación en pruebas reales.

4.1. Herramientas de Software

En general, los drones requieren *software* de control en tierra para configurar la controladora de vuelo, calibrar sensores, ajustar el sistema del radiocontrol, gestionar la telemetría, los modos de vuelo y la navegación por GPS. Estas aplicaciones están disponibles para PC, móviles y diversos sistemas operativos. Entre las más conocidas se encuentran *Mission Planner* y *QGroundControl* [29] [30].

Los *firmwares* para controladoras de vuelo de drones más populares son PX4 y Ardupilot. Estos se comunican con las aplicaciones a través del protocolo MAVLink (*Micro Air Vehicle Link*). Este protocolo es un estándar ampliamente adoptado en vehículos aéreos no tripulados y sistemas robóticos, diseñado para ser liviano y eficiente en plataformas con recursos computacionales y de comunicación limitados. Este protocolo combina un mecanismo de transmisión continua de telemetría basado en un modelo de publicación y suscripción, mediante el cual los módulos del sistema envían datos como actitud, posición GPS o estado del vehículo. Además utiliza intercambios punto a punto utilizados para operaciones que requieren

Capítulo 4. Operaciones con un Dron

confirmación, tales como la lectura y escritura de parámetros del *firmware*, el envío de planes de misión o la ejecución de comandos específicos. La estructura y semántica de los mensajes se definen mediante archivos XML denominados dialectos, a partir de los cuales se generan automáticamente bibliotecas para múltiples lenguajes de programación, asegurando así que cualquier componente del sistema (ya sea el vehículo, una computadora a bordo o una estación de control en tierra) pueda intercambiar información de forma coherente, interoperable y extensible [31].

El equipo de desarrollo de PX4 sugiere el uso de *QGroundControl* (opción que se utiliza en este proyecto), mientras que el equipo de *ArduPilot* prefiere *Mission Planner*. *QGroundControl* ofrece control total de vuelo y planificación de misiones, está pensado tanto para profesionales como para desarrolladores, y al ser de código abierto, permite modificarlo y colaborar en su desarrollo libremente [32] [33]. Contar con alguna de estas aplicaciones es requisito mínimo para poder realizar la configuración y calibración de sensores y realizar un vuelo manual con el radio control. En el repositorio del proyecto se deja un manual para configurar y calibrar los drones utilizados.

Vuelo Autónomo: para habilitar vuelos autónomos, es necesario incorporar herramientas de *software* que permitan, por un lado, establecer una interfaz de comunicación entre la computadora a bordo y la controladora de vuelo, y por otro, disponer de un entorno de simulación que posibilite validar las rutinas programadas antes de su ejecución en campo. Tener una computadora a bordo permite ejecutar rutinas de vuelo autónomo mediante alguna API basada en lenguaje de alto nivel. Por ejemplo, si se desea que un dron despegue, se dirija a una posición objetivo y posteriormente regrese al punto de partida, es necesario contar con funciones o métodos capaces de enviar estas órdenes utilizando un protocolo comprensible para la controladora de vuelo. Para este propósito se utiliza MAVSDK, una colección de bibliotecas que implementan el protocolo MAVLink y habilitan la interacción programática con sistemas basados en PX4. MAVSDK es multiplataforma y proporciona APIs en diversos lenguajes, entre ellos Python, C++, Go y Rust, lo que facilita su integración en distintas arquitecturas de *software* y permite desarrollar comportamientos autónomos de forma modular y escalable [10]. En particular, MAVSDK-Python proporciona un cliente que se comunica con un servidor escrito en C++, el cual se comunica directamente con las controladoras de vuelo. Esta arquitectura permite estructurar misiones autónomas mediante programación asíncrona con la librería de Python *asyncio*, lo que resulta esencial para manejar flujos concurrentes de telemetría, sensores y comandos durante el vuelo[34]. La API expone de forma accesible funcionalidades como despegue, aterrizaje, navegación por *waypoints* y lectura de datos en tiempo real.

Herramientas de Simulación: la validación previa en simulación es indispensable para reducir riesgos operativos y depurar algoritmos antes de

4.2. Experimentos

su ejecución en drones reales. Para ello se utilizan herramientas compatibles con PX4 en modalidad SITL (*Software In The Loop*), donde el *firmware* de la controladora de vuelo se ejecuta en un entorno virtual que replica el comportamiento físico del vehículo [35]. En este escenario, el dron simulado puede ser controlado por MAVSDK, de la misma forma que ocurriría con un dron físico.

El entorno de simulación se implementa utilizando Gazebo, un simulador de robótica 3D ampliamente adoptado que permite modelar vehículos, sensores y escenarios con distintos niveles de detalle[36]. Gazebo posee la capacidad para ejecutar simulaciones complejas con múltiples robots y entornos extensos. Se caracteriza por su motor de física de alta fidelidad, su soporte para modelos en formato SDF (*Simulation Description Format*), que describe robots, modelos del mundo y entornos completos para la simulación. En su versión moderna, Gazebo Ignition introduce una arquitectura modular basada en sistemas y componentes, junto con un *middleware* de transporte que facilita el intercambio eficiente de datos entre módulos del simulador. A través de SITL, PX4 interactúa con Gazebo enviando y recibiendo información de estado mediante interfaces estandarizadas, comportándose exactamente igual que si estuviera conectado a *hardware* real. Esto habilita la ejecución de rutinas de vuelo, pruebas de navegación, experimentación con sensores virtuales y validación de comandos autónomos en un entorno controlado y reproducible [37].

En resumen: PX4 SITL simula el *firmware* de la controladora de vuelo. Gazebo, nos provee una interfaz gráfica de simulación con un dron donde podemos ver sus movimientos. Con MAVSDK generamos código para interactuar con PX4 SITL, de manera adicional, podemos vincular PX4 SITL a *QGroundControl*, para ver en otra interfaz gráfica, por ejemplo la orientación del dron y el recorrido en tiempo real representado en un mapa geográfico. Este entorno de simulación se adquiere desde un repositorio de PX4 y constituye la base para desarrollar, ejecutar y evaluar los programas creados con MAVSDK-Python, permitiendo observar el desempeño de las misiones, depurar posibles fallos y preparar los algoritmos para su posterior validación en pruebas reales. La Figura 4.1 ilustra la interacción entre PX4 SITL, Gazebo, *QGroundControl* y MAVSDK. Además, dejamos en el repositorio un manual para instalar todas estas herramientas y ejecutar un ejemplo básico.

4.2. Experimentos

Previo a realizar un vuelo (ya sea radiocontrolado o autónomo) es necesario realizar la configuración del *firmware* y la calibración de los sensores tanto internos como externos del dron. Para esto redactamos un manual que se encuentra en el repositorio del proyecto. En él, explicamos los pasos

Capítulo 4. Operaciones con un Dron

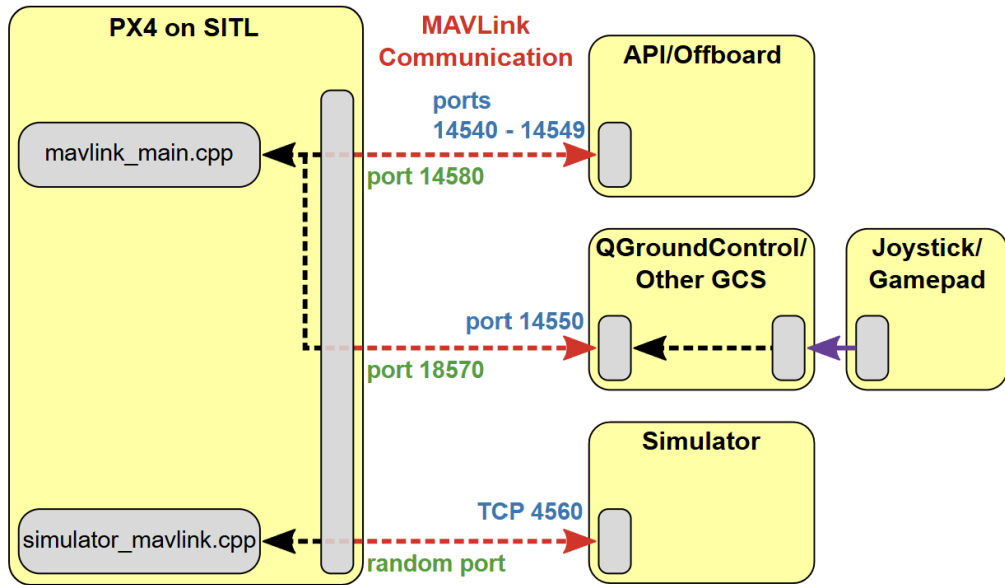


Figura 4.1: Arquitectura general utilizada para el desarrollo y validación de vuelos autónomos mediante PX4 SITL, Gazebo y MAVSDK [35].

necesarios para configurar y calibrar un dron antes de ponerlo a volar. Además, proporcionamos una explicación detallada sobre qué estamos configurando y por qué es importante hacerlo, junto a los fundamentos principales de manejo para ejecutar vuelos radiocontrolados. Los vuelos radiocontrolados son importantes para validar la estabilidad de la aeronave, y dar el paso a las pruebas de vuelo autónomo.

El desarrollo de rutinas de vuelo autónomo requiere comprender qué son los modos de vuelo de los drones que utilizan el *firmware* PX4 y en particular qué es el modo de vuelo *Offboard*. Los modos de vuelo definen cuánto controla el piloto y cuánto la controladora de vuelo. Los modos autónomos funcionan sin intervención humana y permiten automatizar tareas como despegue, regreso al origen, aterrizaje o la ejecución de misiones programadas. Por su parte, los modos manuales, son operados por el piloto con asistencia de la controladora, ofreciendo desde vuelos estables hasta opciones acrobáticas según el modo seleccionado (ver Apéndice B para más detalles). En particular, uno de los modos autónomos es el modo *Offboard*, el cual permite controlar el dron enviando comandos de movimiento en tiempo real desde un sistema externo, lo que habilita la generación dinámica de posiciones, velocidades o actitudes durante la misión. A este modo es posible acceder mediante la librería MAVSDK-Python a través de distintos métodos. Una vez habilitado este modo, el dron responde a los comandos enviados de forma continua hasta que el modo se detiene o se activa un modo de vuelo seguro, como por ejemplo *Hold*.

4.2. Experimentos

Dentro de los métodos disponibles, uno de los más utilizados en el proyecto permite establecer un punto objetivo en el sistema de coordenadas NED (*North-East-Down*), en términos de desplazamientos en los ejes Norte (X), Este (Y) y abajo (Z), junto con un ángulo de rotación respecto al eje Z denominado *Yaw*, lo que habilita el control preciso de trayectorias tridimensionales (más detalles en la sección del Apéndice B.1.1).

En general, el tipo de programas que utilizan estos métodos ejecutan un flujo principal con tareas en paralelo que es lo que permite la librería *asyncio*, en la que se basa MAVSDK-Python. A modo de ejemplo ilustrativo, en este proyecto las rutinas autónomas implementadas integran estas capacidades en un código tanto para drones simulados como para drones físicos.

A continuación se explica el flujo de funcionamiento de la implementación de este código ilustrado en la Figura 4.2. Comienza con la configuración del entorno de registro (datos como posiciones GPS, alturas o cambios en modos de vuelo para posterior procesamiento), la solicitud y validación de parámetros y la inicialización de las tareas de telemetría de forma concurrente.

Dicho programa permite seleccionar de forma interactiva una figura geométrica para ser ejecutada por el dron (línea recta, círculo o cuadrado), así como los parámetros asociados: altitud de vuelo y dimensión característica (distancia, radio o longitud del lado). El sistema se conecta con el dron (real o simulado) mediante la clase *System* de MAVSDK, encargada de intercambiar mensajes MAVLink con la controladora de vuelo ya sea simulada o real. Mientras las tareas de telemetría capturan datos en paralelo, el hilo principal del programa espera estimaciones de posición global y local confiables, realiza el armado (encendido de los motores) y ejecuta el despegue. A continuación, se habilita el modo *Offboard* y se envían las instrucciones de movimiento que permiten al dron trazar la figura seleccionada. Una vez completado el patrón, las tareas de telemetría son canceladas y el dron recibe la orden de regresar al punto de origen, finalizando así la misión autónoma.

Capítulo 4. Operaciones con un Dron

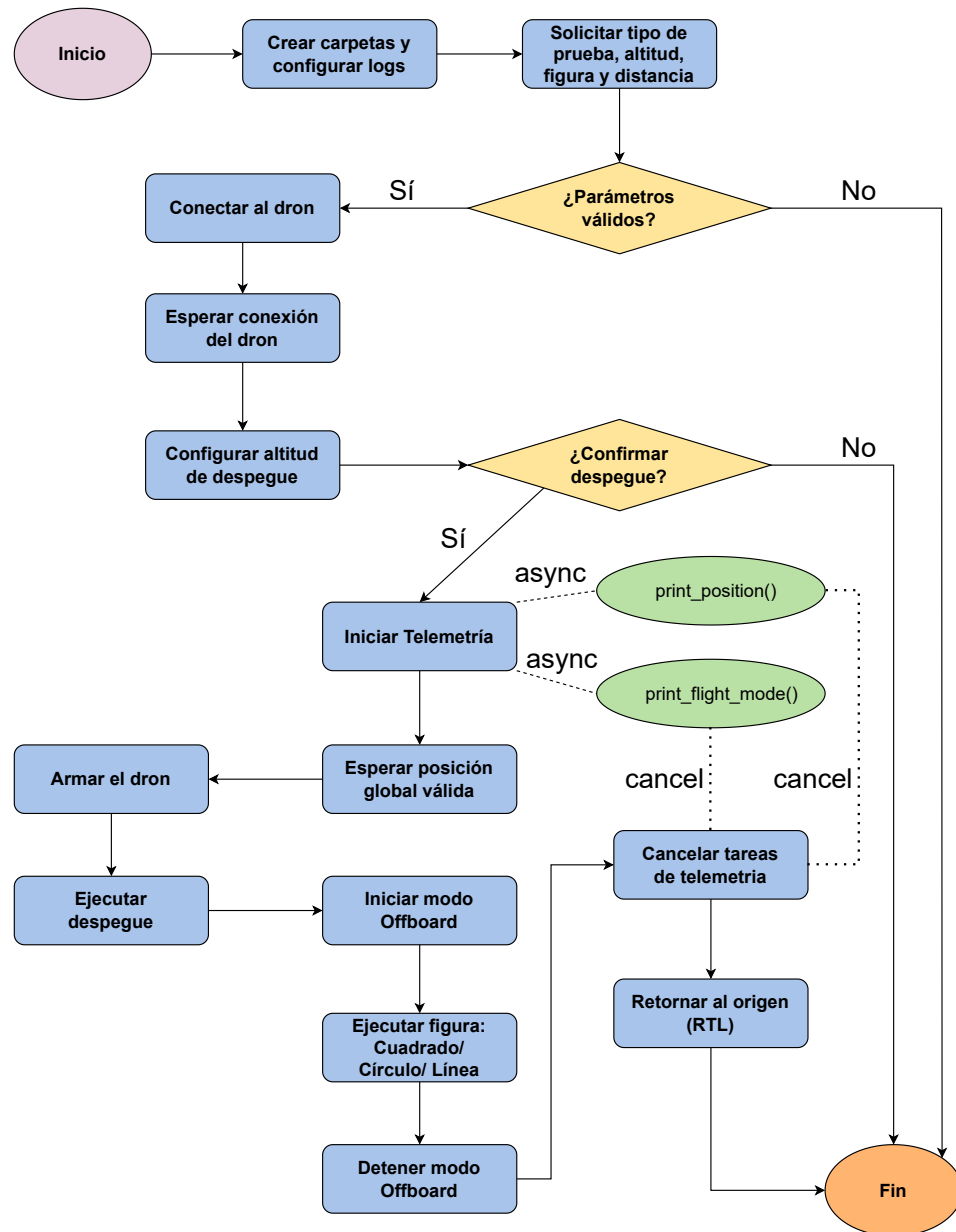


Figura 4.2: Diagrama de flujo de código ejemplo.

4.3. Validación en Entorno Simulado

Previo a realizar la prueba en los drones reales, es de buena práctica realizar una simulación con las herramientas mencionadas en la Sección 4.1. Se muestra a continuación el ejemplo explicado en la sección previa.

4.3. Validación en Entorno Simulado

```
=====
Rutina Interactiva de Vuelo de Figuras con Dron
=====

Seleccione el tipo de dron ('s' para simulado, 'r' para real): s
Ingrese la altura de despegue (entre 1 y 10 metros): 8
Ingrese la figura a volar ('cuadrado', 'circulo' o 'linea'): cuadrado
Ingrese el valor de distancia (lado para cuadrado y linea (entre 1m
Waiting for drone to connect...
Drone connected
¿Are you sure you want to takeoff? (s/n): s
Waiting for drone to have a global position estimate...
Flight mode: HOLD
-- Global position estimate OK
-- Arming
-- Taking off 8.0 meters
Flight mode: TAKEOFF
Flight mode: HOLD
-- Setting initial offboard setpoint
-- Starting offboard
Starting square pattern with side 10.0 m at 8.0 m altitude
-- Square: Moving 10.0 m North
Flight mode: OFFBOARD
-- Square: Moving 10.0 m East
-- Square: Moving 10.0 m South
-- Square: Moving 10.0 m West
-- Stopping offboard
-- Returning to launch (RTL)
```

Figura 4.3: Mensajes por consola del código ejemplo que permiten dar seguimiento al flujo del programa.

En primer lugar, se lanza el simulador PX4 SITL, al hacerlo se abre junto a este una ventana de Gazebo que muestra el dron simulado y luego es necesario abrir *QGroundControl*. Luego, se ejecuta el código ejemplo. En la Figura 4.3 se observa la interacción del código con el usuario a través de mensajes por consola. Para esta prueba en particular, se seleccionó una trayectoria en forma de cuadrado, con lados de 10 m y a una altura constante de 8 m. Durante la ejecución, se muestran mensajes de monitoreo que permiten verificar el estado del sistema a lo largo del experimento. Otra forma de realizar el seguimiento en tiempo real consiste en visualizar los movimientos del dron simulado directamente en Gazebo y observar simultáneamente el seguimiento de la misión en el mapa de *QGroundControl*, como se ilustra en la Figura 4.4. Esto permite corroborar que la trayectoria ejecutada es coherente con la prueba definida.

Como se mencionó en la Sección 4.2, durante la prueba se registran datos de telemetría en archivos de texto. Uno de esos archivos contiene los datos relevados por el GPS en columnas con: Tiempo (s), Latitud (°), Longitud (°), Altitud (m) graficados en la Figura 4.5. En la gráfica de Latitud vs Longitud se puede observar que en el plano horizontal, la trayectoria realizada por el dron es coherente con la trayectoria planificada. No obstante, para realizar un análisis preciso de las distancias, es necesario tener en cuenta que las coordenadas geográficas están expresadas en grados. Por lo tanto,

Capítulo 4. Operaciones con un Dron

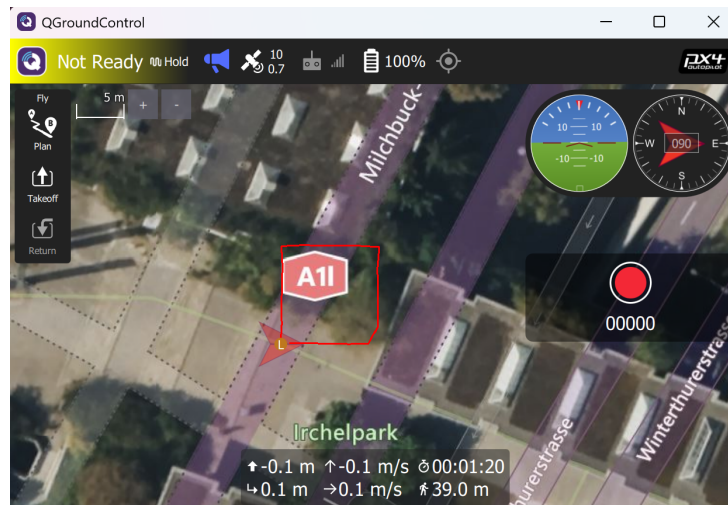


Figura 4.4: Datos del GPS visualizadas en *QGroundControl* al finalizar el cuadrado de lado 10m.

se requiere una conversión a unidades métricas para poder cuantificar la longitud de los segmentos de la trayectoria. En las gráficas de la Figura 4.5, se utiliza una conversión de grados a metros para los datos de latitud y longitud representándolos como distancias métricas respecto al primer dato obtenido como punto de referencia.

Las gráficas de latitud, longitud y altura relativas permiten analizar el comportamiento del dron a lo largo del tiempo. Observando la altura, se identifica claramente la fase de despegue y la posterior estabilización en la cota definida para la prueba. Por su parte, las curvas de latitud y longitud muestran cómo el vehículo va alcanzando cada uno de los puntos de la trayectoria: primero un desplazamiento en una dirección (Norte/Sur), luego en la dirección ortogonal (Este/Oeste), y así sucesivamente, describiendo el patrón cuadrado programado.

El análisis cuantitativo confirma que el dron ejecutó la trayectoria prevista con una precisión acorde al contexto de la simulación. La altura relativa alcanzó un máximo de 8,092 m, con una desviación de apenas 10 cm respecto al valor nominal, dentro de lo esperable para un sistema de control en lazo cerrado. En cuanto a la latitud y la longitud, las diferencias entre los valores máximos y mínimos registrados fueron de aproximadamente 10,7 m en cada eje, coincidiendo con la magnitud prevista para los lados del cuadrado programado. El ligero sobrepase respecto a los 10 m ideales (del orden de 7%) puede atribuirse a oscilaciones normales en el proceso de navegación, ya que el vehículo tiende a cruzar levemente cada punto objetivo antes de estabilizarse. Estas variaciones no representan un error significativo, sino un efecto inherente a la dinámica del control y a la forma en que se registran los valores extremos de posición.

4.4. Pruebas de Campo y Resultados

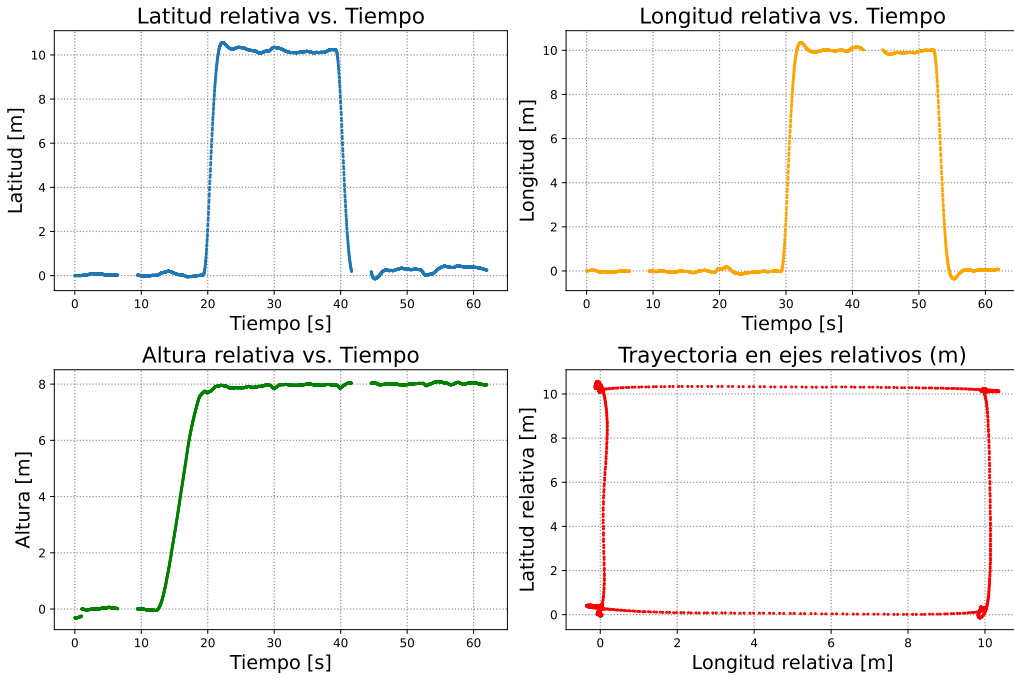


Figura 4.5: Datos del GPS graficados al finalizar el cuadrado de lado 10m y altura 8m. Latitudes y longitudes se expresan en metros relativos al primer dato como punto de referencia

4.4. Pruebas de Campo y Resultados

Luego de la validación en el entorno simulado, se procedió a replicar el mismo tipo de rutina con los drones reales.

En esta sección analizamos los datos correspondientes a la prueba representada en las gráficas de la Figura 4.6. Para este experimento se programó nuevamente una trayectoria en forma de cuadrado, pero reduciendo las dimensiones para mayor seguridad: una altura objetivo de 4 m y lados de 4 m. Esta elección conservadora se debió a que estas fueron las primeras validaciones en vuelo real, y era necesario comenzar con maniobras acotadas para evaluar la confiabilidad de la plataforma.

La trayectoria planificada comenzaba con un desplazamiento hacia el Este, seguido por un movimiento hacia el Sur, luego hacia el Oeste y finalmente el retorno al punto inicial. El análisis de las gráficas de la prueba real muestra que el dron ejecutó la secuencia de movimientos prevista de forma coherente con la rutina programada. La altura relativa al punto de origen alcanzó un máximo de 4.063 m, muy cercano a los 4 m deseados, mientras que los desplazamientos máximos en latitud y longitud fueron de 3.45 m y 3.87 m respectivamente, valores esperables para trayectorias tan cortas y limitadas por la precisión del GPS. La representación en el plano latitud-longitud exhibe una figura similar a un cuadrado, aunque menos precisa que en simulación debido a

Capítulo 4. Operaciones con un Dron

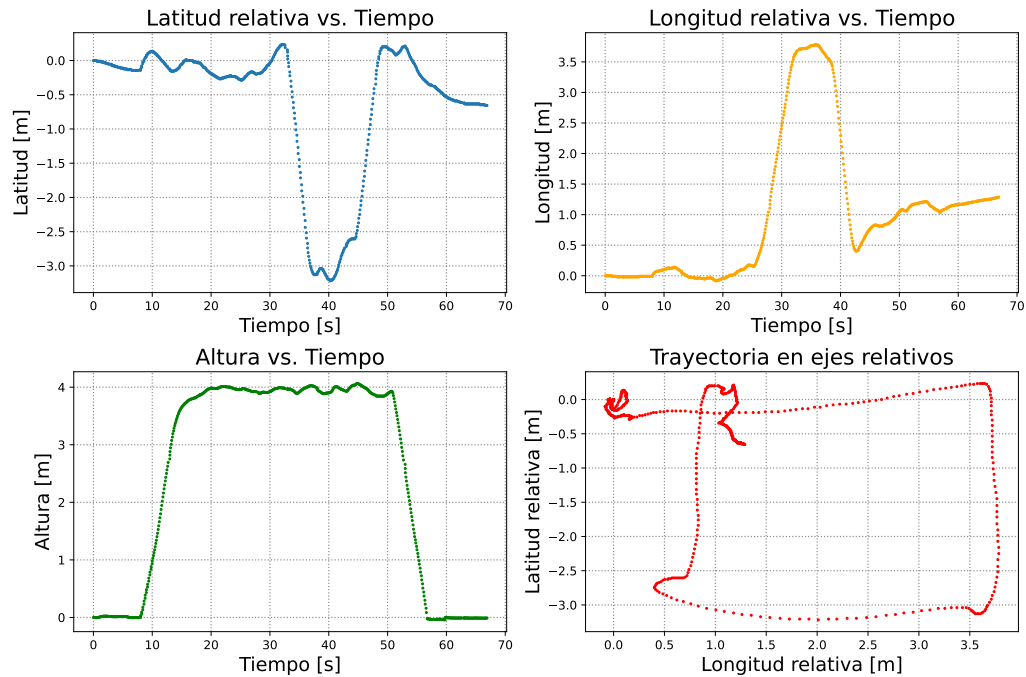


Figura 4.6: Datos GPS del dron durante la ejecución de la trayectoria cuadrada en entorno real. Latitudes y longitudes se expresan en metros relativos al primer dato como punto de referencia.

factores típicos del entorno real, como ruido de posicionamiento, dinámicas del controlador y rachas de viento. En este contexto, pequeñas fluctuaciones del orden de decenas de centímetros tienen un impacto proporcionalmente alto sobre lados de solo 4 metros. Aún así, la forma general y la secuencia temporal coinciden con lo planificado, por lo que la prueba se considera satisfactoria: el dron respondió correctamente a los comandos enviados mediante MAVSDK y las desviaciones observadas se encuentran dentro de lo razonable para vuelos autónomos basados en GPS. En la Figura 4.7 se observa uno de los drones volando en las pruebas realizadas durante esta etapa.

En síntesis, este capítulo estableció las bases de *software* necesarias para operar el dron tanto en forma radiocontrolada como autónoma, integrando herramientas de control, comunicación y simulación. Con estos fundamentos, el siguiente capítulo se centrará en la construcción de las rutinas de vuelo autónomo en enjambre y en su validación progresiva desde el entorno simulado hasta las pruebas reales en campo.

4.4. Pruebas de Campo y Resultados



Figura 4.7: Dron en vuelo de campo

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 5

Operaciones en Enjambre

Este capítulo describe el proceso de despliegue del enjambre, abarcando la configuración de las NUCs, la red ad-hoc y los componentes de software necesarios para coordinar múltiples drones de manera colaborativa. Se detallan las decisiones técnicas adoptadas, la organización interna del sistema y los mecanismos de comunicación que permiten el funcionamiento distribuido del enjambre.

5.1. Configuración de las NUCs y la Red

En el sistema implementado se distinguen dos tipos funcionales de drones: drones de tarea y drones de red. Los drones de tarea son aquellos destinados a ejecutar las misiones principales (como navegación, inspección o recolección de información) y dependen de la red para coordinarse con la estación terrestre o con otros agentes. Por otro lado, los drones de red no participan directamente en la misión, sino que actúan como nodos de infraestructura móvil, aumentando la cobertura y la estabilidad de la malla inalámbrica al servir como repetidores y soportes de reenvío.

Ambos tipos utilizan el mismo *hardware* y la misma configuración base; la distinción surge únicamente del rol asignado en cada experimento, lo que permite que cualquier dron pueda convertirse en nodo de red o de tarea según sea necesario.

A su vez, se designa un nodo para que asuma el rol de *Ground Station*, el cual va a ser el encargado de enviar las instrucciones de a donde debe dirigirse cada dron de red, y cuya identidad se debe pre-configurar en el resto de la flota. Dicho nodo no va a volar conjuntamente con el resto.

Independientemente del rol que tome cada nodo, a todos se les va a configurar que se comuniquen por una misma red inalámbrica ad-hoc, cargando el protocolo de batman-adv en cada uno. Esto implica que todos van a poder rutear los mensajes, al igual que van a poder comunicarse directamente si la

Capítulo 5. Operaciones en Enjambre

distancia lo permite.

Además de la red inalámbrica, se dispone de una red cableada utilizada para la administración remota de las NUCs mediante `ssh` y como puerta de salida a Internet. A través de esta interfaz cableada es posible mantener actualizados los paquetes del sistema operativo y desplegar configuraciones de forma centralizada. Gracias a que la *Ground Station* no estaba diseñada para participar en el vuelo, cumple además el rol *gateway*, actuando como puente entre ambos dominios. Por lo que en la configuración final se emplean tres dominios de red:

redes configuradas

```
bat0: 10.42.2.0/24
eth0: 10.42.1.0/24
wlan0: 10.42.0.0/24
```

Se configuraron rutas para la comunicación entre los distintos dominios de la red, aunque por defecto, el kernel de Linux no reenvía paquetes IP entre interfaces distintas. Para ello es necesario explicitar la habilitación del reenvío del tráfico IPv4 entre interfaces. Solo la *Ground Station* requiere esta configuración, ya que el reenvío interno dentro del *mesh* lo maneja *batman-adv* a nivel de capa 2.

En caso de desear que los nodos puedan acceder a Internet mediante este puente, es necesario aplicar traducción de direcciones (NAT: *Network Address Translation*), enmascarando todos los paquetes originados en la red 10.42.2.0/24 con la IP pública de la interfaz conectada a Internet del equipo conectado por cable a la *Ground Station*. Esto es necesario porque las direcciones privadas del *mesh* no son enrutables fuera del dominio local.

La configuración de cada NUC se automatiza en el arranque mediante la edición de la `crontab` del usuario `root` (`sudo crontab -e`), donde se programan los *scripts* responsables de levantar las interfaces de red, cargar *batman-adv* y lanzar los servicios necesarios. Se utilizó `crontab` en lugar de definir un servicio `systemd` porque la tarea a ejecutar es periódica, simple y acotada, sin necesidad de mantener un proceso en ejecución ni gestionar dependencias, estados o reinicios. `crontab` ofrece una solución más simple y portable (está en todos los Unix, a diferencia de `systemd`) para la ejecución programada de scripts. El flujo general de este proceso se muestra en la Figura 5.1.

5.2. ROS 2

En el Capítulo 4, se describieron las herramientas de *software* necesarias para los vuelos autónomos individuales realizados con cada dron de manera

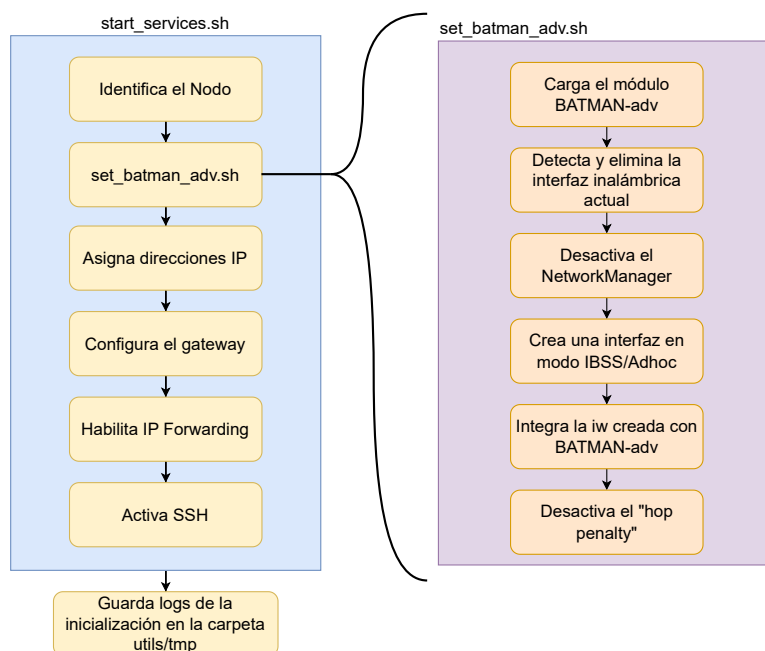


Figura 5.1: Secuencia de servicios lanzados en el arranque de cada NUC

aislada. El siguiente paso es avanzar hacia un sistema orientado a enjambre, donde varios drones puedan coordinarse, compartir información y ejecutar comportamientos cooperativos. Para lograrlo, es necesario contar con un entorno de *software* que permita organizar la lógica interna de cada dron y estructurar la comunicación entre ellos de forma modular y escalable. En este proyecto, ese rol lo cumple ROS 2. Su incorporación permite unificar el funcionamiento interno de los drones y la *Ground Station* mediante mecanismos de comunicación estandarizados, facilitando la integración, el control por módulos segregados y la expansión futura del sistema hacia comportamientos más complejos.

A continuación se presenta una introducción a los conceptos fundamentales de ROS 2, necesaria para entender su uso dentro de la arquitectura del sistema. El *Robot Operating System 2* (ROS 2) es un *framework* de *software* libre diseñado para el desarrollo de aplicaciones robóticas. No es un sistema operativo en sentido estricto, sino un conjunto de herramientas, librerías y convenciones que permiten la construcción de sistemas distribuidos, facilitando la comunicación entre componentes heterogéneos [38]. ROS 2 surge como evolución de ROS 1, superando limitaciones relacionadas con la falta de soporte nativo para comunicaciones distribuidas, la ausencia de calidad de servicio (QoS: *Quality of Service*), la dependencia de un “maestro central” y la escasa portabilidad multiplataforma. Gracias a la integración de un *middleware*

Capítulo 5. Operaciones en Enjambre

estandarizado (DDS: *Data Distribution Service*), ROS 2 se posiciona como una solución robusta y escalable, utilizada tanto en investigación académica como en aplicaciones industriales [39] [40]. En la Figura 5.2 se observa la arquitectura general de ROS 2 como un diagrama de capas.

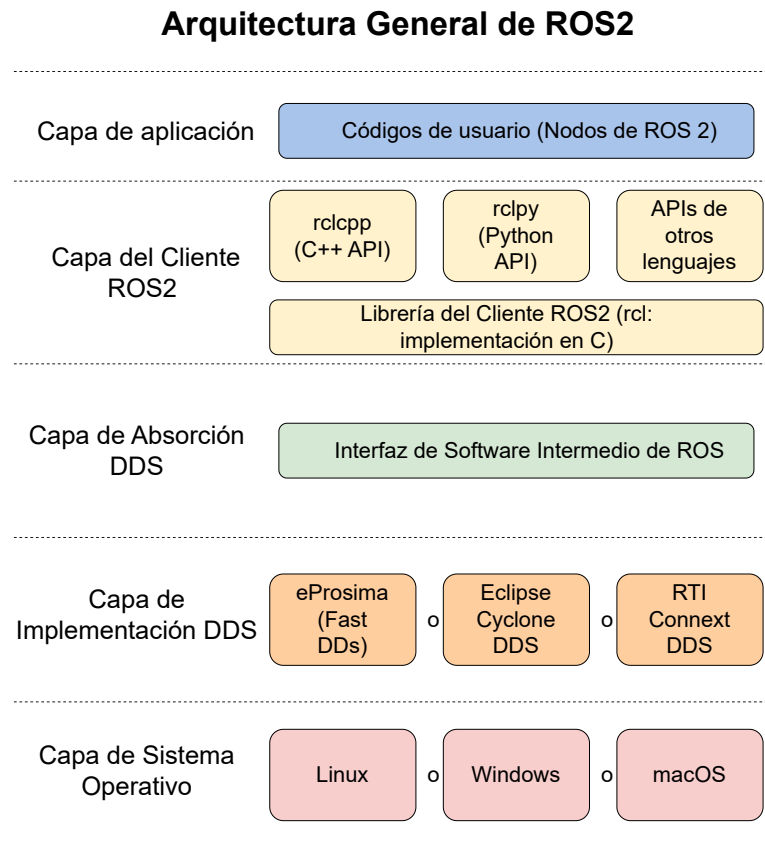


Figura 5.2: Arquitectura de ROS 2 representada bajo un modelo en capas, mostrando la organización jerárquica de sus componentes y la interacción entre los distintos niveles del sistema [41].

5.2.1. Elementos Fundamentales de ROS 2

En primer lugar, los **nodos** representan las unidades básicas de ejecución: cada uno es un proceso capaz de publicar o suscribirse a datos, ofrecer servicios, ejecutar acciones y manejar parámetros. Una mejora significativa respecto

a ROS 1 es la posibilidad de ejecutar múltiples nodos dentro de un mismo proceso o utilizar nodos con ciclo de vida controlado, que transitan por estados como *inactive*, *active* o *finalized*.

La comunicación principal entre nodos se realiza mediante **tópicos**, que implementan un esquema asíncrono basado en el paradigma publicador/suscriptor. Un nodo publicador envía mensajes a un tópico, mientras que otros nodos suscriptores reciben esa información. Cada tópico está asociado a un tipo de dato específico, pudiendo utilizar tanto interfaces estándar de ROS 2 como mensajes personalizados definidos mediante archivos `.msg` o `.srv`, lo que otorga gran flexibilidad. El procesamiento de los mensajes recibidos se maneja mediante **callbacks**, funciones que se ejecutan automáticamente cuando llega nueva información y permiten activar lógica en respuesta a eventos externos.

Además de los tópicos, ROS 2 incorpora otros mecanismos de comunicación como **servicios** y **acciones**. Los servicios ofrecen un intercambio sincrónico *request/response* para operaciones puntuales, mientras que las acciones están diseñadas para tareas de larga duración que requieren retroalimentación periódica y posibilidad de cancelación. A su vez, el uso de *namespaces* permite organizar nodos, tópicos y servicios de forma jerárquica, facilitando la estructuración de sistemas complejos como entornos multi-robot. Si bien estos últimos mecanismos no se utilizan directamente en este proyecto, forman parte de los conceptos fundamentales del ecosistema ROS 2.

Otro componente clave son los **parámetros**, valores configurables asociados a cada nodo que pueden modificarse en tiempo de ejecución. En ROS 2, a diferencia de ROS 1, los parámetros no se gestionan en un servidor global sino dentro de cada nodo, reforzando la modularidad del sistema y permitiendo mayor control sobre el comportamiento individual.

Finalmente, ROS 2 incorpora variables de entorno que definen el alcance de la comunicación entre nodos. Entre ellas, `ROS_DOMAIN_ID` permite separar dominios de ejecución para evitar interferencias entre sistemas distintos que comparten la misma red física, mientras que `ROS_LOCALHOST_ONLY` restringe la comunicación al propio host, lo que resulta útil en etapas de desarrollo o en escenarios donde se requiere limitar el tráfico a nivel local [42].

A continuación damos un ejemplo del uso de algunos de estos elementos. En el sistema que se ejecuta en los drones descrito en la siguiente sección, un “nodo A” publica las posiciones GPS del dron en un tópico. Otro “nodo B”, encargado de registrar esos datos en la base de datos local, se mantiene suscrito a ese mismo tópico. Cuando el “nodo A” envía una nueva posición, en el “nodo B” se ejecuta la función de **callback**, que responde al evento y guarda el dato en la base de datos.

Capítulo 5. Operaciones en Enjambre

5.2.2. Middleware y QoS

ROS 2 emplea el estándar DDS a través de una capa de abstracción llamada RMW (*ROS Middleware Interface*). Esto elimina la dependencia de un maestro central y habilita la configuración de políticas de calidad de servicio (QoS), como fiabilidad, latencia máxima, durabilidad de mensajes o tamaño de colas. Estas capacidades lo diferencian de ROS 1, donde la comunicación se basaba en protocolos propios (TCPROS/UDPROS) sin control de QoS [43] [44].

5.2.3. Paquetes y Estructura de Trabajo

En ROS 2, un paquete es la unidad mínima de *software* que contiene nodos, bibliotecas, interfaces, configuraciones o utilidades relacionadas. Todo desarrollo en ROS 2 se organiza en paquetes, lo que facilita la modularidad y la reutilización de código [45].

La estructura de un paquete varía según el lenguaje de programación utilizado, siendo Python y C++ los más comunes. En el caso de Python, un paquete típico creado incluye varios elementos fundamentales. El archivo `package.xml` contiene los metadatos del paquete, como su nombre, versión, dependencias y licencias. El archivo `setup.py` define el proceso de instalación y ejecución del paquete, mientras que el archivo `setup.cfg` proporciona configuraciones complementarias para dicha instalación. Dentro del directorio `resource/` se incluyen los archivos necesarios para la indexación del paquete dentro del ecosistema ROS 2. La carpeta principal `{nombre_paquete}/` alberga el código fuente en Python, donde se implementan los nodos y utilidades. Finalmente, el directorio opcional `launch/` puede contener archivos `.launch.py`, los cuales describen cómo ejecutar múltiples nodos, definiendo parámetros, remapeos o *namespaces*. En el caso de C++, los paquetes que se crean en lugar de tener el archivo `setup.py`, contienen un archivo `CMakeLists.txt` para gestionar la compilación.

La documentación oficial de ROS 2 ofrece tutoriales completos tanto para Python como para C++ [42], por lo que en esta tesis se detalla principalmente la organización de paquetes en Python, que es el enfoque utilizado en este proyecto.

En las siguientes secciones se expone la arquitectura basada en ROS 2 de la plataforma diseñada y la importancia de los conceptos desarrollados en esta.

5.3. Arquitectura, Diagrama y Flujo de la Plataforma Diseñada

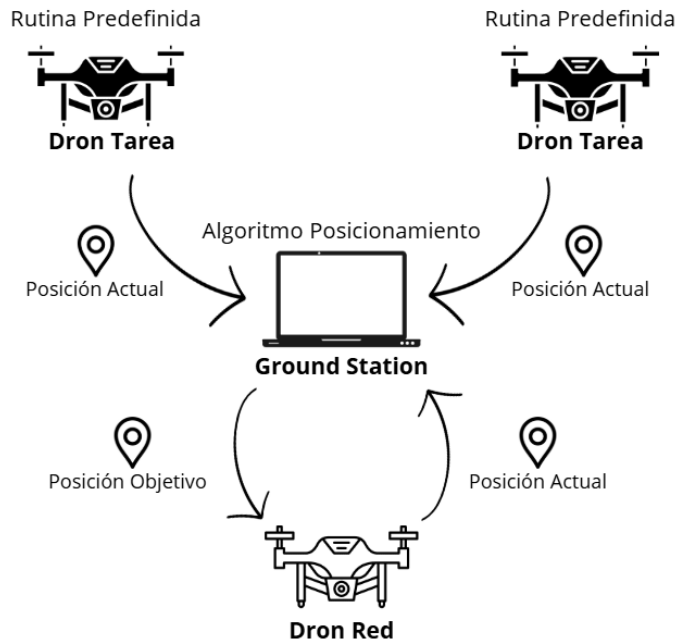


Figura 5.3: Esquema de alto nivel del funcionamiento de la plataforma.

5.3. Arquitectura, Diagrama y Flujo de la Plataforma Diseñada

La plataforma desarrollada implementa un sistema distribuido compuesto por una estación en tierra (*Ground Station*), drones de tarea y drones de red. Cada uno equipado con una computadora a bordo (Intel NUC) que ejecuta un entorno basado en ROS 2 para la organización interna de módulos y MAVSDK para la interacción con la controladora de vuelo, el envío de comandos y la recepción de telemetría. El *software* desplegado es idéntico en todos los drones. El rol de cada uno se define dinámicamente al iniciar el sistema mediante parámetros. Por su parte, la *Ground Station* también ejecuta un entorno basado en ROS 2 y algoritmos de posicionamiento para la reubicación de los drones de red del enjambre como se observa en la Figura 5.3.

La comunicación entre los drones y la *Ground Station* se realiza mediante datagramas UDP sobre una red ad-hoc gestionada con B.A.T.M.A.N., utilizando interfaces Wi-Fi. El objetivo de esta comunicación es officar de puente entre el entorno de ROS 2 de cada dron y el de la *Ground Station*. Cada dron transmite periódicamente su posición GPS y métricas de los enlaces de red, simultáneamente permanece a la espera de instrucciones enviadas por la *Ground Station*. Esta última procesa de forma continua la información recibida de todos los agentes y ejecuta un algoritmo centralizado que determina las

Capítulo 5. Operaciones en Enjambre

posiciones objetivo para los drones de red, reenviando estas órdenes por la red como mensajes *broadcast* también en datagramas UDP.

El flujo operativo comienza cuando cada dron ejecuta un **lanzador** que habilita los módulos ROS 2 necesarios y establece su comportamiento: los drones de tarea ejecutan la rutina experimental (trayectorias autónomas o permanencia en tierra), mientras que los drones de red móviles despegan, mantienen vuelo estacionario a una altitud fijada y actualizan su posición en coordenadas NED según las órdenes de posición calculadas por la *Ground Station*. Se cuenta también con la posibilidad de que los drones de red permanezcan en tierra para pruebas estáticas de enlaces y comunicación. En este caso reciben las ordenes de movimiento pero no las ejecutan.

La *Ground Station*, implementada sobre una NUC conectada a una laptop de forma cableada para acceder a ella, ejecuta su propio lanzador que inicializa los módulos de ROS 2. Durante toda la operación, el sistema recopila la información transmitida por los drones, evalúa el estado global del enjambre y difunde nuevas instrucciones de movimiento en ciclos sucesivos en base al resultado de un algoritmo. Además, expone un servidor web para la monitorización en tiempo real de la ubicación y estado de los enlaces entre agentes.

En conjunto, esta arquitectura coordina de manera integrada los componentes distribuidos del enjambre, conectando la operación autónoma de cada dron con la toma de decisiones centralizada y permitiendo la ejecución de misiones colaborativas sobre una red inalámbrica ad-hoc. Permite aislar los entornos de ROS 2 de cada agente y limitarse a transmitir en la red la información relevante (posiciones y estados de enlace). Además permite la ejecución del sistema con los drones en modalidad estática y dinámica.

5.3.1. Arquitectura de Software e Interfaces

En las NUCs montadas en los drones, los módulos de *software* principales se muestran en la Figura 5.4. En ella se incluyen también las interfaces físicas y lógicas (por ejemplo tópicos de ROS 2) que conforman la arquitectura completa del sistema. Estos módulos se implementan en el paquete de ROS 2 que denominamos **drone_movement**, desarrollado en lenguaje Python.

No todos los módulos representados corresponden a nodos de ROS 2, aunque estén incluidos dentro de un paquete. Los nodos en ROS 2 se implementan típicamente como clases que heredan de *Node*, integrando publicadores, suscriptores y lógica interna bajo un enfoque modular orientado a objetos. Si bien ROS 2 no impone un paradigma específico, su diseño favorece este modelo por su capacidad de encapsulación y reutilización del código. En la Figura 5.4, los módulos en color celeste representan módulos que implementan nodos de ROS, mientras que los de color amarillo corresponden a módulos que implementan funciones auxiliares ya sean utilizadas por los

5.3. Arquitectura, Diagrama y Flujo de la Plataforma Diseñada

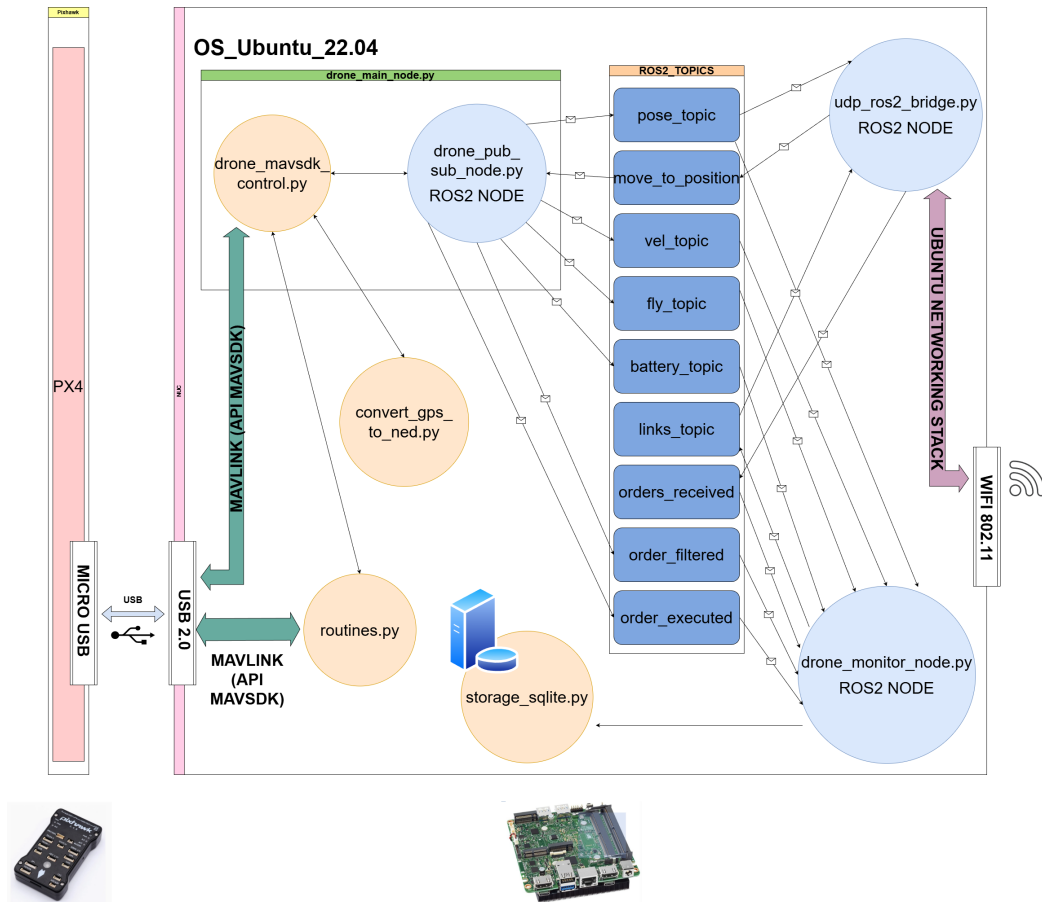


Figura 5.4: Módulos internos y sus principales interfaces. En color celeste se representan módulos que implementan nodos de ROS y en color amarillo módulos con funciones auxiliares.

nodos ROS o para la comunicación con la controladora de vuelo como es el caso de `drone_mavsdk_control.py`. Además, en color azul, se representan los tópicos con los que se vinculan los diferentes nodos internos. Para comprender el diagrama de la Figura 5.4, se agrega a continuación un breve resumen de las responsabilidades y acciones de cada módulo.

El nodo principal `drone_main_node` inicializa los subsistemas y determina el rol del dron, mientras que `drone_mavsdk_control` actúa como núcleo operativo gestionando la conexión con el vehículo, el modo Offboard, el armado, las rutinas autónomas y la publicación de telemetría, utilizando las rutinas definidas en `routines` y las conversiones de `convert_gps_to_ned` referenciadas a la coordenada geográfica del punto de inicio. Sobre esta base, `drone_pub_sub_node` implementa la interfaz ROS 2 de mensajería, publicando telemetría y filtrando las órdenes de movimiento dirigidas al dron. El intercambio con la *Ground Station* se realiza mediante `udp_bridge_node`, que envía telemetría compacta y recibe órdenes externas para publicarlas en

Capítulo 5. Operaciones en Enjambre

ROS 2. Paralelamente, `drone_monitor_node` registra de forma autónoma toda la información relevante del vuelo y del estado de los enlaces, almacenándola mediante la capa de persistencia `storage_sqlite`, que mantiene una base de datos local con posiciones, velocidades, batería y ciclo completo de órdenes. En conjunto, estos módulos conforman una arquitectura integrada donde cada componente cumple un rol específico (control, comunicación, conversión, registro o ejecución de rutinas) logrando un sistema distribuido coherente, modular y orientado tanto a la navegación autónoma como al análisis de desempeño de la misión (para más detalles de cada módulo ver Apéndice D).

Por su parte, la arquitectura de la *Ground Station* representada en la Figura 5.5, se ejecutan los nodos y además el módulo `Algoritmo`, encargado de determinar los puntos destino objetivo de los drones de red según el algoritmo empleado y el método de acceso a sus salidas.

A continuación, dejamos un resumen de las funciones principales de cada módulo del diagrama de la Figura 5.5.

El módulo principal `gs_ros_node` coordina el comportamiento colectivo de los drones, recibiendo las posiciones publicadas en ROS 2, ejecutando periódicamente el algoritmo centralizado y generando las consignas de movimiento que publica en el tópico `move_to_position`. La comunicación entre ROS 2 y la red inalámbrica se realiza mediante `gs_udp_bridge`, que recibe por UDP la telemetría enviada por los drones, la convierte a mensajes internos (los publica en el tópico `pose_topic`) y en sentido inverso, retransmite por broadcast las órdenes de posicionamiento emitidas por el algoritmo. Paralelamente, el nodo `gs_monitor` registra en tiempo real la posición, métricas de red y órdenes transmitidas, integrando una interfaz `WebSocket` (se detalla en la próxima Sección) para visualización remota y almacenando toda la información mediante `gs_storage_sqlite`, que mantiene una base de datos local con vuelos, posiciones, métricas y consignas. En conjunto, estos módulos conforman la arquitectura funcional de la *Ground Station*, encargada de recibir telemetría, procesar la lógica centralizada y distribuir órdenes de manera sincronizada mientras registra toda la actividad de la misión (por más detalles ver Apéndice D).

Además, es importante destacar que varios de los tópicos ROS 2 presentes en el diagrama de la *Ground Station* también aparecen en el diagrama de los drones. Esto refleja la intención arquitectónica de simular que todos los agentes (drones y *Ground Station*) comparten un mismo ecosistema ROS, como si ejecutaran dentro de una única red local. En efecto, dado que ROS 2 utiliza un modelo de comunicación distribuido basado en DDS, los nodos pueden descubrirse y comunicarse directamente cuando comparten una misma red local, sin requerir mecanismos adicionales de encaminamiento o pasarela. En este contexto, el uso de componentes intermedios como `gs_udp_bridge` o `udp_bridge_node` no resulta estrictamente necesario, ya que los drones

5.3. Arquitectura, Diagrama y Flujo de la Plataforma Diseñada

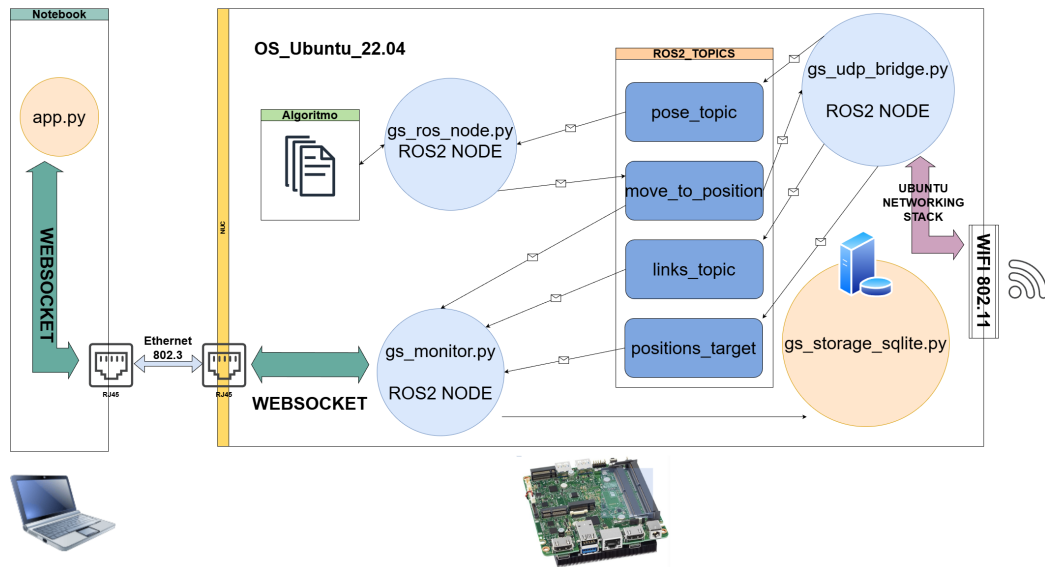


Figura 5.5: Módulos internos y sus principales interfaces en la Ground Station.

podrían intercambiar información directamente a través de los tópicos de ROS 2. No obstante, este enfoque presenta limitaciones prácticas. Por defecto, ROS 2 emplea mecanismos de descubrimiento automático basados en multicast, utilizando todas las interfaces de red (tanto virtuales como físicas) disponibles del sistema. Este comportamiento genera un volumen significativo de tráfico de control y señalización, el cual crece de forma poco escalable a medida que aumenta el número de nodos participantes. En escenarios inalámbricos, como redes ad-hoc entre vehículos aéreos, dicho tráfico puede provocar congestión del medio, aumento de latencia y degradación del rendimiento general de la red. Por este motivo, se incorporaron ambos nodos pasarela para limitar la comunicación al tráfico estrictamente necesario (posiciones de todos los drones, órdenes de movimiento y métricas de enlaces) utilizando la interfaz virtual de batman-adv (bat0), permitiendo el ruteo de paquetes, complementado con el uso de la variable de entorno `ROS_LOCAL_HOST_ONLY`, que confina el resto del tráfico ROS 2 al host local y evita sobrecargar la red distribuida del sistema.

Por último, desarrollamos el paquete **interfaces**, que agrupa tipos de mensajes personalizados para el transporte (a través de los tópicos) de datos de posición utilizados por los distintos módulos del sistema.

- **DronePoseData:** contiene el identificador del dron, coordenadas geográficas, altitud y tipo de agente, unificando el formato de telemetría.
- **ArrayDronePoseData:** agrupa múltiples *DronePoseData* y se utiliza para enviar órdenes o actualizaciones globales desde la *Ground Station*. Cada dron filtra los datos según su identificador.

Capítulo 5. Operaciones en Enjambre

También es importante describir con precisión cómo se gestionan las coordenadas dentro de la plataforma y cuáles son los requisitos que deben cumplir los algoritmos. Por un lado, cada algoritmo recibe como entrada las posiciones GPS de todos los drones, tanto de tarea como de red, y debe devolver como salida nuevas coordenadas GPS que indiquen la posición objetivo a la que debe desplazarse cada dron de red. La plataforma envía a los drones de red un datagrama que contiene directamente las coordenadas GPS del punto objetivo. Si el algoritmo necesita realizar una etapa de adaptación, por ejemplo transformar las coordenadas GPS a UTM (*Universal Transverse Mercator*), o elegir una posición inicial de referencia, como por ejemplo el primer dato recibido de un dron, para trabajar en un sistema relativo expresado en metros puede hacerlo de manera interna. Sin embargo, es obligatorio que la salida final siempre consista en posiciones GPS para cada dron de red.

Esta exigencia se vincula con la forma en que los drones ejecutan los desplazamientos. Cada dron de red utiliza un método del modo *Offboard* de la librería MAVSDK, con coordenadas NED como se vió en la Sección 4.2. En este sistema, las posiciones se expresan en metros y están referenciadas al punto en el que la controladora de vuelo realizó el *boot*. Por esta razón, cada dron debe iniciar el experimento exactamente en el lugar donde fue energizado o reiniciado. Al comenzar el experimento, el dron almacena su primera coordenada GPS como referencia absoluta y, cada vez que recibe una nueva orden expresada en coordenadas GPS, calcula internamente el desplazamiento en metros relativo a esta referencia inicial para convertirlo a coordenadas NED compatibles con el método.

Por último, es importante señalar el comportamiento dinámico durante los desplazamientos: si un dron ya alcanzó su punto objetivo, permanece en espera hasta recibir una nueva orden. En cambio, si está en movimiento hacia un destino y arriba una orden distinta, actualiza inmediatamente el punto objetivo y descarta la instrucción previa, respondiendo en tiempo real a las indicaciones generadas por el algoritmo en la *Ground Station* (para más detalles sobre coordenadas ver el Apéndice B.1.1).

5.4. Monitoreo en tiempo real

En un sistema compuesto por múltiples drones móviles operando sobre una red ad-hoc, es fundamental disponer de un mecanismo de monitoreo en tiempo real que permita observar la posición de cada nodo y los enlaces que se forman entre ellos. Herramientas que recopilen una amplia variedad de información para un posterior análisis son importantes. Pero también es necesario desarrollar una herramienta que permita monitorear en tiempo real los experimentos realizados, permitiendo detectar rápidamente anomalías, pérdidas de conectividad o fallos en un nodo, lo que facilita la toma de

5.4. Monitoreo en tiempo real

decisiones durante las pruebas.

Para implementar esta funcionalidad se desarrolló una interfaz web accesible desde cualquier navegador, basada en una arquitectura cliente-servidor. En donde un equipo actúa como servidor central y el resto de los dispositivos operan como clientes visualizadores. La Figura 5.6 muestra la vista principal de la interfaz desarrollada para monitorear la red.

El servidor levanta una aplicación web desarrollada con **Flask** [46], que se encarga de servir los archivos HTML, hojas de estilo y *scripts JavaScript* que componen la interfaz del monitoreo. Para el envío de datos en tiempo real se emplea una conexión **WebSockets**, un protocolo que permite comunicación bidireccional y persistente entre cliente y servidor, evitando la necesidad de realizar consultas periódicas (*polling*). De esta manera, cada vez que se reciben datos actualizados desde los drones, el servidor puede transmitirlos inmediatamente a todos los clientes conectados.

Por su parte, los clientes acceden al monitoreo ingresando la dirección IP del servidor en un navegador web. Al hacerlo, se establece una conexión **WebSocket** con el servidor, a través de la cual reciben los datos de posición y topología en formato JSON. Estos datos son procesados mediante **scripts JavaScript** en el lado del cliente, que actualizan dinámicamente el mapa o diagrama SVG que representa los drones y sus enlaces.

El flujo de datos comienza con la ejecución de un experimento en el entorno ROS2, donde se levanta un nodo denominado nodo Monitor. Este nodo se conecta como cliente a la aplicación del servidor, enviando de manera periódica la información de la red, compuesta por las posiciones actuales de cada dron, y de las métricas de *batman-adv* sobre los enlaces entre nodos (vecinos, rutas preferenciales y calidad de los enlaces). El servidor, a su vez, almacena y gestiona estos datos, poniéndolos a disposición de todos los clientes conectados a la interfaz web.

Dado que los clientes deben conectarse a la dirección IP del servidor, todos los dispositivos deben encontrarse dentro de la misma red o contar con una ruta que les permita alcanzar su dirección. Esto da lugar a dos posibles alternativas. Una alternativa es levantar el servidor en la *Ground Station*, accediendo al monitoreo desde la laptop conectada mediante un enlace *Ethernet*, o directamente accediendo como cliente desde la misma *Ground Station* (en caso de contar con un monitor conectada a la misma). La otra alternativa es levantar el servidor en la laptop, conectada por Wi-Fi a una red compartida entre varios equipos y simultáneamente cableada a la *Ground Station*. En este caso, esta última puede comunicarse con la laptop a través del enlace cableado (gracias a una ruta predefinida), mientras que otros dispositivos, al estar en la misma red inalámbrica, también pueden acceder a la interfaz web (por ejemplo, desde un teléfono móvil).

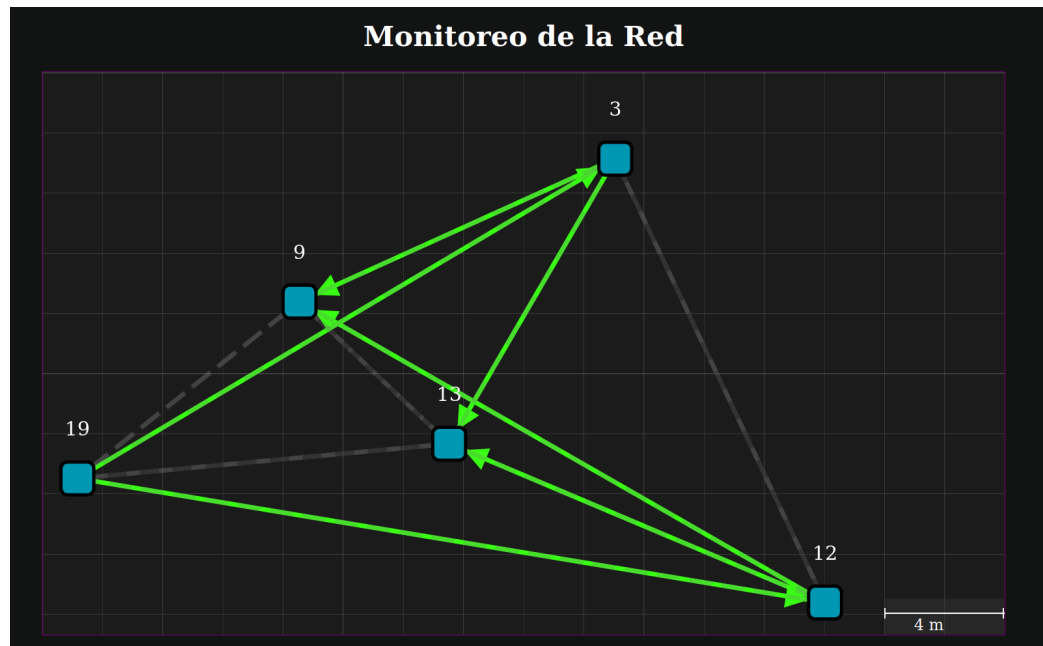


Figura 5.6: Página de Monitoreo de la Red.

5.5. Adaptación de la Plataforma a Nuevos Algoritmos

Durante la integración de nuevos algoritmos en la plataforma surgió un inconveniente recurrente: muchos de estos módulos utilizan librerías de Python que entran en conflicto con las dependencias ya instaladas en el entorno principal asociado a ROS 2. Para evitar estos conflictos y garantizar la reproducibilidad de las pruebas, se diseñó un mecanismo basado en la ejecución desacoplada de procesos mediante el uso de entornos virtuales independientes de Python, además de un cliente y un servidor de la herramienta *ZeroMQ* (ZMQ). Esta herramienta es una biblioteca de mensajería de alto rendimiento que implementa patrones de comunicación como *request-reply*, *publish-subscribe* y *push-pull*, ofreciendo colas de mensajes livianas, asíncronas y de baja latencia sin requerir un *broker* intermedio. Su diseño orientado a *sockets* la convierte en una solución eficiente para interconectar procesos distribuidos, manteniendo independencia entre los entornos de ejecución [47].

En este esquema, el nodo de la *Ground Station* encargado de invocar el algoritmo no ejecuta directamente dicho código, sino que ejecuta un cliente ZMQ que se conecta a un servidor local, el cual se ejecuta de manera persistente durante todo el experimento en un entorno virtual separado. Este servidor actúa como intermediario entre el nodo ROS 2 y el módulo que ejecuta el algoritmo recibiendo solicitudes del cliente, ejecutando el código con sus propias dependencias y devolviendo los resultados sin interferir con el entorno del sistema principal. Finalmente, para garantizar la compatibilidad con

5.5. Adaptación de la Plataforma a Nuevos Algoritmos

los módulos de control del enjambre, es necesario acordar y estandarizar el formato de salida que produce cada nuevo algoritmo. En particular, se requiere que los resultados sean convertidos a las estructuras de datos esperadas por los drones, posiciones objetivo expresadas en coordenadas geográficas (latitud, longitud y altitud). Este paso asegura que la integración con la red de drones se realice de manera consistente y sin ambigüedades, independientemente de la complejidad interna del algoritmo incorporado.

A modo de conclusión, este capítulo sienta las bases para un sistema de drones, donde cada uno opera de manera autónoma pero coordinada dentro del enjambre. A partir de este despliegue, es posible avanzar hacia los algoritmos de control, coordinación y experimentación que se desarrollan en los capítulos siguientes.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 6

Algoritmos de Posicionamiento

En términos generales, un algoritmo de posicionamiento para drones de red ajusta de forma autónoma y tridimensional la ubicación de drones que actúan como nodos de comunicación. Su objetivo es optimizar la topología de una red entre agentes (otros drones, robots, sensores) para maximizar la conectividad, la cobertura y, en algunas aplicaciones, la eficiencia energética, garantizando una comunicación robusta y adaptable. Un algoritmo procesa ciertas entradas y como salida genera las trayectorias (puntos objetivo y opcionalmente velocidades) para los drones. Opera bajo restricciones de cinemática, energía y parámetros de seguridad. Las soluciones pueden ser centralizadas o distribuidas, y se emplean distintas técnicas para obtener las posiciones. Funciona en un ciclo continuo de percepción, optimización y comando para mantener una red aérea inteligente y resiliente.

6.1. Implementación Propuesta

Recordamos que consideramos un conjunto de agentes de tarea realizando sus operaciones, mientras que otros agentes, llamados agentes de red, se posicionan estratégicamente para mantener la conectividad entre todos los agentes. El objetivo es determinar las mejores posiciones para estos agentes de red dada la distribución espacial de los agentes de tarea en ese instante. Este algoritmo es una primera aproximación que es válida en el escenario concreto que estamos planteando y particularmente que la conectividad mejora incluso cuando se tiene en cuenta únicamente el posicionamiento. En este sentido, el algoritmo genera tantos grupos de agentes de tarea como agentes de red disponibles. Cada agente de red se encarga de brindarle conectividad a un grupo de agentes de tarea. Estos grupos, llamados *clusters*, tienen la propiedad de que se solapan entre sí; es decir, comparten elementos con los *clusters* vecinos.

Se exploraron métodos clásicos de agrupamiento, como *K-means* y *Fuzzy C-*

Capítulo 6. Algoritmos de Posicionamiento

means [48] [49], los mismos comparten algunas ideas en la solución propuesta, como la formación de *clusters* y el solapamiento de estos. Sin embargo, *K-means* fue descartado porque tiende a maximizar la compacidad interna de los grupos y la separación entre ellos, lo que contraviene la necesidad de mantener comunicación entre grupos. *Fuzzy C-means*, por su parte, permite que un dato pertenezca parcialmente a varios *clusters* (propiedad acorde con el problema), pero no está diseñado para optimizar posiciones físicas.

Desarrollar un algoritmo propio brinda control sobre su funcionamiento. Además, tanto *K-means* como *Fuzzy C-means* requieren fijar una semilla para obtener resultados deterministas y no es trivial predefinir posiciones iniciales para los agentes de red que no pertenecen a *clusters*, lo cual es fundamental en este proyecto ya que estamos trabajando en un espacio geográfico particular. A su vez, estos algoritmos no contemplan casos excepcionales como *clusters* vacíos (ya que cada agente de red representa un clúster debe tener agentes de tarea asignados), *clusters* con un único agente de tarea, o *clusters* duplicados (dos agentes de red asignados exactamente al mismo *cluster*); tales situaciones podrían provocar colisiones entre drones. El algoritmo desarrollado incorpora mecanismos específicos para manejar estas condiciones y prevenir colisiones. De esta forma, el problema de agrupamiento se replanteó como uno de localización de instalaciones FLP (*Facility Location Problem*) dinámico, centrado en ubicar estratégicamente los agentes de red móviles para brindar cobertura de comunicación.

6.2. C-DFLP: Clustering Dinamic Facility Location Problem

Para abordar esta problemática, se diseñó un algoritmo que determina la ubicación espacial de los agentes de red. El proceso consta de tres etapas: primero, se crean y se forman, como se explica en la Sección 6.2.1, tantos grupos de agentes de tarea como agentes de red haya disponibles (por ejemplo, con dos agentes de red se definen dos grupos de agentes de tarea; con tres agentes de red, tres grupos de agentes de tarea). En segundo lugar, a partir de esos grupos iniciales, se calculan las posiciones objetivo centroides a las que debe desplazarse cada agente de red para mejorar la conectividad del sistema. Por último, se ejecuta un procedimiento de prevención de colisiones para garantizar la seguridad durante el vuelo.

La estrategia de formación de *clusters* se basa en dos criterios: cercanía y solapamiento controlado. En una primera etapa, para cada agente de tarea se identifican sus dos agentes de red más próximos, de este modo, los agentes de tarea quedan asociados simultáneamente a dos agentes de red (*clusters*) vecinos, y pueden actuar como elementos de solape entre *clusters* adyacentes; la Figura 6.1 ilustra este proceso. Luego, mediante las combinaciones dos a dos de los agentes de red se identifican aquellos agentes de tarea por cercanía

6.2. C-DFLP: Clustering Dinamic Facility Location Problem

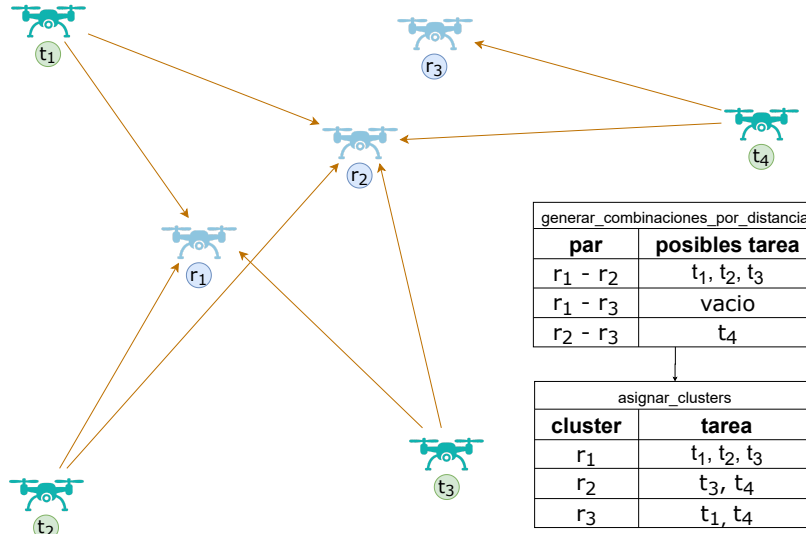


Figura 6.1: Diagrama elección de clúster mediante cercanía y solapamiento.

y se asigna a ambos *clusters* el agente de tarea que este mas cerca de ambos al mismo tiempo. El resto de los agentes de tarea se incorporan al clúster de su agente de red más cercano. Finalmente, se garantiza que cada clúster resultante cuente con al menos dos agentes de tarea asignados, evitando configuraciones vacías o triviales.

Las siguientes secciones explican en profundidad la introducción presentada. En la Figura 6.2 se muestra un ejemplo de reubicación de los agentes de red tras una iteración típica del algoritmo, donde cada dron de red ocupa la posición del centroide de su grupo de agentes de tarea.

6.2.1. Flujo de ejecución

El algoritmo propuesto opera con las posiciones iniciales de todos los agentes, tanto de red como de tarea, como datos de entrada, bajo el supuesto que el centro de coordenadas a partir del cual se realizan los cálculos esta al mismo orden de distancia que la separación promedio de los agentes. A partir de las posiciones, realiza una serie de pasos para determinar una configuración final eficiente. En detalle, se forman agrupaciones iniciales de agentes de tarea basadas en sus dos agentes de red más cercanos ($r_x - r_y$); todos los agentes de tarea que comparten la misma pareja de agentes de red vecinos se agrupan en un mismo clúster inicial. Luego, el algoritmo asigna formalmente cada agente de tarea a uno o más agentes de red formando así los grupos designados a cada agente de red y calcula nuevos centroides para estos. Esta asignación constituye el núcleo lógico del método, pues define cómo se reparten los drones de tarea entre los drones de red y determina las posiciones objetivo

Capítulo 6. Algoritmos de Posicionamiento

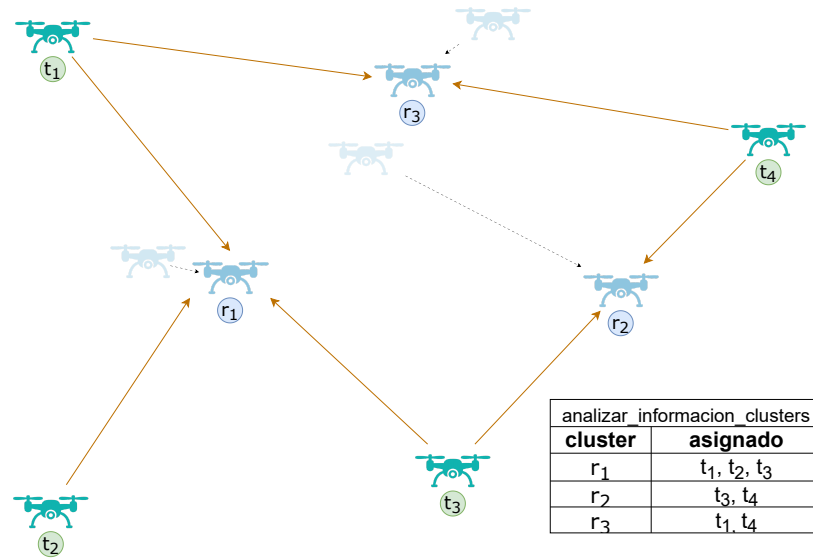


Figura 6.2: Diagrama Posición de Agentes de Red.

(centroides) hacia las cuales deberán desplazarse los drones de red. Se realizan tres iteraciones, este numero fue asignado tras la practica y experiencia para nuestro escenario específico. Al final de cada iteración se obtiene así una coordenada actualizada para cada agente de red, que representa la ubicación ideal de este algoritmo para cubrir su clúster. Con un mismo conjunto inicial de agentes de tarea y de red, el procedimiento es determinista: siempre produce la misma configuración final de posiciones con un mismo conjunto de entradas, garantizando resultados reproducibles. A continuación se describe este flujo de ejecución de forma conceptual, destacando el propósito de cada etapa.

Validaciones iniciales

Si no hay agentes de red disponibles, o si el número de agentes de tarea es menor que dos, el algoritmo finaliza inmediatamente sin realizar cálculos (no hay suficientes datos para una asignación significativa). Si existe exactamente un agente de red, la solución es directa: se calcula el centroide de todas las posiciones de los agentes de tarea y se ubica el único agente de red en ese punto, quedando posicionado de manera estratégica para brindar conectividad al conjunto de agentes de tarea, ya que este algoritmo no considera que los agentes de tarea tengan la posibilidad de rutear. En caso contrario (cuando se cuenta con dos o más agentes de red y de tarea), se procede con la ejecución completa del algoritmo.

6.2. C-DFLP: Clustering Dinamic Facility Location Problem

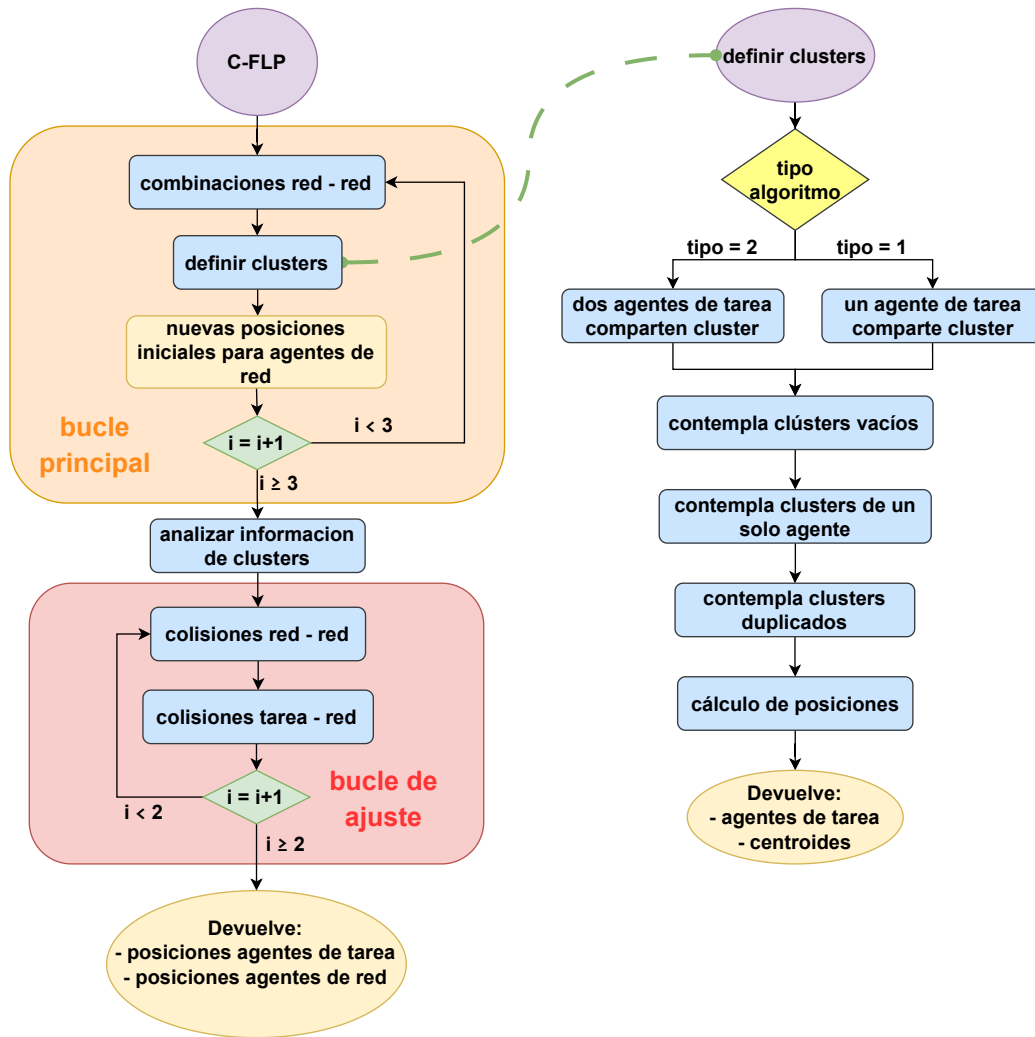


Figura 6.3: Diagrama Flujo principal del algoritmo.

Bucle iterativo principal

En el caso general, el algoritmo ingresa en un bucle iterativo que refina progresivamente la ubicación de los agentes de red. En cada iteración, las posiciones finales calculadas para los agentes de red (los centroides resultantes de los *clusters* en la iteración previa) pasan a ser las posiciones iniciales de la siguiente. Este mecanismo realimenta el cálculo mejorando la posición final de los agentes. Durante cada iteración se efectúan dos operaciones fundamentales, correspondientes al proceso de asignación de *clusters* mediante combinaciones de agentes de red, ilustrado en la Figura 6.3.

Capítulo 6. Algoritmos de Posicionamiento

Bucle de reubicación por colisiones

Una vez definidas las agrupaciones en cada iteración y ubicados provisionalmente los agentes de red en sus centroides correspondientes, es fundamental asegurar que estos reposicionamientos no resulten en colisiones (ni entre dos drones de red, ni entre un dron de red y uno de tarea). Para ello, el algoritmo incorpora una etapa de posprocesamiento dedicada a ajustes de seguridad. En primer lugar, se analiza la configuración resultante y se calcula un centroide específico para cada agente de red, el cual actúa como punto de referencia o “dirección de escape” ante potenciales colisiones. Con esta información se aplican dos ajustes correctivos. Primero, si dos agentes de red quedaron demasiado próximos, el sistema los desplaza a lo largo de la línea recta que los une hasta alcanzar una distancia mínima prefijada. Este desplazamiento garantiza una separación segura entre cualquier par de drones de red. Segundo, si algún agente de red terminó ubicado muy cerca de uno de sus agentes de tarea asignados (riesgo de colisión física), dicho agente de red se reubica ligeramente en dirección a su centroide específico. Este movimiento aleja al dron de red del agente de tarea en cuestión, asegurando una distancia prudencial que evita colisiones directas.

Estas correcciones se llevan a cabo en un número limitado de iteraciones adicionales (dos), lo que busca eliminar las situaciones de riesgo sin comprometer el carácter determinista del algoritmo ni prolongar demasiado el tiempo de ejecución. De todos modos, en la práctica los drones se vuelan a diferentes alturas por lo que el riesgo de colisión teórico es nulo. Dado que la asignación de *clusters* es la etapa medular del algoritmo, a continuación se describe en detalle dicho subproceso y los criterios aplicados en cada paso.

Flujo de asignación de clusters

La Figura 6.3 muestra esquemáticamente el procedimiento interno de asignación de agentes de tarea a los agentes de red, compuesto por una secuencia de etapas encadenadas que buscan asegurar la conectividad, el balance en la distribución de agentes de tarea y la consistencia geométrica en la formación de los *clusters*. Inicialmente, los agentes de tarea se distribuyen entre los distintos agentes de red utilizando una de dos estrategias posibles (denominadas Algoritmo Distendido y Algoritmo Compacto). La diferencia principal entre ambas radica en el manejo de los agentes de tarea compartidos entre *clusters* adyacentes: la Estrategia Distendida permite que un mismo agente de tarea pertenezca simultáneamente a dos *clusters* (es decir, que sea atendida por dos agentes de red a la vez), mientras que la Estrategia Compacta emplea dos agentes de tarea compartidos por cada par de *clusters* vecinos. Este solapamiento controlado garantiza la existencia de agentes de tarea compartidos entre *clusters*, favoreciendo la continuidad topológica de la

6.3. Algoritmo GIOD Centralized

red y mejorando la conectividad global, tal como se explicó anteriormente.

Seguidamente, el algoritmo corrige las asignaciones anómalas o poco robustas que puedan haber surgido. Se comprueba que ningún agente de red quede sin agentes de tarea asignados (*cluster* vacío); de encontrarse alguno, a ese agente se le asignan los dos disponibles más cercanos para asegurar que cubra al menos al menos dos agentes de tarea. Asimismo, se evita que algún clúster quede con un solo agente de tarea; en ese caso, se añade al *cluster* el dron de tarea más cercano, evitando un centroide no deseado coincidente con la posición de un solo dron de tarea. Por último, se detectan *clusters* duplicados, es decir, situaciones en que dos drones de red comparten exactamente el mismo conjunto de drones de tarea. Esa circunstancia (que solo puede ocurrir en *clusters* con dos agentes de tarea) se resuelve ubicando a dichos agentes de red en posiciones equidistantes respecto a esos los compartidos, eliminando el riesgo de colisión.

Con la asignación depurada mediante estas correcciones, se recalculan los centroides de cada grupo de agentes de tarea para obtener las posiciones finales de los agentes de red en esa iteración.

Pseudocódigo

Se presentan los pseudocódigos de los algoritmos que complementan los diagramas de flujo mostrados y lo explicado anteriormente. Se comienza mostrando en el Pseudocódigo 1 el bloque principal del algoritmo C-DFLP con las llamadas correspondientes. En los pseudocódigos, *task* representa los agentes de tarea y *net* los agentes de red descritos en este capítulo.

Por otro lado, la creación de *clusters* y asignación de posiciones de los agentes de tarea se muestra en la subrutina encargada de esto. El Pseudocódigo 2 detalla la misma.

En resumen, el algoritmo C-DFLP desarrolla un proceso iterativo y modular que primero agrupa los drones de tarea en función de la cercanía a los drones de red, luego asigna y ajusta esos grupos aplicando criterios de conectividad y seguridad, y, finalmente, calcula mejores ubicaciones para los drones de red.

6.3. Algoritmo GIOD Centralized

Se toma como comparación el algoritmo GIOD (*Graph Infrastructure On Demand*) Centralized [15] [50], su objetivo al igual que en C-DFLP es optimizar una red de comunicación móvil ad-hoc determinando la posición óptima de los agentes de red para maximizar el desempeño de comunicación de los agentes de tarea. Sin embargo, este algoritmo considera explícitamente que el medio inalámbrico es compartido, una limitación clave frecuentemente desatendida

Algorithm 1 Algoritmo C-DFLP

Require: Posiciones ($task$); Posiciones iniciales (net)

Ensure: Posiciones objetivo (net)

```

1: function C-DFLP( $task, net$ )
2:   if  $net = 0$  or  $task < 2$  then
3:     return error
4:   else if  $net = 1$  then
5:     return CENTROIDEGLOBAL( $task$ )
6:   end if
7:    $net \leftarrow$  INITPOS( $task, net$ )
8:   repeat
9:      $clusters \leftarrow$  ASIGNARCLUSTER( $task, net$ )
10:     $net \leftarrow$  ACTUALIZARCENTROIDES( $clusters$ )
11:  until CRITERIODEPARADA( $net, clusters$ )
12:   $net \leftarrow$  COLISIONESYSEGURIDAD( $net, task$ )
13:  return  $net$ 
14: end function

```

Algorithm 2 Subrutina de Asignación de Clústeres

```

1: function ASIGNARCLUSTERES( $task, net$ )
2:   Inicializar  $clusters$  como listas vacías
3:   for cada tarea  $t$  en  $task$  do
4:      $(i, j) \leftarrow$  MINIMIZADISTANCIA( $t, net_i, net_j$ )
5:     Añadir  $t$  a  $clusters[i]$  y  $clusters[j]$ 
6:   end for
7:   for  $k \leftarrow 1$  to  $|net|$  do
8:     if  $clusters[k]$  está vacío then
9:        $clusters[k] \leftarrow$  DOSTAREASMASCERCANAS( $k, task$ )
10:    else if  $|clusters[k]| = 1$  then
11:       $clusters[k] \leftarrow$  TAREAMASCERCANA( $k, task$ )
12:    end if
13:  end for
14:  EQUILIBRARCLUSTERSDUPLICADOS( $clusters$ )
15:  return  $clusters$ 
16: end function

```

6.3. Algoritmo GIOD Centralized

en otros enfoques. El sistema se basa en una *Equivariant Graph Neural Network* (EGNN), capaz de capturar la física subyacente de la conectividad de la red y modelar cómo las interacciones entre nodos se ven afectadas por su disposición espacial. Se toma el mismo contexto de agentes de tarea y agentes de red como drones repetidores móviles, cuya ubicación debe ser ajustada estratégicamente. El comportamiento del enlace inalámbrico se representa mediante un modelo de canal que describe la degradación de la tasa de datos en función de la distancia, y la red neuronal EGNN, cargada a partir de un modelo preentrenado específicamente para el canal proporcionado, predice la calidad global de conectividad a partir de la configuración espacial de todos los agentes.

Entrenamiento o Inferencia

Este algoritmo opera en régimen de inferencia y no de entrenamiento en línea. El modelo EGNN utilizado fue previamente entrenado, y el *script* principal se limita a cargar dicho modelo y ejecutar el ciclo de optimización sobre la topología actual de la red. Esta característica habilita su aplicación en escenarios cercanos al tiempo real: para la escala de cantidad de drones considerada en este trabajo, el proceso de optimización puede ejecutarse en el orden de mili-segundos. Siendo recomendable, no obstante, el uso de unidades de procesamiento gráfico (GPU: *Graphics Processing Unit*) para garantizar un desempeño aún más eficiente.

6.3.1. Optimización y Gradientes

La contribución central de este enfoque radica en transformar dicho modelo de predicción en una herramienta de optimización continua. A partir de la salida del modelo, se utiliza la funcionalidad de autodiferenciación de *PyTorch* para calcular el gradiente de la conectividad con respecto a las posiciones de los agentes de comunicación. Este gradiente, representado como un vector para cada dron, indica la dirección en el plano (x,y) hacia la cual debe moverse para incrementar en la mayor medida posible la conectividad total de la red según el modelo considerado. En cada iteración, el algoritmo toma como entrada las posiciones de los agentes de tarea y las posiciones actuales de los agentes de comunicación, construye el grafo correspondiente, evalúa la conectividad en el modelo y luego calcula el gradiente. Sobre esta base, se define un tamaño de paso adecuado y se actualizan las posiciones de los agentes de comunicación como la posición actual más un desplazamiento proporcional al gradiente. El procedimiento se repite hasta que la norma del gradiente cae por debajo de un umbral definido, indicando la convergencia a un máximo local de conectividad. Se trata, por tanto, de un método de ascenso por gradiente en el cual el gradiente no es predicho por la red neuronal, sino calculado a partir de ella

Capítulo 6. Algoritmos de Posicionamiento

mediante las herramientas de diferenciación automática de PyTorch usando el modelo.

Capítulo 7

Entorno de Simulación Desarrollado

El objetivo de este capítulo es presentar en detalle el entorno de simulación diseñado específicamente para evaluar y validar los algoritmos de posicionamiento implementados en un enjambre de drones. En él se describe la arquitectura utilizada y las herramientas de software que intervienen, permitiendo así comprobar el correcto funcionamiento del sistema antes de las pruebas de vuelo físico.

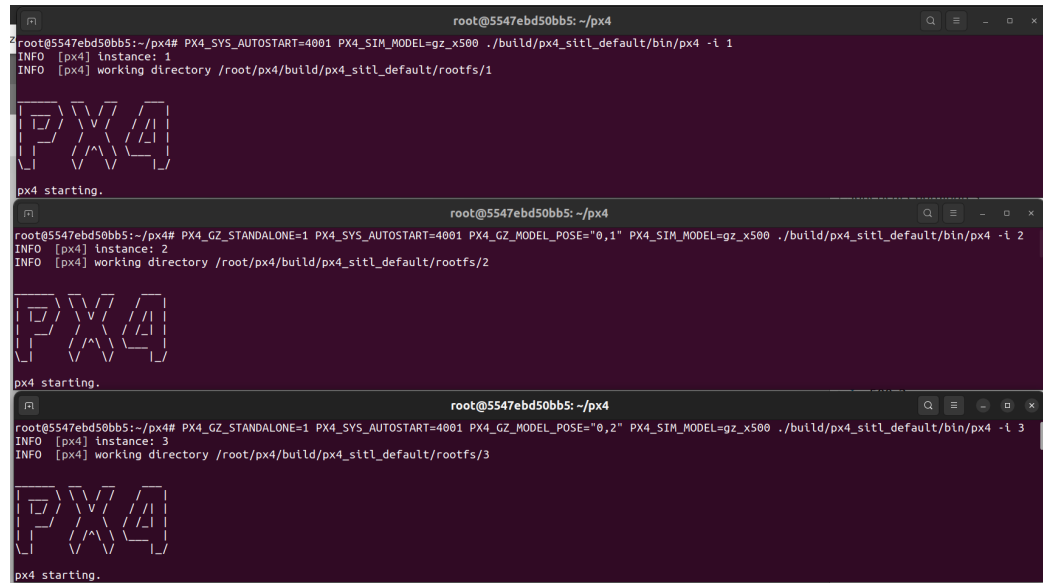
7.1. Desafíos Técnicos y Abordajes Propuestos

En la Sección 4.1 se expuso la relevancia de la simulación y las herramientas utilizadas para validar rutinas de vuelo autónomo en un único dron. Sin embargo, dichas herramientas resultaron insuficientes para evaluar algoritmos de posicionamiento en enjambres, ya que no contemplaban la ejecución simultánea de múltiples vehículos ni la emulación de una red distribuida. Por este motivo, fue necesario incorporar nuevos componentes que permitieran abordar los desafíos asociados a la simulación concurrente de instancias de SITL, al control diferenciado de múltiples vehículos y a la reproducción de la comunicación entre ellos. A continuación se describen los principales problemas identificados y las soluciones implementadas.

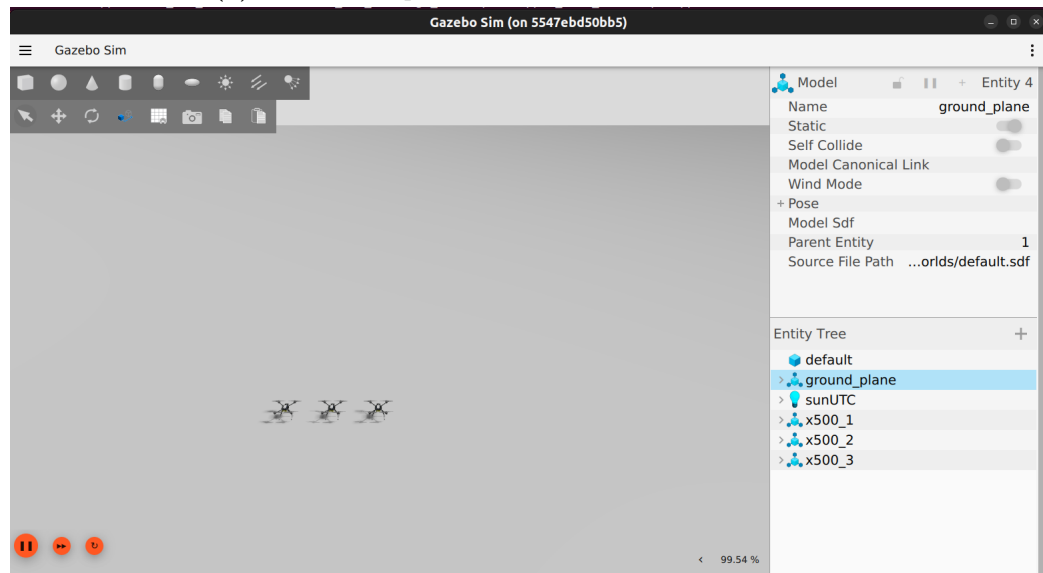
7.1.1. Simulación y Control de Múltiples Instancias de SITL

En esta parte, lo primero que se tuvo que resolver fue como simular los diferentes drones para realizar un experimento. Se resolvió simulando varios drones de forma simultánea dentro de un mismo entorno virtual. Para ello se aprovechó la capacidad de PX4 SITL de lanzar múltiples instancias mediante la asignación de un número de instancia al inicio de la simulación. Esta funcionalidad permitió ejecutar procesos independientes en diferentes terminales, todos conectados a un mismo mundo de Gazebo.

Capítulo 7. Entorno de Simulación Desarrollado



(a) Terminales que lanzan diferentes instancias SITL.



(b) GUI de Gazebo con tres SITL lanzados en el mismo mundo.

Figura 7.1: Ventanas necesarias para simular tres SITL desde una sola computadora.

SITL utiliza la arquitectura cliente-servidor de Gazebo, donde una instancia actúa como servidor y las restantes se ejecutan como clientes (ver Figura 7.1). En la Figura 7.1a se muestra la configuración del entorno mediante la variable `PX4_GZ_STANDALONE=1`, que permite iniciar una instancia en modo independiente (el cual le indica a SITL que no intente lanzar un servidor en Gazebo y que se conecte a otro ya existente).

Luego de lograr la ejecución simultánea, fue necesario garantizar un control

7.1. Desafíos Técnicos y Abordajes Propuestos

independiente para cada vehículo. Para lograrlo se utilizó el hecho de que cada instancia de SITL abre un canal de comunicación por un puerto UDP único, lo cual permitió que se pudiera trabajar con MAVSDK de forma análoga a como se hizo en la sección de despliegue individual. Si se desea ver con más detalle como fue la conexión de MAVSDK a múltiples instancias de SITL en una misma computadora, véase el Apéndice D.1.1.

7.1.2. Simulación de la Comunicación Entre Drones y Virtualización con Docker

El siguiente desafío consistió en simular la comunicación entre los drones. En un sistema real, los drones se comunican mediante mensajes UDP transmitidos por *broadcast* en una red ad-hoc basada en batman-adv. Sin embargo, cuando todas las instancias se ejecutan en la misma máquina, comparten la misma pila de red, lo que impide reproducir ese comportamiento.

Para solucionarlo se adoptó una arquitectura basada en **Docker**. Esta es una plataforma abierta que permite el desarrollo, la distribución y ejecución de aplicaciones. **Docker** permite empaquetar y ejecutar aplicaciones dentro de un entorno aislado llamado contenedor, el cual se caracteriza por ser ligero y tener todo lo necesario (sistema operativo, librerías, etc) para ejecutar la aplicación, lo que hace que esta pueda correrse sin importar lo que haya en el equipo anfitrión [51].

La capacidad de aislamiento de los contenedores de **Docker** permitió implementar una topología de comunicación más cercana a la realidad dentro del enjambre. Cada contenedor, construido a partir de una imagen definida en un **Dockerfile** (archivo que indica las herramientas que tiene la imagen que se corre en el contenedor), se comportó como un equipo independiente. Además, las redes virtuales de **Docker** permitieron definir topologías equivalentes a una LAN. Otra herramienta utilizada fue **Docker Compose**, esta se utiliza para la definición y ejecución de aplicaciones con múltiples contenedores. Con ella, mediante el armado de un archivo **.yaml**, permite gestionar la ejecución de múltiples contenedores y redes de **Docker** [52].

A partir de estas herramientas se diseñó una arquitectura donde cada contenedor representa un dron individual. Dentro de cada contenedor se ejecuta una instancia de PX4 SITL, mientras que la comunicación entre ellos se realiza mediante una red virtual dedicada de **Docker**. La Figura 7.2 ilustra cómo se lanzarían múltiples SITL en una computadora, en comparación a cómo se haría en múltiples equipos.

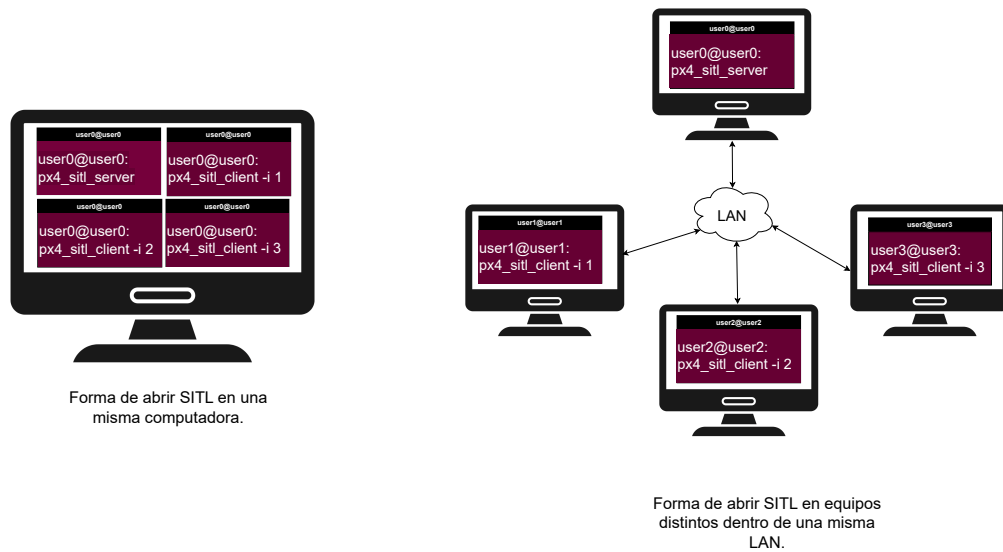


Figura 7.2: Visualización de las formas posibles de lanzar SITL.

7.1.3. Integración de ROS 2 y Ejecución de Múltiples Procesos en Cada Contenedor

Una vez que se implementó **Docker**, se agregó a la imagen que ya corría SITL y MAVSDK, las herramientas necesarias para correr ROS 2, siendo este el último componente necesario para probar el entorno del proyecto, con esto se pudo verificar que las rutinas de movimiento implementadas y los nodos creados para correr algoritmos de posicionamiento, funcionaban de la forma esperada.

Con estas implementaciones, se tuvo que los contenedores de **Docker** iban a correr en paralelo, instancias de SITL y los nodos de ROS 2 con la arquitectura detallada en la Sección 5.3 que fueran a utilizar. Estos nodos serían los que controlan el movimiento de los drones (que cuentan con las funcionalidades de MAVSDK), y los nodos para la comunicación entre ellos (encargados de manejar la comunicación por UDP). Para lograr que los contenedores corran todo lo necesario, se utilizaron dos herramientas. Para lanzar de forma paralela todos los nodos que utilizan los drones, se optó por utilizar la herramienta que trae ROS 2 para esto, la cual permite por medio de un archivo de entrada, configurar todos los nodos de un determinado sistema. Luego de esto, cada contenedor que se lanza para representar un dron lanza dos comandos de forma concurrente, uno es la instancia de SITL y el otro es lanzador de los nodos de ROS 2. Para esto se utilizó la herramienta de Linux **supervisor**, que sirve para correr múltiples procesos en sistemas UNIX. En esta herramienta solo se necesita configurar un archivo que describa los procesos a ser ejecutados.

7.2. Solución Propuesta

Se diseñó una imagen de **Docker** utilizada por los contenedores involucrados en la simulación de un experimento. Dicha imagen se construyó a partir de la imagen oficial de ROS 2 Humble [53], a la que se incorporaron las librerías numéricas necesarias para ejecutar el algoritmo propuesto, junto con los binarios de MAVSDK y PX4 SITL.

También se definió qué se entiende por “experimento”. Este se describe mediante un archivo `.yaml` ejecutado con **Docker-compose**, el cual define y orquesta la ejecución de los siguientes servicios en contenedores: uno encargado de ejecutar el servidor de Gazebo; otro destinado a los nodos ROS 2 de la *Ground Station*; y, finalmente, N contenedores que ejecutan SITL y los nodos ROS 2 correspondientes a los drones. Además, el archivo define dos redes: una destinada a la comunicación entre las instancias de SITL y el servidor de Gazebo, y otra reservada exclusivamente para la comunicación por UDP. La Figura 7.3 muestra el flujo general de servicios y redes definidos para un experimento que corre en una sola computadora.

Los contenedores no cuentan con un espacio de trabajo propio dentro de la imagen, sino que, al momento de su creación, acceden mediante un punto de montaje a una carpeta que contiene el *workspace* de ROS 2. Cada contenedor realiza una copia interna de ese *workspace* para evitar conflictos durante la edición. Esta decisión se tomó con el objetivo de facilitar el desarrollo y prueba de paquetes de ROS 2, permitiendo trabajar desde un entorno local y trasladarse a la simulación sin necesidad de construir nuevas imágenes de **Docker**.

Finalmente, se desarrolló una herramienta en Python basada en plantillas Jinja2 que automatiza la creación de estos archivos. Esta herramienta permite generar un experimento completando un cuestionario, evitando la edición manual de cada archivo y facilitando la configuración de diferentes escenarios.

7.3. Implementación del Algoritmo GIOD Centralized

Para implementar el algoritmo *GIOD Centralized*, se tuvo que adaptar la arquitectura propuesta anteriormente. Esto fue debido a que la imagen que corre el contenedor de la *Ground Station* no tiene las librerías necesarias para correr el algoritmo *GIOD Centralized*. Por esto se decidió crear una nueva imagen basada en la que se implementó anteriormente, agregándole estas librerías faltantes en un entorno virtual de Python como se detalló en la Sección 5.5.

Se decidió crear una imagen nueva, debido a que esta iba a ser utilizada únicamente por la *Ground Station*, y el agregar las librerías faltantes en un entorno virtual hizo que la imagen de **Docker** creada para esto aumentara de

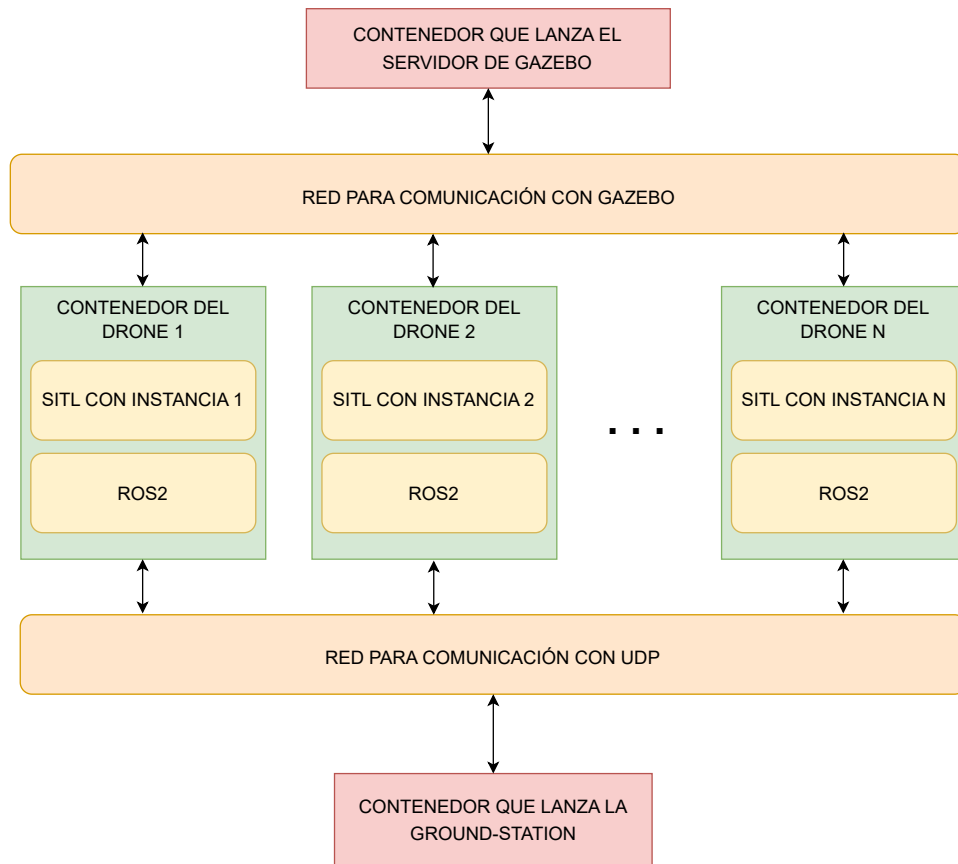


Figura 7.3: Esquema de los servicios y redes para la implementación de un experimento usando el algoritmo C-DFLP.

tamaño. Esto hizo que se pudiera correr los contenedores que forman parte de la simulación del experimento con las imágenes acordes a lo que se busca representar con cada contenedor, para mas información sobre buenas prácticas para trabajar con **Docker**, consultar [54].

En la Figura 7.4 se puede observar un esquema de los servicios que se usan para simular un experimento de vuelo en enjambre usando el algoritmo *GIOD Centralized*.

7.4. Visualización de Trayectorias

Con el esquema de trabajo presentado a lo largo del capítulo, se buscó observar en un entorno simulado el vuelo en enjambre de los agentes de tarea y de los agentes de red. Dado que en este caso los drones volaron a una altura fija durante todo el experimento, resultó de interés disponer de un trazado

7.4. Visualización de Trayectorias

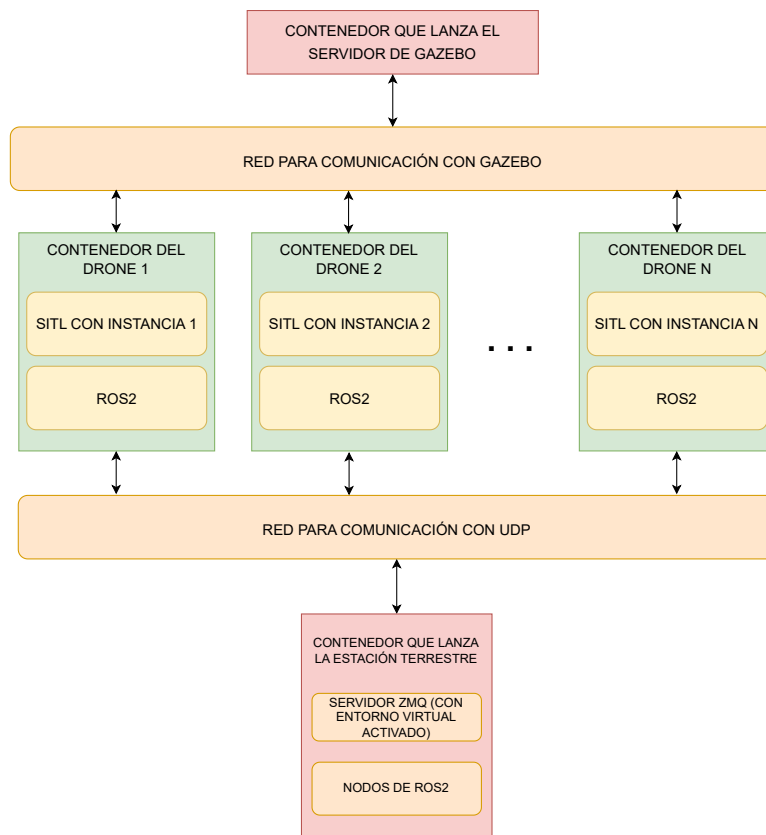


Figura 7.4: Esquema de los servicios y redes definidos para la implementación de un experimento usando el algoritmo *GIOD Centralized*.

bidimensional que permitiera visualizar el recorrido realizado por cada agente. De este modo fue posible analizar con mayor claridad el comportamiento de cada dron dentro del entorno simulado.

Se pudo observar en dos dimensiones lo que se hizo, conectando los contenedores que representan drones a *QGroundControl*. Un ejemplo de como se visualizó esto se puede ver en la Figura 7.5.

Capítulo 7. Entorno de Simulación Desarrollado

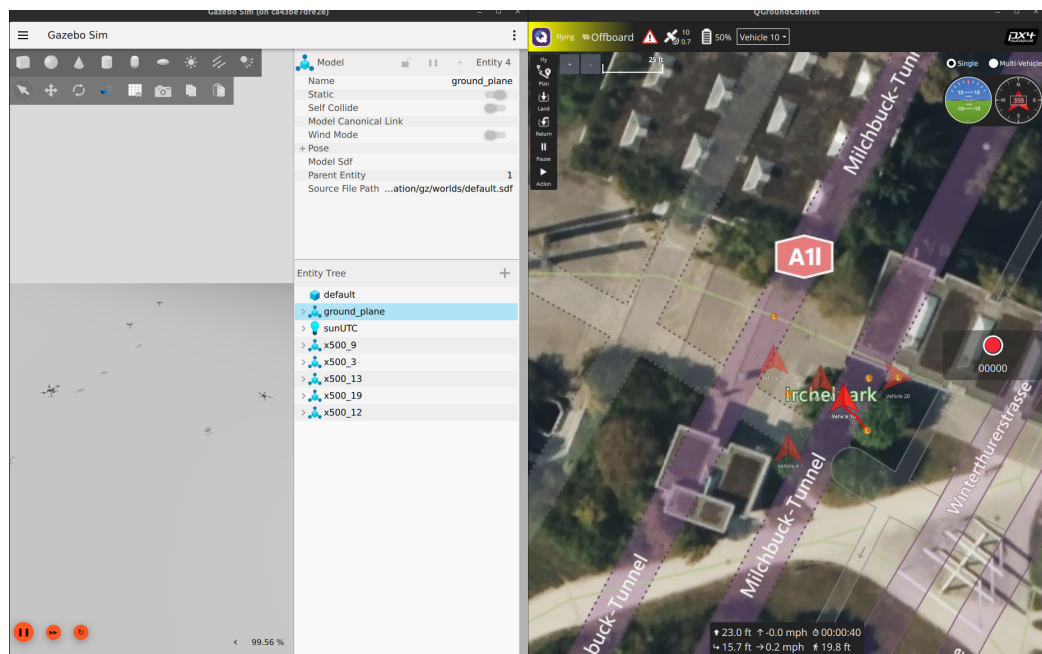


Figura 7.5: Visualización de la trayectoria con *QGroundControl* en simulación.

Capítulo 8

Experimentos en Enjambre y Resultados

En este capítulo se detallan los experimentos realizados para verificar el funcionamiento integral de la plataforma desarrollada. A lo largo del proceso de desarrollo, se ejecutaron numerosas pruebas parciales en módulos específicos. Sin embargo, aquí se presentan los ensayos consolidados en un entorno operativo común, abarcando tanto el vuelo coordinado de múltiples drones como el comportamiento de la red de comunicaciones en condiciones reales. En este contexto, la ejecución de los experimentos requirió el desarrollo de un conjunto de herramientas auxiliares destinadas a garantizar condiciones controladas, reproducibles y seguras durante las pruebas. Dichas herramientas abarcan desde mecanismos para la gestión dinámica de la potencia de transmisión, sincronización temporal y rutinas de validación de la calibración de sensores. Asimismo, se implementaron utilidades para el registro estructurado de datos provenientes tanto de la red como de los sistemas de telemetría y control, así como herramientas de postprocesamiento orientadas al análisis y la visualización de los resultados. El detalle de estas herramientas, junto con los procedimientos de inicialización y validación asociados, se presenta en el Apéndice F.

8.1. Diseño y Resultados de los Experimentos

En esta sección se describen los experimentos de validación diseñados para comprobar el funcionamiento integral de la plataforma y sus resultados. El objetivo es evaluar de forma global el desempeño coordinado del sistema completo, verificando la interacción entre los subsistemas y su comportamiento bajo condiciones reales.

La plataforma fue concebida para ejecutar algoritmos de posicionamiento autónomo, aunque su arquitectura también permite analizar el rendimiento de los enlaces de red que transportan la información de telemetría y control. De este modo, los experimentos se orientan tanto a la validación funcional de

Capítulo 8. Experimentos en Enjambre y Resultados

la navegación y coordinación entre drones como al estudio del flujo de datos y la estabilidad de la comunicación.

Dado que la plataforma experimental permite ejecutarse con los drones en modos estáticos y dinámicos, en la primera prueba los drones permanecen en tierra, pero los módulos de comunicación y control operan normalmente, enviando y recibiendo mensajes de posición, métricas de red y órdenes de movimiento, sin ejecutar el armado ni el despegue. Esta modalidad permite realizar pruebas sobre transmisión, latencia, cobertura y ruteo sin riesgo físico, además de validar el transporte en la red de los datos con el formato correcto tanto desde los drones a la *Ground Station*, así como también las ordenes inferidas de los algoritmos a los drones de red. Por otro lado, en la modalidad dinámica algunos drones despegan y vuelan mientras otros permanecen estáticos: los drones de red actualizan su posición según las órdenes enviadas por la *Ground Station*, mientras que los drones de tarea siguen trayectorias predefinidas, se mantienen fijos en un punto o son trasladados de forma manual. Ambas modalidades comparten la misma arquitectura de red y flujo de mensajes, garantizando equivalencia funcional entre pruebas completamente estáticas y dinámicas.

Los experimentos se diseñaron bajo estas premisas, manteniendo una configuración topológica inicial común con una estación terrestre y un conjunto de drones de red y de tarea, que en las pruebas que implican vuelo, lo realizan a diferentes alturas para evitar colisiones entre ellos. Además los drones implementan la reducción en la potencia de sus antenas debido a la explicación del en la Sección F.1 del Apéndice F.

8.1.1. Prueba de Validación Estática

Este experimento se desarrolla en modo completamente estático y tiene como objetivo validar la operatividad de la plataforma sin movimiento físico de los drones, verificando su capacidad para ejecutar pruebas controladas sobre el estado de los enlaces de comunicación. Se comparan dos configuraciones de drones de red, manteniendo fijas las posiciones de los drones de tarea como se observa en la Figura 8.1: una disposición para los drones de red es propuesta por el algoritmo GIOD Centralized y la otra corresponde a un algoritmo externo a nuestro trabajo utilizado como referencia que se encuentra en el Artículo [14] que oficia de “competidor”. Este algoritmo se basa en un esquema de optimización distribuida sobre grafos, donde los nodos ajustan de manera iterativa sus enlaces y posiciones en función de métricas locales de conectividad y capacidad. La asignación de rutas y recursos se formula como un problema de optimización convexa, resuelto de forma descentralizada mediante intercambio periódico de información entre nodos vecinos.

El propósito principal de esta prueba no es evaluar el desempeño detallado de la red, sino confirmar que la plataforma permite ejecutar el entorno

8.1. Diseño y Resultados de los Experimentos

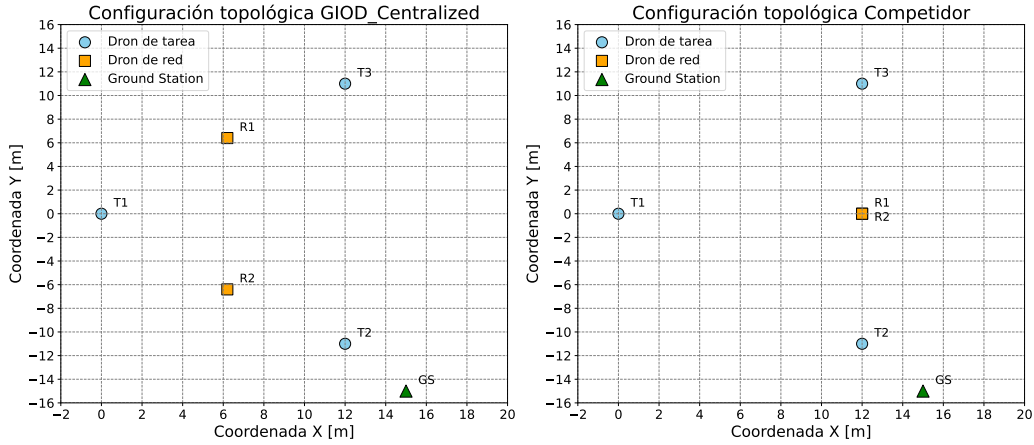


Figura 8.1: Comparativa entre la configuración (para las mismas posiciones de los drones de tarea) de los drones de red propuesta por GIOD_Centralized y la propuesta de su competidor.

completo bajo condiciones estáticas y realizar mediciones de la tasa de transmisión de datos. Se podría haber comparado también con la posición propuesta por el algoritmo C-DFLP, de igual forma para reducir la prueba se decidieron comparar estas dos configuraciones por su diferencia topológica marcada como se observa en la Figura 8.1.

Para evaluar el desempeño de la red en la configuración estática, se ejecutaron pruebas de tráfico mediante la herramienta `iperf3`, configurada en modo bidireccional y utilizando el protocolo TCP durante 30 segundos. El objetivo fue medir la calidad de los enlaces de comunicación entre los drones de tarea. Las pruebas se realizaron de forma simultánea siguiendo una topología en anillo, estableciendo enlaces entre los pares $T2-T1$, $T1-T3$ y $T3-T2$. De esta manera, se obtuvo una caracterización completa del comportamiento de la red en todos los segmentos del anillo de comunicación. Cabe mencionar que el uso de TCP (tanto en estas pruebas como en las dinámicas que se ven más adelante en este capítulo) se fundamenta en que conceptualmente uno de los objetivos de la infraestructura de red de la plataforma es transportar tráfico de aplicación entre drones de tarea, que podrían utilizar el protocolo TCP en capa de transporte. Se podrían también haber realizado pruebas de `iperf3` utilizando UDP, tanto entre drones de tarea o mismo entre drones de red y la *Ground Station*, ya que utilizan este protocolo en capa de transporte para el tráfico de gestión y control de red.

Ejecución, Resultados y Análisis

La prueba de validación estática permitió confirmar que la plataforma es capaz de ejecutar el entorno sin necesidad de vuelo, lo que resulta especialmente útil para evaluar métricas de red (como las tasas de transmisión entre nodos)

Capítulo 8. Experimentos en Enjambre y Resultados

en condiciones controladas y sin los riesgos operativos asociados al vuelo. Este modo de operación facilita además el registro detallado de los enlaces y las posiciones, permitiendo ser reproducido. Validado el entorno estático, es posible realizar ensayos donde parte o la totalidad de los nodos de tarea permanecen en tierra mientras los drones de red ajustan sus posiciones respecto a ellos del mismo modo que lo harían durante un vuelo.

Se llevaron a cabo varias pruebas sobre las capacidades del canal inalámbrico aprovechando la plataforma, que rutea los mensajes y registra los enlaces generados. En estas pruebas, se ejecutaron sesiones de `iperf3` entre todos los pares de agentes de tarea, y se graficó la suma del *throughput* agregado (en kbps) resultante de todos esos flujos concurrentes. Se hicieron pruebas variando la cantidad y posiciones de los drones de red, para observar su efecto en el *throughput* de datos entre los de tarea, haciendo experimentos iniciales con ninguno, y después agregando dos, en las posiciones a las que converge el algoritmo de posicionamiento GIOD Centralized, y el competidor al mismo. Se compararon en cada caso dos escenarios, uno fue con ROS 2 ejecutándose, en el cual todos los nodos (tanto de red como de tarea) permanecen en una rutina estática (sin despegar) pero mantienen el intercambio usual de mensajes ROS 2 vía UDP (telemetría, métricas de B.A.T.M.A.N., órdenes), y otro sin ROS 2, donde los nodos permanecen encendidos y batman-adv sigue realizando el enrutamiento, pero no se generan los mensajes adicionales del entorno ROS 2 ni se registra información en la base de datos. La Figura 8.2 muestra los resultados comparativos para ambas configuraciones, incluyendo el caso en el que no se utilizaron drones de red.

La prueba se realizó en distancias reducidas, y presentó fuertes asimetrías entre los valores obtenidos del *throughput*. En una red ad-hoc/MANET, este tipo de configuración satura fácilmente el medio inalámbrico, que es de acceso compartido (CSMA/CA). Si un nodo presenta mejor señal-ruido o recibe ACKs con menor latencia, va a tener menos *backoff*, obteniendo más “turnos” de transmisión, haciendo que sus flujos compitan mejor contra los otros, terminando por acaparar el canal. Si el medio está saturado, los paquetes de confirmación (ACK) se demoran. Esto genera un efecto dominó en donde uno de los flujos empieza a perder ACKs, bajando la ventana de congestión (*cwnd*) del protocolo TCP, quedándose atrás y provocando otros flujos lo superan fácilmente.

Además, la contención aumenta significativamente cuando el tráfico es bidireccional: con tres nodos ejecutando `iperf3` simultáneo en ambos sentidos, se generan seis flujos compitiendo por un único medio. Considerando también que `iperf3` bidireccional no garantiza simetría entre los flujos. Cada dirección establece su propia ventana efectiva de transmisión, determinada por el mínimo entre la ventana de congestión (*cwnd*) y la ventana de recepción (*rwnd*) de cada flujo TCP sin sincronización, por lo que puede pasar que un

8.1. Diseño y Resultados de los Experimentos

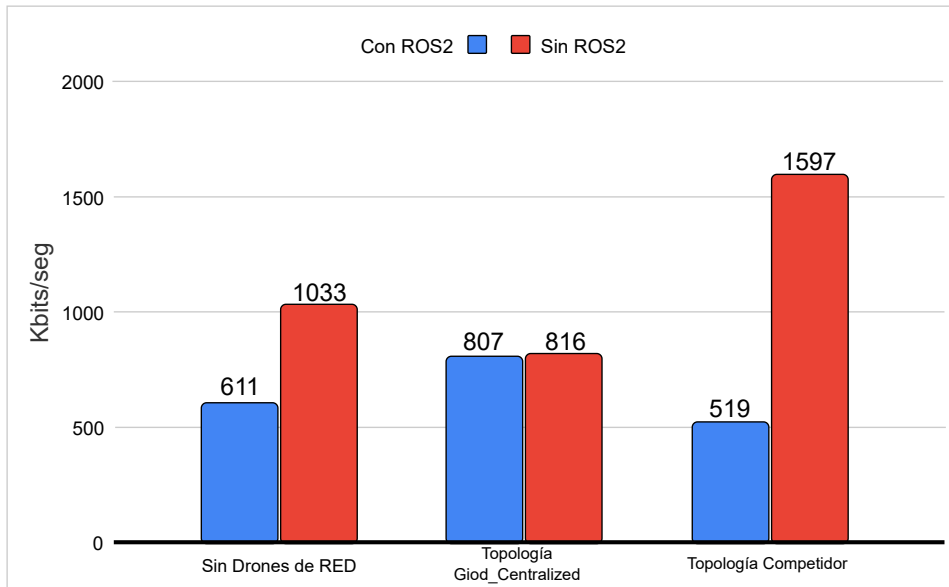


Figura 8.2: Gráfico de barras comparativo del throughput agregado en pruebas estáticas, evaluando el impacto de la presencia de nodos de red y de la ejecución de ROS 2.

lado establezca ventanas mayores, aumentando las asimetrías, desplazando al otro. En estas condiciones, pequeños desbalances físicos provocan que un nodo domine el tráfico y que los resultados aparenten asimetrías fuertes que no reflejan la calidad real del enlace. La atenuación de la potencia realizada en cada dron (detallada en la Sección F.1) incentivó dichos desbalances físicos. Además de que las medidas fueron tomadas mientras los drones permanecían estáticos sobre escaleras metálicas, lo cuál pudo haber contribuido en la interferencia de la comunicación inalámbrica.

Por lo tanto, los resultados obtenidos reflejan principalmente el efecto de la contención y la competencia de tráfico generada por la metodología de prueba, y no la calidad intrínseca de los enlaces. Más allá de estas limitaciones, lo relevante es que la plataforma demostró ser apta para realizar este tipo de experimentos estáticos de manera controlada y repetible.

Para futuros trabajos, hubiera sido deseable realizar pruebas de `iperf3` de a un enlace por vez, midiendo la capacidad de cada vínculo bajo la topología establecida por el algoritmo de posicionamiento. También el tener en consideración que la *Ground Station* es un nodo que rutea, y que para éstas pruebas estáticas se pudo haber usado como un agente de tarea. A su vez, evaluar previo a la realización de las pruebas del canal entre múltiples nodos, el efecto individual que sufre cada dron al ser apantalladas sus antenas Wi-Fi. Y obtener nuevas formas de elevar cada uno de los drones durante las pruebas, evitando colocarlos sobre materiales metálicos. Estas alternativas

Capítulo 8. Experimentos en Enjambre y Resultados

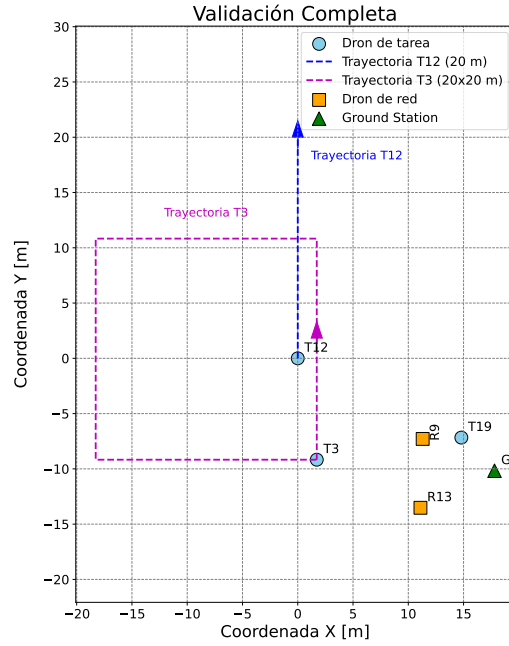


Figura 8.3: Prueba de validación completa con drones de tarea y de red volando

permitirían aprovechar plenamente las capacidades de la plataforma para el análisis detallado del comportamiento de la red.

8.1.2. Prueba de Validación Completa

Esta prueba es de validación del enjambre completo. Se ilustran las posiciones iniciales de los drones en la Figura 8.3. Si bien cualquier dron puede officiar tanto de agente de red como de tarea, en el diseño de esta prueba se indica el identificador del dron que efectivamente fue usado para vincularlo con los resultados obtenidos directamente. Así, en el diseño, *T19* indica que el dron con el identificador 19, está officiando de agente de tarea. La idea es probar el comportamiento de los drones de tarea *T12* y *T3* en vuelo, junto con los drones de red (*R9* y *R13*) coordinados por la *Ground Station*. El dron *T12* realiza un desplazamiento en línea recta, manteniéndose estático 15 s cada 10 m. El dron *T3* ejecuta un vuelo en forma de cuadrado de 20 m de lado, siguiendo su trayectoria predefinida. Por su parte, el dron *T19* se mantiene estático como punto de anclaje del enjambre a la *Ground Station*. Para el caso de los drones de red, se ubican en las posiciones indicadas por la *Ground Station*, generadas por el algoritmo C-DFLP con el objetivo de analizar la estabilidad del vuelo, la cobertura de red y la calidad de los enlaces provistas por el TQ de batman-adv.

8.1. Diseño y Resultados de los Experimentos

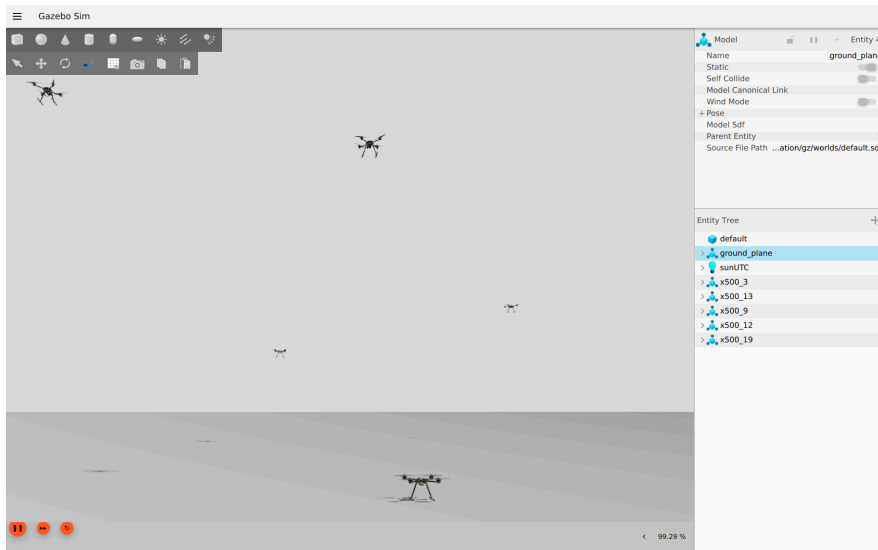


Figura 8.4: Drones volando en Gazebo con experimento realizado.

Implementación previa en entorno de simulación

Previo a realizar el experimento diseñado en la Figura 8.3, se implementó esta prueba en el entorno de simulación, para poder tener una primera estimación de las trayectorias que realizarán los drones que conforman el enjambre.

Para la prueba en el entorno de simulación, se preparó el archivo que utiliza *Docker Compose* para lanzar los contenedores que forman parte del experimento. En este archivo se especifica que van a correr siete contenedores, uno que va a levantar el servidor de Gazebo, otro que va a actuar como *Ground Station* y los cinco restantes van a ser los que representen a los drones que forman parte del experimento. En la Figura 8.4 se observa como se ven los drones en el mundo por defecto que trae el PX4 SITL en Gazebo.

Para verificar que todos los drones se muevan de forma coherente, se corrió la simulación, se fue observando como se movían los drones en Gazebo y en *QGroundControl*, aplicación en la cual se conectaron las instancias de PX4 SITL para monitorear su trayectoria de forma más sencilla en el plano (X,Y). Esta forma de probar el experimento nos permitió verificar que todos los nodos de ROS 2 funcionaran correctamente previo a hacer una prueba de vuelo real. En las Figuras 8.5 y 8.6 se observan las diferentes trayectorias que realizaron todos los drones que conforman el enjambre en el experimento diseñado.

A partir de la comparación entre las trayectorias visualizadas en *QGroundControl* y los desplazamientos observados en Gazebo, se verificó que el comportamiento de los drones coincidió con el esperado. Esto permitió avanzar con las pruebas de vuelo reales con la tranquilidad de que los nodos de ROS

Capítulo 8. Experimentos en Enjambre y Resultados

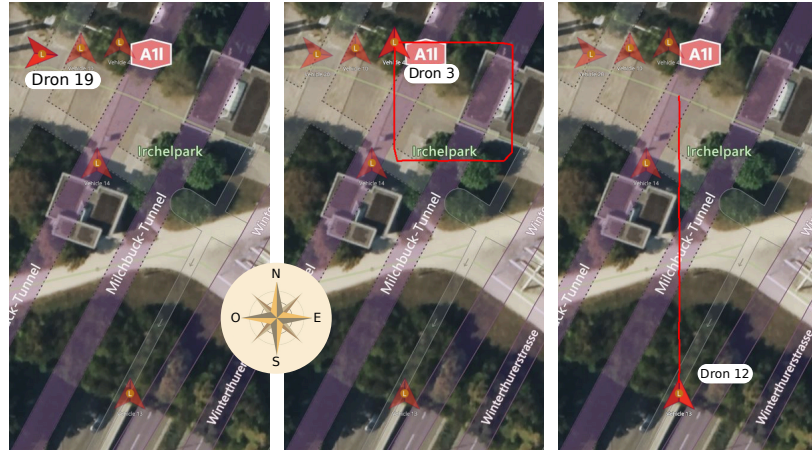


Figura 8.5: Trayectorias realizadas en el experimento por los drones de tarea utilizando el algoritmo de posicionamiento C-DFPL.

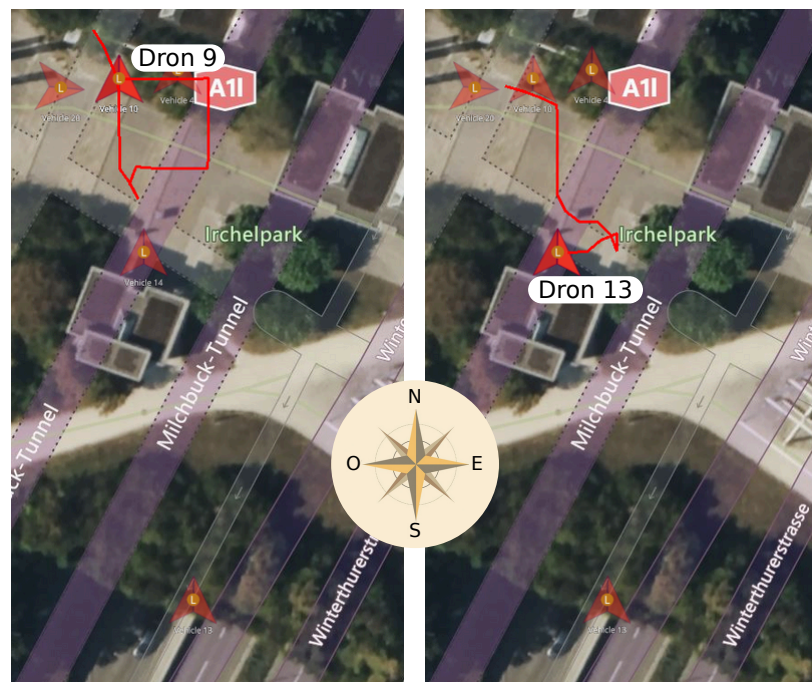


Figura 8.6: Trayectorias realizadas en el experimento por los drones de red utilizando el algoritmo de posicionamiento C-DFPL.

8.1. Diseño y Resultados de los Experimentos



Figura 8.7: Despegue del enjambre en prueba validación completa en campo.

2 y las rutinas de movimiento implementadas con MAVSDK funcionaban correctamente.

Ejecución, Resultados y Análisis

A continuación, se realiza la prueba de validación completa del enjambre en campo, utilizando cinco drones de manera simultánea, cuatro de ellos volando de manera autónoma, siguiendo el diseño presentado en la Sección 8.1.2. La Figura 8.7 ilustra este experimento, en el cual se obtiene un enjambre de drones autónomos ejecutando una rutina de vuelo coherente, combinando trayectorias de agentes de tarea con el posicionamiento dinámico de los agentes de red. La estación terrestre coordina el despliegue, supervisa el estado de cada vehículo y asegura que el algoritmo de posicionamiento se ejecute conforme a lo previsto, mientras los drones despegan de forma ordenada y alcanzan sus respectivas posiciones objetivo.

El registro detallado de datos permite reproducir los experimentos realizados y obtener una observación precisa del comportamiento del enjambre, lo que resulta fundamental para corroborar el buen funcionamiento de la flota. La Figura 8.8 muestra el experimento de validación mencionado, donde las trayectorias fueron reconstruidas a partir de los registros generados por la controladora de vuelo. No obstante, el mismo análisis podría haberse realizado utilizando los datos almacenados en la base de datos; de hecho, la trayectoria correspondiente a la simulación fue extraída directamente de dicha base. Las trayectorias mostradas surgen del posicionamiento calculado mediante el algoritmo C-DFLP y permiten visualizar con claridad cómo un dron de tarea

Capítulo 8. Experimentos en Enjambre y Resultados

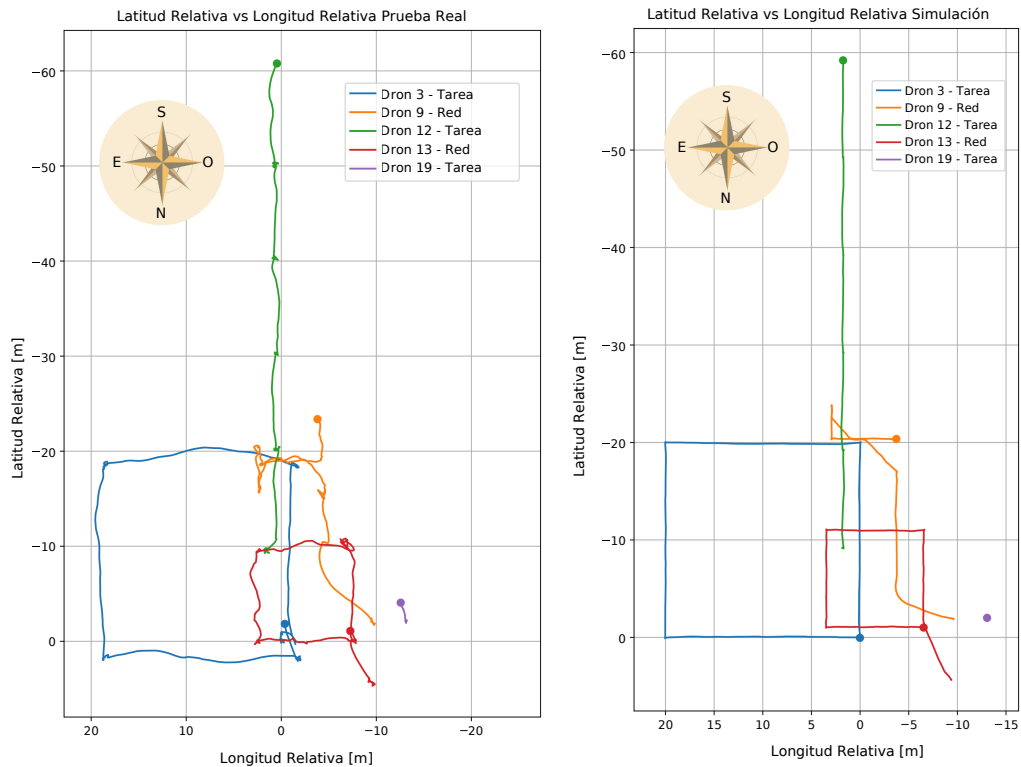


Figura 8.8: Trayectorias reportadas por los datos de GPS de todos los drones en la prueba de validación completa referidas a la latitud y longitud inicial del dron 3 que oficia de origen del gráfico. Luego se convierten esas diferencias a metros y se expresan las longitudes en función de las latitudes para visualizar la trayectoria de cada dron. En el gráfico a la izquierda se observan los resultados de las pruebas de campo y a la derecha los de la simulación.

recorre una línea recta de 50 m en dirección Sur, otro dron de tarea describe un cuadrado de 20 m de lado, mientras que un tercer dron de tarea permanece en una posición fija a modo de referencia. A partir de estos movimientos, los agentes de red se reubican de forma coordinada para mantener la conectividad entre los distintos agentes de tarea durante toda la misión.

Se observa que la trayectoria real se asemeja fuertemente a la trayectoria simulada, tanto en la forma global de los recorridos como en la disposición relativa de los drones, lo que confirma la consistencia entre los resultados obtenidos en campo y los obtenidos en el entorno de simulación. Esta correspondencia respalda la solidez del modelo empleado y refuerza la validez de la plataforma como herramienta de experimentación para el estudio de algoritmos de posicionamiento y coordinación en enjambres de vehículos aéreos no tripulados.

La validación de la cobertura se analiza mediante las métricas de calidad de enlace (TQ) provistas por batman-adv, previamente descritas en la Sección 3.3.5. En la Figura 8.9 se presenta la evolución temporal del promedio del TQ entre

8.1. Diseño y Resultados de los Experimentos

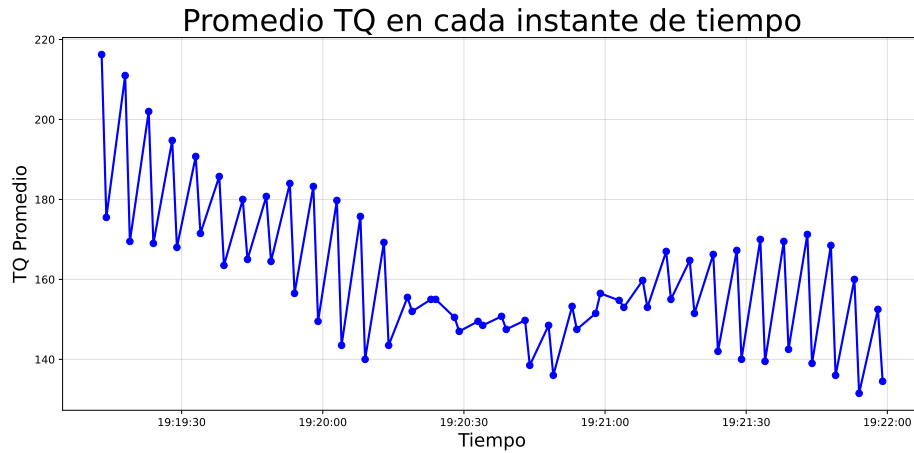


Figura 8.9: Evolución del TQ promedio entre los drones de tarea durante el vuelo de validación.

todos los pares de drones de tarea, con el objetivo de obtener una visión global del estado de la red durante el experimento. Inicialmente, se observa una caída breve en el valor promedio del TQ, correspondiente al período de convergencia del protocolo, en el que los nodos comienzan a intercambiar OGMS y estiman la calidad de los enlaces. Descenso asociado también por el alejamiento progresivo de los drones de tarea, previo al posicionamiento definitivo de los drones de red en sus ubicaciones asignadas. Una vez que los drones de red alcanzan su posición objetivo y las rutas se establecen, el promedio del TQ tiende a mantenerse estable, a pesar de que uno de los drones de tarea continúa alejándose, y otro variando su distancia al realizar un cuadrado, lo cual implica cambios de rutas. Esto confirma que los drones de red lograron sostener la conectividad y la calidad del enlace entre los agentes de tarea, asegurando una malla estable a medida que la distancia entre ellos variaba.

De forma complementaria, en la Figura 8.10 se representa específicamente la calidad del enlace entre el dron 12 al dron 19, par seleccionado debido a que la distancia entre ellos aumenta de manera monótona durante el experimento. A medida que la distancia aumenta la calidad del enlace disminuye progresivamente, hasta que la vista directa deja de ser viable, y se aprecia como utiliza nodos intermedios para mantener estable la comunicación. Finalmente, en la Tabla 8.1 se muestran las rutas activas registradas en el momento de máxima separación del dron 12 respecto del dron 19. Los valores de TQ y los nodos intermedios evidencian que efectivamente se utilizó ruteo *multihop*, lo cual valida que los drones de red operaron como repetidores móviles para extender la cobertura de la red.

Capítulo 8. Experimentos en Enjambre y Resultados

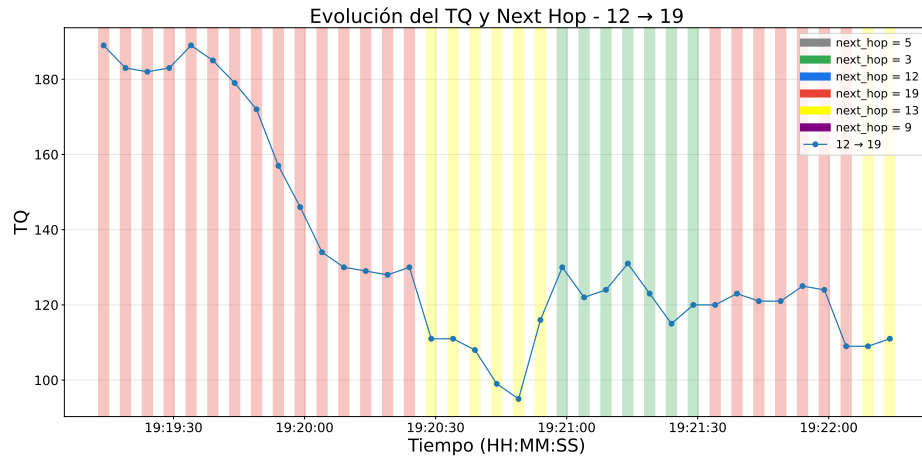


Figura 8.10: Calidad del enlace en la comunicación del dron 12 al 19 durante el experimento, mostrando a quien le mandaba el mensaje para llegar al destino.

Origen	Destino	Próximo Salto	TQ
19	3	9	177
19	12	3	101
12	3	3	230
12	19	3	115
3	12	12	139
3	19	13	122

Tabla 8.1: Rutas activas registradas en el instante de mayor separación entre los drones 12 y 19

8.1.3. Prueba de Validación Dinámica

Esta prueba se diseñó para evaluar el desempeño del algoritmo C-DFLP bajo condiciones mixtas de operación. El objetivo principal es analizar el impacto de la incorporación progresiva de drones de red en la cobertura y el rendimiento de los enlaces de comunicación durante el desplazamiento de un dron de tarea. En la Figura 8.11 se ilustra la configuración inicial (sin drones de red) y la final (con dos drones de red). Para esta prueba, los drones $T12$ y $T19$ se mantienen estáticos, mientras que el dron $T3$ se desplaza manualmente siguiendo una trayectoria lineal. Cada 10 m recorridos se ejecuta una medición de rendimiento de red mediante `iperf3` en modo TCP bidireccional, con intervalos de muestreo de 20 segundos, replicando la misma metodología empleada en las pruebas estáticas. Posteriormente, se repite el procedimiento agregando un dron de red al enjambre, y luego con dos.

En las pruebas sin drones de red, se planea hacer una cantidad indefinida de puntos hasta perder la conectividad. Luego, en las pruebas con drones de red se busca extender al menos 10 m de la distancia máxima alcanzada en las

8.1. Diseño y Resultados de los Experimentos

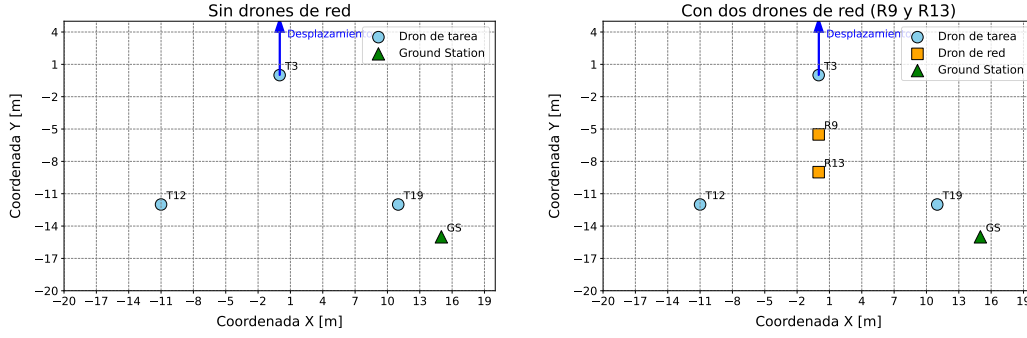


Figura 8.11: Evolución de la configuración topológica del entorno: sin drones de red y con dos drones de red.

prueba sin drones de red.

Se evalúa el vuelo autónomo de los drones de red, verificando que actualicen su posición de manera estable y coherente con las órdenes enviadas por la *Ground Station* en respuesta al movimiento del dron *T3*. En cada escenario se registran las tasas de transferencia y la estabilidad de los enlaces (a través de las métricas de batman-adv), comparando el comportamiento del sistema con y sin agentes de red. Además, se propone calcular un factor de desempeño (Q_d) de la red del enjambre en las tres configuraciones. Similar a la prueba estática, consiste en calcular en cada distancia “d” la suma del *throughput* de los flujos concurrentes de n drones de la siguiente forma: siendo $T_{d,i,j}$ el *throughput* del flujo del enlace entre el dron de tarea i y el j , en la distancia d :

$$Q_d = \sum_{i=1}^n \sum_{j=1, i \neq j}^n (T_{d,i,j}) \quad (8.1)$$

Ejecución, Resultados y Análisis

Durante esta etapa experimental se buscó validar dos aspectos fundamentales de la plataforma: por un lado, verificar que los drones de red ejecutaran correctamente las órdenes de movimiento enviadas desde la *Ground Station*, posicionándose de manera coordinada para brindar cobertura según lo previsto; y por otro, evaluar simultáneamente el rendimiento del canal inalámbrico entre los drones de tarea durante el avance del experimento. Para ello, se presentan diversas representaciones gráficas que permiten analizar de manera conjunta las trayectorias ejecutadas, las métricas obtenidas mediante *iperf3* y la información de enlace registrada por batman-adv.

En el escenario inicial sin drones de red, la Figura 8.12 muestra las trayectorias en coordenadas relativas (metros) respecto al punto de inicio del dron *T3*. Los puntos de mayor densidad indican zonas donde los drones permanecieron más tiempo, resultado del período necesario para estabilizar las

Capítulo 8. Experimentos en Enjambre y Resultados

mediciones de `iperf3` en cada posición. Sobre esta misma figura se representan las tasas de transmisión registradas para cada distancia y para cada sentido de los flujos, diferenciadas mediante una codificación por colores.

En este escenario se logró mantener conectividad y obtener métricas de rendimiento hasta los 20 metros respecto del punto inicial. La Figura 8.13 ilustra los enlaces directos establecidos entre cada nodo (representados en color verde) antes de que alguno quedara aislado. Cuando dos nodos no cuentan con un enlace directo, pero existe una secuencia de enlaces que permite la comunicación entre ellos, la red opera a través de saltos múltiples, tal como es característico en redes ad-hoc y gestionado aquí mediante el protocolo batman-adv.

En este contexto, el dron *T19* resultó ser el eslabón más débil de la red, ya que fue el primero en quedar aislado. Aunque la distancia entre *T19* y *T12* era la menor del escenario, no formaban un enlace directo. Esto se explica considerando que la potencia de transmisión fue modificada para ajustar el alcance a distancias adecuadas para los experimentos, tal como se detalla en la Sección F.1. A pesar de poseer el mismo *hardware*, estas modificaciones manuales pueden generar variaciones significativas en la potencia efectiva transmitida o recibida por cada nodo. A esto se suma que tanto *T19* como *T12* estaban ubicados sobre estructuras metálicas (como se observa en las Figuras 8.14 y 8.15), lo que probablemente introdujo atenuaciones adicionales. En contraste, el dron *T3* fue trasladado a mano, permaneciendo alejado de superficies metálicas, lo que explica que *T12* y *T19* se comunicaran preferentemente a través de *T3* pese a estar entre sí a menor distancia.

Para complementar el análisis, se incluyen también las métricas internas del protocolo batman-adv, en particular la calidad del enlace TQ para cada distancia, junto con el próximo salto para alcanzar al destinatario como se observa en la Figura 8.16. Estas mediciones revelan que, independientemente de la distancia, los enlaces de menor calidad eran aquellos asociados al dron *T19*, reforzando la hipótesis de que su antena fue la más afectada por las modificaciones manuales y por su ubicación durante la prueba.

A continuación se presentan los resultados obtenidos al incorporar de manera progresiva los drones de red en el experimento. En primera instancia, se analiza el escenario con un dron de red, y posteriormente el caso con dos drones de red. Cabe destacar que, durante la prueba con dos drones de red, se registró una falla inesperada en el GPS del dron *T3*, el cual dejó de enviar datos de posición durante un intervalo prolongado. Aún así, los análisis mostrados a continuación se centran en el rango de distancias donde sí se dispone de datos válidos, rango que además supera la distancia máxima alcanzada en la prueba inicial sin drones de red, evidenciando un incremento efectivo en la cobertura lograda por la red.

Las Figuras 8.17 y 8.19 muestran las trayectorias ejecutadas junto con

8.1. Diseño y Resultados de los Experimentos

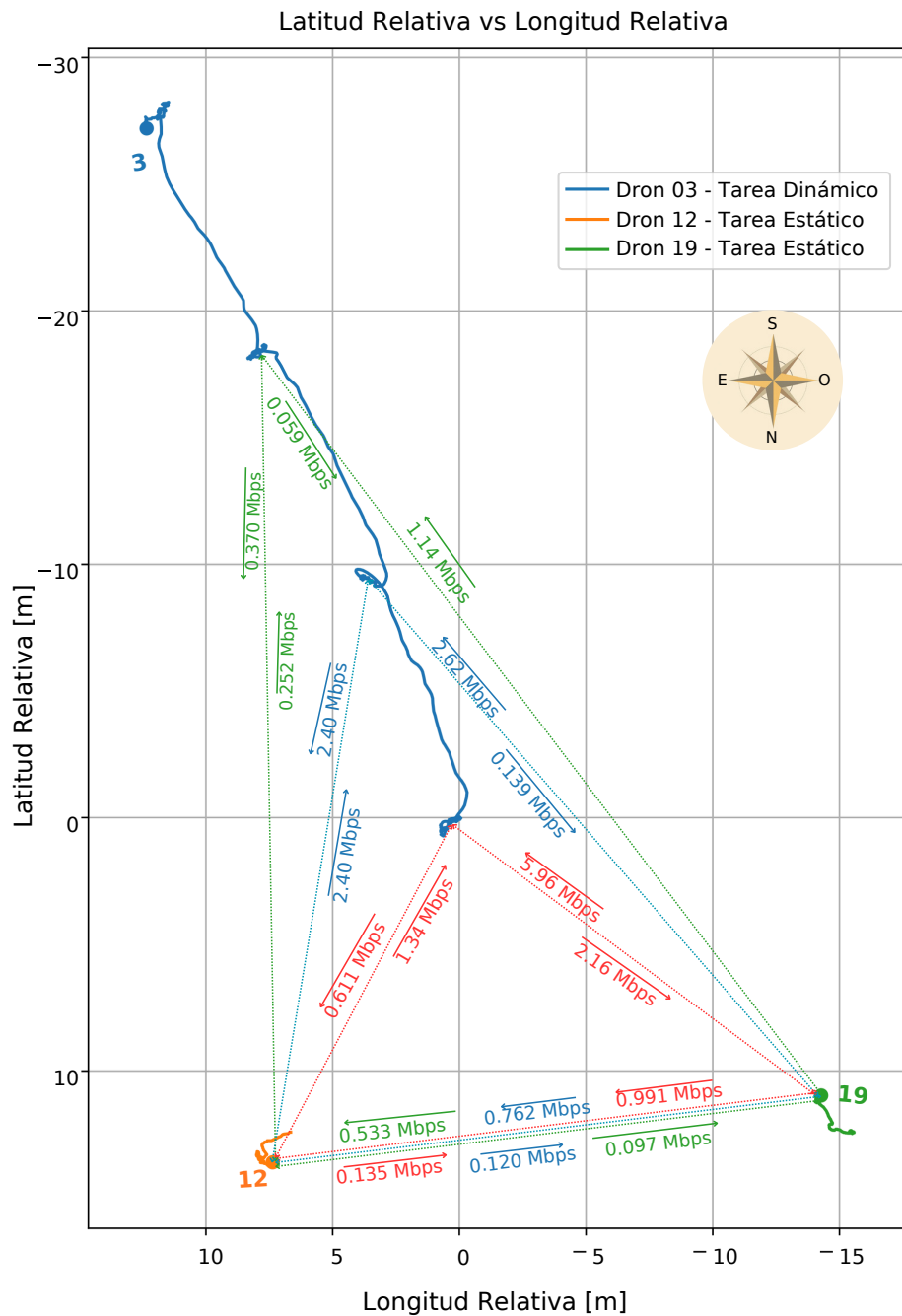


Figura 8.12: Trayectorias reportadas por los datos de GPS de todos los drones en la prueba de validación dinámica sin drones de red, referidas a la latitud y longitud inicial del dron 3 que oficia de origen del gráfico. Luego se convierten esas diferencias a metros y se expresan las longitudes en función de las latitudes para visualizar la trayectoria de cada dron. También se observan en diferentes colores (en cada punto de medición) los anillos formados por los drones de tarea con las tasas de transmisión en Mbps en ambos sentidos para cada par de drones reportados por iperf3.

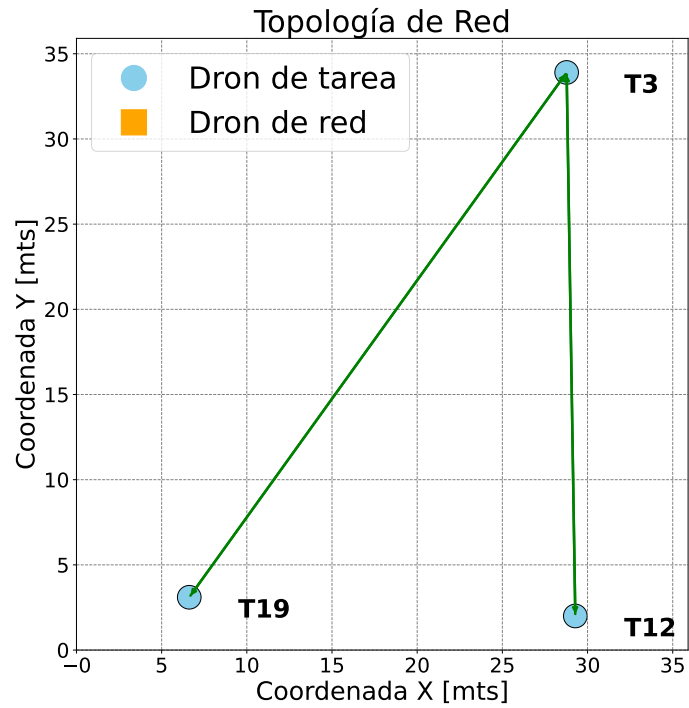


Figura 8.13: Topología de la red, representando los enlaces directos entre cada nodo, en el experimento realizado sin agentes de red a la máxima distancia alcanzable.

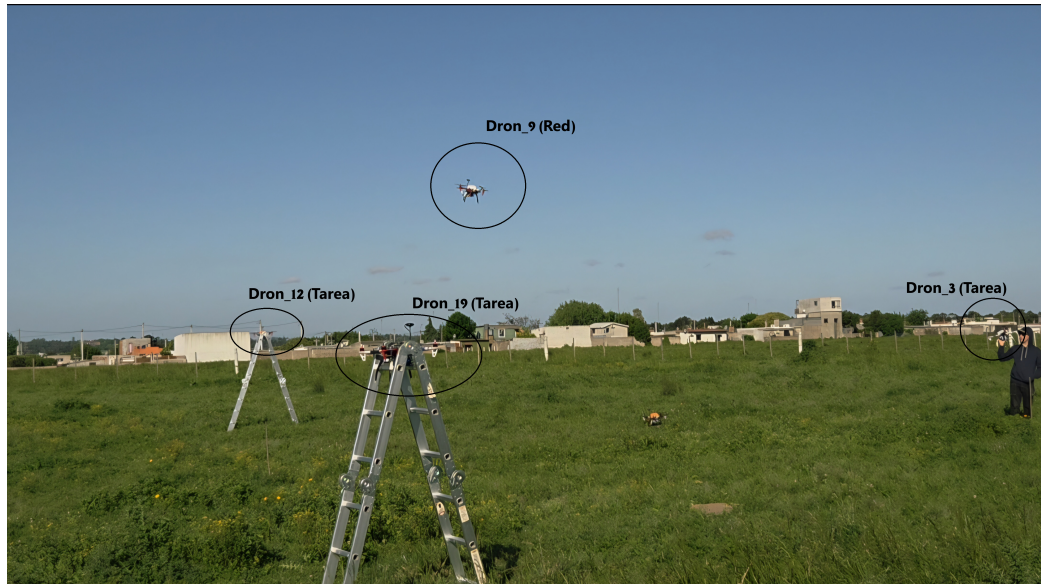


Figura 8.14: Prueba de enjambre con un dron de red dinámico y un dron de tarea móvil de forma manual.

8.1. Diseño y Resultados de los Experimentos

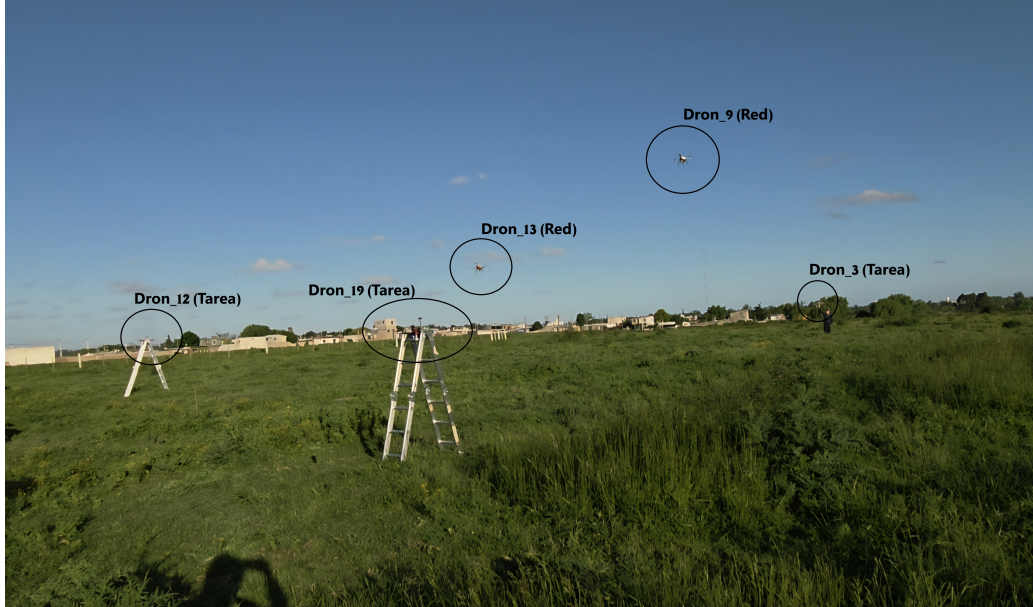


Figura 8.15: Prueba de enjambre con dos drones de red dinámicos y un dron de tarea móvil de forma manual.

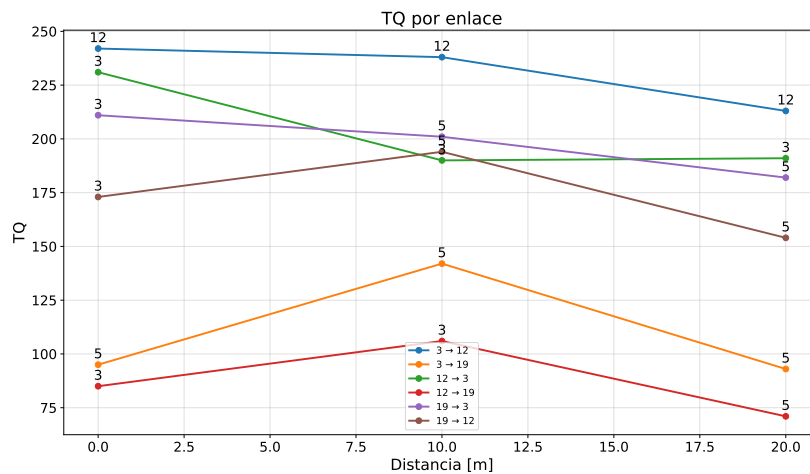


Figura 8.16: Calidad de cada enlace en función de la distancia del dron 3, sin drones de red.

Capítulo 8. Experimentos en Enjambre y Resultados

los valores de *throughput* obtenidos mediante `iperf3`. Lo más relevante en estas representaciones es la evidencia visual del comportamiento autónomo de los drones de red: estos ajustan continuamente sus posiciones en función del movimiento del dron *T3*, cumpliendo las órdenes generadas por el algoritmo implementado en la *Ground Station*.

En el caso de la configuración con un dron de red, se observa que dicho nodo tiende a ubicarse en el centroide formado por los tres drones de tarea, adoptando una posición estratégica para maximizar la conectividad entre ellos. Por su parte, en el escenario con dos drones de red (Figura 8.19), el dron *R9* se posiciona consistentemente en el centroide definido por el dron *T19* y el dron *T3*, mientras que el dron *R13* se desplaza en relación al conjunto completo conformado por los tres drones de tarea. Este comportamiento confirma que el algoritmo de posicionamiento dinámico genera una geometría coherente con el objetivo de optimizar la cobertura. Complementariamente, las Figuras 8.18 y 8.20 permiten observar con mayor detalle la evolución temporal de las posiciones relativas. En estas se aprecia claramente cómo los drones de red responden de forma reactiva a los cambios en la trayectoria del dron de tarea. En particular, en la Figura 8.20 se evidencia la interrupción del GPS del dron *T3* entre los segundos 120 y 190 aproximadamente. A pesar de esta condición adversa, la plataforma mantuvo un comportamiento robusto: los drones de red conservaron la última posición válida hasta que el dron *T3* recuperó señal (alrededor del segundo 200), momento en el cual actualizaron su ubicación inmediatamente, reestableciendo la geometría de operación prevista.

Aprovechando la ampliación observable en la cobertura obtenida con la incorporación de drones de red, se procedió a extender el experimento aumentando la distancia hasta 60 m. El objetivo fue verificar si, además de mejorar el *throughput* puntual, la plataforma era capaz de sostener la comunicación *multihop* entre todos los drones de tarea en un escenario de mayor exigencia. Para este análisis se consultaron las tablas de ruteo registradas en la base de datos para cada nodo.

Los resultados muestran que, aún en esta distancia extendida, la conectividad entre todos los drones de tarea se mantuvo en todo momento, existiendo siempre al menos un camino entre cada par de ellos. Esta topología se observa en la Figura 8.21. De manera particular, se evidencia que el dron *T19* recurrió al dron de red *R9* para enrutar sus mensajes, lo que confirma nuevamente que *T19* fue el eslabón más débil del enlace y que la presencia del nodo de red resultó crítica para sostener su conectividad. Este comportamiento valida que la plataforma no solo extiende el alcance efectivo de la red, sino que asegura la interconexión de todos los drones de tarea mediante comunicación *multihop*, demostrando que el dron de red cumplió adecuadamente su función de brindar cobertura y soporte de enlace.

Finalmente, en la Figura 8.22 se resume el desempeño global de la red

8.1. Diseño y Resultados de los Experimentos

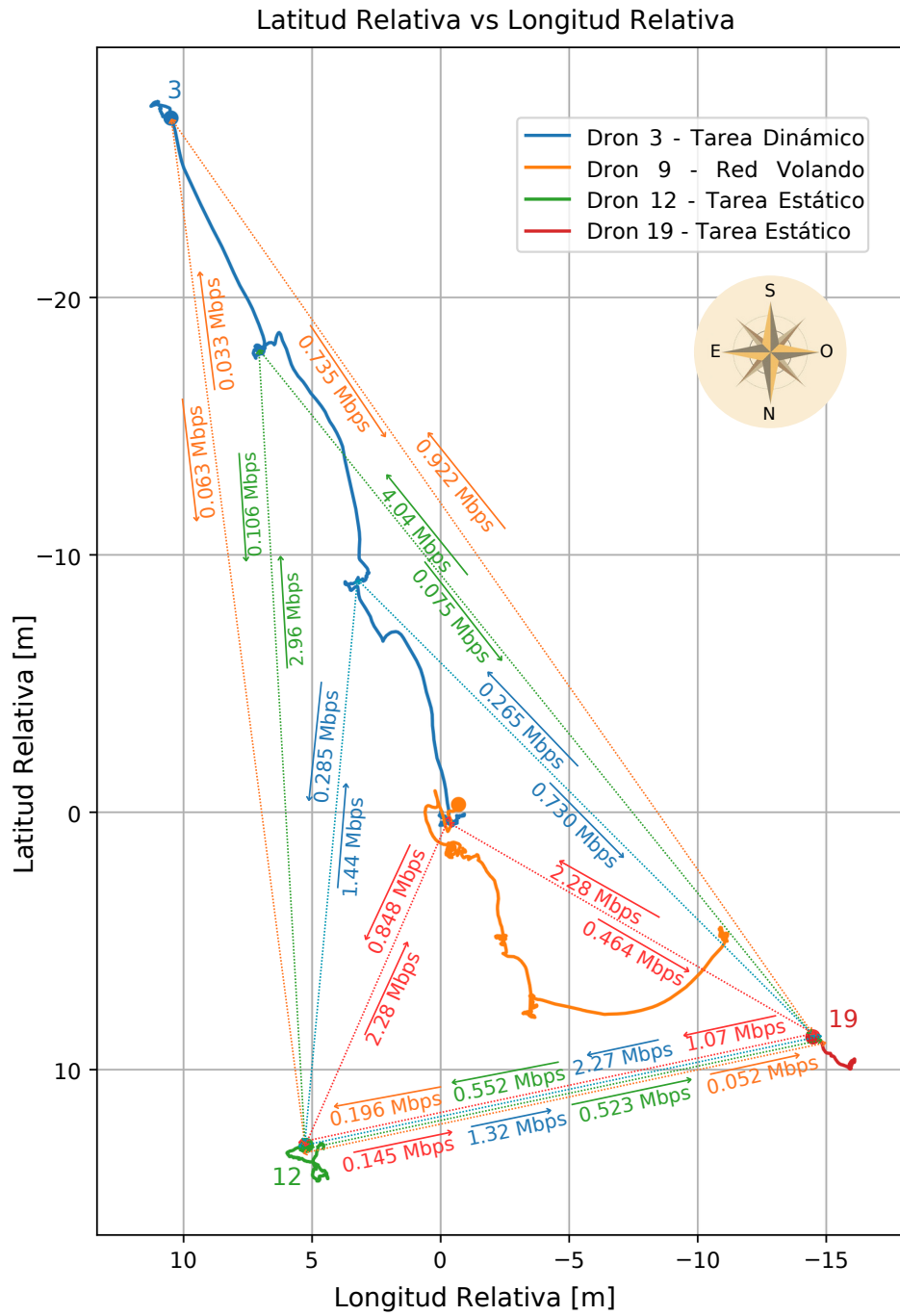


Figura 8.17: Trayectorias reportadas por los datos de GPS de todos los drones en la prueba de validación dinámica con un dron de red, referidas a la latitud y longitud inicial del dron 3 que oficia de origen del gráfico. Luego se convierten esas diferencias a metros y se expresan las longitudes en función de las latitudes para visualizar la trayectoria de cada dron. También se observan en diferentes colores (en cada punto de medición) los anillos formados por los drones de tarea con las tasas de transmisión en Mbps en ambos sentidos para cada par de drones reportados por iperf3.

Capítulo 8. Experimentos en Enjambre y Resultados

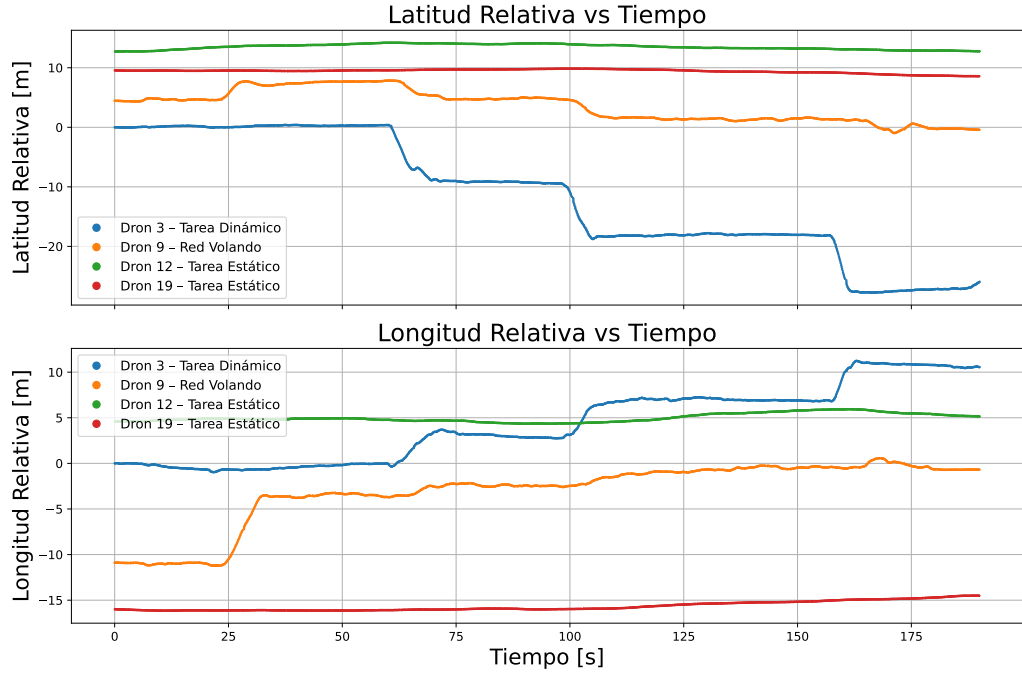


Figura 8.18: Resultados de la prueba de validación dinámica con un dron de red en gráficos de latitud y longitud relativas al punto inicial del dron 3 expresadas en metros en función del tiempo.

mediante el cálculo del *throughput* agregado Q_d definido en la Ecuación 8.1, en conjunto con el promedio del parámetro TQ de B.A.T.M.A.N. entre cada par de nodos de tarea.

Para cada distancia evaluada y para cada una de las configuraciones (sin drones de red, con uno y con dos drones de red), se obtiene la suma de los flujos concurrentes registrados por *iperf3*. Estos valores provienen directamente de las mediciones presentadas en los anillos de las Figuras 8.19, 8.17 y 8.12, que representan el rendimiento de cada enlace en cada punto del recorrido. En este caso al igual que en las pruebas estáticas, se vuelven a encontrar asimetrías en esta medición debido a la competencia por el acceso al medio. Inicialmente independientemente de la cantidad de drones de red parece ser que el *throughput* disminuye con la distancia. No obstante, se observó el punto máximo de este factor a los 20 m con un solo dron de red en el enjambre. Esto igualmente no nos caracteriza la red ya que en este caso, dos de los seis flujos acapararon todo el canal inalámbrico, logrando esa mejora que no es representativa de la totalidad de la red.

Este experimento permitió verificar que la plataforma efectivamente extiende el rango de comunicación entre los nodos de tarea, aprovechando el ruteo mediante los drones de red cuando la separación entre dos nodos supera el alcance directo del enlace. Sumado a esto, se confirma que los drones de red

8.1. Diseño y Resultados de los Experimentos

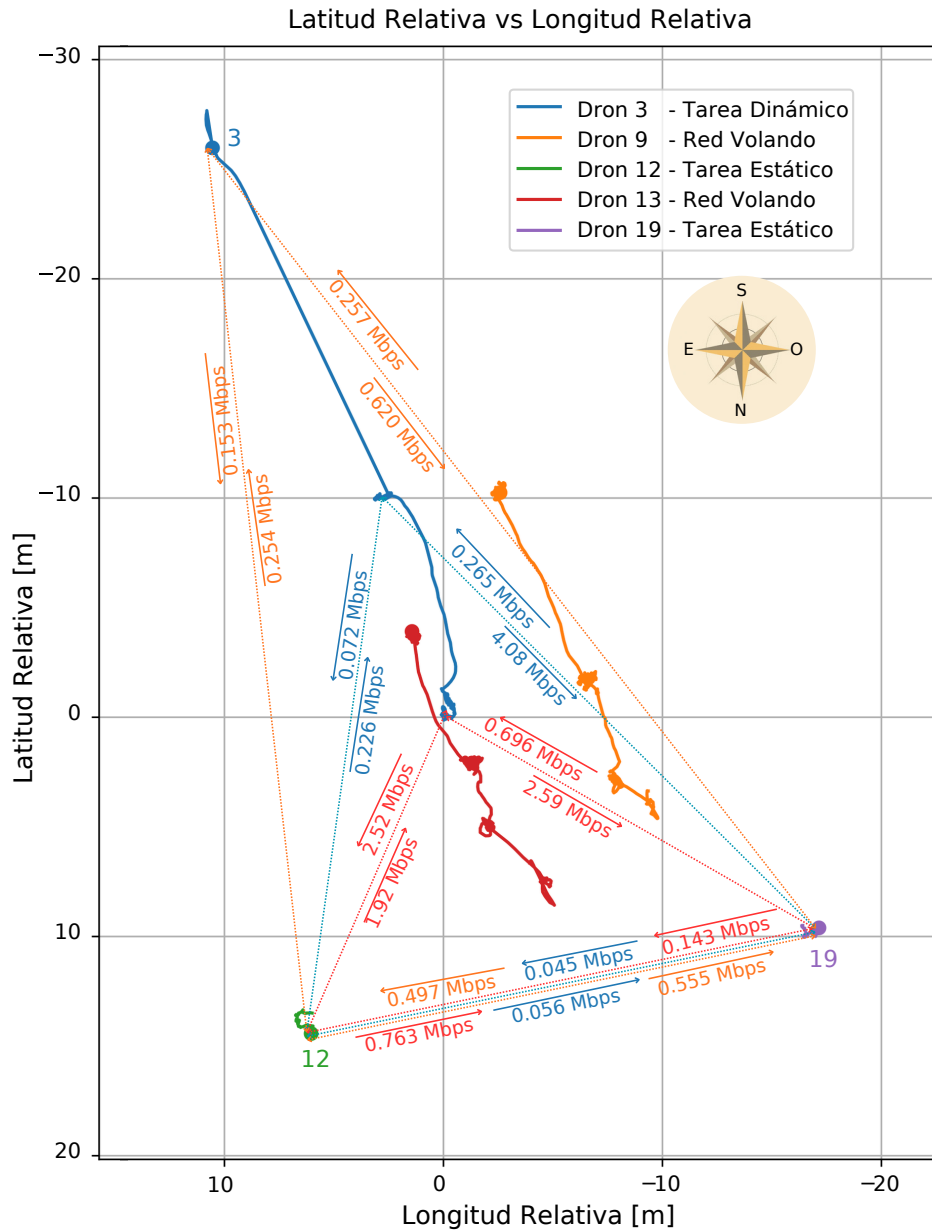


Figura 8.19: Trayectorias reportadas por los datos de GPS de todos los drones en la prueba de validación dinámica con dos drones de red, referidas a la latitud y longitud inicial del dron 3 que oficia de origen del gráfico. Luego se convierten esas diferencias a metros y se expresan las longitudes en función de las latitudes para visualizar la trayectoria de cada dron. También se observan en diferentes colores (en cada punto de medición) los anillos formados por los drones de tarea con las tasas de transmisión en Mbps en ambos sentidos para cada par de drones reportados por iperf3.

Capítulo 8. Experimentos en Enjambre y Resultados

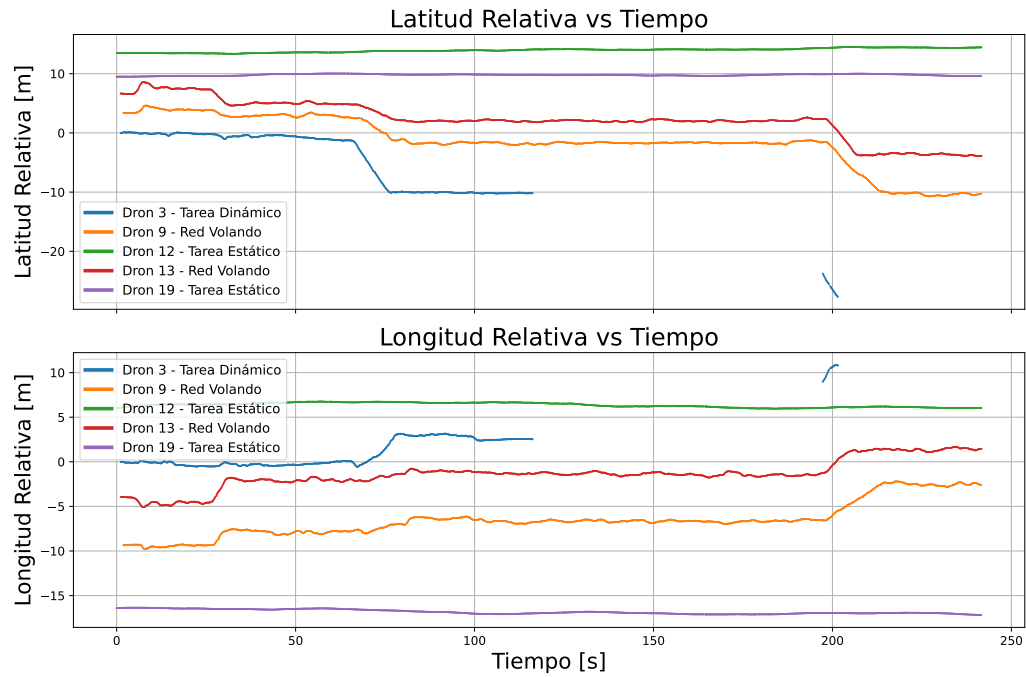


Figura 8.20: Resultados de la prueba de validación dinámica con dos dron de red en gráficos de latitud y longitud relativas al punto inicial del dron 3 expresadas en metros en función del tiempo.

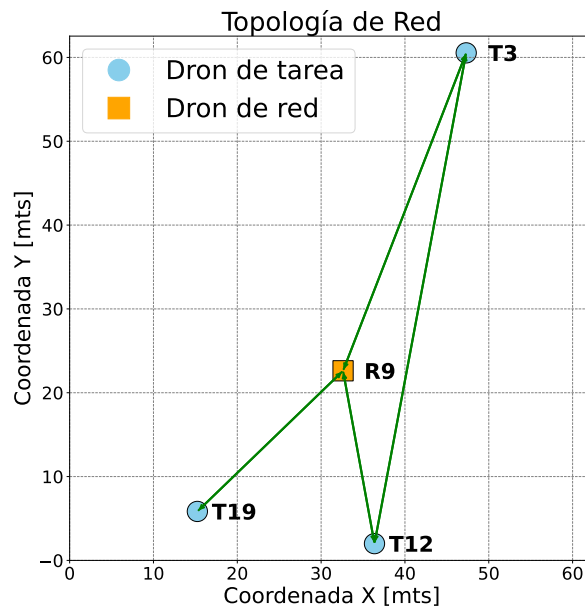


Figura 8.21: Representación de los enlaces directos entre cada nodo en el experimento realizado con tres drones de tarea y un dron de red, cada uno en su posición global.

8.1. Diseño y Resultados de los Experimentos

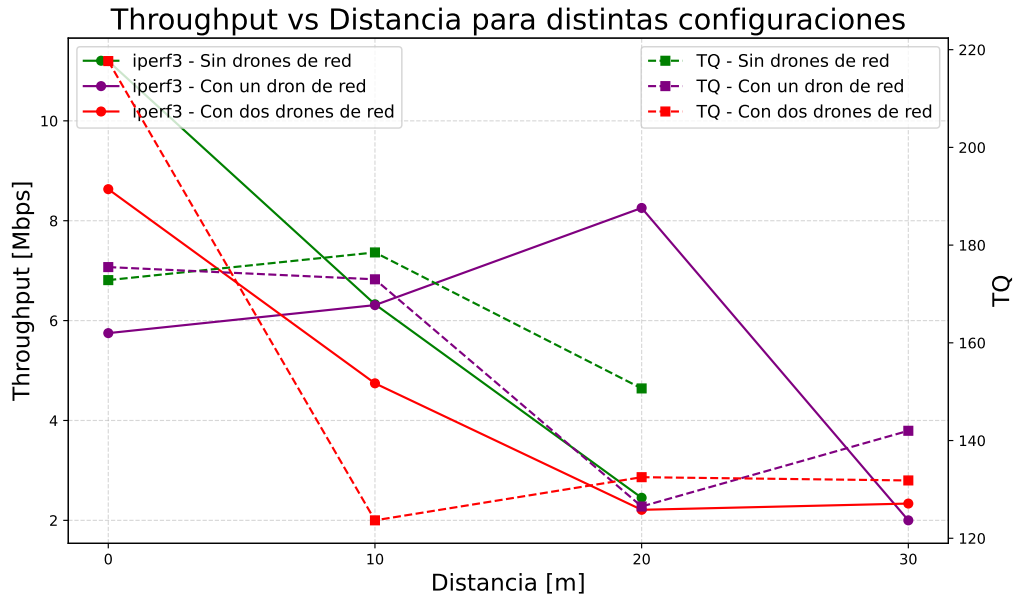


Figura 8.22: Total del throughput de cada enlace tomado con la herramienta iperf3 en conjunto con el promedio de la estimación de la calidad del enlace brindada por B.A.T.M.A.N., en función de la distancia para las distintas configuraciones de la red.

cumplen adecuadamente su función dentro de la plataforma y se posicionan acorde al algoritmo que corre la *Ground Station*. Finalmente se confirma nuevamente, que la plataforma permite hacer pruebas del canal, aún con drones de red volando.

8.1.4. Prueba de Validación con Nuevos Algoritmos

El objetivo de esta prueba es validar los aspectos descritos en la Sección 5.5. En particular, se busca comprobar que si los algoritmos aceptan y devuelven los tipos de datos descritos en 5.3.1. La implementación del cliente y servidor ZMQ, resultan eficientes y adecuadas para la plataforma. Para ello, se realizó una evaluación básica empleando la misma configuración planteada en la prueba anterior 8.1.3, utilizando un único dron de red, con el fin de verificar que responde correctamente a las órdenes generadas por el algoritmo GIOD Centralized visto en la Sección 6.3.

Ejecución, Resultados y Análisis

En la Figura 8.23 se presentan las trayectorias obtenidas durante la prueba de validación para los algoritmos nuevos con el algoritmo GIOD Centralized. Se verifica que el dron de red *R9* corrige su posición en función de las salidas provistas por el algoritmo, manteniendo coherencia con los desplazamientos

Capítulo 8. Experimentos en Enjambre y Resultados

del dron 3 de tarea (desplazado manualmente). En comparación con los movimientos analizados en las pruebas anteriores con el algoritmo C-DFLP, se observa que la naturaleza de estas correcciones difiere significativamente. En este caso, los ajustes de posición se realizan de manera más “suave”, lo cual responde a la propia lógica del algoritmo, que devuelve un punto objetivo en coordenadas geográficas pero, en relación a la optimización de la dirección hacia la cual debería desplazarse el dron para mejorar la conectividad.

Asimismo, se confirmó la eficiencia del enfoque basado en un servidor externo persistente, encargado de ejecutar el algoritmo y entregar los resultados cuando la *Ground Station* lo solicite. Esta separación favorece la escalabilidad y modularidad de la solución, evitando que el algoritmo se ejecute en el mismo entorno operativo de la *Ground Station*. Adicionalmente, se incorporó un temporizador para medir el tiempo de ejecución del algoritmo, lo cual resulta relevante para algoritmos de mayor complejidad o latencia, permitiendo ajustar la frecuencia con la cual la *Ground Station* actualiza las posiciones de los drones de red mediante consultas al servidor externo.

8.1. Diseño y Resultados de los Experimentos

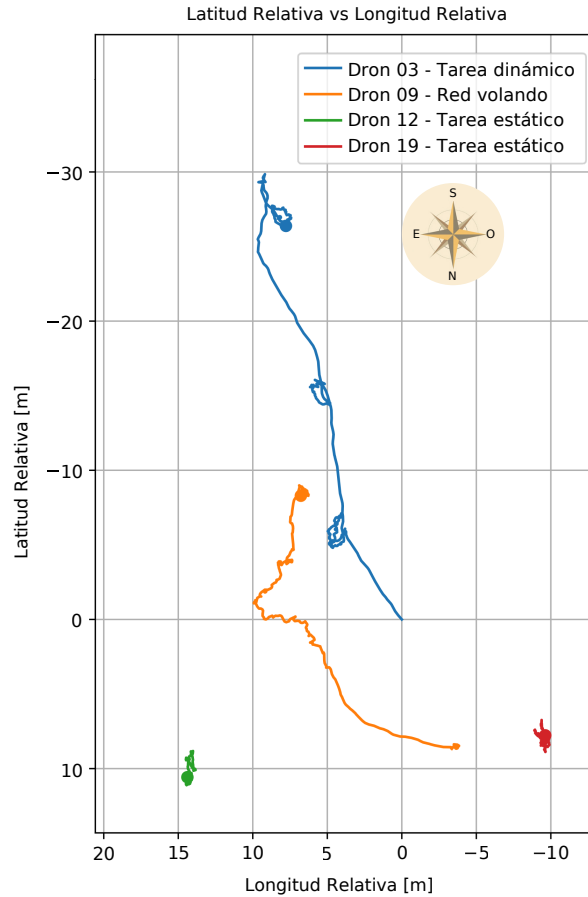


Figura 8.23: Resultados de la trayectoria de la prueba de validación de nuevos algoritmos con un dron de red en gráficos de latitud en función de la longitud.

Este capítulo integró y documentó todos los aspectos vinculados a la ejecución de pruebas reales de campo con el enjambre de drones, incluyendo las precauciones operativas, la logística requerida, los datos recolectados y las herramientas empleadas para su posterior análisis. A través de los experimentos desarrollados se comprobó que la plataforma es capaz de ejecutar correctamente el algoritmos de posicionamiento, actualizar las posiciones de los drones de red en tiempo real y sostener comunicaciones mediante una red ad-hoc *multihop*. Asimismo, se verificó la posibilidad de medir y analizar las tasas de transmisión de los enlaces, información que resulta especialmente relevante ante la eventual incorporación de agentes de tarea de otro tipo que pretendan utilizar la infraestructura de red para operar.

En síntesis, este capítulo resume un tramo fundamental del proyecto, en el cual las pruebas de campo (siempre demandantes en términos de planificación y logística) permitieron validar en un entorno real tanto el comportamiento del algoritmo como el funcionamiento coordinado de la plataforma.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 9

Conclusiones

El proyecto cumplió el objetivo general de disponer de una plataforma funcional para el vuelo coordinado en enjambre, capaz de integrar drones de tarea y drones de red gestionados por una estación terrestre que calcula posiciones objetivo y supervisa la misión. El sistema mostró un comportamiento estable tanto en simulaciones como en vuelos reales, incluyendo escenarios con hasta cinco drones operando de manera simultánea. En relación con los objetivos específicos, se diseñó y construyó una flota de vehículos con una arquitectura de *hardware* homogénea (que abarca estructura, sistema de propulsión, controladora de vuelo y computadora a bordo), debidamente configurada y calibrada para vuelo autónomo. Se implementó una red ad-hoc basada en batman-adv que soporta el intercambio de telemetría, métricas de enlace y órdenes de movimiento entre los nodos. Asimismo, se desarrolló una arquitectura de software sobre ROS 2, MAVSDK y PX4 que integra funciones de control, comunicación y monitoreo en tiempo real. Finalmente, se diseñó y ejecutó un conjunto de experimentos individuales y en enjambre que permitió validar tanto la plataforma como los algoritmos de posicionamiento integrados.

Sobre esta base, se obtuvo una plataforma robusta para la experimentación con drones que integra simulación y pruebas de algoritmos de posicionamiento en escenarios donde resulta crucial mantener un flujo de información constante entre los agentes. La solución desarrollada se caracteriza por su escalabilidad, al permitir incorporar nuevos parámetros y variantes de algoritmos de posicionamiento de manera parametrizable y generalizable, y por su potencial de ampliación mediante la integración futura de modelos de simulación de red más detallados dentro del propio entorno. Cabe señalar que una parte importante del trabajo intermedio (incluyendo pruebas exploratorias y desarrollo de herramientas auxiliares para validar la red, la simulación y el entorno en ROS 2) fue determinante para consolidar la plataforma actual. En este contexto, la plataforma cumple adecuadamente el objetivo de ejecutar algoritmos de posicionamiento para drones de red con distintos criterios de

Capítulo 9. Conclusiones

optimización, como por ejemplo para el aumento de la cobertura.

Las pruebas realizadas fueron satisfactorias de acuerdo con el análisis presentado en el capítulo de resultados; sin embargo, el desarrollo del proyecto permitió identificar diversas oportunidades de mejora que orientan líneas de trabajo futuras tanto en el plano técnico como organizativo. En primer lugar, un control más sofisticado del enjambre, con mayor capacidad de control, abriría la puerta a experimentar con algoritmos de posicionamiento descentralizados. Esto implicaría adaptar la plataforma para que cada dron de red pueda calcular localmente sus objetivos a partir de información parcial del entorno, intercambiando únicamente la información estrictamente necesaria con sus vecinos, y permitiría estudiar fenómenos como la convergencia del enjambre, la estabilidad de la formación y la sensibilidad frente a fallas o retrasos en la comunicación.

En segundo lugar, los estudios sobre el canal de comunicación podrían profundizarse mediante experimentos en espacios mas amplios, que incluyan ensayos a mayores distancias, escenarios con menos restricciones sobre la potencia de transmisión y la comparación sistemática de diferentes protocolos de ruteo para redes ad-hoc, evaluando su desempeño en términos de *throughput*, retardo, pérdida de paquetes y comportamiento frente a la movilidad. En este mismo contexto, resultaría natural incorporar mecanismos de ciberseguridad en el tráfico compartido, tales como cifrado, autenticación y gestión básica de claves, de manera de evaluar no solo el rendimiento bruto de la red, sino también su capacidad de operar de forma segura ante posibles interferencias o accesos no autorizados.

Otro eje de mejora consiste en realizar ensayos con drones más potentes, capaces de transportar mayor carga útil y disponer de tiempos de vuelo más prolongados; ello permitiría integrar sensores adicionales o equipos de comunicaciones más exigentes y, al mismo tiempo, diseñar experimentos donde los agentes de tarea transmitan datos de aplicación reales, como flujos de vídeo o mediciones de sensores, para analizar la calidad de experiencia de extremo a extremo y no solo métricas de red abstractas.

Desde el punto de vista de la ingeniería del software y de la infraestructura de soporte, se identificó la conveniencia de mantener una organización más estricta y homogénea de los repositorios y de la estructura del código, así como dedicar más tiempo al desarrollo y pulido de las herramientas auxiliares: una página de monitoreo más completa y flexible, una base de datos persistente para conservar el historial de vuelos y métricas, y nuevos mundos en los entornos de simulación que representen escenarios más diversos y desafiantes (por ejemplo, entornos urbanos, obstáculos complejos o condiciones operativas particulares).

Finalmente, la realización de experimentos específicos orientadas a estudiar la resiliencia a fallas (mediante la introducción controlada de fallos de nodos,

enlaces o componentes de la infraestructura, y el diseño de mecanismos de reconfiguración automática y tolerancia a errores) permitiría robustecer la aplicación y consolidar la plataforma como un entorno de experimentación aún más sólido. Si bien las limitaciones de *hardware* y el tamaño acotado del equipo exigieron priorizar tareas y acotar ciertos alcances, el trabajo realizado deja sentadas las bases para que futuras generaciones puedan extender la plataforma, migrarla a *hardware* más moderno y explorar en profundidad estas líneas de mejora, aprovechando la misma como un punto de partida estable, escalable y reutilizable para nuevos proyectos en enjambres de drones y redes inalámbricas ad-hoc.

Esta página ha sido intencionalmente dejada en blanco.

Apéndice A

Pixhawk 2.4.8

En esta sección se explican detalles de la controladora de vuelo utilizada en el proyecto.

A.1. Microcontroladores

- **STM32F427VIT6 (FMU – Flight Management Unit):** Es el microcontrolador principal de la Pixhawk 2.4.8. Se trata de un MCU de la familia de 32-bit, STM32 de STMicroelectronics, con núcleo ARM Cortex-M4 con unidad de punto flotante (FPU). Funciona a 168 MHz y dispone de 256 KB de RAM y 2 MB de memoria Flash. Este procesador ejecuta el sistema operativo en tiempo real NuttX y el firmware de vuelo (PX4v1.10.1 en nuestro caso, pero podría ser ArduPilot). Sus prestaciones (CPU de 32-bit, alto clock, FPU) le permiten realizar cálculos intensivos necesarios para controlar la aeronave (por ejemplo, filtros Kalman, control PID, navegación). Cabe señalar que debido a una errata de hardware en revisiones antiguas de este chip, en algunas Pixhawk la flash utilizable estaba limitada a 1 MB. No obstante, muchas unidades 2.4.8 modernas usan el silicio revisado (Rev3) sin esa limitación. Rol en el control de vuelo: el STM32F427 (a veces referido como PX4FMU) se encarga de leer todos los sensores, ejecutar los algoritmos de estimación de estado y control, gestionar la comunicación con periféricos (GPS, radio telemetría, etc.) y coordinarse con el co-procesador de E/S. En esencia, es el “cerebro” de la controladora [55] [56].
- **STM32F100C8T8 (IO MCU – Entrada/Salida):** Es un microcontrolador menos potente que el de la FMU, es de 32 bits (Cortex-M3) corriendo a 24 MHz, con 8 KB de RAM y 64 KB de Flash. Actúa como co-procesador dedicado a las funciones de entrada/salida. En la arquitectura Pixhawk se encarga específicamente de generar las señales

Apéndice A. Pixhawk 2.4.8

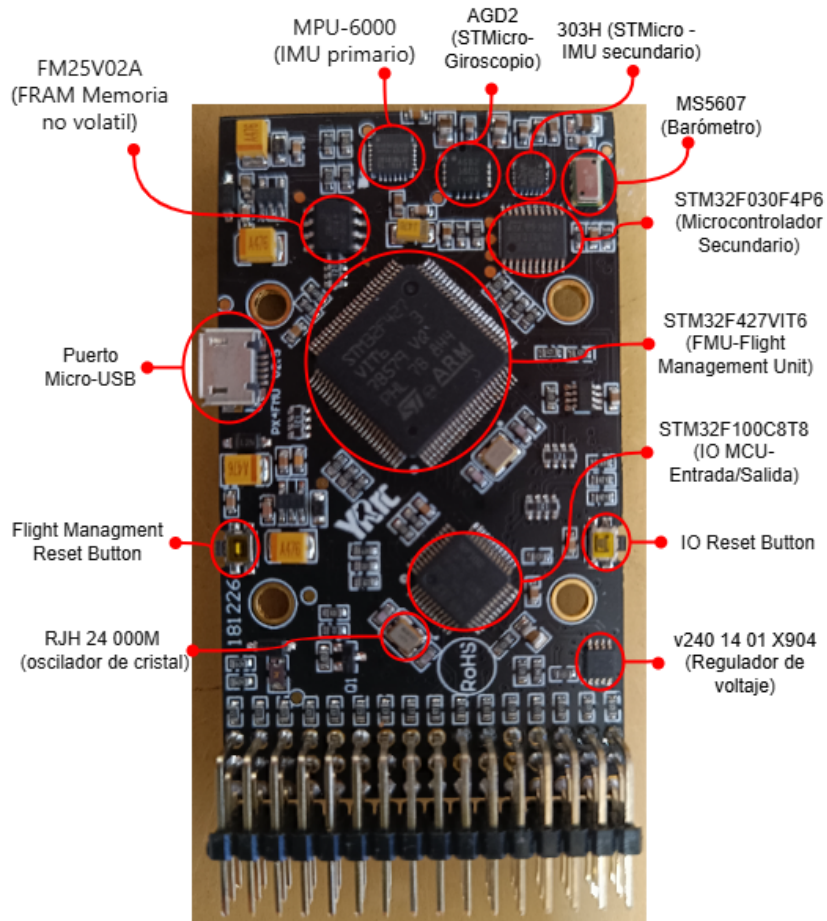


Figura A.1: Detalles de los microcontroladores y sensores integrados en la placa de la Pixhawk 2.4.8.

PWM para los motores(o servos si los hubiese) y de manejar las entradas PPM/S.Bus del receptor de radio. Su inclusión permite que las tareas de tiempo real más críticas (por ejemplo, mantener la generación de PWM a 400 Hz) y funciones de failsafe se ejecuten independientemente del hilo principal en el FMU. En caso extremo de que el FMU deje de responder, el STM32F100 puede tomar el control y poner a los actuadores en un estado seguro. Además, este IO MCU integra la lógica de mixing de servos/motores: combina las salidas virtuales de control (roll, pitch, yaw, throttle) en comandos específicos para cada motor. Tener un microcontrolador separado para E/S alivia la carga del procesador principal y añade una capa de seguridad. Este chip corre un firmware de E/S más simple (PX4IO-ardupilot IO) supervisado por el FMU [55] [57].

- **STM32F030F4P6 (Secundario y Auxiliar):** Adicionalmente, se

integra un tercer microcontrolador STM32 de la familia F0, basado en el núcleo Cortex-M0. Este componente posee recursos más modestos, con una frecuencia de reloj de hasta 48 MHz, 16 Kbytes de memoria Flash y 4 Kbytes de RAM. Sus funciones se centran en tareas de bajo nivel, como la gestión de la lógica del botón de arranque, el control de las luces LED RGB, la inicialización del sistema y la operación como puente USB en modo bootloader, liberando así a la Unidad de Vuelo (FMU) de dichas responsabilidades [55] [58].

A.2. Sensores

- **MPU-6000 (InvenSense):** El IMU primario, es un modulo de 6 ejes que combina acelerometro triaxial y giroscopio triaxial en un solo chip. Con tecnologia MEMS y comunicacion SPI o I2C con el FMU. Este IMU es la referencia principal y es fundamental para estimar la actitud (inclinacion) del dron y estabilizar el vuelo. Suele montarse al centro de la placa.
- **L3GD20 (STMicro, marcado como “AGD2”):** Giroscopio de 3 ejes.
- **STM32F030F4P6 (Secundario y Auxiliar)**

Esta página ha sido intencionalmente dejada en blanco.

Apéndice B

Conceptos Útiles

En esta sección se mencionan algunos conceptos útiles para comprensión del documento.

- **Firmware:** El firmware de un dron es el conjunto de software que está instalado directamente en el hardware de la controladora de vuelo corriendo en un RTOS (“Real Time OS”) y que controla su funcionamiento básico. Es una especie de “cerebro” que le permite ejecutar las funciones que definen el comportamiento del dron, como la navegación, el control del vuelo, la respuesta a comandos del piloto, la estabilización en el aire, la gestión de la batería, y la interacción con los sensores (como cámaras o sistemas GPS) [59] [60].

Este firmware se encuentra en una memoria no volátil dentro del dron, lo que significa que a diferencia del software que se ejecuta en una computadora, no se borra cuando el dron se apaga. Los fabricantes suelen actualizar el firmware periódicamente para mejorar el rendimiento, añadir nuevas funciones o corregir errores, y es esencial para garantizar que el dron funcione de manera segura y eficiente.

En resumen, el firmware de un dron es el software embebido en el hardware que gestiona sus operaciones y asegura que todas sus partes trabajen de manera coordinada [61].

- **Airframe:** El airframe de un dron es la estructura física que sostiene todos los componentes del vehículo aéreo no tripulado (VANT). En términos simples, es el “cuerpo” del dron, que incluye el chasis o armazón y todos los elementos que lo componen, como los brazos, las hélices, los motores y los puntos de montaje para la cámara, la batería, los sensores y la electrónica.
- **Flight Modes:**

Apéndice B. Conceptos Útiles

Los modos de vuelo proporcionan distintos tipos y niveles de automatización del vehículo y asistencia del piloto automático al usuario (piloto). Los modos autónomos están totalmente controlados por el piloto automático y no requieren la intervención del piloto o del control remoto. Se utilizan, por ejemplo, para automatizar tareas comunes como el despegue, el regreso a la posición de origen y el aterrizaje. Otros modos autónomos ejecutan misiones preprogramadas, siguen una baliza GPS o aceptan comandos de una computadora externa o una estación terrestre. Los modos manuales son controlados por el usuario (a través de los joysticks o palancas de control remoto) con la ayuda del piloto automático. Los distintos modos manuales permiten distintas características de vuelo; por ejemplo, algunos modos permiten realizar acrobacias, mientras que otros son imposibles de volar y mantendrán la posición o el rumbo contra el viento [62] [63].

Dependiendo de la aeronave, PX4 soporta distintos modos de vuelo, para el caso de los multicopteros transcribimos los más relevantes que utilizamos en el proyecto:

Modos Manuales:

- **Position:** el modo “Position” es uno de los más recomendados para principiantes, ya que es el que ofrece mayor asistencia en el control manual del dron. Utiliza el sensor de presión barométrica y el receptor GPS para estabilizar el dron tanto en el eje vertical (altitud) como horizontal (posición). En este modo, el dron mantiene una posición constante y puede corregir su trayectoria automáticamente, lo que facilita el vuelo, especialmente para quienes están aprendiendo.

Este modo requiere una buena estimación de la posición global, proporcionada por el GPS, para armar y volar correctamente. La precisión del GPS es crucial para que el dron se mantenga estable, ya que si la señal es débil o imprecisa, la capacidad de mantener la posición podría verse afectada. Además, el modo “Position” es ideal para vuelos de corta distancia y en entornos donde se puede confiar en una señal GPS estable [64].

- **Altitude:** a diferencia del modo “Position”, el modo “Altitude” proporciona estabilidad vertical utilizando el sensor de presión barométrica, pero deja a cargo del piloto la estabilidad horizontal, ya que no incluye frenado ni mantenimiento de la posición en el plano horizontal. En este modo, el dron mantiene una altitud constante, pero no corrige ni controla su desplazamiento lateral o

hacia adelante/atrás. Esto significa que el piloto debe gestionar manualmente los movimientos horizontales.

Por otro lado, el modo “Altitude” no depende de la señal GPS para mantener la altitud, lo que lo hace adecuado en entornos donde la señal GPS puede ser débil o inestable. Sin embargo, si bien no usa GPS para la estabilización vertical, puede aprovecharse del GPS para otras funciones de navegación, como la velocidad [65].

Modos Autónomos:

- **Hold:** el modo “Hold” permite que el dron permanezca suspendido en su posición, tanto horizontal como verticalmente, resistiendo vientos y otras fuerzas externas. En este modo, el dron utiliza tanto el GPS como el barómetro para mantener su posición y altitud. Si el dron detecta fuerzas externas como vientos fuertes, ajusta su orientación y motores para contrarrestar esas influencias y mantener la posición estable.

Este modo es especialmente útil en situaciones donde se requiere que el dron se mantenga en una ubicación precisa, sin necesidad de intervención constante del piloto. Sin embargo, es importante tener en cuenta que la precisión en la estabilización de la posición depende de la calidad de la señal GPS y de las condiciones atmosféricas. En áreas con una señal GPS débil o en espacios cerrados, la efectividad del modo “Hold” podría verse limitada [66].

- **Return:**

el modo “Return” tiene como objetivo principal regresar al punto de despegue o a un punto de destino seleccionado previamente. En este modo, el dron primero desciende a una altitud segura, la cual es ajustable por el piloto antes de iniciar el vuelo. Posteriormente, se dirige automáticamente hacia las coordenadas GPS del punto de despegue, utilizando su sistema de navegación para calcular la mejor ruta. Una vez que alcanza la ubicación de origen, el dron realiza un aterrizaje autónomo en ese punto.

Este modo es ideal en situaciones donde el piloto pierde la referencia visual del dron o en caso de pérdida de señal de control. Sin embargo, la efectividad del “Return” depende de una señal GPS sólida, ya que el dron necesita una ubicación precisa para iniciar el regreso. Además, si el punto de despegue o el destino seleccionado se encuentra en un área con obstáculos o condiciones que interfieran con el GPS, el regreso automático podría verse comprometido [67].

- **Mission:**

Apéndice B. Conceptos Útiles

el modo “Mission” permite configurar previamente coordenadas GPS de distintos waypoints (puntos de referencia) para que el dron navegue automáticamente entre ellos. En este modo, el piloto define una ruta que el dron seguirá de forma autónoma, visitando cada waypoint en el orden establecido. Durante el vuelo, el dron ajusta su trayectoria y altitud en función de los waypoints programados, lo que permite realizar vuelos de largo alcance o tareas repetitivas sin intervención constante del piloto.

Este modo es muy útil para misiones de mapeo, inspección o cualquier tarea que requiera que el dron pase por ubicaciones específicas de forma precisa. La exactitud en la navegación dependerá de la calidad de la señal GPS y de la calibración de los sensores del dron. Si el GPS es inestable o el área presenta interferencias, la precisión en el seguimiento de la ruta podría verse afectada [68].

- **Takeoff:** el modo “Takeoff” permite que el vehículo despegue de manera vertical hasta alcanzar una altitud segura antes de cambiar automáticamente al modo “Hold”. Durante el despegue, el dron se eleva de forma controlada, asegurando que se estabilice verticalmente mediante el uso del barómetro y otros sensores. Una vez alcanzada la altitud configurada, el dron se posiciona en el modo “Hold”, donde permanece suspendido en su ubicación, manteniendo su altitud y posición horizontal frente a vientos u otras fuerzas externas [69].
- **Land:** el modo “Land” permite que el dron aterrice de forma controlada. Durante este proceso, el dron utiliza principalmente el sensor de presión barométrica para medir su altitud y realizar un descenso suave y estable. Sin embargo, también es posible combinar el barómetro con sensores adicionales, como un sensor LIDAR (Light Detection and Ranging), para mejorar la precisión del aterrizaje, especialmente en entornos con poca visibilidad o cuando se requiere una mayor exactitud en la medición de la distancia al suelo [70].
- **Offboard:** el modo “Offboard” permite que el vehículo obedezca comandos de posición, velocidad o actitud proporcionados externamente, generalmente a través de MAVLink o ROS 2 (Robot Operating System 2). En este modo, el dron se desconecta de los controles tradicionales del piloto y sigue las órdenes enviadas por una computadora o sistema externo que proporciona setpoints (puntos de referencia) de navegación.

Este modo es especialmente útil en aplicaciones avanzadas como misiones autónomas o tareas en las que el dron debe ser controlado por un sistema externo, como un vehículo terrestre, un servidor

de control de misión o un sistema de control en tiempo real. La comunicación constante y confiable entre el dron y el sistema externo es crucial para el éxito de la misión. El uso de MAVLink o ROS 2 permite una integración flexible y precisa de los comandos de vuelo, ya que estos protocolos soportan la transmisión de datos complejos y en tiempo real. Es el modo ideal para los “nodos de red” de nuestro proyecto [71].

■ **Safety:**

Es un mecanismo que permite al dron ejecutar una acción determinada de seguridad. Por ejemplo, según el porcentaje de la capacidad de la batería. La “Failsafe Action” es la acción que elegimos (en nuestro caso, que aterrice) cuando la batería alcanza el nivel denominado “Battery Failsafe Level”. El “Battery Warn Level” es el nivel a partir del cual se registra un aviso (warning) en los registros del equipo debido al bajo nivel de batería, mientras que el “Battery Emergency Level” es el nivel en el que el dron aterriza por defecto ya que no tiene suficiente batería para continuar volando. Se detallan a continuación las acciones safety configurables:

- **RC Loss Failsafe Trigger:** le permite al dron ejecutar la acción configurada en “Failsafe Action” (en nuestro caso, que aterrice) si se pierde la comunicación entre el receptor y el transmisor (Control Remoto), y no se puede restablecer durante un tiempo de espera denominado “RC Loss Timeout”, el cual es configurable (dos segundos en nuestro caso).
- **Geofence Failsafe Trigger:** dispara una acción configurable en “Action on breach” (mantenerse en el aire, en nuestro caso) cuando el dron cruza virtualmente un perímetro delimitado por un cilindro virtual, cuyo centro está en la “home position”, con una “Max Altitude” configurable de 30 m (en nuestro caso) y un “Max Radius” también configurable de 200 metros (en nuestro caso).
- **Return To Launch Settings:** es una acción de seguridad que activa el modo de retorno a la ubicación de despegue. Se configura la opción “Climb to altitude of”, que define la altura a la que el dron regresa a la latitud y longitud del punto de despegue, y la acción a realizar al llegar a ese punto, que en nuestro caso es “Land immediately”.
- **Land Mode Settings:** cuando una acción de seguridad se activa y pone al dron en modo de aterrizaje, se configuran tanto la velocidad de descenso (“Landing Descent Rate”), que en nuestro caso es de 0.7 m/s, como el tiempo que espera para desarmarse después de

Apéndice B. Conceptos Útiles

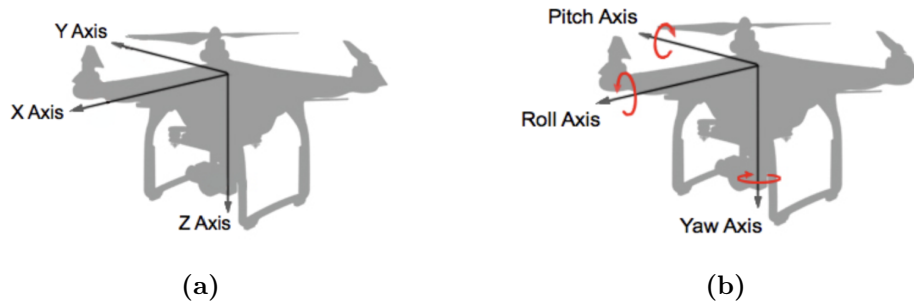


Figura B.1: Sistema de referencia corporal [73]

tocar tierra (“Disarm After”), que en nuestro caso son dos segundos [72].

- **Parameters:** Al modificar un parámetro, es fundamental leer detenidamente la descripción proporcionada por QGC, ya que explica qué estamos modificando. Además, es importante revisar las advertencias (‘warnings’) sobre los posibles efectos secundarios que podría generar la modificación.

B.1. Sistemas de coordenadas

El movimiento de la aeronave se describe en función de la posición y orientación de los ejes que definen el sistema de coordenadas o marco de referencia. Se pueden clasificar en marcos de referencia corporal y marco de referencia mundial.

B.1.1. Sistema de coordenadas corporales:

El sistema de coordenadas corporales está definido con respecto a la aeronave. Se establecen tres ejes ortogonales con origen en el centro de masa: el eje X apunta hacia la parte frontal, el eje Y hacia el lado derecho y, siguiendo la regla de la mano derecha, el eje Z apunta hacia la parte inferior de la aeronave. En este marco de referencia, los desplazamientos positivos a lo largo de los ejes X, Y y Z corresponden a traslación hacia adelante, hacia la derecha y hacia abajo, respectivamente.

La rotación de la aeronave también se expresa en función de estos ejes, utilizando la regla de la mano derecha para definir las direcciones positivas. Las rotaciones alrededor de los ejes X, Y y Z se denominan alabeo (roll), cabeceo (pitch) y guiñada (yaw), respectivamente como se observa en la Figura B.1b.

B.1. Sistemas de coordenadas

Sistema de coordenadas terrestres (mundiales):

Un sistema de coordenadas global comúnmente utilizado en aeronáutica es el sistema NED (North-East-Down), en el que los ejes X, Y y Z positivos están alineados con las direcciones norte, este y hacia abajo, respectivamente.

Aunque puede resultar poco intuitivo que el eje Z positivo apunte hacia abajo, esta convención es útil porque mantiene la coherencia con la regla de la mano derecha y con la definición estándar de los ángulos de rumbo utilizados en navegación. En este sistema, un rumbo de 0° corresponde al norte y un rumbo de $+90^\circ$ al este.

El origen del sistema NED generalmente se establece en un punto fijo del marco de referencia mundial, como la ubicación de despegue en nuestro caso[73].

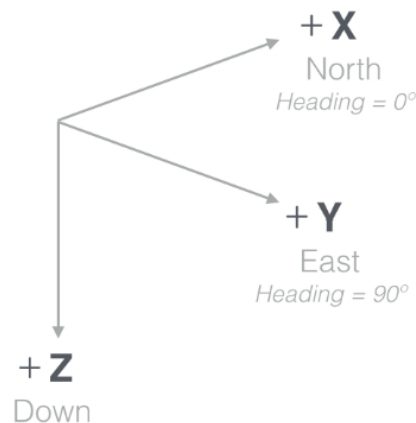


Figura B.2: Ejes en sistema de referencia NED [73].

Sistema de coordenadas geodésicas El sistema de coordenadas geodésicas representa una posición terrestre mediante tres componentes: latitud (ϕ), longitud (λ) y altitud (h). La latitud es el ángulo entre la normal al elipsoide de referencia (generalmente WGS-84) y el plano ecuatorial. La longitud corresponde al ángulo entre el meridiano de referencia (Greenwich) y el meridiano histórico del punto. La altitud es la distancia perpendicular entre el punto y el elipsoide como se observa en las Figuras B.3a y B.3b. Este sistema, usado por GPS, permite una localización precisa sobre un modelo matemático de la Tierra [74] [75] [76].

Apéndice B. Conceptos Útiles

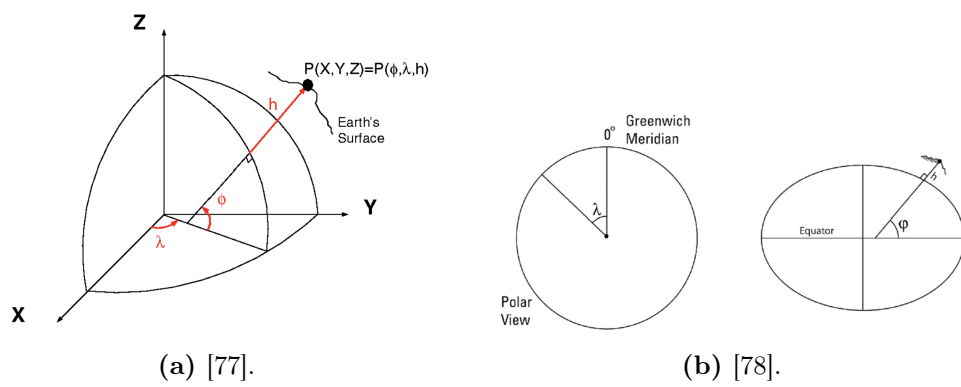


Figura B.3: Sistema de coordenadas geodésicas.

Apéndice C

Alternativas de comunicación NUC-Pixhawk

En esta sección se explican alternativas que se consideraron para la comunicación entre la NUC y el Pixhawk.

Para comunicar la controladora de vuelo con la computadora a bordo, se optó por usar la API MAVSDK-Python. Pero también existen otras herramientas para lograr la comunicación entre estas dos componentes, las cuales queremos comentar.

La primera de las herramientas que queremos comentar es el uso de la herramienta MAVROS, esto permite la comunicación entre nodos de ROS y nodos que se comunican mediante MAVlink. Esta herramienta resuelve el problema de la comunicación entre las componentes mencionadas, pero con la limitante de que está disponible únicamente para ROS. En este proyecto nos planteamos usar ROS2, que es la siguiente versión de este programa. Esta elección es debido a que la fecha de fin de vida útil de ROS iba a ser en el año de entrega de este documento (2025), y de ahí el interés de utilizar la siguiente versión del programa. Para más información acerca de ROS se puede consultar el siguiente enlace [79].

Debido al interés de utilizar ROS2 en el proyecto, se encontró que el fabricante de la controladora de vuelo con la que trabajamos PX4, ya ofrece una solución para comunicar nodos de ROS2 con tópicos uORB de PX4. En versiones del firmware como la 1.13 se utilizaba el middleware FastRTPS, y a partir de la versión 1.14 se empieza a utilizar el middleware uXRCE-DDS. En la Figura C.1 se muestra un esquema del funcionamiento del puente.

A modo de resumen, el funcionamiento de este puente consta de un cliente que se ejecuta en el PX4 y un agente que se ubica en la computadora a bordo, se comunican de forma bidireccional por medio de mensajería que puede ser UDP, TCP o personalizado. El agente funciona como un proxy para que el cliente se suscriba a tópicos del espacio DDS global. La razón por la cual se decidió no trabajar con esta opción es debido a que estamos trabajando con

Apéndice C. Alternativas de comunicación NUC-Pixhawk

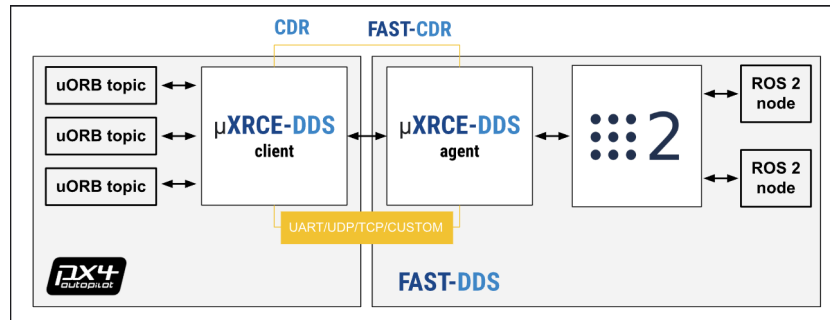


Figura C.1: Ilustración del puente uXRCE-DDS. Imagen de [80].

el hardware pixhawk 2.4.8, estamos condicionados a trabajar con la versión más estable de firmware para este dispositivo, la cual es la versión 1.10. Para dicha versión no se encuentra disponible el puente uXRCE-DDS. Si se desea trabajar con este puente en versiones más modernas de hardware, se pueden consultar los siguientes enlaces [80] [81].

Apéndice D

Modulos de Software en Drones, Ground Station y entorno de Simulación

En esta sección se explican de forma detallada los módulos de Software implementados en los drones, en la *ground station* y el entorno de simulación.

D.0.1. Drones

ROS2 Package

```
drone_movement/  
  package.xml  
  setup.py  
  setup.cfg  
  resource/  
    drone_movement  
drone_movement/  
  __init__.py  
  drone_main_node.py  
  drone_mavsdk_control.py  
  drone_pub_sub_node.py  
  drone_monitor_node.py  
  udp_ros2_bridge.py  
  routines.py  
  storage_sqlite.py  
  convert_gps_to_ned.py
```

drone_main_node.py

Módulo principal del sistema. Inicializa los subsistemas ROS 2 y MAVSDK, creando el nodo *DronePubSub* y estableciendo el enlace con el dron mediante

Apéndice D. Módulos de Software en Drones, Ground Station y entorno de Simulación

la API MAVSDK. Según los parámetros iniciales, configura el modo de operación (*task* o *net*) y coordina el ciclo ROS 2, gestionando el intercambio de telemetría y control con el Pixhawk.

`drone_mavsdk_control.py`

Este módulo centraliza la gestión completa del dron a través de MAVSDK, abarcando la conexión con el vehículo, la ejecución de rutinas de vuelo y la publicación continua de telemetría. Implementa funciones asincrónicas para armado, despegue y control en modo Offboard, permitiendo desplazamientos hacia posiciones GPS convertidas a coordenadas NED y el seguimiento de trayectorias predefinidas según el tipo de dron (*task* o *net*). Mantiene actualizados los tópicos ROS 2 con datos de posición, velocidad, batería y modo de vuelo, y gestiona la finalización segura de las misiones mediante la desactivación del modo Offboard y restauración de parámetros. Integra rutinas externas definidas en *routines.py* y utilidades de conversión de coordenadas (*gps_2_ned*), actuando como el núcleo funcional de interacción con el dron físico.

`drone_pub_sub_node.py`

Este módulo implementa el nodo *DronePubSub*, que actúa como interfaz entre el dron y el entorno ROS 2, publicando y suscribiéndose a tópicos de telemetría y control. Publica datos de posición, velocidad, modo de vuelo y batería, además de gestionar los canales de órdenes filtradas y ejecutadas. Se suscribe al tópico *move_to_position*, desde donde recibe las posiciones objetivo de todos los drones, filtra la correspondiente a su identificador y la reenvía al sistema de control. El acceso concurrente a la posición objetivo se protege mediante un mecanismo de exclusión mutua, asegurando la coherencia de los datos. En conjunto, el módulo funciona como un puente de mensajería entre ROS 2 y MAVSDK, garantizando la sincronización de la información de vuelo.

`udp_bridge_node.py`

Este módulo implementa el nodo *UDPBridgeNode*, que enlaza ROS 2 con la Estación Terrestre mediante mensajes UDP. Convierte los datos de posición publicados en *pose_topic* a un formato compacto y los envía periódicamente hacia la GS, asegurando interoperabilidad y bajo consumo de ancho de banda. A su vez, recibe órdenes de movimiento, las decodifica y publica en *move_to_position* como estructuras *ArrayDronePoseData*. Dispone de temporización configurable, manejo concurrente de recepción y validación de mensajes. En conjunto, actúa como un puente bidireccional entre la red de drones y la GS para el intercambio eficiente de telemetría y control.

drone_monitor_node.py

Este módulo implementa el nodo *Monitor*, encargado de registrar la telemetría, eventos de control y métricas de red en una base de datos SQLite local. Al iniciarse, crea las tablas necesarias y registra un nuevo vuelo asociado al dron y misión. Se suscribe a los tópicos *pose_topic*, *vel_topic*, *fly_topic*, *battery_topic*, *orders_received*, *order_filtered* y *order_executed*.

Se almacenan posición, velocidad, modo de vuelo, batería y órdenes de movimiento. Además, ejecuta periódicamente la recolección de métricas de red, que publica en *links_topic* en formato JSON. En conjunto, actúa como un sistema de registro autónomo que conserva datos de vuelo y red aun sin conexión con la Ground Station.

storage_sqlite.py

Este módulo implementa la capa de almacenamiento local de cada dron mediante una base de datos SQLite ligera. Registra de forma persistente los datos de vuelo, telemetría y órdenes, incluso sin conexión con la estación terrestre. Al iniciarse, crea el directorio *logs/* y el archivo *drone_monitor.db*, generando automáticamente las tablas descritas en la Tabla ???. Las funciones de inserción permiten almacenar posiciones, velocidades, modos de vuelo, estado de batería y ciclo completo de órdenes (recibidas, filtradas y ejecutadas). Este esquema de registro autónomo facilita la trazabilidad de misiones y el análisis posterior del desempeño del sistema distribuido.

routines.py

Este módulo implementa las rutinas de vuelo autónomo en modo offboard mediante MAVSDK. Permite ejecutar trayectorias predefinidas o dinámicas y combinar movimientos con pruebas de red (*iperf*). La función principal, *routine_task*, selecciona y ejecuta la rutina asignada según el tipo de tarea.

Incluye rutinas como *do_square* (patrón cuadrado), *do_line* (trayectoria lineal), *do_static* (posición fija), *do_go_and_back* (ida y regreso) y *routine_from_csv*, que interpreta archivos de *waypoints* CSV con soporte GPS y NED. También dispone de *move_direction_for_iperf* para evaluar la conectividad en vuelo.

Entre sus utilidades complementarias, *run_iperf_client* ejecuta pruebas de rendimiento de red, y *rc* permite lanzar comandos externos del sistema. Su ejecución asíncrona posibilita enviar *setpoints* continuos mientras se realizan otras tareas, manteniendo un diseño modular, extensible y orientado al análisis de rendimiento y estabilidad durante el vuelo.

Apéndice D. Módulos de Software en Drones, Ground Station y entorno de Simulación

convert_gps_to_ned.py

Este módulo realiza la conversión entre coordenadas geográficas (*latitud*, *longitud*) y el sistema de referencia local *NED* (North-East-Down), utilizado internamente por el dron para representar desplazamientos relativos. La función principal, *gps_2_ned()*, transforma una posición objetivo en coordenadas GPS a desplazamientos relativos respecto a una posición de referencia (*home*), obteniendo las componentes norte y este en metros.

Esta conversión permite traducir misiones definidas en coordenadas absolutas a comandos de movimiento relativos.

D.0.2. Ground Station

ROS2 Package

```
gs_movement/  
  package.xml  
  setup.py  
  setup.cfg  
  resource/  
    gs_movement  
  gs_movement/  
    Algoritmo/  
    __init__.py  
    gs_monitor.py  
    gs_ros_node.py  
    gs_storage_sqlite.py  
    gs_udp_bridge.py
```

gs_ros_node.py

Este módulo implementa el nodo principal de la *Ground Station* en ROS 2, responsable de coordinar el comportamiento colectivo de los drones en misión. Recibe las posiciones publicadas en *pose_topic*, y ejecuta periódicamente el algoritmo centralizado seleccionado. A partir de las posiciones objetivo calculadas, publica en *move_to_position* mensajes *ArrayDronePoseData* con las consignas de movimiento para los drones de red. Mantiene además estructuras para mapear identificadores y registrar posiciones de drones de tarea y red, junto con temporizadores que controlan la ejecución del algoritmo y el cierre del experimento. En síntesis, el nodo centraliza la información, aplica la lógica cooperativa de posicionamiento y distribuye las órdenes de vuelo de forma sincronizada.

gs_udp_bridge.py

Este módulo implementa el nodo de comunicación UDP de la estación base, que actúa como puente entre la red de drones y ROS 2. Recibe por UDP los mensajes de posición de los drones, los convierte a *DronePoseData* y los publica en *pose_topic*. A su vez, se suscribe a *move_to_position* para reenviar por UDP las consignas de movimiento generadas por el planificador. Utiliza un socket en modo *broadcast* para el envío y otro de recepción en un hilo separado, evitando bloqueos en el ciclo principal. También gestiona los tópicos *links_topic* y *positions_target* para monitoreo de enlaces y consignas transmitidas. En conjunto, *gs_udp_bridge_node.py* constituye la interfaz entre la comunicación inalámbrica UDP y la mensajería interna de ROS 2.

gs_monitor.py

Este módulo implementa el nodo de monitoreo de la estación base, encargado de registrar y visualizar en tiempo real el estado de la red y la posición de los drones. Integra una base de datos local, la mensajería ROS 2 y comunicación WebSocket con una interfaz externa. Inicializa el almacenamiento mediante *gs_storage_sqlite*, crea las tablas necesarias y asocia un *flight_number* único a cada sesión. Se suscribe a *pose_topic* para registrar posiciones, a *links_topic* para almacenar métricas de red (en formato JSON) y a *positions_target* para guardar las órdenes de movimiento. Mantiene un hilo WebSocket que transmite posiciones y métricas al servidor externo para visualización remota. En conjunto, *gs_monitor_node.py* centraliza el registro, almacenamiento y monitoreo en tiempo real de las misiones.

gs_storage_sqlite.py

Este módulo implementa la capa de persistencia de la estación terrestre mediante una base de datos local SQLite, destinada a registrar la información generada durante cada vuelo sin requerir un servidor externo. Al iniciarse, crea el directorio *logs/* y el archivo *ground_station.db*, definiendo las tablas principales: **vuelos** (identificador, nombre y tiempo de inicio), **posiciones** (coordenadas y marca temporal), **métricas** (datos de red y diagnóstico) y **órdenes** (consignas enviadas y número de secuencia). Durante la misión, las funciones *crear_vuelo()*, *insertar_posicion()*, *insertar_metrica()* e *insertar_orden()* registran en tiempo real la telemetría, las métricas y las órdenes transmitidas por el nodo de monitoreo. El uso de SQLite permite mantener un registro persistente y confiable de las misiones, adecuado para entornos desconectados o con recursos limitados.

D.1. Entorno de simulación

Se explica de forma detallada como fue la conexión de MAVSDK a diferentes instancias de SITL, lanzados desde una misma computadora.

D.1.1. Conexión de MAVSDK a múltiples instancias SITL

Cada instancia de *SITL* abre un canal de comunicación por UDP en el puerto `14540 + instance_id`, lo que permitió conectar cada vehículo desde *MAVSDK* utilizando la misma lógica aplicada para un único *drone*. El fragmento de código D.1.1 muestra la configuración dinámica de estas conexiones:

Conexión de la API MAVSDK al conectar con múltiples instancias de SITL

```
1 grpc_port = 50050 + instance_id
2 udp_port = 14540 + instance_id
3 drone = System(port=grpc_port)
4 await drone.connect(system_address="udp://:udp_port")
```

1. El valor de *grpc_port* debía estar libre, utilizándose el número de instancia por simplicidad.
2. Los *udp_port* se definieron para permitir la conexión con cada instancia (ver Figura 4.1).
3. En caso de no especificarse *port*, su valor por defecto es 50050.

Este esquema permitió ejecutar rutinas de vuelo diferenciadas para cada instancia de *SITL* utilizando una única computadora.

Apéndice E

Tablas de los experimentos realizados

En esta sección se detallan algunas de las tablas que se obtuvieron de los experimentos.

Punto	Enlace	Subida [Mbps]	Bajada [Mbps]
Punto1	3→19	2.16	5.96
Punto1	12→3	0.611	1.34
Punto1	19→12	0.991	0.135
Punto2	3→19	0.139	2.62
Punto2	12→3	2.40	0.286
Punto2	19→12	0.762	0.120
Punto3	3→19	0.0596	1.14
Punto3	12→3	0.252	0.370
Punto3	19→12	0.533	0.097

Tabla E.1: Velocidades de subida y bajada por punto y enlace sin drones de red

Apéndice E. Tablas de los experimentos realizados

Punto	Enlace	Subida [Mbps]	Bajada [Mbps]
Punto1	3→19	0.464	2.28
Punto1	12→3	0.942	0.848
Punto1	19→12	1.07	0.145
Punto2	3→19	0.730	0.265
Punto2	12→3	1.44	0.285
Punto2	19→12	2.27	1.32
Punto3	3→19	0.075	4.04
Punto3	12→3	2.96	0.106
Punto3	19→12	0.552	0.523
Punto4	3→19	0.735	0.922
Punto4	12→3	0.0325	0.0628
Punto4	19→12	0.196	0.0516

Tabla E.2: Velocidades de subida y bajada por punto y enlace con un dron de red

Punto	Enlace	Subida [Mbps]	Bajada [Mbps]
Punto1	3→19	2.59	0.696
Punto1	12→3	1.92	2.52
Punto1	19→12	0.143	0.763
Punto2	3→19	4.08	0.265
Punto2	12→3	0.226	0.072
Punto2	19→12	0.045	0.0561
Punto3	3→19	0.161	0.270
Punto3	12→3	0.520	0.133
Punto3	19→12	0.187	0.940
Punto4	3→19	0.257	0.620
Punto4	12→3	0.254	0.153
Punto4	19→12	0.497	0.555

Tabla E.3: Velocidades de subida y bajada por punto y enlace con 2 drones de red

Apéndice F

Preparación Inicial del Entorno Real

Con el objetivo de garantizar la robustez, repetibilidad y fiabilidad de los procedimientos experimentales, se diseñó y desarrolló un conjunto de utilidades de *software* y protocolos de operación. Estos automatismos, implementados mediante *scripts* personalizados, tienen la función crítica de minimizar el error humano, verificar la integridad del sistema y asegurar el correcto almacenamiento de los datos distribuidos. A continuación, se detallan las metodologías y herramientas desarrolladas para la preparación del entorno de vuelo.

F.1. Gestión y Atenuación de la Potencia de Transmisión

Las pruebas preliminares evidenciaron que el alcance nominal de la red Wi-Fi excedía significativamente los rangos permitidos por la autonomía de las baterías de los drones y el espacio físico disponible, dificultando las pruebas en distancias cortas que buscan escenarios de pérdida de conexión. Por esta razón, se estableció una estrategia híbrida de atenuación de potencia, combinando limitaciones por *software* con modificaciones físicas en el *hardware*.

Inicialmente, se desarrolló un procedimiento para disminuir la potencia de transmisión (*TX power*) de la tarjeta de red mediante comandos del sistema operativo Linux. La configuración se realiza a través de la herramienta *iw*, tal como se muestra en el siguiente bloque:

```
1 ip link set wlan0 up # Activacion de la interfaz si no esta operativa
2 iw dev wlan0 set txpower fixed 100 # Configuracin a 100 mBm (1 dBm)
```

Listing F.1: Configuración de potencia por software

Es importante notar que el parámetro **fixed** establece la potencia en 1 % de decibelios-milivatios (dBm/100). Por tanto, un valor de 100 corresponde a 1 dBm. Del mismo modo, se utiliza el valor 2200 para restablecer la potencia a su nivel nominal de 22 dBm. Para validar esta configuración y caracterizar el comportamiento de la red, se diseñó y ejecutó un experimento de medición

Apéndice F. Preparación Inicial del Entorno Real

de alcance (RSSI y pérdida de paquetes). El diseño experimental constó de 9 escenarios de prueba, combinando 3 niveles de potencia (22, 10 y 1 dBm) con 3 alturas de vuelo (1, 2 y 3,6 m). Estas mediciones se realizaron para observar cómo varía la señal de la red a diferentes alturas y establecer una altura mínima de vuelo. Dado que a 3,6 m la señal se comporta de forma más estable, se decidió volar los drones por encima de este valor.

El procedimiento comenzó determinando el umbral de desconexión a potencia mínima (1 dBm). Mediante el envío continuo de paquetes ICMP (ping) desde la *Ground Station*, se determinó que la pérdida de conectividad ocurría aproximadamente a los 150 m. Con base en este hallazgo, se definió una discretización espacial de 10 puntos de medida separados por 15 m entre sí. La Figura F.1 presenta los resultados obtenidos tras realizar *iperf* a 3,6 m de altura, cota que se aproxima a la altitud operativa de los experimentos detallados en las secciones de diseño y análisis de resultados 8.1.

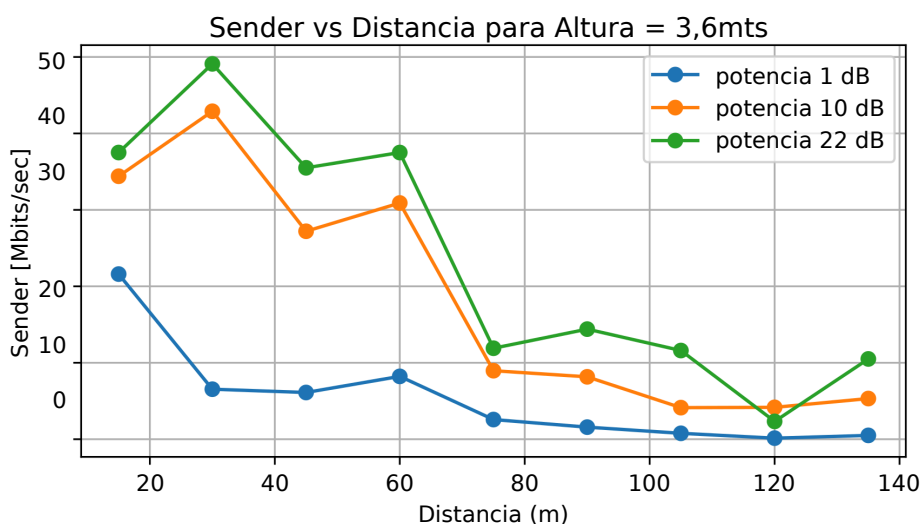


Figura F.1: Resultados de la prueba de potencia a 3,6 m de altura

Tras analizar los resultados, se concluyó que la atenuación por *software* era insuficiente para los objetivos del proyecto, ya que los drones mantenían conexión estable hasta los 60 – 80 m, como se puede observar en el promedio mostrado, incluso a potencia mínima. Para lograr escenarios de desconexión controlada dentro de un área de vuelo reducida, fue necesario implementar una intervención física en el *hardware*. Se cubrieron las antenas Wi-Fi utilizando un material conductor (papel aluminio) para generar una atenuación pasiva adicional. Esta modificación permitió reducir el alcance efectivo a aproximadamente 20 m con una potencia de 5 dBm, tal como se aprecia en la Figura F.2, cumpliendo con los requisitos experimentales.

Adicionalmente, se programó una lógica de control dinámico dentro de

F.2. Protocolo de Calibración y Verificación del Control

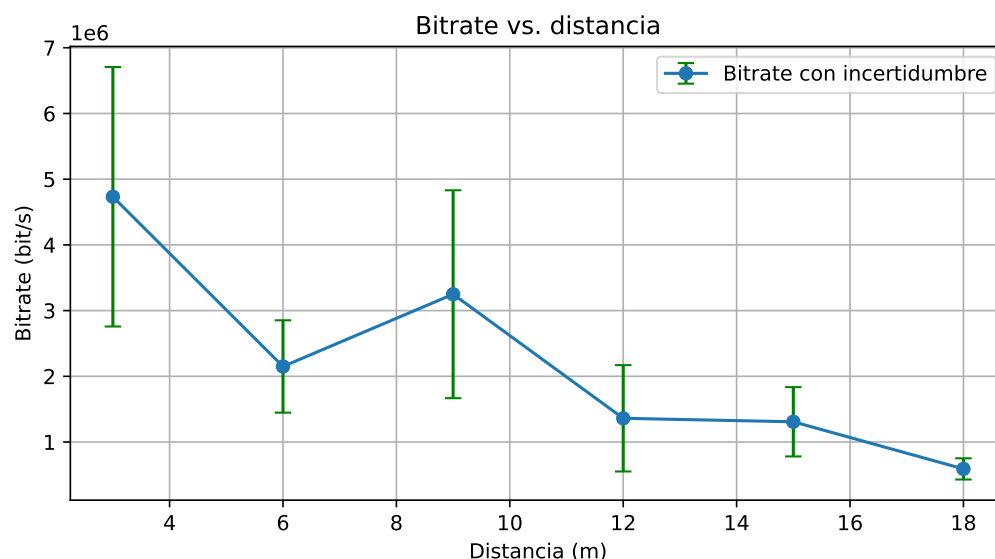


Figura F.2: Resultados del throughput con potencia a 5dBm y con las antenas apantalladas, en función de la distancia

las rutinas de ROS 2. Por seguridad, el sistema inicia siempre con potencia máxima (22 dBm). Los *scripts* desarrollados monitorean el estado del vuelo y reducen automáticamente la potencia a 5 dBm únicamente cuando el dron entra en modo *Offboard* y se restablece cuando el experimento termina para garantizar la fiabilidad del enlace de mando con la *Ground Station*. Esto se realizó así para asegurarnos que cada dron recibiera la orden de despegue, ya que partiendo desde el piso, es donde la señal era mas debil. Esto se diseñó de este modo para asegurar la recepción confiable de la orden de despegue por parte de cada dron, ya que la fase inicial en tierra corresponde a la condición más desfavorable para el enlace inalámbrico en términos de atenuación y obstrucciones.

F.2. Protocolo de Calibración y Verificación del Control

Para asegurar la integridad física de los equipos y la validez de los datos posicionales, se estableció un protocolo estricto de calibración al inicio de cada jornada experimental. Todo vehículo aéreo involucrado es sometido a una calibración de sensores mediante el *software QGroundControl*.

Como medida de seguridad crítica, previo a la ejecución de las misiones autónomas, se realiza un vuelo de prueba manual. Este procedimiento permite verificar la estabilidad mecánica y respuesta de los controladores PID del dron, y la correcta vinculación y calibración del radiocontrol.

Si se detecta cualquier anomalía o deriva (*drift*) durante esta fase, se aborta

Apéndice F. Preparación Inicial del Entorno Real

la operación y se repite el proceso de calibración. La precisión del control manual es fundamental, ya que constituye el último recurso de seguridad para que el operador recupere el control ante un eventual fallo del sistema autónomo.

F.3. Sincronización Temporal Distribuida (NTP)

Dado que la arquitectura del sistema implica la operación simultánea de múltiples agentes y una estación terrestre, la coherencia temporal de los datos registrados es vital para su posterior fusión y análisis. Cualquier desfase temporal (*clock drift*) entre los nodos invalidaría las métricas de rendimiento de la red y la correlación de eventos distribuidos.

Para mitigar este problema, se configuró el protocolo NTP (*Network Time Protocol*) utilizando **Chrony**, una implementación optimizada para redes con variabilidad en la latencia y conexiones intermitentes, condiciones típicas en redes de drones (MANETs).

Con el fin de automatizar este proceso crítico y evitar errores de configuración manual en campo, se desarrollaron *scripts* específicos que gestionan el despliegue de la arquitectura cliente-servidor. Estas herramientas, disponibles en el repositorio del proyecto, permiten levantar un servidor NTP local en la *Ground Station* y forzar la sincronización inmediata de todos los nodos aéreos al inicio de la jornada. Esta configuración garantiza que todos los registros (*logs*) del sistema compartan una referencia temporal unificada con precisión de milisegundos, requisito indispensable para la validación cruzada de los datos experimentales.

F.4. Inicialización y Validación de Sensores

El proceso de puesta en marcha del sistema de cómputo a bordo requiere verificar la correcta interfaz entre el computador de alto nivel (NUC) y la controlador de vuelo (Pixhawk). Realizar esta verificación manualmente mediante inspección de terminales es propenso a errores y consume tiempo valioso de vuelo. Por esta razón, se desarrolló una suite de herramientas de validación automática, escritas en Python y Bash, que se incluye en el repositorio del proyecto. Estos *scripts* realizan un diagnóstico secuencial que abarcan tanto la comprobación de disponibilidad del puerto serial y el flujo de telemetría (mensajes MAVLink) entre la NUC y la controlador de vuelo, como el monitoreo del estado energético, alertando sobre niveles de tensión críticos antes del despegue, y la verificación de la convergencia de los estimadores de posición (GPS) y la lectura correcta de los sensores de altura (LIDAR/Barómetro).

F.5. Registros de Datos Obtenidos

Finalmente, estas rutinas automatizan el reinicio del *firmware* PX4 una vez posicionado el dron, paso fundamental para establecer el origen del sistema de coordenadas local (NED) relativo al punto de despegue (*Home Position*), asegurando que las trayectorias en modo *Offboard* se ejecuten con la referencia espacial correcta (se detalla en el Apéndice D.0.1 el módulo que implementa esta función y quien la utiliza).

F.5. Registros de Datos Obtenidos

En esta sección se describe el esquema de registro y aprovechamiento de los datos generados. En primer lugar, se presentan los registros nativos de la controladora Pixhawk, que almacenan información detallada de cada vuelo y como utilizar la misma para visualizar y depurar el comportamiento de un dron tras un vuelo. A continuación, se introduce la base de datos implementada en SQLite3, que integra métricas de red, variables de vuelo y comandos intercambiados entre los nodos. Finalmente, se describe la realización de una herramienta de reproducción de vuelos que utiliza esta base de datos central para visualizar las trayectorias y enlaces del enjambre y facilitar el análisis posterior de las misiones.

F.5.1. Registros del Pixhawk

La controladora de vuelo incorpora una tarjeta SD en la que se registran datos cada vez que se realiza un vuelo. El volumen de información almacenada y cuando comienza a almacenarse dependen de la configuración del *firmware* de la Pixhawk 2.4.8. En particular, el parámetro `SDLOG_PROFILE` define qué conjuntos de datos se incluyen en el archivo de registro de vuelo, mientras que `SDLOG_MODE` controla el modo de inicio del registro y resulta clave para el diagnóstico, ya que determina el momento exacto a partir del cual la aeronave comienza a guardar información (al armar, desde el encendido, etc.). Ambos parámetros pueden ajustarse desde *QGroundControl*, de forma análoga a los parámetros descritos en el Apéndice B. Esta capacidad de registro es fundamental para el análisis posterior de los vuelos, ya que permite detectar anomalías e identificar errores o comportamientos inesperados durante las misiones.

Para realizar este análisis se utiliza la herramienta en línea Flight Review de PX4, que genera un conjunto de gráficas, una vista satelital de la trayectoria seguida por el dron y una reproducción en tres dimensiones de la misión, entre otros recursos. Además, permite descargar el conjunto de parámetros configurados en esa aeronave, lo que resulta especialmente útil cuando se desea comparar el comportamiento de un dron que presenta fallos con otro que opera correctamente.

Apéndice F. Preparación Inicial del Entorno Real

En el repositorio del proyecto se incluye un *script* que compara los parámetros entre dos drones y verifica que los parámetros clave estén correctamente configurados para garantizar un vuelo estable y un registro de datos adecuado. Asimismo, se proporcionan funciones para procesar los datos almacenados en la tarjeta SD, generar gráficas y reproducir vuelos en enjambre, lo que facilita el análisis del comportamiento conjunto de los vehículos, dado que la página *Flight Review* de PX4 permite analizar solo un dron por vez.

F.5.2. Registros de Datos en SQL

Los datos almacenados por el PX4 brindan información muy variada y útil, además de ser una herramienta con la que contamos por defecto. Sin embargo, no guarda datos específicos sobre la red ni sobre la arquitectura desarrollada en ROS 2. Para llevar a cabo un proceso de identificación de errores en *software* o *hardware* más controlado, fue necesario crear una estructura que combinara toda esta información y la relacionara entre sí, de modo que sean fácilmente consultable. Por ejemplo, los mensajes de ROS 2 intercambiados en un determinado experimento, qué rutas seguían los mensajes enviados y en qué posiciones se encontraba cada nodo. Para ello se creó una base de datos implementada en SQLite3. Se eligió esta base por su ligereza y compatibilidad con comandos Python, lenguaje usado en la arquitectura de ROS 2 implementada, por lo que se pueden cargar datos nuevos fácilmente desde un nodo.

Cada dron mantiene un registro local de información relevante sobre su operación y su entorno de red. Los datos se almacenan de manera estructurada en dos archivos independientes, sin establecer relaciones cruzadas entre tablas de distintas bases, con el objetivo de simplificar el acceso y minimizar el procesamiento requerido en vuelo. El nodo Monitor (nodo de ROS 2) es el encargado de gestionar el proceso de registro de datos localmente, almacenando por un lado las métricas de la red obtenidas a través de batman-adv, como la tabla de nodos en la red detectados, las rutas activas, la calidad de cada camino posible y el último momento en que cada nodo fue visto. Por otro lado, también almacena un conjunto reducido de datos brindados por el Pixhawk, tales como la posición, la velocidad, el nivel de la batería y el modo de vuelo del dron. Además, se registran las ordenes de movimiento con posiciones objetivo recibidas desde la *Ground Station*, la interpretación y ejecución de las mismas realizadas por el dron cuando oficia de agente de red.

Una vez completados los vuelos, se realiza un proceso de consolidación de datos. En esta etapa, todos los archivos generados por los drones son transferidos a la *Ground Station*, que se encarga de verificar que no estén corruptos, depurar posibles inconsistencias y realizar la fusión de la información en una base de datos central, unificada y estructurada. En la Figura F.3 se muestra una representación del esquema resultante, donde puede apreciarse la

organización de las tablas, los campos almacenados en cada una y las relaciones definidas mediante claves foráneas, que permiten vincular eficientemente los datos pertenecientes a un mismo experimento o nodo.

Si bien, habían otras alternativas a una base de datos, dando cómo ejemplo la herramienta ofrecida por ROS2 llamada `rosvbag2` [82], la cuál cumple correctamente la función de captura de datos, estando orientado principalmente a la grabación y reproducción de mensajes. Ésta no resulta óptima para tareas de análisis y para el acceso selectivo a la información. El uso de SQLite3 permite realizar consultas estructuradas, filtrado y procesamiento eficiente de los datos, facilitando el postprocesamiento y la integración con futuras aplicaciones.

F.5.3. Reproducción de Vuelos

Esta herramienta fue desarrollada con el objetivo de visualizar y analizar los experimentos realizados, permitiendo observar la evolución de las posiciones y enlaces entre los drones en una representación bidimensional (vista superior). Su propósito principal es facilitar el post-análisis de las misiones, evaluando si el comportamiento observado durante el experimento coincide con el esperado o si existieron desvíos en la ejecución.

La reproducción se realiza a través de la misma interfaz web utilizada para el monitoreo en tiempo real, pero en este caso los datos provienen de la base de datos central, donde mediante consultas SQL, se devuelven las trayectorias y enlaces registrados durante el vuelo correspondiente.

La lógica del *front-end* (implementada en *JavaScript*) es prácticamente la misma que en el modo en vivo. La diferencia principal radica en la forma en que se obtienen los datos: al seleccionar el archivo a reproducir, el back-end consulta la base de datos y envía al cliente todas las coordenadas y enlaces preordenados temporalmente, lo que puede generar un breve tiempo de carga inicial.

Una vez cargados los datos, al presionar el botón “*Play*”, el sistema inicia la reproducción simulando el comportamiento en tiempo real: los datos son enviados secuencialmente según su marca temporal (*timestamp*) al gestor correspondiente (*handler*), que se encarga de actualizar la posición y el estado de los enlaces en pantalla. De esta manera, la herramienta de reproducción emula fielmente la lógica de actualización del monitoreo en vivo, pero sobre datos históricos, lo que permite realizar análisis visuales detallados y comparar distintas ejecuciones de una misma misión.

Apéndice F. Preparación Inicial del Entorno Real

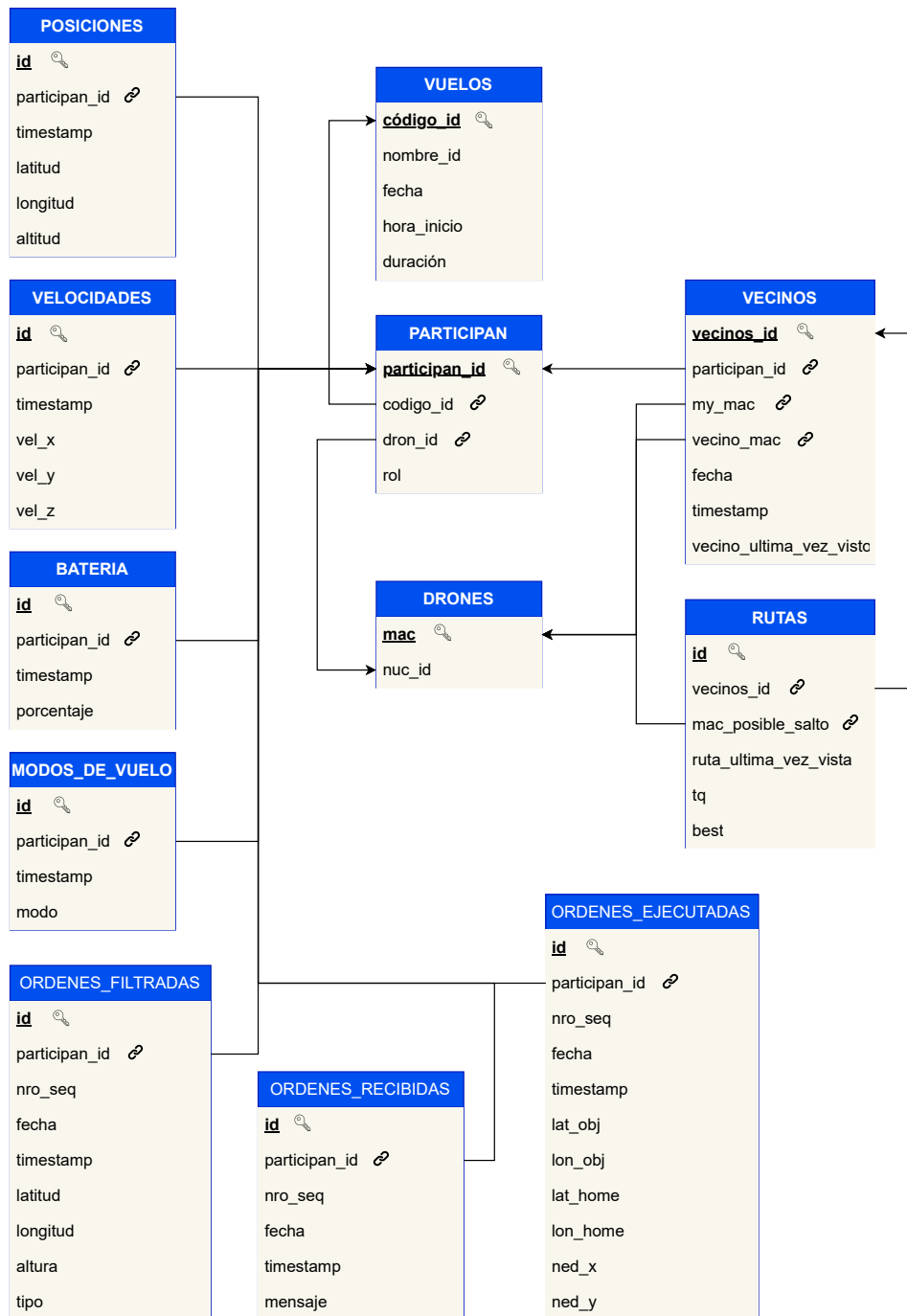


Figura F.3: Modelo relacional de la base de datos, donde se muestran las tablas almacenadas, los datos que contienen y las relaciones entre ellas, las cuales permiten vincular la información durante las consultas.

Índice de tablas

2.1. Componentes principales de la estructura mecánica.	11
2.2. Componentes del sistema de control, detección y cómputo. . .	15
2.3. Componentes del sistema de propulsión y alimentación eléctrica.	17
8.1. Rutas activas registradas en el instante de mayor separación entre los drones 12 y 19	88
E.1. Velocidades de subida y bajada por punto y enlace sin drones de red	127
E.2. Velocidades de subida y bajada por punto y enlace con un dron de red	128
E.3. Velocidades de subida y bajada por punto y enlace con 2 drones de red	128

Esta página ha sido intencionalmente dejada en blanco.

Índice de figuras

2.1.	En orden de derecha a izquierda se observa el chasis inferior y superior junto con la placa de montaje de la OBC, los brazos, el tren de aterrizaje, la carcasa protectora del OBC junto con el soporte de un regulador de voltaje, y finalmente, el soporte de la batería junto con un velcro.	10
2.2.	Comenzando de arriba a la izquierda, los componentes que conforman el control y sensado del Dron son: La controladora de vuelo Pixhawk, la computadora a bordo con sus antenas, el gps, el Lidar para medir distancias, el radio control con su receptor, y el regulador de voltaje Pololu.	12
2.3.	Pixhawk 2.4.8 por dentro y por fuera.	12
2.4.	Componentes que conforman la alimentación y propulsión del Dron.	15
2.5.	Drones ensamblados listos para volar.	17
2.6.	Guía visual de la totalidad de las piezas del dron.	18
3.1.	Mensajes encapsulados con la trama de batman-adv.	27
4.1.	Arquitectura general utilizada para el desarrollo y validación de vuelos autónomos mediante PX4 SITL, Gazebo y MAVSDK [35].	34
4.2.	Diagrama de flujo de código ejemplo.	36
4.3.	Mensajes por consola del código ejemplo que permiten dar seguimiento al flujo del programa.	37
4.4.	Datos del GPS visualizadas en <i>QGroundControl</i> al finalizar el cuadrado de lado 10m.	38
4.5.	Datos del GPS graficados al finalizar el cuadrado de lado 10m y altura 8m. Latitudes y longitudes se expresan en metros relativos al primer dato como punto de referencia	39
4.6.	Datos GPS del dron durante la ejecución de la trayectoria cuadrada en entorno real. Latitudes y longitudes se expresan en metros relativos al primer dato como punto de referencia.	40
4.7.	Dron en vuelo de campo	41

Índice de figuras

5.1.	Secuencia de servicios lanzados en el arranque de cada NUC .	45
5.2.	Arquitectura de ROS 2 representada bajo un modelo en capas, mostrando la organización jerárquica de sus componentes y la interacción entre los distintos niveles del sistema [41].	46
5.3.	Esquema de alto nivel del funcionamiento de la plataforma. . .	49
5.4.	Módulos internos y sus principales interfaces. En color celeste se representan módulos que implementan nodos de ROS y en color amarillo módulos con funciones auxiliares.	51
5.5.	Módulos internos y sus principales interfaces en la Ground Station.	53
5.6.	Página de Monitoreo de la Red.	56
6.1.	Diagrama elección de clúster mediante cercanía y solapamiento.	61
6.2.	Diagrama Posición de Agentes de Red.	62
6.3.	Diagrama Flujo principal del algoritmo.	63
7.1.	Ventanas necesarias para simular tres SITL desde una sola computadora.	70
7.2.	Visualización de las formas posibles de lanzar SITL.	72
7.3.	Esquema de los servicios y redes para la implementación de un experimento usando el algoritmo C-DFLP.	74
7.4.	Esquema de los servicios y redes definidos para la implementación de un experimento usando el algoritmo <i>GIOD Centralized</i> . . .	75
7.5.	Visualización de la trayectoria con <i>QGroundControl</i> en simulación.	76
8.1.	Comparativa entre la configuración (para las mismas posiciones de los drones de tarea) de los drones de red propuesta por <i>GIOD_Centralized</i> y la propuesta de su competidor.	79
8.2.	Gráfico de barras comparativo del throughput agregado en pruebas estáticas, evaluando el impacto de la presencia de nodos de red y de la ejecución de ROS 2.	81
8.3.	Prueba de validación completa con drones de tarea y de red volando	82
8.4.	Drones volando en Gazebo con experimento realizado.	83
8.5.	Trayectorias realizadas en el experimento por los drones de tarea utilizando el algoritmo de posicionamiento C-DFPL. . .	84
8.6.	Trayectorias realizadas en el experimento por los drones de red utilizando el algoritmo de posicionamiento C-DFPL.	84
8.7.	Despegue del enjambre en prueba validación completa en campo.	85

8.8. Trayectorias reportadas por los datos de GPS de todos los drones en la prueba de validación completa referidas a la latitud y longitud inicial del dron 3 que oficia de origen del gráfico. Luego se convierten esas diferencias a metros y se expresan las longitudes en función de las latitudes para visualizar la trayectoria de cada dron. En el gráfico a la izquierda se observan los resultados de las pruebas de campo y a la derecha los de la simulación.	86
8.9. Evolución del TQ promedio entre los drones de tarea durante el vuelo de validación.	87
8.10. Calidad del enlace en la comunicación del dron 12 al 19 durante el experimento, mostrando a quien le mandaba el mensaje para llegar al destino.	88
8.11. Evolución de la configuración topológica del entorno: sin drones de red y con dos drones de red.	89
8.12. Trayectorias reportadas por los datos de GPS de todos los drones en la prueba de validación dinámica sin drones de red, referidas a la latitud y longitud inicial del dron 3 que oficia de origen del gráfico. Luego se convierten esas diferencias a metros y se expresan las longitudes en función de las latitudes para visualizar la trayectoria de cada dron. También se observan en diferentes colores (en cada punto de medición) los anillos formados por los drones de tarea con las tasas de transmisión en Mbps en ambos sentidos para cada par de drones reportados por iperf3.	91
8.13. Topología de la red, representando los enlaces directos entre cada nodo, en el experimento realizado sin agentes de red a la máxima distancia alcanzable.	92
8.14. Prueba de enjambre con un dron de red dinámico y un dron de tarea móvil de forma manual.	92
8.15. Prueba de enjambre con dos drones de red dinámicos y un dron de tarea móvil de forma manual.	93
8.16. Calidad de cada enlace en función de la distancia del dron 3, sin drones de red.	93

Índice de figuras

8.17. Trayectorias reportadas por los datos de GPS de todos los drones en la prueba de validación dinámica con un dron de red, referidas a la latitud y longitud inicial del dron 3 que oficia de origen del gráfico. Luego se convierten esas diferencias a metros y se expresan las longitudes en función de las latitudes para visualizar la trayectoria de cada dron. También se observan en diferentes colores (en cada punto de medición) los anillos formados por los drones de tarea con las tasas de transmisión en Mbps en ambos sentidos para cada par de drones reportados por iperf3.	95
8.18. Resultados de la prueba de validación dinámica con un dron de red en gráficos de latitud y longitud relativas al punto inicial del dron 3 expresadas en metros en función del tiempo.	96
8.19. Trayectorias reportadas por los datos de GPS de todos los drones en la prueba de validación dinámica con dos drones de red, referidas a la latitud y longitud inicial del dron 3 que oficia de origen del gráfico. Luego se convierten esas diferencias a metros y se expresan las longitudes en función de las latitudes para visualizar la trayectoria de cada dron. También se observan en diferentes colores (en cada punto de medición) los anillos formados por los drones de tarea con las tasas de transmisión en Mbps en ambos sentidos para cada par de drones reportados por iperf3.	97
8.20. Resultados de la prueba de validación dinámica con dos dron de red en gráficos de latitud y longitud relativas al punto inicial del dron 3 expresadas en metros en función del tiempo.	98
8.21. Representación de los enlaces directos entre cada nodo en el experimento realizado con tres drones de tarea y un dron de red, cada uno en su posición global.	98
8.22. Total del throughput de cada enlace tomado con la herramienta iperf3 en conjunto con el promedio de la estimación de la calidad del enlace brindada por B.A.T.M.A.N., en función de la distancia para las distintas configuraciones de la red.	99
8.23. Resultados de la trayectoria de la prueba de validación de nuevos algoritmos con un dron de red en gráficos de latitud en función de la longitud.	101
A.1. Detalles de los microcontroladores y sensores integrados en la placa de la Pixhawk 2.4.8.	108
B.1. Sistema de referencia corporal [73]	116
B.2. Ejes en sistema de referencia NED [73].	117
B.3. Sistema de coordenadas geodésicas.	118

C.1. Ilustración del puente uXRCE-DDS. Imagen de [80].	120
F.1. Resultados de la prueba de potencia a 3,6 m de altura	130
F.2. Resultados del throughput con potencia a 5dBm y con las antenas apantalladas, en función de la distancia	131
F.3. Modelo relacional de la base de datos, donde se muestran las tablas almacenadas, los datos que contienen y las relaciones entre ellas, las cuales permiten vincular la información durante las consultas.	136

Esta página ha sido intencionalmente dejada en blanco.

Bibliografía

- [1] Wikipedia contributors, *Vehículo aéreo no tripulado*, Accedido: 26-nov-2025, 2025. dirección: https://es.wikipedia.org/wiki/Veh%C3%ADculo_a%C3%A9reo_no_tripulado#cite_note-1.
- [2] APD Drones, *¿Qué es un Drone y para qué sirve?*, Accedido: 26-nov-2025, 2025. dirección: <https://idc.apddrones.com/educacion/que-es-un-drone-y-para-que-sirve/>.
- [3] Built In, *What is a dron?*, Accedido: 26-nov-2025, 2025. dirección: https://builtin.com/drones?_x_tr_hist=true.
- [4] Ferrovial, *Drones: qué son, tipos de drones y para qué sirven*, Accedido: 26-nov-2025, 2025. dirección: <https://www.ferrovial.com/es/innovacion/tecnologias/drones/>.
- [5] Lumasky, *que es un enjambre de drones*, Accedido: 26-nov-2025, 2025. dirección: <https://lumasky.show/es/journal/que-es-un-enjambre-de-drones/>.
- [6] APD Drones, *enjambre de enjambre*, Accedido: 26-nov-2025, 2025. dirección: <https://idc.apddrones.com/noticias/enjambre-de-drones/>.
- [7] Wikipedia contributors, *Inteligencia de enjambre*, Accedido: 26-nov-2025, 2025. dirección: https://es.wikipedia.org/wiki/Inteligencia_de_enjambre.
- [8] Built In, *drone swarm*, Accedido: 26-nov-2025, 2025. dirección: <https://builtin.com/articles/drone-swarm>.
- [9] S. Eckelmann, *B.A.T.M.A.N. Advanced Documentation Overview*, Accedido: 28-nov-2025, 2025. dirección: <https://www.open-mesh.org/projects/batman-adv/wiki>.
- [10] MAVSDK, *MAVSDK*, Accedido: 12-feb-2025, 2025. dirección: <https://mavsdk.mavlink.io/main/en/index.html>.
- [11] R. García, *Proyecto Tero*, Accedido: 26-nov-2025, 2025. dirección: <https://iie.fing.edu.uy/proyectos/tero/>.

Bibliografía

- [12] G. Cabrera Núñez, A. Castro Ibarburu y S. Salmini Rodríguez, “ENDRA: Enjambre de Drones Autónomos”, 2021.
- [13] R. Ercoli, F. Navadian, J. Urrisa, P. Monzón y F. Benavides, “Exploración robótica colaborativa de interiores.”, en *Congreso Argentino de Sistemas Embebidos–CASE 2024, San Miguel de Tucumán, Provincia de Tucumán, Argentina, 8-9 ago. 2024*, pp. 1-4., CASE, 2024.
- [14] M. Calvo-Fullana, M. Gerasimenko, D. Mox et al., “A Networked Multi-Agent System for Mobile Wireless Infrastructure on Demand”, *IEEE Transactions on Robotics*, 2024.
- [15] M. del Castillo, A. Ribeiro y F. Larroca, “EGNN-based Topology Control in Wireless Mobile Infrastructure on Demand with Shared Access Restrictions”, en *Proceedings of the 3rd GNNet Workshop on Graph Neural Networking Workshop*, 2024, págs. 46-52.
- [16] *FlameWhee 450*FlameWhee 450, Accedido: 26-nov-2025, 2025. dirección: https://dl.djicdn.com/downloads/flamewheel/en/F450_User_Manual_v2.2_en.pdf.
- [17] M. McNabb, *Tethered and Unstoppable: How Fiber-Optic Drones Are Rewriting the Rules of Battlefield Control*, Accedido: 25-nov-2025, 2025. dirección: <https://dronelife.com/2025/07/21/tethered-and-unstoppable-how-fiber-optic-drones-are-rewriting-the-rules-of-battlefield-control/>.
- [18] A. P. Mack, *Cómo son los nuevos drones de fibra óptica que evaden los inhibidores en el frente de la guerra en Ucrania*, Accedido: 25-nov-2025, 2025. dirección: <https://www.latercera.com/tendencias/noticia/como-son-los-nuevos-drones-de-fibra-optica-que-evaden-los-inhibidores-en-el-frente-de-la-guerra-en-ucrania/>.
- [19] Fortinet., *¿Qué es una red inalámbrica? Tipos de redes inalámbricas*, Accedido: 25-nov-2025, 2025. dirección: <https://www.fortinet.com/lat/resources/cyberglossary/wireless-network>.
- [20] Wikipedia contributors, *Red inalámbrica*, Accedido: 25-nov-2025, 2025. dirección: https://es.wikipedia.org/wiki/Red_inal%C3%A1mbrica.
- [21] TechTarget Contributor, *The 4 Different Types of Wireless Networks*, Accedido: 25-nov-2025, 2025. dirección: <https://www.techtarget.com/searchnetworking/tip/The-4-different-types-of-wireless-networks>.
- [22] Taoglas, *Understanding Infrastructure Mode in Wireless Networking Technology*, Accedido: 25-nov-2025, 2025. dirección: <https://www.taoglas.com/blogs/understanding-infrastructure-mode-in-wireless-networking-technology/>.

- [23] Hytera, *¿Qué es una red ad-hoc y por qué es importante?*, Accedido: 25-nov-2025, 2025. dirección: <https://www.hytera.com/la/connect/blog/que-es-una-red-ad-hoc>.
- [24] 7SIGNAL, Inc., *What Is an Ad Hoc Network?*, Accedido: 25-nov-2025, 2025. dirección: <https://www.7signal.com/ad-hoc-network>.
- [25] TechTarget Contributor, *Wireless Ad Hoc Network (WANET)*, Accedido: 25-nov-2025, 2025. dirección: <https://www.techtarget.com/searchmobilecomputing/definition/ad-hoc-network>.
- [26] S. Dunlavey, *Ad-Hoc Networks Explained: Understanding Their Nature and Impact*, Accedido: 25-nov-2025, 2025. dirección: <https://agileblue.com/resource/ad-hoc-networks-explained-understanding-their-nature-and-impact-2/>.
- [27] C. Taylor, *¿Cómo elegir el mejor protocolo de enrutamiento para su red ad hoc?*, Accedido: 25-nov-2025, 2025. dirección: <https://www.linkedin.com/advice/0/how-do-you-choose-best-routing-protocol-your-ad-hoc?lang=es&lang=es&originalSubdomain=es>.
- [28] Relay, Frame and V-Series, ITUT, “OSI model”, *Transport*, vol. 10, n.º 225, pág. 1,
- [29] ArduPilot Development Team, *Mission Planner – Ground Control Station for ArduPilot*, Accedido: 11-jun-2025, 2025. dirección: <https://ardupilot.org/planner/>.
- [30] QGroundControl Development Team, *QGroundControl – Drone Control Ground Control Station*, Accedido: 11-jun-2025, 2025. dirección: <https://qgroundcontrol.com/>.
- [31] mavlink, *MAVlink*, Accedido: 26-feb-2025, 2025. dirección: <https://mavlink.io/en/>.
- [32] Dronecode Project, Inc., *QGroundControl*, Accedido: 26-nov-2025, 2025. dirección: <https://qgroundcontrol.com/>.
- [33] R. Alvarez-Torrico, *Introducción a Ardupilot y PX4 - Parte 1: Soluciones de Autopiloto Para Drones*, Accedido: 11-jun-2025, 2025. dirección: <https://tecbolivia.com/index.php/articulos-y-tutoriales-microcontroladores/101-introduccion-a-ardupilot-y-px4-parte-1>.
- [34] P. S. Foundation, *asycio-Python*, Accedido: 27-feb-2025, 2025. dirección: <https://docs.python.org/3/library/asyncio.html>.
- [35] PX4 Development Team, *PX4 Simulation Documentation*, Accedido: 15-nov-2025, 2025. dirección: <https://docs.px4.io/main/en/simulation/>.

Bibliografía

- [36] Wikipedia contributors, *Gazebo (simulador)*, Accedido: 26-nov-2025, 2025. dirección: [https://en.wikipedia.org/wiki/Gazebo_\(simulator\)](https://en.wikipedia.org/wiki/Gazebo_(simulator)).
- [37] PX4 Development Team, *Simulación con Gazebo*, Accedido: 26-nov-2025, 2025. dirección: https://docs.px4.io/main/en/sim_gazebo_gz/.
- [38] MathWorks, *Robot Operating System (ROS 2) Basic Concepts*, Accedido: 30-sep-2025, 2022. dirección: <https://www.mathworks.com/help/ros/gs/robot-operating-system-ros2-basic-concepts.html>.
- [39] Open Robotics, *Changes between ROS 1 and ROS 2*, Accedido: 30-sep-2025, 2020. dirección: <https://design.ros2.org/articles/changes.html>.
- [40] Robotics Backend, *ROS1 vs ROS2 – Practical overview*, Accedido: 30-sep-2025, 2021. dirección: <https://roboticsbackend.com/ros1-vs-ros2-practical-overview>.
- [41] Automatic Addison, *ROS 2 Architecture Overview*, Accedido: 01-oct-2025, 2021. dirección: <https://automaticaddison.com/ros-2-architecture-overview/>.
- [42] Open Robotics, *ROS 2 Documentation — Tutorials*, Accedido: 30-sep-2025, 2023. dirección: <https://docs.ros.org/en/humble/Tutorials.html>.
- [43] Open Robotics, *About different middleware vendors*, Accedido: 30-sep-2025, 2023. dirección: <https://docs.ros.org/en/humble/Concepts/Intermediate/About-Different-Middleware-Vendors.html>.
- [44] S. Paul, D. Lephuoc y M. Hauswirth, “Performance Evaluation of ROS2-DDS middleware implementations facilitating Cooperative Driving in Autonomous Vehicle”, *arXiv preprint arXiv:2412.07485*, 2024.
- [45] Open Robotics, *ROS 2 Documentation — Creating a package*, Accedido: 30-sep-2025, 2023. dirección: <https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-Your-First-ROS2-Package.html>.
- [46] Pallets Projects, *Flask Documentation*, Accessed: 26-dic-2025, 2025. dirección: <https://flask.palletsprojects.com/>.
- [47] T. Z. authors, *ZeroMQ: High-Performance Asynchronous Messaging Library*, Accedido: 20-nov-2025, 2025. dirección: <https://zeromq.org/>.
- [48] J. A. Hartigan y M. A. Wong, “A K-Means Clustering Algorithm”, *Journal of the Royal Statistical Society Series C: Applied Statistics*, vol. 28, n.º 1, págs. 100-108, dic. de 1988, ISSN: 0035-9254. DOI: 10.2307/2346830. eprint: https://academic.oup.com/jrssc/article-pdf/28/1/100/48619400/jrssc_28_1_100.pdf. dirección: <https://doi.org/10.2307/2346830>.

- [49] J. C. Bezdek, R. Ehrlich y W. Full, “FCM: The fuzzy c-means clustering algorithm”, *Computers Geosciences*, vol. 10, n.º 2, págs. 191-203, 1984, ISSN: 0098-3004. DOI: [https://doi.org/10.1016/0098-3004\(84\)90020-7](https://doi.org/10.1016/0098-3004(84)90020-7). dirección: <https://www.sciencedirect.com/science/article/pii/0098300484900207>.
- [50] M. Del Castillo, *GIOD Centralized*, Accedido: 30-nov-2025, 2025. dirección: https://gitlab.fing.edu.uy/mdelcastillo/giod_centralized.
- [51] Docker, Inc., *Docker Documentation*, Accedido: 21-nov-2025, 2025. dirección: <https://docs.docker.com/get-started/docker-overview/>.
- [52] Docker, Inc., *Docker Compose Documentation*, Accedido: 22-nov-2025, 2025. dirección: <https://docs.docker.com/compose/>.
- [53] Open Source Robotics Foundation, *Imagen oficial de ros2 humble full desktop*, Accedido: 28-nov-2025, 2025. dirección: <https://hub.docker.com/layers/osrf/ros/humble-desktop-full/images/sha256-71ae08a6a0aae71a2f981e066c8a1d7dd76e956abf419c04626a0c746c3ebf4f>.
- [54] Docker, Inc., *Mejores Prácticas en Docker*, Accedido: 23-nov-2025, 2025. dirección: <https://docs.docker.com/build/building/best-practices/>.
- [55] PX4 Development Team, *Pixhawk-1*, Accedido: 26-nov-2025, 2025. dirección: https://docs.px4.io/main/en/flight_controller/pixhawk.html.
- [56] STMicroelectronics, *STM32F427*, Accedido: 26-nov-2025, 2025. dirección: <https://www.st.com/en/microcontrollers-microprocessors/stm32f427-437.html>.
- [57] STMicroelectronics, *STM32F100C8*, Accedido: 26-nov-2025, 2025. dirección: <https://www.st.com/en/microcontrollers-microprocessors/stm32f100c8.html>.
- [58] STMicroelectronics, *STM32F030F4*, Accedido: 26-nov-2025, 2025. dirección: <https://www.st.com/en/microcontrollers-microprocessors/stm32f100c8.html>.
- [59] Control Drone, *Cómo funcionan los Drones*, Accedido: 26-nov-2025, 2025. dirección: <https://www.controlatron.es/como-funcionan-los-drones/>.
- [60] A. Abbas, “Real Time Operating Systems (RTOS) For Drones”, en, *ResearchGate*, 2018, Accedido: 2025-06-12. dirección: https://www.researchgate.net/publication/328131120_Real_Time_Operating_SystemsRTOS_For_Drones.

Bibliografía

- [61] Prometec, *Aprendiendo algo del Firmware para un dron*, Accedido: 26-nov-2025, 2025. dirección: <https://www.prometec.net/firmware-para-un-dron/>.
- [62] PX4 Development Team, *Flight Modes Overview*, Accedido: 26-nov-2025, 2025. dirección: https://docs.px4.io/v1.14/en/getting_started/flight_modes.html.
- [63] PX4 Development Team, *Flight Modes (Multicopter)*, Accedido: 26-nov-2025, 2025. dirección: https://docs.px4.io/main/en/flight_modes_mc/.
- [64] PX4 Development Team, *Position Mode (Multicopter)*, Accedido: 26-nov-2025, 2025. dirección: https://docs.px4.io/main/en/flight_modes_mc/position.html.
- [65] PX4 Development Team, *Altitude Mode (Multicopter)*, Accedido: 26-nov-2025, 2025. dirección: https://docs.px4.io/main/en/flight_modes_mc/altitude.html.
- [66] PX4 Development Team, *Hold Mode (Multicopter)*, Accedido: 26-nov-2025, 2025. dirección: https://docs.px4.io/main/en/flight_modes_mc/hold.html.
- [67] PX4 Development Team, *Return Mode (Multicopter)*, Accedido: 26-nov-2025, 2025. dirección: https://docs.px4.io/main/en/flight_modes_mc/return.html.
- [68] PX4 Development Team, *Mission Mode (Multicopter)*, Accedido: 26-nov-2025, 2025. dirección: https://docs.px4.io/main/en/flight_modes_mc/mission.html.
- [69] PX4 Development Team, *Takeoff Mode (Multicopter)*, Accedido: 26-nov-2025, 2025. dirección: https://docs.px4.io/main/en/flight_modes_mc/takeoff.html.
- [70] PX4 Development Team, *Land Mode (Multicopter)*, Accedido: 26-nov-2025, 2025. dirección: https://docs.px4.io/main/en/flight_modes_mc/land.html.
- [71] PX4 Development Team, *Offboard Mode*, Accedido: 26-nov-2025, 2025. dirección: https://docs.px4.io/main/en/flight_modes_mc/land.html.
- [72] PX4 Development Team, *Safety Configuration (Failsafes)*, Accedido: 26-nov-2025, 2025. dirección: <https://docs.px4.io/main/en/config/safety.html>.

- [73] DJI Developer, *Flight Control – DJI Mobile SDK Documentation*, Accedido: 08-jul-2025, 2016. dirección: <https://developer.dji.com/mobile-sdk/documentation/introduction/flightController-concepts.html>.
- [74] Wikipedia contributors, *World Geodetic System*, Accedido: 08-jul-2025, 2025. dirección: https://en.wikipedia.org/wiki/World_Geodetic_System.
- [75] Wikipedia contributors, *Coordenadas geodésicas*, Accedido: 09-jul-2025, 2024. dirección: https://es.wikipedia.org/wiki/Coordenadas_geod%C3%A9sicas.
- [76] GISGeography, *WGS84 – World Geodetic System (WGS-84)*, Accedido: 09-jul-2025, 2025. dirección: <https://gisgeography.com/wgs84-world-geodetic-system>.
- [77] NOAA National Geodetic Survey, *XYZ Conversion Tool*, Accedido: 08-jul-2025, 2025. dirección: <https://geodesy.noaa.gov/TOOLS/XYZ/xyz.shtml>.
- [78] ResearchGate-Scientific Diagram, *Representation of geodetic coordinates: latitude, longitude and height above the reference ellipsoid*, Accedido: 08-jul-2025, 2025. dirección: https://www.researchgate.net/figure/Representation-of-geodetic-coordinates-latitude-ph-longitude-l-and-height-above-the_fig2_236028762.
- [79] S. Loretz, *ROS*, Accedido: 5-mar-2025, 2025. dirección: <https://wiki.ros.org/Distributions>.
- [80] PX4 Development Team, *puente uXRCE-DDS*, Accedido: 5-mar-2025, 2025. dirección: https://docs.px4.io/main/en/ros2/user_guide.
- [81] PX4 Development Team, *uXRCE-DDS*, Accedido: 5-mar-2025, 2025. dirección: https://docs.px4.io/main/en/middleware/uxrce_dds.html.
- [82] Automatic Addison, *ROS 2: Recording a bag from a node (Python)*, Accedido: 29-dic-2025, 2021. dirección: <https://docs.ros.org/en/humble/Tutorials/Advanced/Recording-A-Bag-From-Your-Own-Node-Py.html>.

Esta es la última página.
Compilado el miércoles 31 diciembre, 2025.
<http://iie.fing.edu.uy/>