Universidad Nacional
de Educación a Distancia

Universidad
Complutense de Madrid

Máster en Ingeniería de Sistemas y Control

Integración del dron Crazyflie 2.1 en un sistema ROS 2 mediante micro-ROS

Alumno: **Badr Alioua**

Director: **Francisco José Mañas Álvarez**

Curso: 2024-2025

Defensa: septiembre 2025

Máster en Ingeniería de Sistemas y Control

Integración del dron Crazyflie 2.1 en un sistema ROS 2 mediante micro-ROS

Proyecto específico propuesto por un profesor

Alumno: **Badr Alioua**

Director: **Francisco José Mañas Álvarez**

Autorización

Autorizamos a la Universidad Complutense y a la UNED a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firma del alumno

Badr Alioua

Firmado digitalmente por Badr Alioua
Fecha: 2025.08.23 11:17:59 +02'00'

Agradecimientos

Quisiera aprovechar estas líneas para expresar mi agradecimiento a todas las personas que me han acompañado y apoyado a lo largo de mi etapa universitaria. A todo el profesorado, por su dedicación y por compartir su conocimiento. En especial, quiero dar las gracias a mis tutores del proyecto, por su paciencia, su guía constante y por haber sido una fuente de motivación para seguir aprendiendo y superándome.

Resumen

Este Trabajo Fin de Máster presenta un estudio sobre la integración de ROS 2 Humble en el firmware del Crazyflie, un pequeño vehículo aéreo no tripulado y código abierto. La meta principal de este proyecto era llevar a cabo una revisión de versiones anteriores del firmware del Crazyflie como primer paso para identificar limitaciones y errores existentes, y así desarrollar un agente y un cliente de micro-ROS que se ejecutan de forma estable en un entorno Linux. Este desarrollo tenía como objetivo establecer una comunicación fluida con ROS 2 Humble, aprovechando las funcionalidades y mejoras ofrecidas por una versión más reciente del firmware del Crazyflie.

Para alcanzar este objetivo, se comenzó con familiarizarse con la estructura de micro ros, proporcionando el contexto necesario para entender las mejoras y las capacidades que este sistema operativo robótico ofrece. Después, se realizó una revisión profunda de una versión previa del firmware del Crazyflie, identificando aquellos aspectos que requerían mejoras o presentaban fallas. A base de este análisis, se diseñó y desarrolló un agente de micro-ROS y un cliente correspondiente que no solo se ajustan a los requisitos operativos de un entorno Linux, sino que también garantizan una integración y comunicación eficaces con ROS 2 Humble.

Como resultado final del trabajo, se ha conseguido que la implementación del agente y del cliente de micro-ROS desarrollados logra una comunicación estable con ROS 2 Humble. Esta Comunicación permite al Dron publicar su posición y suscribirse a agentes que estén a su alcance. Este Trabajo no solo contribuye a la mejora del funcionamiento y la integración de sistemas robóticos complejos como el Crazyflie con plataformas de software avanzado como ROS 2 Humble, sino que también establece una base sólida para futuras investigaciones y desarrollos en este campo emergente.

Palabras clave: ROS 2 Humble, micro-ROS, Crazyflie, Publicador/Suscriptor Integración Firmware

Abstract

This Master's Thesis presents a study on the integration of ROS 2 Humble into the Crazyflie firmware, a small open-source unmanned aerial vehicle (UAV). The main goal of this project was to review previous versions of the Crazyflie firmware as a first step to identify existing limitations and errors, and to develop a micro-ROS agent and client that runs stably in a Linux environment. This development aimed to establish smooth communication with ROS 2 Humble, taking advantage of the functionalities and improvements offered by a more recent version of the Crazyflie firmware.

To achieve this objective, the work began by familiarizing with the structure of micro-ROS, providing the necessary context to understand the improvements and capabilities offered by this robotic operating system. Then, an in-depth review of a previous version of the Crazyflie firmware was conducted, identifying areas that required improvement or presented issues. Based on this analysis, a micro-ROS agent and corresponding client were designed and developed. These not only meet the operational requirements of a Linux environment but also ensure effective integration and communication with ROS 2 Humble.

As the result of the work, the implementation of the developed micro-ROS agent and client achieved stable communication with ROS 2 Humble. This communication allows the drone to publish its position and subscribe to agents within its reach. This Thesis not only contributes to improving the functionality and integration of complex robotic systems like Crazyflie with advanced software platforms like ROS 2 Humble, but also lays a solid foundation for future research and development in this emerging field.

Keywords: ROS 2 Humble, micro-ROS, Crazyflie, Publisher/Subscriber model, Firmware integration

Índice general

Contenido

Índice de figuras

Índice de Códigos

Índice de tablas

Lista de Acrónimos

Acrónimo	Significado
BLE	Bluetooth de Baja Energia (del inglés Bluetooth Low Energy)
DDS	Servicio de Distribución de Datos (del inglés Data Distribution Service)
GNU	GNU's Not Unix
GPG	GNU Privacy Guard
IMU	Unidad de Medición Inercial (del inglés Inertial Measurement Unit)
LTS	Soporte a Largo Plazo (del inglés Long-Term Support)
MCU	Unidad de Microcontrolador (del inglés Microcontroller Unit)
micro-ROS	Versión de ROS 2 adaptada para microcontroladores
PC	Ordenador Persona (del inglés Personal Computer)
RAM	Memoria de acceso aleatorio (del inglés Random Access Memory)
ROS	Sistema Operativo para Robot (del inglés Robot Operating System
ROS 2	Sistema Operativo para Robot (del inglés Robot Operating System versión 2)
RTOS	Sistema operative de tiempo real (del inglés Real-Time Operating System)
UAV	Vehículo Aéreo No Tripulado (del inglés Unmanned Aerial Vehicle)
USB	Bus Serie Universal (del inglés Universal Serial Bus)
XRCE-DDS	Entorno con recursos extremadamente limitados (del inglés eXtremely Resource Constrained Environments - Data Distribution Service)

1. Introducción

En este capítulo, se introducen los pilares fundamentales de este Trabajo Fin de Master (TFM), centrándose en dos elementos fundamentales: por un lado, el avance de la tecnología ROS, una herramienta esencial en el desarrollo de sistemas robóticos; y por otro lado, su notable capacidad para poder integrarse de manera flexible y eficiente en sistemas robóticos con recursos limitados. A continuación, se presentarán los objetivos principales propuestos en este trabajo y finalmente se incluirá una descripción de la estructura de la memoria.

1.1. Motivación

En la última década, la robótica ha experimentado un crecimiento exponencial, no solo en términos de aplicaciones industriales y de investigación, sino también en su inclusión en la vida cotidiana. Este avance ha sido posible gracias al desarrollo de tecnologías innovadoras que permiten a los robots realizar tareas cada vez más complejas de manera autónoma. Una de estas tecnologías es *Robot Operating System* (ROS), un framework de software flexible para el desarrollo de aplicaciones robóticas, que ha evolucionado desde la creación de ROS en el año 2007 hasta la actualidad con versiones como ROS 2. Esta versión promete mejoras significativas en términos de seguridad, compatibilidad con sistemas empotrados e introduce mejoras en las herramientas de visualización, abriendo un nuevo horizonte de posibilidades para la robótica de pequeña escala [1].

Crazyflie es un micro-dron o Vehículo Aéreo No Tripulado UAV (del inglés *Unmanned Aerial Vehicle*) que ha ganado popularidad en el campo de la investigación debido a su pequeño tamaño, versatilidad y capacidad de personalización. Desarrollado por la compañía sueca Bitcraze, Crazyflie es un dron de código abierto que ha sido usado por investigadores de diversas disciplinas para una amplia gama de aplicaciones [2].

En referencia a su tamaño compacto, Crazyflie es ideal para interiores y experiencias de enjambres [3] y puede ser equipado con una variedad de sensores,

cámaras y otros dispositivos dependiendo de las necesidades de la investigación. Esto lo convierte en una herramienta ideal para experimentar con algoritmos de control, navegación autónoma, percepción del entorno y otras áreas de investigación relacionadas con la robótica y la inteligencia artificial.

Con el fin de potenciar sus capacidades y facilitar su integración en entornos robóticos más complejos, cobra especial importancia la mejora de su compatibilidad con plataformas estándares muy extendidas en la industria y academia, como ROS 2. En esta línea, micro-ROS se ha desarrollado como una solución clave, ofreciendo una versión de ROS 2 mejorada para dispositivos con limitaciones de recursos, tal como los microcontroladores que operan dentro de Crazyflie [4].

El presente Trabajo Fin de Máster se centra en el desarrollo de una versión actualizada del agente de micro-ROS para el Crazyflie, partiendo como base del análisis detallado de versiones previas del firmware del UAV. A lo largo del proyecto, no solo se identificaron y evaluaron las dependencias críticas de las librerías utilizadas, sino que también se diseñó y construyó un nuevo cliente de micro-ROS adaptado a los requisitos de las plataformas modernas. Esta actualización no se limitó únicamente a la corrección de errores existentes, sino que supuso una mejora significativa en la eficiencia y el aprovechamiento de los recursos internos del dispositivo, impulsando su rendimiento y estabilidad en entornos ROS 2. Por lo tanto, la motivación detrás de este proyecto radica en aprovechar las posibilidades que micro-ROS ofrece para la gestión y control de robots autónomos, específicamente para el Crazyflie 2.1.

1.2. Contexto tecnológico

Micro-ROS constituye un conjunto de librerías de software que facilita el desarrollo de aplicaciones robóticas destinadas a ser implementadas en microcontroladores, los cuales suelen contar con recursos computacionales limitados. Este marco de trabajo está específicamente diseñado para su uso con ROS 2 y permitiendo una comunicación bidireccional entre nodos ROS 2 que operan en un sistema externo y la aplicación de micro-ROS que se ejecuta en un microcontrolador, como por ejemplo, un Crazyflie 2.1, placas de la familia Arduino etc. Micro-ROS es un proyecto de código abierto y puede ser altamente beneficioso para cualquier especialista o investigador en robótica que tenga como objetivo integrar microcontroladores de bajo nivel en un sistema robótico.

Este TFM se desarrolla en el contexto de la plataforma Robotic Park [5], una plataforma destinada a la investigación en sistemas robóticos distribuidos y colaboración multi-agente. Dentro de esta plataforma, los crazyflies desempeñan un

papel central como unidad aérea ligera y versátil, ideal para la experimentación en entornos interiores y escenarios de enjambres robóticos. El presente proyecto contribuye a esta línea de trabajo mediante la mejora de la integración de los crazyflies como agentes distribuidos, sentando las bases para su utilización coordinada en sistemas multi-agente.

1.3. Objetivos del Trabajo fin de Máster

Este TFM tiene como objetivo principal investigar cómo se puede lograr una efectiva integración entre micro-ROS y ROS 2 Humble dentro del entorno del firmware del Crazyflie 2.X. El estudio se centra en los desafíos clave:

a. **Investigar la integración efectiva** entre micro-ROS y ROS 2 Humble dentro del firmware del Crazyflie 2.X, realizando un examen detallado de las iteraciones anteriores del firmware del Crazyflie con el objetivo de detectar y subsanar cualquier limitación o error existente.

b. **Diseñar y desarrollar un agente y un cliente de micro-ROS** que funcionen de manera óptima en un entorno Linux y que, al mismo tiempo, logre una comunicación fluida con ROS 2 Humble, utilizando para ello una versión más reciente del firmware de este UAV

   Dentro de este marco, el crazyflie asume un rol crucial como nodo publicador y suscriptor dentro del sistema ROS 2. Su principal función es publicar de manera continua su posición, permitiendo que otros agentes interesados se suscriban a esta información. Además, está diseñado para aceptar suscripciones, lo que le permite recibir datos relevantes de otros nodos dentro de la red.

c. **Implementar al Crazyflie como nodo publicador y suscriptor** dentro del sistema ROS 2, permitiéndole publicar su posición y recibir comandos desde otros nodos.

La Figura 1.1 ilustra la arquitectura general del sistema propuesto, donde se observa la interacción usuario-UAV mediante suscripciones a otro agente ROS. Este esquema permite tanto el control del vehículo aéreo como la visualización de datos recolectados por un agente instalado en un entorno Linux. Por una parte, los datos generados de manera constante por los nodos son captados por el Crazyflie, que, a su vez, cambia su comportamiento dependiendo del comando recibido. Este proceso de recopilación y transmisión de información se realiza mediante el intercambio de comandos entre los tres componentes del sistema, utilizando un esquema

Publisher/Subscriber.

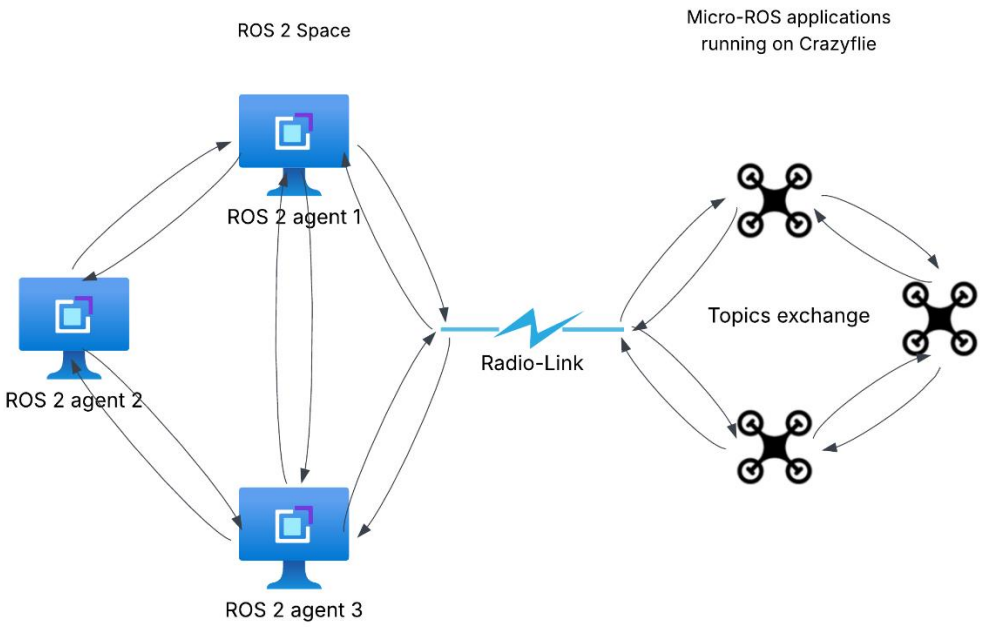

Figura 1.1 Esquema general del sistema

1.4. Planificación temporal del proyecto

Para estructurar el desarrollo de este Trabajo Fin de Máster y asegurar una ejecución ordenada, se estableció una planificación temporal basada en un diagrama de Gantt. En este diagrama se reflejan las principales fases del proyecto, desde el estudio preliminar hasta la documentación final, asignando a cada etapa un periodo estimado de duración. La Tabla 1.1 muestra la planificación del trabajo.

Tarea		Estado	Cronograma
Estudio preliminar de ROS y micro-ROS		Completada	14 sep. 24 – 5 oct. 24
Preparación del entorno		Completada	6 oct. 24 – 13 oct. 24
Análisis del firmware		Completada	19 oct. 24 – 9 nov. 24
Actualización de las librerías y makefiles	Subtarea 1: Depuración de errores	Completada	10 nov. 24 – 4 ene. 25
	Subtarea 2: Compilación continua		
Desarrollo del agente y cliente		Completada	29 dic. 24 – 23 mar. 25
Integración y validación		Completada	18 ene. 25 – 23 mar. 25
Documentación y resultados		Completada	16 feb. 25 – 26 abr. 25
Redacción y revisión de la memoria		Completada	15 mar. 25 – 27 jul. 25

Tabla 1.1 Línea de tiempo del proyecto

1.5. Estructura de la memoria

El trabajo desarrollado está organizado en seis capítulos. En el primer capítulo se introducen los aspectos fundamentales que se desarrollarán en los siguientes capítulos, y se definen los objetivos y los resultados alcanzados. El segundo capítulo recorre la evolución de ROS, destacando las limitaciones y ventajas de micro-ROS en entornos con recursos limitados, mientras que el tercero detalla las tecnologías seleccionadas y su relevancia para alcanzar los objetivos del proyecto.

Los capítulos siguientes profundizan en el desarrollo práctico. El cuarto describe el proceso de implementación del sistema, acompañado de diagramas de actividad que explican su funcionamiento interno. El capítulo cinco analiza las pruebas realizadas para validar el cumplimiento de los requisitos establecidos. Finalmente, el capítulo seis recoge las conclusiones del estudio y propone posibles líneas futuras de trabajo. Además, se incluyen dos apéndices: uno con el presupuesto del proyecto y otro con un manual de uso del sistema.

2. Marco teórico

2.1. ROS

Robot Operating System (ROS) es un entorno de software versátil y escalable, práctico para el diseño y la programación en robótica. No se trata de un sistema operativo en el sentido convencional, sino más bien de una plataforma que proporciona interfaces, componentes y herramientas para crear aplicaciones robóticas eficientes. ROS permite integrar componentes de los robots (sensores, actuadores y procesadores) a través de mensajes y tópicos. Eso facilita la gestión de dispositivos a bajo nivel, la implementación de funciones comunes, la comunicación entre procesos y la administración de paquetes de software. Desde sus inicios, ROS ha evolucionado hasta convertirse en un referente indiscutible dentro del campo de la investigación y el desarrollo en robótica, sustentado por una comunidad de usuarios dinámica y un extenso repositorio de herramientas y librerías [6].

Origen de ROS

ROS se originó en 2007 como resultado de una colaboración entre la Universidad de Stanford y Willow Garage, una empresa de robótica. El objetivo era crear un marco de trabajo que facilitara la investigación y el desarrollo en robótica mediante un conjunto de herramientas y librerías estandarizadas que promueven la reutilización de código en la comunidad robótica. La primera versión oficial, ROS 1, se lanzó en 2010, bajo el nombre "*Box Turtle*" marcando el inicio de una era de innovación en el diseño, desarrollo e implementación de sistemas robóticos [7].

Desde entonces, se han sucedido numerosas distribuciones, cada una con mejoras progresivas en funcionalidad, estabilidad y compatibilidad.

Evolución de ROS

La trayectoria de ROS ha sido marcada por un desarrollo y crecimiento notables

desde su inicio. Cada nueva versión ha traído consigo avances y añadido funcionalidades que han enriquecido su ecosistema. La comunidad que rodea a ROS se ha expandido de manera impresionante, lo que ha aportado distintos conjuntos de paquetes que abarcan desde la navegación autónoma hasta técnicas avanzadas de manipulación y percepción. Esta versatilidad para tratar con distintos tipos de hardware y software ha jugado un rol importante en la amplia adopción de ROS, tanto en proyectos de investigación como en soluciones comerciales. La adaptabilidad y la capacidad de integración de ROS han cimentado su posición como una herramienta esencial en el ámbito de la robótica.
La última versión de ROS 1 fue Noetic Ninjemys, lanzada en 2020, y está pensada como la última versión con soporte a largo plazo, LTS (del inglés *Long-Term Support*), compatible con Ubuntu 20.04. Con Noetic, se cerró el ciclo de actualizaciones mayores de ROS 1, dando paso a la consolidación de ROS 2 como sucesor natural tal y como se muestra en la Figura 2.

Figura 2.1 Evolución de ROS [8]

ROS 2

La transición significativa hacia ROS 2, que tuvo lugar en el año 2015, marcó un paso importante en el desarrollo de esta plataforma. Concebido y desarrollado desde sus cimientos, ROS 2 se propuso superar ciertas restricciones observadas en ROS 1, enfocándose particularmente en aspectos críticos como la operación en tiempo real, la seguridad y la capacidad de escalado del sistema. ROS 2 tiene una interfaz de middleware abstracta, a través de la cual se proporciona serialización, transporte y detección Actualmente todas las implementaciones de esta interfaz se basan en el estándar DDS (del inglés *Data Distribution Service*). Esto permite que ROS 2

proporcione varias políticas de calidad de servicio que mejoran la comunicación en diferentes redes. ROS 2 también constituye una mejora sustancial en términos de rendimiento, seguridad y capacidad para trabajar en entornos de sistemas distribuidos. Además, ha sido diseñado con una mayor facilidad de uso en mente, orientado específicamente hacia su aplicación en productos comerciales y situaciones que demandan una alta fiabilidad y eficiencia [9].

La primera versión oficial de ROS 2 fue Ardent Apalone, publicada en diciembre de 2017. Desde entonces, el nombre de cada versión ha seguido un orden alfabético donde la primera palabra la vota la comunidad de desarrolladores y la segunda se debe asemejar a una especie real de tortuga. La versión utilizada en este trabajo es ROS 2 Humble Hawksbill, lanzada en mayo de 2022, y es una versión con soporte a largo plazo (LTS), lo que la hace especialmente adecuada para proyectos de investigación y desarrollo sostenido. Como parte de la serie de lanzamientos de ROS 2, Humble continúa con el compromiso de mejorar la usabilidad, la seguridad y el soporte para una variedad aún mayor de hardware y sistemas operativos. Esta versión trae consigo mejoras significativas en la estabilidad y nuevas funcionalidades, asegurando que los desarrolladores y los investigadores tengan acceso a un conjunto de herramientas robusto y flexible para sus proyectos de robótica

La última versión disponible de ROS 2 es Kilted Kaiju, lanzada en mayo de 2025, que incluye nuevas funcionalidades y mejoras de rendimiento, aunque aún no está consolidada. En la siguiente tabla comparativa se puede apreciar las diferencias entre las dos versiones [9] [10]:

Características	ROS	ROS 2
Año de lanzamiento	2010 (Box Turtle)	2017 (Ardent Apalone)
Middleware de comunicación	Personalizado (ROS Master)	Basado en DDS (Data Distribution Service)
Tiempo real	No nativo	Sí, soporte para tiempo real
Seguridad	Limitada	Mejorada (DDS-Security)
Arquitectura	Centralizada	Distribuida
Compatibilidad multi-robot	Limitada	Mejorada
Sistemas operativos soportados	Principalmente Linux	Linux, Windows, MacOS
Lenguajes principales	C++, Python	C++, Python (mejorado soporte a otros)
Soporte a dispositivos embebidos	Muy limitado	Sí, mediante micro-ROS
Última versión estable	ROS Noetic Ninjemys (2020)	ROS 2 Kilted Kaiju (2025)

Tabla 2.1 Comparación entre ROS y ROS 2

micro-ROS

Reconociendo la necesidad de extender las capacidades de ROS a dispositivos de recursos limitados, como microcontroladores, la comunidad de ROS introdujo micro-ROS. Esta iniciativa representa una adaptación de ROS 2 para sistemas embebidos, permitiendo que incluso los dispositivos con restricciones de hardware puedan participar en el ecosistema de ROS. Micro-ROS se presenta como un puente entre el mundo de los microcontroladores y el más amplio universo de ROS, facilitando la creación de sistemas robóticos más integrados y versátiles.

Una de las principales contribuciones de micro-ROS es su middleware basado en XRCE-DDS (del inglés *eXtremely Resource Constrained Environments Data Distribution Service*), el cual fue especialmente diseñado para entornos extremadamente restringidos en recursos. Esto asegura una comunicación eficiente y confiable incluso bajo condiciones de hardware limitado y en redes potencialmente inestables.

Desde su lanzamiento, micro-ROS ha permitido a investigadores y desarrolladores llevar las ventajas de ROS a plataformas antes consideradas inaccesibles, tales como drones ligeros, sensores inteligentes, y robots móviles pequeños. Con ello, micro-ROS ha impulsado el desarrollo de sistemas robóticos embebidos y de bajo consumo, creando nuevas oportunidades y expandiendo considerablemente el alcance de aplicaciones robóticas [4] [11].

ROS-Industrial

Es una extensión de ROS pensada para su aplicación en entornos industriales. Su objetivo principal es llevar las ventajas del ecosistema ROS, como la modularidad, el desarrollo colaborativo y la facilidad de integración, al mundo de la automatización y la manufactura. Proporciona controladores, librerías y herramientas específicamente diseñadas para trabajar con robots industriales, como brazos robóticos, sensores industriales y sistemas de visión artificial. Es ampliamente utilizado en sectores como la automoción, la logística y la producción avanzada [12] [13].

2.2. Limitaciones de micro-ROS

Dependencia a la red de comunicación

Uno de los desafíos más presentes que enfrenta ROS en el escenario de los sistemas empotrados es su profunda necesidad de conectividad de red. Funcionando bajo una arquitectura de publicación y suscripción, ROS facilita el intercambio de mensajes entre nodos mediante tópicos. Este modelo asume una red estable y segura, una tarea no siempre realizable en aplicaciones prácticas. Tal es el caso de flotas robóticas operando en ambientes cambiantes, donde pueden surgir inconvenientes como latencias, congestión de red, o fallos en la conexión. Estos desafíos pueden perjudicar la eficacia y fiabilidad del sistema, y en ciertas circunstancias, resultar en pérdida o desincronización de datos.

Carencia de seguridad

La seguridad es otra limitación prominente de ROS en contextos distribuidos, dada su falta de mecanismos de seguridad integrados, incluyendo cifrado, autenticación y autorización. Esta carencia lo expone a posibles amenazas de seguridad, como intrusiones maliciosas o accesos no autorizados. Un agente hostil podría, por ejemplo, interceptar, modificar o insertar mensajes en el sistema, comprometiendo su integridad o funcionalidad. De igual forma, individuos no autorizados podrían acceder o manipular robots, exponiendo datos sensibles o

causando perjuicios. Aunque es posible mitigar estos riesgos con herramientas o marcos de seguridad externos, su integración y eficacia con ROS podrían no ser óptimas. En este sentido, SROS2, una extensión de ROS 2 basada en el estándar DDS-Security, ofrece funcionalidades nativas para mejorar significativamente la seguridad mediante cifrado, autenticación y gestión de acceso a recursos. No obstante, el uso extendido de SROS2 aún requiere un esfuerzo considerable en configuración y gestión, especialmente en sistemas embebidos y distribuidos con limitaciones de recursos, lo que mantiene la seguridad como un área de mejora continua dentro del ecosistema ROS [14].

Problemas de compatibilidad

La compatibilidad presenta una tercera barrera para ROS en ambientes distribuidos. Si bien existen soluciones como ROS bridge, que permiten la comunicación entre ROS 1 y ROS 2, estas herramientas pueden introducir cierta complejidad y afectar el rendimiento o la simplicidad del sistema. En general, dentro de una misma versión de ROS, las distintas distribuciones son compatibles entre sí, lo que facilita la interoperabilidad. Sin embargo, cuando se busca aprovechar mejoras significativas de nuevas versiones, suele ser necesario migrar o adaptar el código y los paquetes existentes, lo que puede generar desafíos de compatibilidad entre librerías o dependencias

2.3. Alcance de micro-ROS

Micro-ROS ha alcanzado progresos notables en su integración con sistemas basados en microcontroladores, abriendo el espectro de aplicaciones en robótica e IoT que pueden beneficiarse de la estructura y herramientas de ROS 2 en dispositivos de recursos limitados. Esta iniciativa se centra en ajustar ROS 2 para su operación eficiente en microcontroladores y otros sistemas empotrados, marcando un avance fundamental hacia una robótica más accesible y versátil [10]. Algunos de los avances alcanzados se reflejan en lo siguiente [15]:

- **Herramientas y Bibliotecas**: Se han desarrollado herramientas específicas y librerías para facilitar la integración de micro-ROS en proyectos de microcontroladores. Esto incluye soporte para configuraciones personalizadas, facilitando la comunicación entre dispositivos de recursos limitados y sistemas más complejos basados en ROS 2.

- **Compatibilidad con Plataformas**: micro-ROS ha mejorado su compatibilidad con una amplia gama de plataformas de microcontroladores y sistemas operativos en tiempo real (RTOS), como NuttX, FreeRTOS, y Zephyr. Esto

facilita su implementación en una variedad más amplia de dispositivos y aplicaciones

- **Comunicación y Middleware**: micro-ROS utiliza XRCE-DDS (*eXtremely Resource Constrained Environments Data Distribution Service*) como su middleware de comunicación, diseñado específicamente para entornos con recursos limitados. Esto ha mejorado significativamente la eficiencia y fiabilidad de la comunicación entre nodos en sistemas empotrados.

- **Casos de Uso y Aplicaciones**: La comunidad de micro-ROS ha demostrado su aplicabilidad en una variedad de casos de uso, desde robots autónomos pequeños hasta sistemas IoT complejos, evidenciando su flexibilidad y potencial en el campo de la robótica y más [16].

- **Desarrollo y Soporte Comunitario**: La comunidad en torno a micro-ROS ha crecido, contribuyendo con documentación, tutoriales y soporte técnico, lo que facilita la adopción y el desarrollo de proyectos basados en micro-ROS [15].

2.4. Micro-UAVs: potencial, aplicaciones y limitaciones

Los micro-UAVs representan una categoría emergente dentro de la robótica aérea, caracterizada por su ligereza, bajo coste, y alta maniobrabilidad. Su reducido tamaño les permite operar de forma segura en entornos cerrados, donde los UAVs tradicionales no podrían desenvolverse con facilidad [17] [18]

En el ámbito de los UAV se establece una clasificación por tamaño que incluye cuatro categorías principales: pico-UAV (menos de 5 cm y 50 gramos), destinados a vigilancia discreta; nano-UAV (5-15 cm, 50-200 gramos), ideales para misiones breves en interiores; micro-UAV (15-30 cm, 200 gramos a 2 kg), que equilibran maniobrabilidad y autonomía, útiles en investigación y operaciones en interiores; y mini-UAV (30 cm a 2 m), con mayor carga útil y autonomía, utilizados principalmente en exteriores para tareas especializadas como inspección o monitoreo.

Potencial y ventajas

Gracias a su tamaño compacto, los micro-UAVs ofrecen múltiples ventajas y posibilidades operativas, siendo especialmente adecuados para realizar operaciones en interiores, como inspecciones en almacenes, laboratorios o edificios [19]. Además, su reducido tamaño facilita misiones coordinadas en enjambre, donde

varias unidades pueden trabajar conjuntamente en tareas complejas. Esta versatilidad los convierte en plataformas ideales para la investigación académica, permitiendo validar algoritmos avanzados de control, percepción y navegación. También destacan en aplicaciones de inspección en espacios reducidos o de difícil acceso, como tuberías o estructuras industriales, así como en misiones críticas de búsqueda y rescate en entornos urbanos o zonas de derrumbe [20], aprovechando su capacidad para acceder a lugares que serían inaccesibles para drones de mayor tamaño. Además, la modularidad de plataformas como el Crazyflie 2.1 permite equiparlos con sensores adicionales (como cámaras, sensores de distancia o módulos de IA), aumentando considerablemente sus capacidades.

Limitaciones actuales

Pese a sus numerosas ventajas, los micro-UAVs enfrentan desafíos técnicos significativos que limitan su aplicabilidad en ciertos contextos. Su capacidad de carga útil es reducida debido a restricciones estructurales y de peso, lo que impide transportar sensores o equipos adicionales sin comprometer la estabilidad y el rendimiento del vuelo. En cuanto a la autonomía, la mayoría de estos dispositivos dependen de baterías de baja capacidad, lo que limita su tiempo de vuelo a entre 15 y 30 minutos, restringiendo su uso en misiones prolongadas [20]. Además, su capacidad de procesamiento es limitada; sin módulos adicionales, el procesador interno de estos drones no puede ejecutar tareas complejas de percepción o inteligencia artificial en tiempo real [21]. Por último, debido a su ligereza, los micro-UAVs son especialmente sensibles a perturbaciones ambientales como corrientes de aire o vibraciones, lo que puede afectar su estabilidad y control durante el vuelo. Estas limitaciones abren la puerta a la incorporación de extensiones como el AI-deck, que permiten dotar a los micro-UAVs de capacidad de procesamiento local para tareas avanzadas como visión por computador, navegación autónoma o aprendizaje automático.

3. Tecnologías y elementos

En este capítulo se describen las tecnologías, plataformas y elementos empleados y necesarios para el desarrollo del proyecto. La elección de cada elemento está basada en su adecuación para cumplir con los objetivos plateados en el trabajo.

Además, se detallarán tanto los componentes de software como de hardware, explicando brevemente sus características, funcionalidades y el papel que juegan dentro del sistema.

3.1. ROS 2 Humble

ROS 2 Humble Hawksbill es la distribución de ROS 2 seleccionada para este trabajo. Su elección se debe a la estabilidad y soporte a largo plazo (LTS), así como a las mejoras en seguridad, comunicación en sistemas distribuidos y compatibilidad con entornos embebidos. Además, cuenta con una comunidad activa y amplia documentación, lo cual facilita su adopción en proyectos de investigación.

ROS 2 Humble proporciona un sistema de publicación/suscripción robusta basado en DDS, lo que permite una comunicación eficiente entre nodos incluso en redes no deterministas, como las que se presentan en sistemas inalámbricos con microcontroladores.

3.2. micro-ROS

micro-ROS es una extensión de ROS 2 diseñada para su uso en microcontroladores con recursos limitados. Su integración permite que dispositivos como Crazyflie participen como nodos dentro del ecosistema ROS, enviando y recibiendo información en tiempo real.

Para este proyecto, micro-ROS se ha empleado en el firmware del Crazyflie, permitiendo al dron publicar su posición y suscribirse a comandos desde otros

agentes ROS. Esto ha sido posible gracias a la implementación de un cliente micro-ROS dentro del firmware y su comunicación con un agente ROS 2 desplegado en un entorno Linux; concretamente Ubuntu Jammy (22.04 LTS).

3.3. Crazyflie 2.1

Crazyflie 2.X es un micro dron de código abierto y alta modularidad, desarrollado por Bitcraze [22]. Su pequeño tamaño y versatilidad lo hacen ideal para entornos de investigación.

En este proyecto, el Crazyflie actúa como un nodo ROS 2 gracias a la integración de micro-ROS en su firmware. El sistema establece un canal de comunicación bidireccional que permite: (1) la publicación de su posición, y (2) la recepción de comandos de control desde un agente ROS externo.

3.3.1. Especificaciones técnicas

Figura 3.1 Crazyflie 2.1 [22]

El cuadricóptero Crazyflie 2.1 está equipado con los siguientes componentes y características técnicas:

- **Procesador principal (MCU)**: STM32F405 ARM Cortex-M4 a 168 MHz, con FPU (Unidad de Punto Flotante) integrada y 192 kB de RAM.

- **Coprocesador de radio y carga**: nRF51822 SoC con ARM Cortex-M0 a 16 MHz

y 16 kB de RAM.

- **Sensores integrados**: IMU de 3 ejes: acelerómetro, giroscopio (Bosch BMI088). Sensor de presión barométrico (BMP388).

- **Comunicación inalámbrica**: Radio de 2.4 GHz (protocolo propio de Bitcraze). Bluetooth Low Energy (BLE) para configuración y control básico.
  Alcance de la Comunicación hasta 1km dependiendo del entorno.

- **Alimentación**: Batería LiPo de 3.7 V y 240 mAh, con carga vía micro-USB o deck de expansión.

- **Tamaño**: 92 mm de diámetro entre ejes de motor.

- **Peso**: 27 g (incluyendo batería).

- **Puertos de expansión**: Dos puertos de expansión (uno frontal y uno trasero) para conectar decks adicionales (sensores, cámaras, etc.).

- **Sistema operativo**: FreeRTOS.

- **Compatibilidad**: Totalmente compatible con herramientas de desarrollo de código abierto de Bitcraze y adaptable mediante firmware personalizado.

Estas características hacen del Crazyflie 2.1 una plataforma óptima para experimentar con tecnologías como micro-ROS, facilitando su integración en entornos robóticos avanzados y distribuidos.

3.3.2. Expansiones y accesorios

Flow Deck v2

El Flow Deck v2 permite al Crazyflie realizar posicionamiento relativo en interiores sin necesidad de sistemas externos como GPS. Funciona mediante sensores ópticos y ultrasónicos:

- **Sensor óptico**: PMW3901, capaz de medir el flujo óptico horizontal en superficies con textura.

- **Sensor de distancia**: VL53L1x (ToF – Time of Flight), proporciona medición precisa de altura con un rango de hasta ~4 metros.

- **Conectividad**: Se acopla al puerto de expansión inferior.
- **Uso principal**: Posicionamiento local, vuelo estacionario, navegación autónoma en interiores.
- **Peso**: 1.6g.
- **Tamaño (ancho x alto x largo):** 21x28x4mm.
- Diseñado para montaje debajo del Crazyflie 2.X.

Figura 3.2 Flow Deck v2 [22]

Multiranger Deck

El Multiranger Deck proporciona detección de obstáculos en cinco direcciones, ampliando la percepción espacial del Crazyflie:

- **Sensores**: 5 sensores de distancia VL53L0x (uno en cada lateral y uno hacia abajo).
- **Rango de medición**: hasta ~2 metros por sensor.
- **Frecuencia de actualización**: ~20 Hz por sensor.
- **Uso principal**: Prevención de colisiones, navegación autónoma, mapeo básico del entorno.

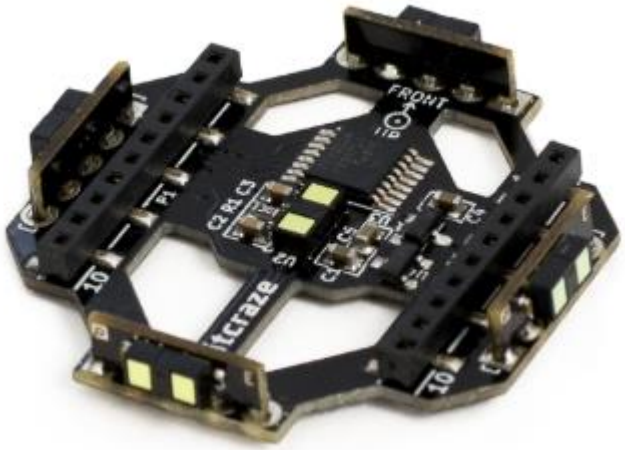

Figura 3.3 Multiranger Deck [22]

Crazyradio PA

El Crazyradio PA es un transceptor USB que permite la comunicación inalámbrica robusta de largo alcance entre el PC y el Crazyflie:

- **Frecuencia**: 2.4 GHz (protocolo propietario Bitcraze).
- **Potencia de transmisión**: Hasta +20 dBm (PA = Power Amplifier).
- **Alcance**: hasta 1 km en condiciones óptimas.
- **Velocidad de datos**: Hasta 2 Mbps.
- **Interfaz**: USB 2.0.
- **Uso principal**: Establecimiento de conexión inalámbrica confiable entre el entorno Linux (agente ROS 2) y el dron.
- **Firmware** de código abierto.
- **Actualización del firmware por USB**.
- **Baja latencia**.
- **Peso**: 6g.

- **Tamaño (ancho x alto x largo)**: 58x16x6.5 mm (incluyendo conectores).

Figura 3.4 Crazyradio PA [22]

3.4. PC Gamepad

Para controlar el dron de forma sencilla e intuitiva, se recomienda usar mandos de PlayStation o de Xbox. Una alternativa más económica sería usar un mando genérico que se puede comprar por internet. Este periférico permite enviar comandos de movimiento de forma intuitiva y directa, facilitando la evaluación del sistema tanto en vuelo libre como en escenarios de navegación asistida.

Figura 3.5 Gamepad para PC

3.5. Entorno Software

El desarrollo de este proyecto requiere tanto elementos hardware como software. A continuación, se comentarán las herramientas empleadas para llevar a cabo el desarrollo de este trabajo.

Máquina Virtual

Se ha decidido trabajar con una Máquina virtual para aislar el entorno y garantizar la reproducibilidad de las pruebas. En este ámbito, *VMWare Workstation 16 player* será nuestra herramienta en este caso. Esta máquina dispone de 4 procesadores virtuales, 6 GB de memoria RAM asignada y 70GB de espacio como se muestra en la Figura 3.6

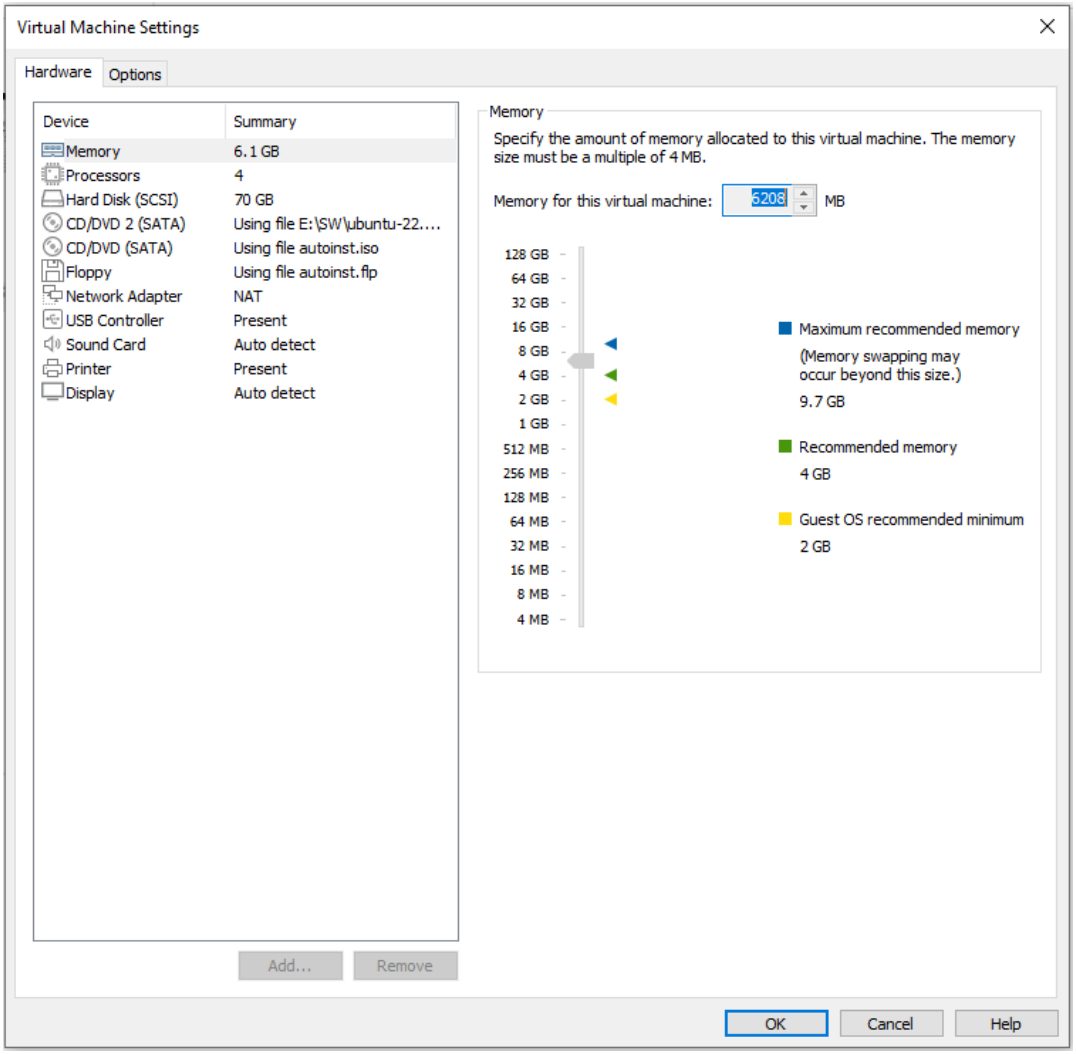

Figura 3.6 Características de la máquina virtual

Ubuntu 22.04

El sistema operativo base en el que se ha desplegado la máquina virtual. Esta distribución de Linux ofrece estabilidad, soporte a largo plazo y compatibilidad plena con ROS 2 Humble ya que ofrece las herramientas y dependencias necesarias.

CFlib

Crazyflie Python library, es una librería desarrollada por Bitcraze que permite la comunicación y control del Crazyflie desde un ordenador. Su uso es esencial para validar la comunicación entre el Crazyradio PA y el Crazyflie antes de activar los nodos ROS.

Crazyflie client

El cliente del Crazyflie es una aplicación desarrollada por Bitcraze que permite interactuar con el dron de forma sencilla. Se puede usar en dos formas: con interfaz gráfica tal y como se muestra en la Figura 3.7 o sin (*Headless client*).

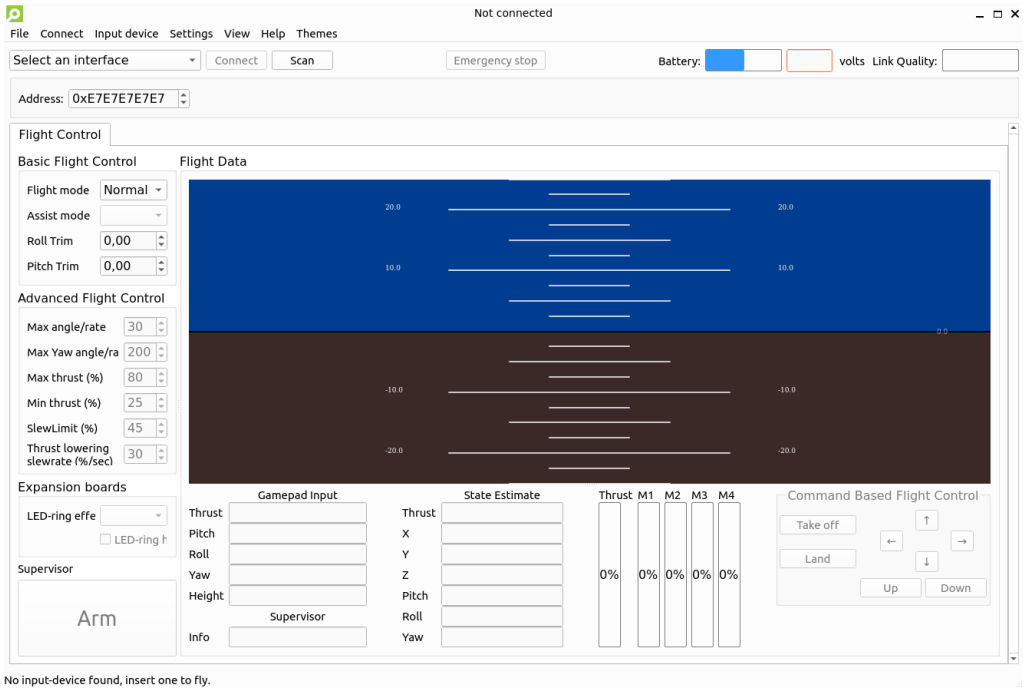

Figura 3.7 Cliente del Crazyflie

4. Desarrollo e implementación

En este capítulo se detalla las etapas prácticas llevadas a cabo para lograr la integración efectiva entre el firmware del Crazyflie y el ecosistema de ROS 2 Humble, a través del uso de micro-ROS. El proceso comienza con la preparación del entorno de desarrollo finaliza con la implementación final del agente y cliente micro-ROS, destacando las decisiones técnicas adoptadas y las soluciones aplicadas a los desafíos encontrados.

Se explica también cómo se configuró el sistema para permitir una comunicación fluida entre el agente desplegado en Linux y el cliente embebido en el dron, así como los pasos seguidos para adaptar el firmware del Crazyflie. También se presentarán los elementos clave que permitieron establecer la arquitectura Publisher/Subscriber y asegurar la estabilidad del sistema. Este capítulo marca la transición del diseño teórico a una solución funcional y comprobable.

4.1. Preparación del entorno de desarrollo

4.1.1. Configuración de la máquina virtual y el sistema operativo

Antes de comenzar con la implementación del sistema, fue necesario preparar un entorno de trabajo estable y reproducible. Para ello, se optó por utilizar una máquina virtual con Ubuntu 22.04 LTS, ya que esta versión del sistema operativo es totalmente compatible con ROS 2 Humble y cuenta con soporte a largo plazo. Además, permite un mejor control sobre las dependencias y posibles conflictos que podrían surgir durante el desarrollo.

Se utilizó VMWare Workstation 16 Player como plataforma para la máquina virtual, principalmente por su facilidad de uso y buen rendimiento. La elección de una máquina virtual también facilitó el aislamiento del entorno de pruebas, evitando afectar otros sistemas del equipo de desarrollo.

4.1.2. Instalación y configuración de ROS 2 Humble

Todos los pasos para la instalación y configuración básica de ROS 2 Humble

se proporcionan en la documentación de ROS [23]

En primer lugar, se asegura que el sistema cuenta con los componentes básicos para manejar repositorios externos. Luego, se añade el repositorio oficial de ROS 2 junto con su clave GPG (del inglés *GNU Privacy Guard*) para garantizar la autenticidad de los paquetes. Una vez configurado el repositorio, se actualiza el sistema y se procede a instalar el metapaquete ros-humble-desktop, que incluye las herramientas y bibliotecas esenciales para el desarrollo con ROS 2. Finalmente, se instalan también herramientas de desarrollo (ros-dev-tools) útiles para compilar y gestionar paquetes ROS, tal y como se muestra en el siguiente código:

```
$ sudo apt install software-properties-common
$ sudo add-apt-repository universe

$sudo apt update && sudo apt install curl -y
$ sudo curl -sSL
        https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -
        o /usr/share/keyrings/ros-archive-keyring.gpg

$ echo "deb [arch=$(dpkg --print-architecture) signed-
        by=/usr/share/keyrings/ros-archive-keyring.gpg]
        http://packages.ros.org/ros2/ubuntu $(. /etc/os-release && echo
        $UBUNTU_CODENAME) main" | sudo tee
        /etc/apt/sources.list.d/ros2.list > /dev/null

$ sudo apt update
$ sudo apt upgrade

$ sudo apt install ros-humble-desktop
$ sudo apt install ros-dev-tools
```

Código 4.1 *Instalación de ROS 2 Humble*

4.1.3. Instalación e integración de micro-ROS en Linux

En primer lugar, se crea el espacio de trabajo, una carpeta llamada microros_ws, donde se clonó el paquete micro_ros_setup desde GitHub, asegurando que la rama utilizada coincidiera con la distribución instalada de ROS 2 (en este caso, Humble). Una vez hecho, se actualizan las dependencias de ROS y se instala la herramienta pip para gestionar componentes de Python.

Con todo el entorno listo, se utilizó colcon build para compilar las herramientas de micro-ROS y cargar sus variables del entorno. Esto se consigue con la siguiente lista de comandos [24]

```
# Source the ROS 2 installation
$ source /opt/ros/humble/setup.bash

# Create a workspace and download the micro-ROS tools
$ mkdir microros_ws
$ cd microros_ws
$ git clone -b $ROS_DISTRO https://github.com/micro-
ROS/micro_ros_setup.git src/micro_ros_setup

#Missing
$ rosdep init

# Update dependencies using rosdep
$ sudo apt update && rosdep update
$ rosdep install --from-paths src --ignore-src -y

# Install pip
$ sudo apt-get install python3-pip

# Build micro-ROS tools and source them
$ colcon build
$ source install/local_setup.bash
```

Código 4.2 Instalación de micro-ROS en Linux

Con el paquete de micro-ROS ya compilado, el entorno está listo para generar el firmware del Crazyflie. Este proceso se realiza con tres pasos definidos por el sistema de construcción de micro-ROS:

1. Crear: descarga los repositorios necesarios y las herramientas de compilación cruzada específicas para la plataforma hardware. También incluye una colección de aplicaciones micro-ROS listas para usar.

2. Configurar: permite seleccionar qué aplicación se va a compilar, además de configurar parámetros como el tipo de transporte, la IP y el puerto del agente (en caso de usar UDP) o el dispositivo serie.

3. Compilar: realiza la compilación cruzada, generando los binarios específicos para la plataforma destino.

4. Flashear: graba los binarios generados en la memoria del dispositivo, dejándolo listo para ejecutar la aplicación micro-ROS seleccionada.

En este punto, únicamente se crea el firmware con el siguiente comando. Los pasos restantes se desarrollarán en los apartados siguientes:

```
$ ros2 run micro_ros_setup create_firmware_ws.sh freertos crazyflie21
```

4.1.4. Instalación del Cliente Crazyflie cfclient

Como primer paso, se ha considerado establecer permisos para el gestor de dispositivos udev con el fin de usar el Crazyradio sin permisos de super usuario

```
$ sudo groupadd plugdev
$ sudo usermod -a -G plugdev $USER
```

Código 4.3 Configuración de permisos de dispositivos udev (a)

Después de reiniciar sesión:

```
cat <<EOF | sudo tee /etc/udev/rules.d/99-bitcraze.rules > /dev/null
# Crazyradio (normal operation)
SUBSYSTEM=="usb",    ATTRS{idVendor}=="1915",    ATTRS{idProduct}=="7777",
MODE="0664", GROUP="plugdev"
# Bootloader
SUBSYSTEM=="usb",    ATTRS{idVendor}=="1915",    ATTRS{idProduct}=="0101",
MODE="0664", GROUP="plugdev"
# Crazyflie (over USB)
SUBSYSTEM=="usb",    ATTRS{idVendor}=="0483",    ATTRS{idProduct}=="5740",
MODE="0664", GROUP="plugdev"
EOF
```

Código 4.4 Configuración de permisos de dispositivos udev (b)

Recargar las udev-rules:

```
$ sudo udevadm control --reload-rules
$ sudo udevadm trigger
```

Código 4.5 Recarga de los permisos udev

Para evitar problemas de compatibilidades entre las distintas versiones de librerías usadas en el desarrollo del proyecto, es conveniente instalar el cliente en un entorno virtual de Python:

```
$ mkdir cfclient
$ cd cfclient
$ sudo apt install python3.10-venv
```

```
$ python3 -m venv env
$ source env/bin/activate
$ pip install cflib
$ pip install cfclient
```

Código 4.6 Instalación del cliente cfclient

Finalmente se puede iniciar el cliente (Figura 4.1) con el siguiente comando:

```
~cfclient$ cfclient
```

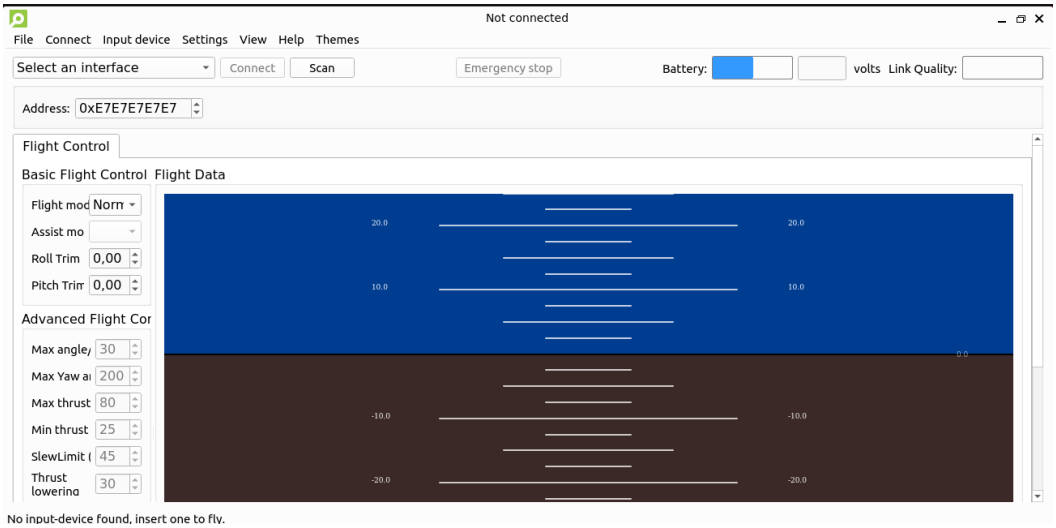

Figura 4.1 Cliente para el control del Crazyflie

Con estos pasos completados, el entorno de desarrollo queda preparado para comenzar con la integración del cliente micro-ROS en el firmware del Crazyflie, lo cual se aborda en una sección posterior.

4.2. Análisis del firmware anterior de Crazyflie

4.2.1. Limitaciones detectadas

El firmware del Crazyflie adaptado para ROS 2 Galactic fue uno de los primeros experimentos para integrar micro-ROS con este dron, y aunque permitió validar conceptos clave, presenta varias limitaciones importantes en comparación con lo que se busca en una implementación más moderna y robusta.

1. **Dependencias obsoletas o desactualizadas** [25]

- El soporte para ROS 2 Galactic ya está *end-of-life*, lo que significa que las herramientas y librerías relacionadas ya no reciben actualizaciones ni parches de seguridad.

- Algunas dependencias en el firmware (como versiones específicas del middleware XRCE-DDS o del build system) no son compatibles con ROS 2 posteriores.

2. **Funcionalidad limitada a publicación**

- En la mayoría de los casos, el firmware solo permitía al Crazyflie publicar su estado, pero no tenía soporte funcional real para suscribirse a otros nodos y ejecutar comandos en respuesta.

- Esto limitaba mucho su integración en sistemas ROS más complejos donde el dron debe reaccionar a eventos externos.

3. **Ausencia de QoS (Quality of Service) configurables** [26]

El soporte para políticas de QoS en versiones tempranas era muy básico o inexistente, dificultando la gestión de comunicaciones en redes inestables (como las redes inalámbricas usadas por el Crazyflie).

4. **Sin mecanismos robustos de reconexión**

Si se pierde la conexión entre el agente ROS y el cliente en el dron, no había una forma estable de reconectar automáticamente sin reiniciar el sistema.

5. **Escalabilidad limitada**

No está preparado para escenarios multi-agente. Es decir, solo puede integrarse de forma rudimentaria en un sistema ROS mayor, no se puede utilizar en flotas o con múltiples tópicos/servicios de forma dinámica.

4.2.2. Estrategia para la migración

En esta fase se ha tomado un enfoque simple y directo para completar la migración. Uno que no consiste en reconstruir el entorno desde cero, sino en tomar el firmware existente compatible con ROS 2 Galactic como referencia, e ir actualizando progresivamente los componentes claves para garantizar su funcionamiento con ROS 2 Humble. Este enfoque permitió conservar parte de la estructura y lógica del sistema original, de forma paralela se adaptaban sus

elementos a las nuevas versiones de las herramientas y librerías.

Lo primero fue identificar los archivos relevantes dentro del repositorio, incluyendo tanto los scripts de compilación como los archivos de configuración propios de micro-ROS. Entre ellos, tuvo especial importancia el archivo colcon.meta, ya que en él se especifica la activación del *custom transport*, una configuración esencial para establecer la comunicación personalizada entre el agente micro-ROS y el cliente embebido en el Crazyflie dentro del espacio ROS.

Una vez revisada la estructura de configuración, se procedió a actualizar los parámetros correspondientes para que fueran compatibles con las versiones recientes de micro XRCE-DDS y de las herramientas de compilación usadas por ROS 2 Humble. Esto implicó también una revisión de las rutas, opciones de compilación y dependencias definidas en el sistema de construcción.

El segundo paso fue adaptar el *Makefile* del firmware del Crazyflie, que se encarga de compilar e integrar los módulos necesarios dentro del entorno FreeRTOS. También fue necesario ajustar las rutas a las nuevas librerías situadas en el repositorio Github. Este paso fue crucial para el avance del trabajo ya que se generaba un error en la compilación (se darán más detalles sobre este punto en el apartado 4.5) Además, se corrigieron nombres de variables y para garantizar que las nuevas cabeceras y fuentes de micro-ROS estuvieran correctamente enlazadas durante la compilación.

La Figura 4.2 muestra el diagrama de flujo con los pasos seguidos durante la migración del firmware original hacia una versión compatible con ROS 2 Humble Esta estrategia de migración gradual ofreció un mayor control sobre cada modificación, permitiendo verificar el impacto de cada cambio sin comprometer el funcionamiento general y las dependencias del sistema. Con esta aproximación, se logró actualizar el firmware original manteniendo su estructura funcional, pero ahora compatible con ROS 2 Humble y preparado para nuevas funcionalidades como la suscripción a tópicos. Los pasos seguidos se reflejan en el siguiente diagrama de flujo.

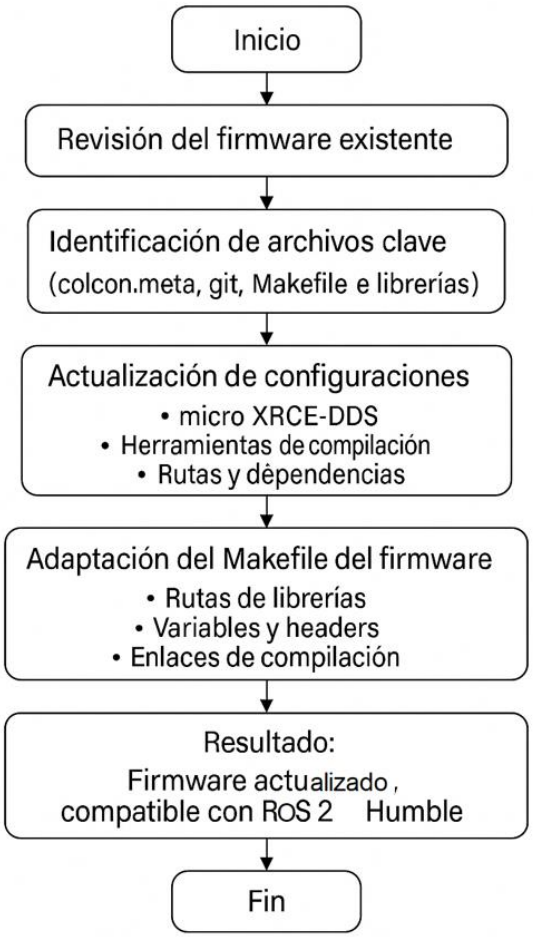

Figura 4.2 Diagrama de Flujo de la estrategia de migración

4.3. Desarrollo del cliente micro-ROS

El desarrollo de la aplicación de micro-ROS que se ejecuta en el dron, se llevó a cabo con el objetivo de habilitar una comunicación fluida y bidireccional con un agente ROS 2 Humble desplegado en Ubuntu. A continuación, se detallan los principales aspectos del proceso de implementación y las decisiones técnicas adoptadas. En la Figura 4.3 se presenta un diagrama de estado de la aplicación que ejecuta el cliente micro-ROS.

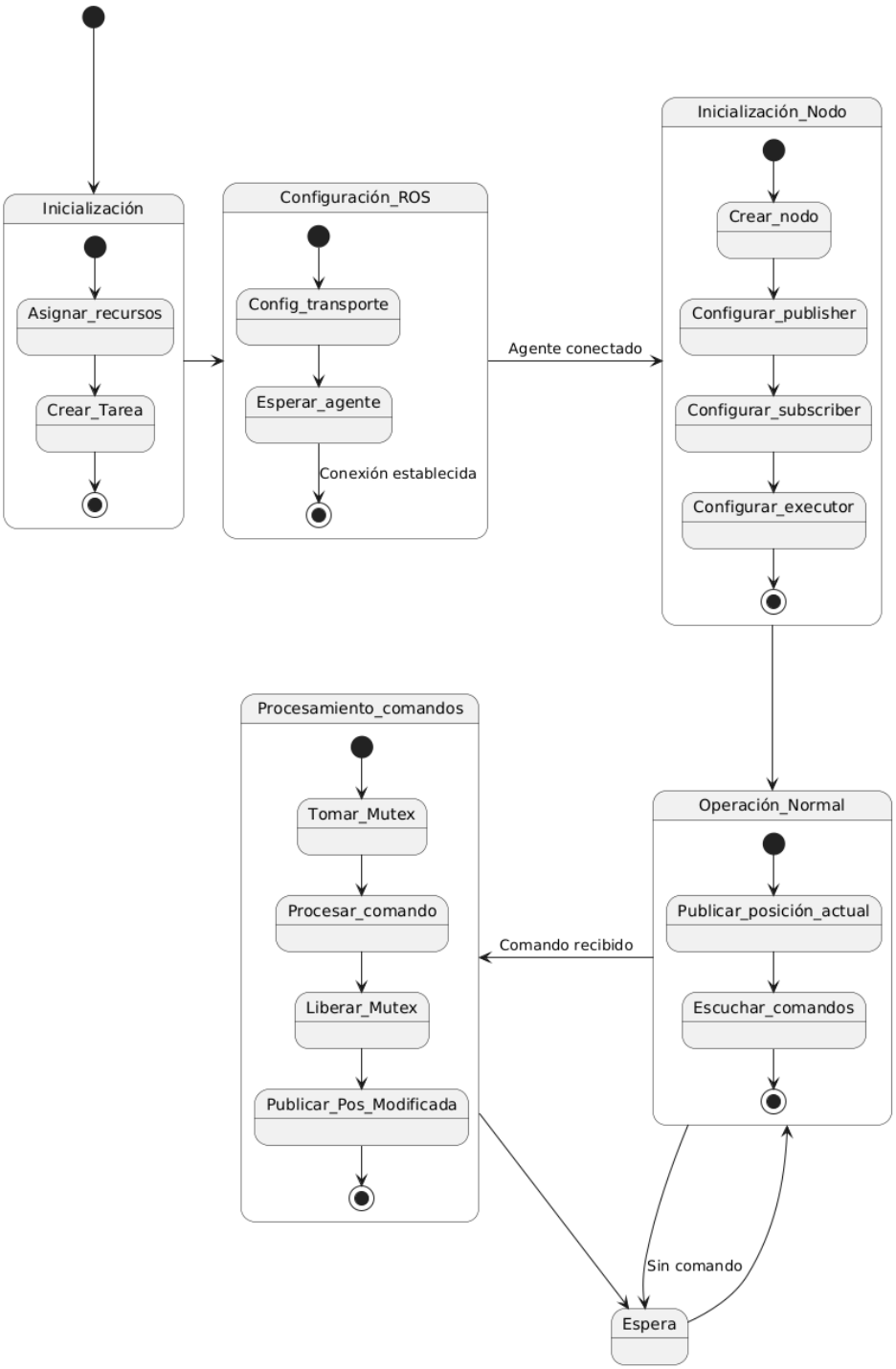

Figura 4.3 Diagrama de estado del cliente micro-ROS

4.3.1. Inicialización de la aplicación

El primer estado de la Figura 4.3 se refiere al *main* de la aplicación donde se asignan los recursos del *FreeRTOS* como la memoria y el *mutex*. La creación de la tarea principal también se realiza en este punto tal y como se muestra en la Figura 4.4.

```
void appMain(){
    // TaskHandle_t task_primary, task_secondary;
    absoluteUsedMemory = 0;
    usedMemory = 0;

    rcl_allocator_t freeRTOS_allocator = rcutils_get_zero_initialized_allocator();
    freeRTOS_allocator.allocate = __crazyflie_allocate;
    freeRTOS_allocator.deallocate = __crazyflie_deallocate;
    freeRTOS_allocator.reallocate = __crazyflie_reallocate;
    freeRTOS_allocator.zero_allocate = __crazyflie_zero_allocate;

    if (!rcutils_set_default_allocator(&freeRTOS_allocator)) {
        DEBUG_PRINT("Error on default allocators (line %d)\n",__LINE__);
        vTaskSuspend( NULL );
    }

    xMutex = xSemaphoreCreateBinaryStatic(&xMutexBuffer);
    xSemaphoreGive(xMutex);

    STATIC_MEM_TASK_CREATE(microros_primary, microros_primary, "microROSprimary", NULL, 3);
}
```

Figura 4.4 Inicialización del Cliente

4.3.2. Comunicación entre el Agente y el cliente

Para el establecimiento de la comunicación entre el Crazyflie y el Agente se utiliza un transporte personalizado basado en el protocolo CRTP (*Crazy RealTime Protocol*). Este transporte personalizado se caracteriza por cuatro funciones principales que permiten abrir o cerrar una conexión y escribir o leer datos.

- *crazyflie_serial_open*
- *crazyflie_serial_close*
- *crazyflie_serial_write*
- *crazyflie_serial_read*

El segundo estado “Configuración_ROS” de la Figura 4.3 es responsable de la configuración del transporte personalizado y se lleva a cabo con la función `rmw_uros_options_set_custom_transport()`, proporcionando parámetros clave como el canal de radio y el puerto, así como un búfer de comunicación definido para manejar la transmisión de mensajes. Con el transporte ya listo para operar, se queda a la espera de la conexión del agente como se muestra en el extracto de código de la Figura 4.5.

```
transport_args primary_args = { .radio_channel = 65, .radio_port = 9, .crtp_buffer = &crtp_buffer_primary[0] };
RCCHECK(rmw_uros_options_set_custom_transport(
    true,
    (void *) &primary_args,
    crazyflie_serial_open,
    crazyflie_serial_close,
    crazyflie_serial_write,
    crazyflie_serial_read,
    rmw_options));

// Wait for agent connection
while(RMW_RET_OK != rmw_uros_ping_agent_options(100, 1, rmw_options))
{
    vTaskDelay(250/portTICK_RATE_MS);
}

rclc_support_init_with_options(&support, 0, NULL, &init_options, &allocator);
```

Figura 4.5 Configuración e inicialización del transporte personalizado

4.3.3. Creación y configuración del nodo micro-ROS

El estado "Inicializacion_Nodo" de la Figura 4.3 se encarga de configurar completamente el nodo ROS crazyflie_node, incluyendo la creación de un publicador en el tópico */drone/pose*, responsable de publicar continuamente la posición actual del Crazyflie, y un suscriptor en el tópico */neighbour_topic*, encargado de recibir comandos externos tal y como se muestra en la Figura 4.6.

```
// create node
rcl_node_t node;
RCCHECK(rclc_node_init_default(&node, "crazyflie_node", "", &support));

// Create publisher 1
rcl_publisher_t pub_pose;
RCCHECK(rclc_publisher_init_best_effort(
    &pub_pose,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(geometry_msgs, msg, Pose),
    "/drone/pose"));

// Create Subscribers
rcl_subscription_t sub_pose;
RCCHECK(rclc_subscription_init_best_effort(
    &sub_pose,
    &node,
    ROSIDL_GET_MSG_TYPE_SUPPORT(std_msgs, msg, Int32),
    "/neighbour_topic"));
```

Figura 4.6 Configuración del nodo micro-ROS

4.3.4. Ejecución del Bucle principal

En el estado "Operación_Normal" de la Figura 4.3, el sistema ejecuta de manera continua las tareas principales del nodo: lee la posición actual del dron y la publica regularmente en el espacio ROS. Paralelamente, verifica de manera constante si se

han recibido comandos externos a través del suscriptor. Dependiendo de si existe o no un comando recibido, el sistema tomará la decisión de entrar en un estado de procesamiento específico o de mantener la publicación rutinaria de la posición actual.

4.3.5. Procesamiento de comandos

Cuando se recibe un comando externo, el sistema entra en este estado específico para gestionar dicho comando. Para evitar condiciones de carrera, se utiliza un mutex que asegura el acceso exclusivo a los datos compartidos. Una vez adquirido el control, el sistema lee el valor del comando y actualiza la posición del dron en consecuencia. Esta nueva posición se publica inmediatamente en el tópico /drone/pose, y posteriormente se libera el mutex para que el ciclo normal del cliente pueda continuar con su ejecución.

El comportamiento del sistema en este estado se describe de forma más clara en la Figura 4.7, que representa una máquina de estados simplificada. El proceso comienza en un estado inicial de exploración, donde la posición X del dron se mantiene constante a la espera de comandos. Al recibir un nuevo mensaje, el sistema evalúa su contenido: si el comando es igual a 0, se interpreta como una instrucción para cambiar al estado de formación, aumentando la posición en X en 20 cm. Si el comando es igual a 1, se considera una orden para continuar explorando, lo que implica un desplazamiento negativo de 20 cm en el eje X. Finalmente, tras aplicar la modificación correspondiente, se publica la nueva posición y el sistema retorna al estado de espera.

Esta lógica de control representa una solución de ejemplo, diseñada para validar el correcto funcionamiento del cliente micro-ROS utilizando un único agente. Permite comprobar que el dron responde adecuadamente a los comandos recibidos y actualiza su estado dentro del ecosistema ROS 2. Sin embargo, en un escenario real, lo esperado sería una comunicación distribuida entre múltiples agentes, donde cada uno comparte su estado y posición con el resto. Esta cooperación permitiría alcanzar objetivos colectivos, como la formación autónoma o la cobertura de un entorno, lo que se corresponde con el verdadero potencial de una arquitectura multiagente basada en ROS 2.

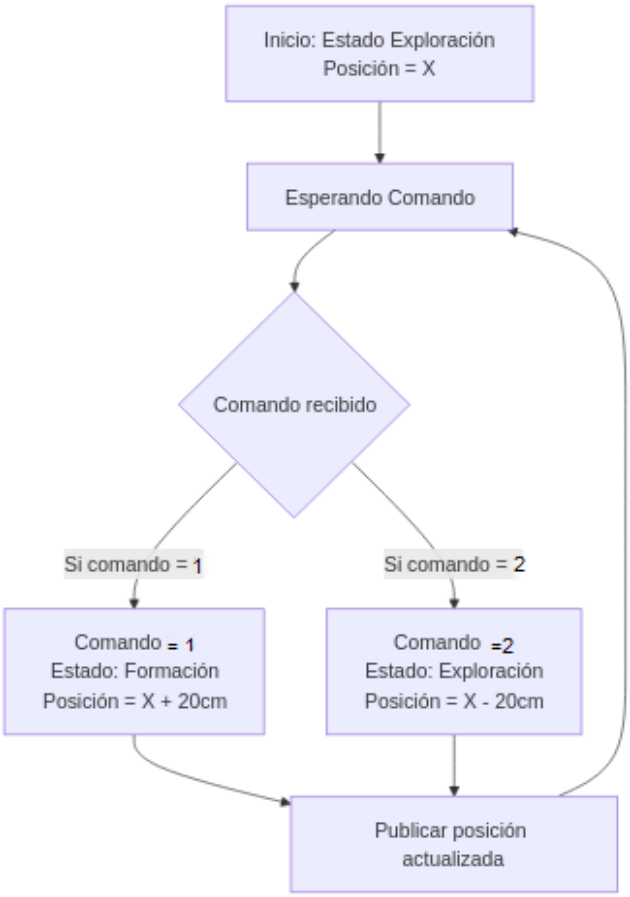

Figura 4.7 Maquina de estados del procesamiento de comandos

4.3.6. Estado de espera

El ultimo bloque no es un estado operacional, sino solo una pausa controlada para regular el ciclo de ejecución del sistema, permitiendo mantener una operación estable y consistente. Al finalizar esta pausa, el sistema regresa automáticamente al estado de operación normal para continuar con el ciclo habitual de publicación y escucha de comandos.

4.4. Integración del sistema completo

Ahora que se ha actualizado el firmware y creado la aplicación micros-ROS, El siguiente paso es configurar este cliente que se ejecutara en el Crazyflie. Para ello se utiliza el comando:

```
$ ros2 run micro_ros_setup configure_firmware.sh crazyflie_demo
```

Este comando permite seleccionar la aplicación crazyflie_demo, que contiene al

cliente desarrollado. Durante esta fase también se configuran parámetros como el tipo de transporte (en este caso, CRTP personalizado), así como las rutas necesarias para la compilación cruzada.

Después de configurar correctamente el entorno, se procede a compilar el firmware mediante compilación cruzada con el comando:

```
$ ros2 run micro_ros_setup build_firmware.sh
```

Este paso genera los binarios específicos para la plataforma Crazyflie, integrando tanto *FreeRTOS* como las bibliotecas de micro-ROS. En este punto pueden aparecer errores relacionados con dependencias o configuración del Makefile, que deben ser revisados para garantizar una compilación exitosa.

Finalmente, una vez generado el firmware, se flashea directamente en la memoria del Crazyflie con:

```
$ ros2 run micro_ros_setup flash_firmware.sh
```

4.5. Problemas encontrados y soluciones aplicadas

Uno de los primeros inconvenientes apareció justo después de ejecutar el comando:

```
$ ros2 run micro_ros_setup create_firmware_ws.sh freertos crazyflie21
```

se detectó que el fichero de configuración de submódulos Git, ubicado en microros_ws/firmware/crazyflie_firmware/.git/config.txt, estaba incompleto. Al intentar compilar, el sistema mostraba errores relacionados con archivos o carpetas que no se encontraban disponibles, lo cual impedía avanzar.

Para resolver este problema, fue necesario editar manualmente dicho archivo e incluir las siguientes líneas correspondientes a los submódulos necesarios:

```
[submodule "vendor/CMSIS"]
    url = https://github.com/ARM-software/CMSIS_5.git
[submodule "vendor/FreeRTOS"]
    url = https://github.com/FreeRTOS/FreeRTOS-Kernel.git
[submodule "vendor/cmock"]
```

```
    url = https://github.com/throwtheswitch/cmock.git
[submodule "vendor/libdw1000"]
    url = https://github.com/bitcraze/libdw1000.git
[submodule "vendor/unity"]
    url = https://github.com/throwtheswitch/unity.git
```

Código 4.7 Fuentes de submódulos de librerías

Estos submódulos representan dependencias externas que el proyecto incorpora para facilitar el desarrollo, depuración y funcionamiento del firmware en un entorno embebido como el Crazyflie. Desde el sistema operativo (FreeRTOS), hasta pruebas automáticas (Unity + CMock), y bibliotecas de hardware específicas (CMSIS y libdw1000).

Un segundo desafío surgió a raíz del cambio de estructura en la nueva versión de la librería CMSIS. El Makefile original del proyecto estaba diseñado para trabajar con la versión antigua (CMSIS 4), por lo que al incluir CMSIS 5, comenzaron a aparecer errores de compilación relacionados con rutas incorrectas o archivos que ya no se encontraban en la misma ubicación.

La solución fue localizar el Makefile responsable de incluir esta librería y modificarlo para que apuntara a las nuevas rutas de CMSIS 5. Este ajuste requirió cierta revisión manual de la estructura del repositorio y entender qué archivos específicos estaban siendo utilizados.

Ambos errores, aunque no complejos, pueden ser bloqueantes para quienes sigan la instalación sin conocimiento previo de estas inconsistencias. Documentarlos fue importante para asegurar una base sólida para futuras ampliaciones del proyecto.

5. Pruebas y resultados

Este capítulo tiene como objetivo verificar que la integración entre el cliente micro-ROS embebido en el Crazyflie y el agente ROS 2 desplegado en Linux funciona de la forma esperada. Para ello, se plantearon una serie de pruebas que permitieran evaluar tanto la comunicación entre nodos como el comportamiento del dron al publicar y recibir mensajes dentro del entorno ROS.

Estas pruebas no solo buscan confirmar que el sistema actualizado compila y se ejecuta sin errores, sino también que cumple con los requisitos funcionales para este proyecto. Entre ellos, destaca la capacidad del Crazyflie para publicar su posición periódicamente en el tópico /drone/pose, así como recibir comandos externos a través del tópico /neighbour_topic y reaccionar de forma adecuada

Esto establece una base sólida para validar que la solución desarrollada puede ser usada para escenarios más complejos, como el control multi-agente o la navegación autónoma.

5.1. Entorno de pruebas

Para validar el correcto funcionamiento del sistema desarrollado, se preparó un entorno de pruebas que combina hardware específico con herramientas del ecosistema ROS 2. El objetivo principal fue verificar tanto la publicación de datos desde el Crazyflie como su capacidad de recibir y procesar comandos externos.

5.1.1. Software de apoyo

Para monitorizar y registrar el comportamiento del sistema se utilizaron dos herramientas principales de ROS 2:

- rosbag: se empleó para grabar los mensajes publicados por el dron en el tópico /drone/pose, con el fin de analizar posteriormente su comportamiento. Esta herramienta resultó clave para demostrar que el Crazyflie actualiza correctamente su posición, incluyendo la componente vertical de empuje (*thrust* en Z) y los desplazamientos laterales (X, Y).

- rqt: permitió visualizar en tiempo real los valores de los tópicos publicados, así como la evolución de los datos recibidos. Al combinar su uso con los datos almacenados en rosbag, se logró una validación más completa del sistema.

5.1.2. Configuración del entorno

Para establecer la comunicación entre el Crazyflie y ROS 2, se siguió una secuencia específica de comandos que garantiza una conexión funcional.

```
$ python3 uros_cf_bridge.py --channel 65 --port 9 --controller
```

Este script establece la conexión inicial con el Crazyflie, especificando el canal de radio y el puerto a utilizar. Además, activa el modo controlador para gestionar la interfaz de comunicación con dron tal y como se ce en la siguiente figura.

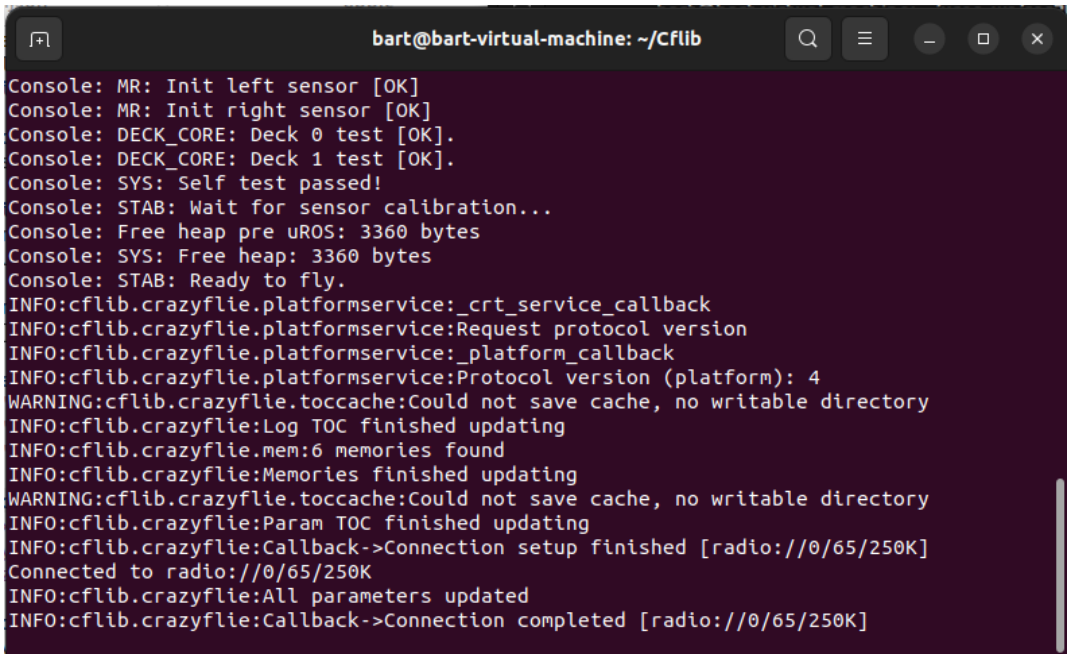

Figura 5.1 Conexión con el Crazyflie

A continuación, en otro terminal se inicia el agente micro-ROS en el entorno ROS 2 Humble. Para ello, se ejecuta:

```
$ ros2 run micro_ros_agent micro_ros_agent serial -f /tmp/uros/port.log
-v6
```

Este comando permite al agente leer el puerto indicado en el archivo port.log, generado por el headless client. El flag -v6 aumenta el nivel de información para facilitar la depuración durante las pruebas como se muestra en la Figura 5.2.

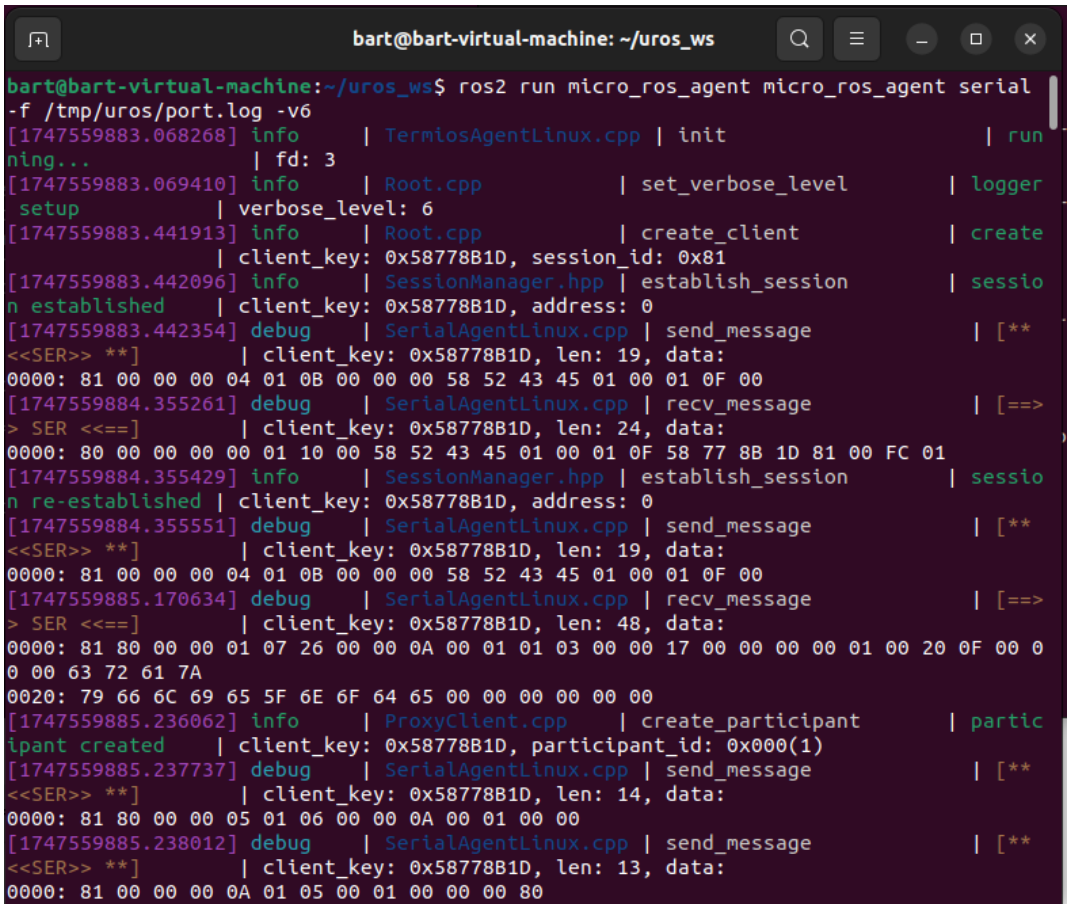

Figura 5.2 Ejecución del agente micro-ROS

Finalmente, en un tercer terminal, una vez establecida la conexión entre el cliente y el agente, se inicia la grabación de datos con rosbag (Figura 5.3):

```
$ ros2 bag record /drone/pose
```

Este comando permite capturar en tiempo real todos los mensajes publicados en el tópico /drone/pose, facilitando posteriormente el análisis y la validación de los datos publicados por el dron.

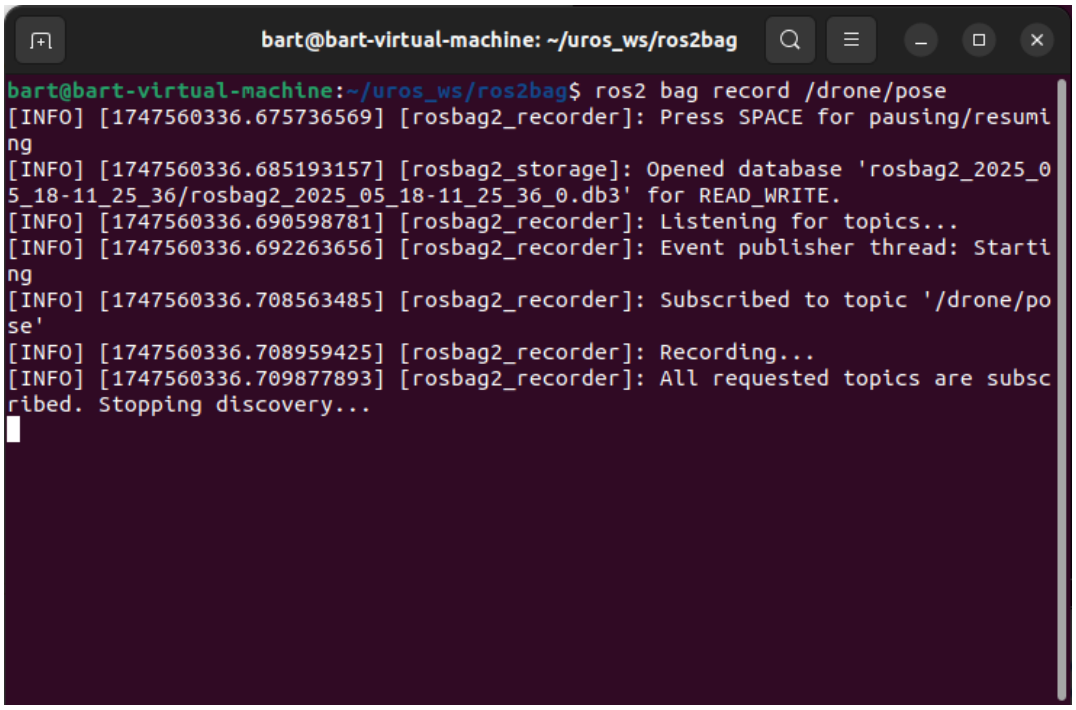

Figura 5.3 Ejecución de la herramienta rosbag

5.2. Publicación de posición: validación con rosbag

Uno de los principales objetivos de este trabajo era verificar que el Crazyflie, una vez actualizado con el cliente micro-ROS, es capaz de publicar su posición en el tópico /drone/pose de forma continua y estable. Para ello, se utilizó la herramienta rosbag, que permite grabar los mensajes publicados en tiempo real y analizarlos posteriormente.

Con la herramienta rosbag, se almacenaron todos los mensajes enviados desde el Crazyflie a través del tópico mencionado. El análisis posterior de la rosbag permitió confirmar que el dron publicaba correctamente las tres componentes de posición: X, Y y Z. En particular, se prestó especial atención a la componente Z, que refleja la intensidad del empuje (*thrust*) generado por los motores, y que es clave para el control vertical del vuelo. En la Figura 5.4 se muestra como el dron sube hasta una altura aproximada de 1.80 metros antes de volver a aterrizar (en el lado derecho de la figura).

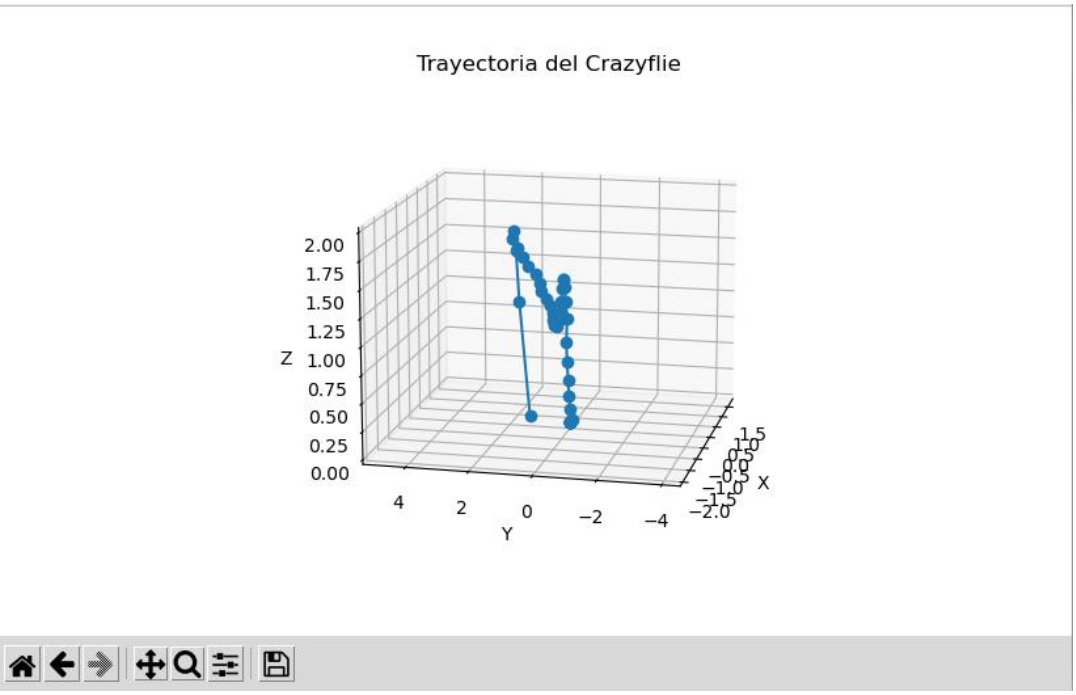

Figura 5.4 Movimiento del dron el eje z

En la Figura 5.5, se observan variaciones claras en las componentes X e Y cuando se aplicaban comandos de desplazamiento lateral desde el entorno ROS o mediante el Gamepad. Estos datos demostraron que el Crazyflie responde correctamente a los comandos de movimiento, y que su estimación de posición es enviada de forma continua al sistema ROS 2.

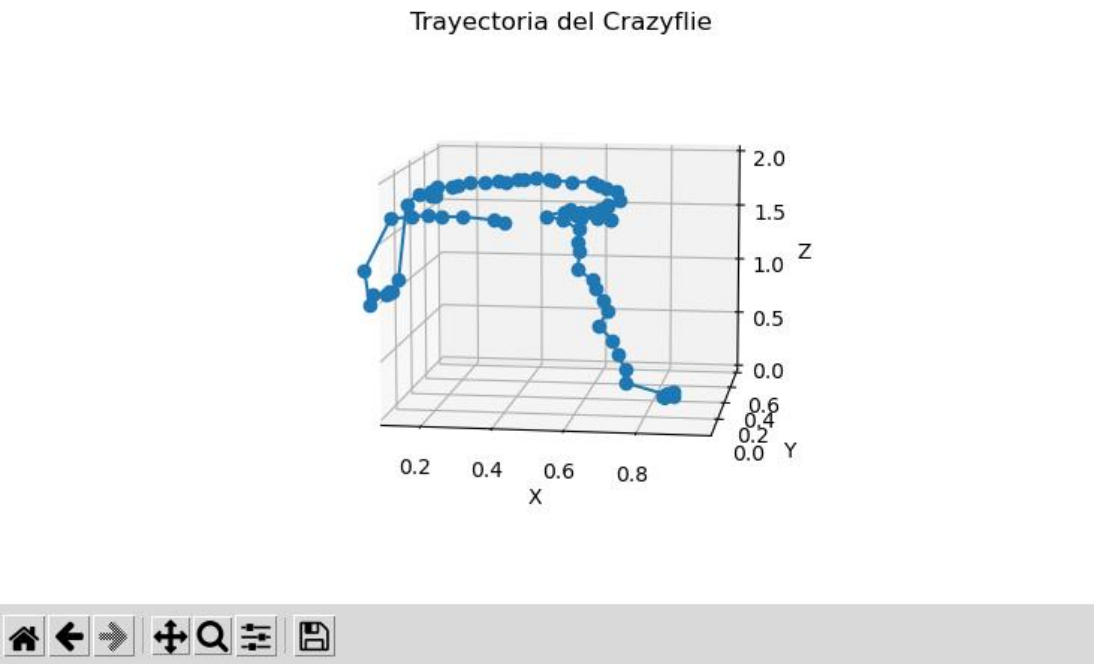

Figura 5.5 Movimiento del dron en los ejes x-y

Durante todo el periodo de esta prueba la comunicación con el dron se ha ejecutado de forma estable y sin interrupciones. Este resultado valida que la actualización del firmware ha sido exitosa en términos de publicación de información relevante hacia ROS 2.

5.3. Visualización de la actividad con rqt

Además de grabar los datos con rosbag, se utilizó la herramienta rqt para visualizar en tiempo real la información publicada por el Crazyflie. Cabe destacar que esta visualización se realizó durante un ensayo distinto al de la grabación previa con rosbag, por lo que es normal que los valores mostrados no coincidan exactamente. Aun así, el objetivo principal de este segundo ensayo fue comprobar de forma inmediata que los mensajes enviados desde el cliente micro-ROS llegaban correctamente al entorno ROS 2 y eran interpretados sin errores.

Para ello, se lanzó el entorno gráfico con el comando:

```
$ rqt
```

Dentro de rqt, se utilizó el plugin rqt_plot para representar gráficamente la evolución de los valores publicados en el tópico /drone/pose. Se monitorizaron en particular las componentes pose.position.x, pose.position.y y pose.position.z, que reflejan el desplazamiento del dron en el espacio tal y como se observa en la siguiente figura.

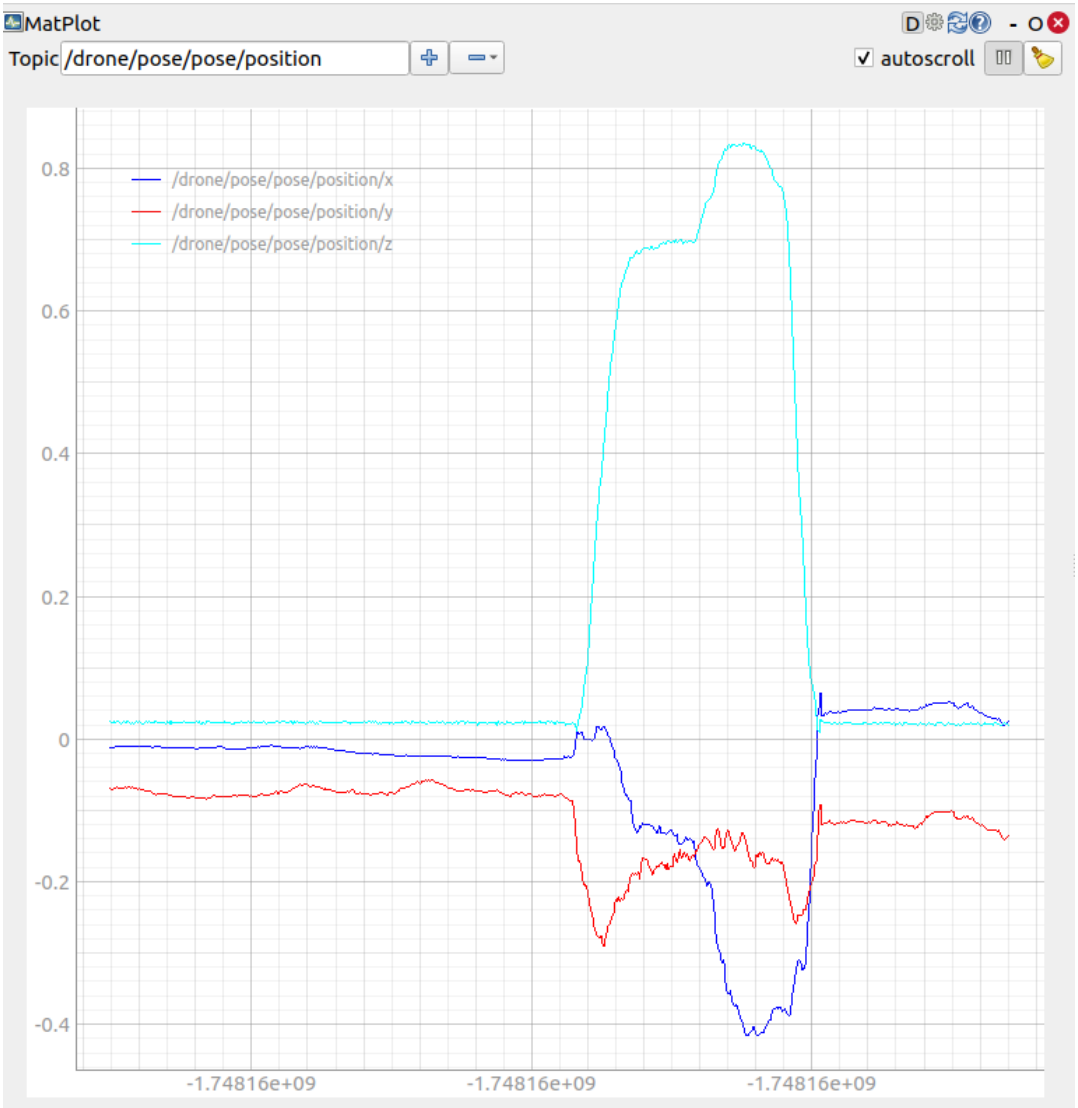

Figura 5.6 Representación del movimiento del dron con rqt

Al observar la evolución de la posición en los tres ejes, se pudo confirmar que los datos respondían de forma coherente a los comandos de control aplicados. Por ejemplo, al generar un desplazamiento en el eje Z (aumentar el empuje), el valor de

pose.position.z aumentaba en tiempo real. Lo mismo ocurría con los desplazamientos laterales sobre los ejes X e Y.

Esta visualización fue especialmente útil durante la fase de ajuste fino, ya que permitía detectar posibles errores en la estimación de la pose o retardos en la publicación.

Gracias a la vista temporal de rqt_plot, se comprobó que la publicación era periódica y no presentaba cortes o saltos inesperados. Este comportamiento estable es indicativo de que tanto el bucle principal del cliente micro-ROS como la conexión con el agente ROS están funcionando correctamente.

En conjunto, el uso de rqt permitió validar de forma visual y rápida el buen funcionamiento del nodo y la correcta integración del Crazyflie en el ecosistema ROS2.

5.4. Comprobación de la suscripción a comandos

Además de publicar su posición en el tópico /drone/pose, el Crazyflie también está diseñado para suscribirse a comandos externos enviados desde otros nodos ROS. Esta funcionalidad permite modificar su comportamiento dinámicamente y validar su rol como nodo suscriptor dentro del sistema ROS 2.

Para esta prueba, se situó el dron en estado de reposo (sin comandos activos) con el objetivo de observar con claridad los cambios provocados exclusivamente por la recepción de un mensaje. A partir de ese estado inicial, se enviaron comandos desde un nodo ROS que publica en el tópico /neighbour_topic, en el cual está suscrito el Crazyflie.
En la Figura 5.7 se representa en una gráfica en tiempo real la posición del dron. Partiendo desde un estado inicial x,y,z ≈ 0,0,0 se publica en el topic /neighbour_topic un mensaje con valor = 1. Tal y como se definió en el capítulo 4, y más concretamente en la Figura 4.7, este valor corresponde a una orden que el cliente micro-ROS interpreta como un comando de desplazamiento positivo en el eje X. Por tanto, al procesar este mensaje, el Crazyflie modifica su posición publicada, aplicando un desplazamiento de +20 cm sobre la coordenada X.

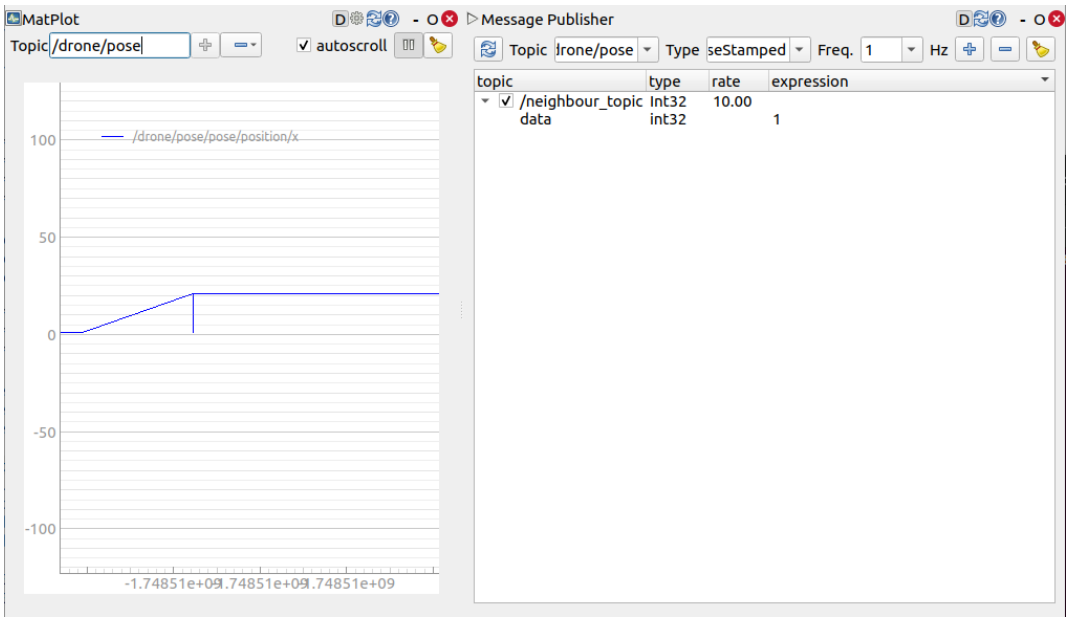

Figura 5.7 Suscripción al neighbour_topic y acción resultada del comando 1

Para la prueba del comando 2, se se ha seguido el mismo proceso anterior cambiando el valor del mensaje a comando = 2. Tal y como se espera, el crazyflie procesa el comando y cambia su posición publicada en el eje X. En este caso sería un desplazamiento negativo de 20 cm de la posición x. La Figura 5.8 presenta el resultado obtenido.

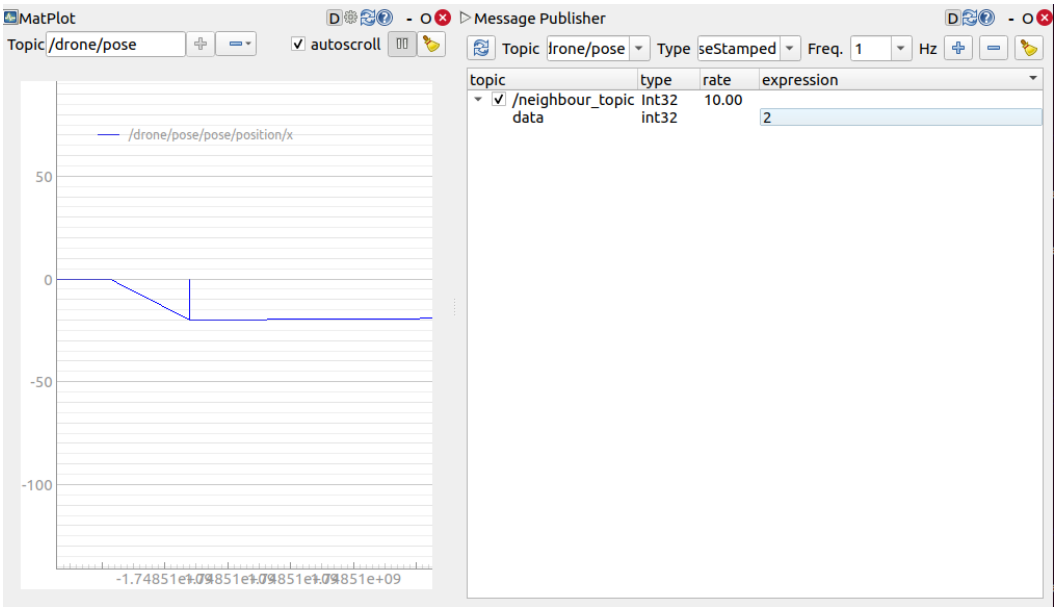

Figura 5.8 Suscripción al /neighbour_topic y acción resultada del comando 2

Desde los dos estados anteriores, si el nodo externo deja de publicar en el /neighbour_topic, el dron vuelve a publicar el estado inicial de su posición tal y como se muestra en la Figura 5.9.

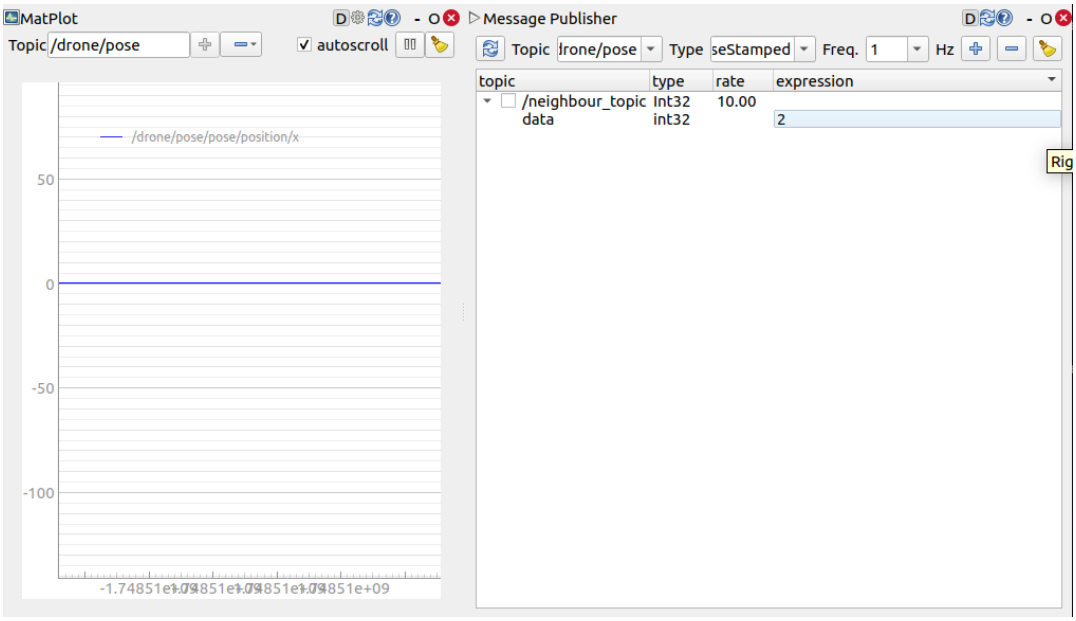

Figura 5.9 Suscripción al /neighbour_topic y acción resultada sin comandos

5.5. Resultados y discusión

Las pruebas realizadas han demostrado que la integración del cliente micro-ROS en el Crazyflie y su comunicación con un agente ROS 2 Humble funciona correctamente, cumpliendo con los objetivos planteados al inicio del proyecto.

En primer lugar, se validó con éxito la capacidad del dron para publicar su posición en tiempo real mediante el tópico /drone/pose, mostrando datos coherentes y estables tanto en rosbag como en rqt. Esto quedó especialmente reflejado en los movimientos sobre el eje Z (thrust) y en las variaciones laterales sobre X e Y, con una publicación continua y sin interrupciones.

En segundo lugar, se comprobó la suscripción a comandos externos a través del tópico /neighbour_topic, tal como se había previsto en el diseño del nodo micro-ROS. Las pruebas mostraron cómo el dron reaccionaba correctamente a los comandos 1 y 2, desplazando su posición en el eje X según lo definido en la máquina de estados (Figura 4.7).

A nivel de comunicación, no se observaron pérdidas de mensajes ni cuelgues del sistema, lo que indica que la arquitectura Agent/Client basada en micro-ROS es

adecuada para escenarios de baja latencia como este. Además, la sincronización entre publicación y recepción no ha generado ningún tipo de errores, y la ejecución sobre FreeRTOS se mantuvo estable durante el proceso de prueba.

Estos resultados sugieren que el Crazyflie puede desempeñar un papel activo como nodo ROS 2 completo, abriendo la puerta a aplicaciones más complejas como la navegación colaborativa, los sistemas multi-agente o el seguimiento de trayectorias. No obstante, también se presentan ciertas limitaciones que podrían abordarse en trabajos futuros, como el soporte para más tipos de comandos, desarrollar una tarea especializada en el tratamiento de los comandos. También, se podría implementar un mecanismo para la reconexión con el agente ROS, en caso de perdido de conexión radio.

En conjunto, la validación funcional ha sido satisfactoria y ha confirmado que la solución propuesta cumple con los requisitos técnicos planteados al inicio del proyecto, aunque deja espacio para mejoras futuras.

6. Conclusiones y Trabajo Futuro

El presente Trabajo Fin de Máster ha logrado cumplir con éxito el objetivo principal planteado: integrar micro-ROS dentro del firmware del Crazyflie y establecer una comunicación estable y funcional con un agente ROS 2 Humble ejecutado en un entorno Linux. Esta integración ha permitido que un dron con recursos muy limitados, como el Crazyflie, se comporte como un nodo ROS 2 completo, capaz de publicar su posición y suscribirse a comandos externos, sentando así las bases para escenarios de mayor complejidad.

A lo largo del proyecto, se ha llevado a cabo una revisión y actualización profunda del firmware del Crazyflie, resolviendo limitaciones heredadas de versiones anteriores, como dependencias obsoletas y estructuras incompatibles. Esta labor incluyó la corrección manual de submódulos, ajustes en los Makefiles y la adaptación a nuevas versiones de librerías como CMSIS. Todo ello ha permitido compilar y ejecutar un cliente micro-ROS completamente funcional en el entorno FreeRTOS del dron.

El sistema implementado ha demostrado en pruebas reales su capacidad para comportarse como un nodo ROS 2 completo: publica su posición en el tópico /drone/pose y reacciona a comandos externos recibidos mediante /neighbour_topic. Herramientas como rosbag y rqt han sido claves para validar estos comportamientos, permitiendo verificar en detalle la evolución de la posición del dron y su respuesta a diferentes escenarios de prueba.

Además, la estabilidad del sistema, la ausencia de errores de comunicación y la correcta ejecución sobre FreeRTOS confirman que la arquitectura basada en micro-ROS es una solución válida para integrar micro-UAVs como el Crazyflie en entornos ROS 2 distribuidos.

Desde el punto de vista funcional, el sistema es capaz de:

- Publicar datos de posición en tiempo real.
- Procesar comandos externos mediante suscripción.
- Reaccionar dinámicamente a estos comandos, modificando su comportamiento según una máquina de estados interna.
- Operar en un entorno ROS 2 sin errores de sincronización.

Limitaciones actuales

Aunque los resultados han sido satisfactorios, también se han identificado algunas limitaciones que abren oportunidades para mejorar:

- El procesamiento de comandos está actualmente gestionado desde el bucle principal, lo cual puede limitar la escalabilidad si se aumentan las tareas simultáneas.
- La reconexión con el agente ROS no está automatizada: si se pierde la conexión, es necesario reiniciar el sistema manualmente.
- La suscripción a comandos está limitada a una estructura muy básica (dos comandos), sin validación ni tratamiento de errores.

Líneas futuras

Este trabajo, aunque no propone un caso de uso 100% completo, sienta una base sólida para futuras líneas de investigación, entre las que destacan:

Desarrollo de una tarea dedicada al tratamiento de comandos, que permita liberar el bucle principal y facilitar futuras ampliaciones del comportamiento.

Implementación de un sistema de reconexión automática con el agente ROS, útil para mantener robustez en escenarios inalámbricos inestables.

Ampliación del protocolo de comandos, incorporando más acciones y parámetros que permitan modificar el comportamiento del dron en tiempo real de forma más dinámica.

Integración de sensores adicionales (como el Multiranger o el AI-deck) para dotar al sistema de capacidades de percepción más avanzadas y procesamiento a bordo.

Pruebas en un entorno multi-dron, permitiendo validar la escalabilidad del sistema en contextos colaborativos o de enjambre.

En definitiva, se ha demostrado que es posible llevar las capacidades de ROS 2 hasta dispositivos embebidos con recursos muy limitados, abriendo nuevas posibilidades para la robótica distribuida y el control cooperativo desde plataformas ligeras como el Crazyflie.“”

7. Referencias

[1] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics,* vol. 7, nº 66, 8 8 2022.

[2] W. Giernacki, M. Skwierczyński, W. Witwicki, P. Wroński, P. Kozierski, "Crazyflie 2.0 quadrotor as a platform for research and education in robotics and control engineering," de *22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, 2017.

[3] K. N. McGuire, C. De Wagter, K. Tuyls, H. J. Kappen, G. C. H. E. de Croon, G. Schoonewille et al., "Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment," *Science Robotics,* vol. 4, nº 35, 2019.

[4] A. Koubaa, Prince Sultan University, College of Computer Science and Informat, Robot Operating System (ROS): The Complete Reference, Springer, 2018.

[5] F. J. Mañas-Álvarez, M. Guinaldo, R. Dormido, S. Dormido, "Robotic park: Multi-agent platform for teaching control and robotics," *IEEE Access,* vol. 11, pp. 34899-34911, 2023.

[6] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler y A. Y. Ng, "ROS: an open-source Robot Operating System," *ICRA Workshop on Open Source Software,* vol. 3, nº 3.2, p. 5, 2009.

[7] "The Origin Story of ROS," 31 10 2017. [En línea]. Available: https://spectrum.ieee.org/the-origin-story-of-ros-the-linux-of-robotics. [Último acceso: 5 4 2025].

[8] "TheConstruct," [En línea]. Available: https://x.com/_TheConstruct_/status/1168418352802516992. [Último acceso: 27 04 2025].

[9] "swri," 10 8 2022. [En línea]. Available: https://www.swri.org/markets/industrial-robotics-automation/blog/the-ros-1-vs-ros-2-transition. [Último acceso: 5 4 2025].

[10] D. Portugal, R. P. Rocha, J. P. Castilho, "Inquiring the robot operating system community on the state of adoption of the ROS 2 robotics middleware," *International Journal of Intelligent Robotics and Applications,* 2024.

[11] J. Park, R. Delgado, B. W. Choi, "Real-time characteristics of ROS 2.0 in multiagent robot systems: An empirical study," *IEEE Access,* vol. 8, p. 154637–154651, 2020.

[12] S. Hoher, F. Weisshardt, U. Eger, U. Reiser, "ISG-virtuosM – das Gateway zwischen industrieller Echtzeit-Steuerungstechnik und dem Open Source Roboterbetriebssystem ROS," de *SPS IPC Drives 2014: Elektrische Automatisierung, Systeme und Komponenten. Internationale Fachmesse und Kongress*, Núremberg, Alemania, 2014.

[13] S. Edwards, C. Lewis, "ROS-Industrial: Applying the Robot Operating System (ROS) to Industrial Applications," de *IEEE International Conference on Robotics and Automation (ICRA) – ECHORD Workshop*, 2012.

[14] V. Mayoral-Vilches, R. White, G. Caiazza, M. Arguedas, "Sros2: Usable cyber security tools for ros 2," *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),* p. 11253–11259, 2022.

[15] “micro-ROS,” [En línea]. Available: https://micro.ros.org/docs/overview/features/. [Último acceso: 5 4 2025].

[16] A. Malki, T. Kołcon, M. Maciaś, “Smart Warehouse as an Example of Micro-ROS Application,” de *Automation 2022: New Solutions and Technologies for Automation, Robotics and Measurement Techniques*, Cham, 2022.

[17] A. Sharma, R. Singh, "A Comprehensive Review of Recent Research Trends on Unmanned Aerial Vehicles (UAVs)," *Systems,* vol. 11, no. 8, p. 400, 2023.

[18] O. Garcia-Salazar, A. Sanchez-Orta, A. J. Muñoz-Vazquez, Advances and Applications in Unmanned Aerial Vehicles, https://doi.org/10.3390/books978-3-7258-3125-8: Machines, 2025.

[19] R. Y. Brogaard, E. Boukas, “Autonomous GPU-based UAS for inspection of confined spaces,” *Robotics and Autonomous Systems,* vol. 172, nº https://doi.org/10.1016/j.robot.2023.104590, p. 104590, 2023.

[20] L. Mingyang, Z. Yibo, H. Chao, H. Hailong, “Unmanned Aerial Vehicles for Search and Rescue: A Survey,” *Remote Sensing, MDPI,* vol. 15, nº 13, 2023.

[21] D. Palossi, A. Loquercio, F. Conti, E. Flamand, D. Scaramuzza, L. Benini, “Ultra-low-power deep learning-powered autonomous nano drones,” *Nature Machine Intelligence,* vol. abs/1805.01831, 2018.

[22] “Bitcraze,” [En línea]. Available: https://www.bitcraze.io/. [Último acceso: 10 05 2024].

[23] “Docs Ros,” [En línea]. Available: https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debs.html. [Último acceso: 2025 04 13].

[24] “microrosorg,” [En línea]. Available: https://micro.ros.org/docs/tutorials/core/first_application_linux/. [Último acceso: 18 04 2025].

[25] “Github,” 4 2020. [En línea]. Available: https://github.com/micro-ROS/micro-ROS_crazyflie_demo. [Último acceso: 5 4 2025].

[26] “micro-ROS_Documentation,” [En línea]. Available: https://micro.ros.org/docs/tutorials/core/first_application_linux/. [Último acceso: 18 04 2025].

[27] Daniel M. Mihalache, Paul T. Hristea, “Micro UAV Swarm for Industrial Applications in Indoor Environment – A Systematic Literature Review,” *Logistics Research, Springer,* vol. 16, nº 1, 2023.

[28] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler y A. Y. Ng, “ROS: an open-source Robot Operating System,” *ICRA Workshop on Open Source Software,* vol. 3, nº 3.2, p. 5, 2009.

Apéndice A

Presupuesto del proyecto

El presente apéndice detalla una estimación del presupuesto necesario para llevar a cabo este Trabajo Fin de Máster. Se incluyen tanto los costes de hardware como el software utilizado, además del tiempo invertido estimado para el desarrollo.

A.1. Coste del material

Elemento	Precio unitario ($)	Cantidad	Total ($)
Crazyflie 2.1	285,00	1	285,00
Flow Deck v2	65,00	1	65,00
Crazyradio PA	45,00	1	45,00
Multi-ranger deck	113,00	1	113
PC personal (uso preexistente)	—	1	0,00
Máquina virtual (VMware Player)	Gratuito	1	0,00
Licencias de software (ROS, etc.)	Gratuito / Open source		0,00

Total estimado hardware/software: 508,00 $ ≈ 434,00 €

A.2. Estimación de tiempo de trabajo

Actividad	Horas estimadas
Estudio previo de micro-ROS y ROS 2	25 h
Preparación del entorno y herramientas	20 h
Análisis y adaptación del firmware	70 h
Desarrollo del cliente micro-ROS	65 h
Pruebas, validación y ajustes	30 h
Redacción de la memoria	60 h
Revisión y correcciones finales	15 h

Total estimado de trabajo: 285 horas

Apéndice B

Manual de uso del sistema

Este apéndice sirve como guía básica para reproducir el entorno de pruebas y ejecutar el sistema desarrollado.

B.1. Requisitos previos

- Ubuntu 22.04 LTS (recomendado en máquina virtual)
- ROS 2 Humble instalado correctamente
- micro-ROS Agent y micro_ros_setup
- Crazyflie 2.1 con Flow Deck
- Crazyradio PA
- Python ≥ 3.8
- Cliente `cfclient` y `cflib` instalados

B.2. Pasos para ejecutar el sistema

1. **Conectar el Crazyradio PA** al PC.
2. **Conectar el mando** al PC
3. **Activar el entorno ROS:**

```
$ source /opt/ros/humble/setup.bash
$ cd ~/microros_ws
$ source install/local_setup.bash
```

4. **Iniciar el cliente del Crazyflie:**

```
$ python3 uros_cf_bridge.py --channel 65 --port 9 --controller
```

5. **En otro terminal, iniciar el agente micro-ROS:**
   antes de ejecutar este comando, hay que esperar a que el Crazyflie comunique que este correctamente conectado

```
$ ros2 run micro_ros_agent micro_ros_agent serial -f
/tmp/uros/port.log -v6
```

B.3. Notas adicionales

- Para actualizar el firmware del Crazyflie, compilar y flashear siguiendo el proceso descrito en el Capítulo 4.
- Se recomienda usar `cfclient` antes de iniciar micro-ROS para verificar que el Crazyflie está operativo.