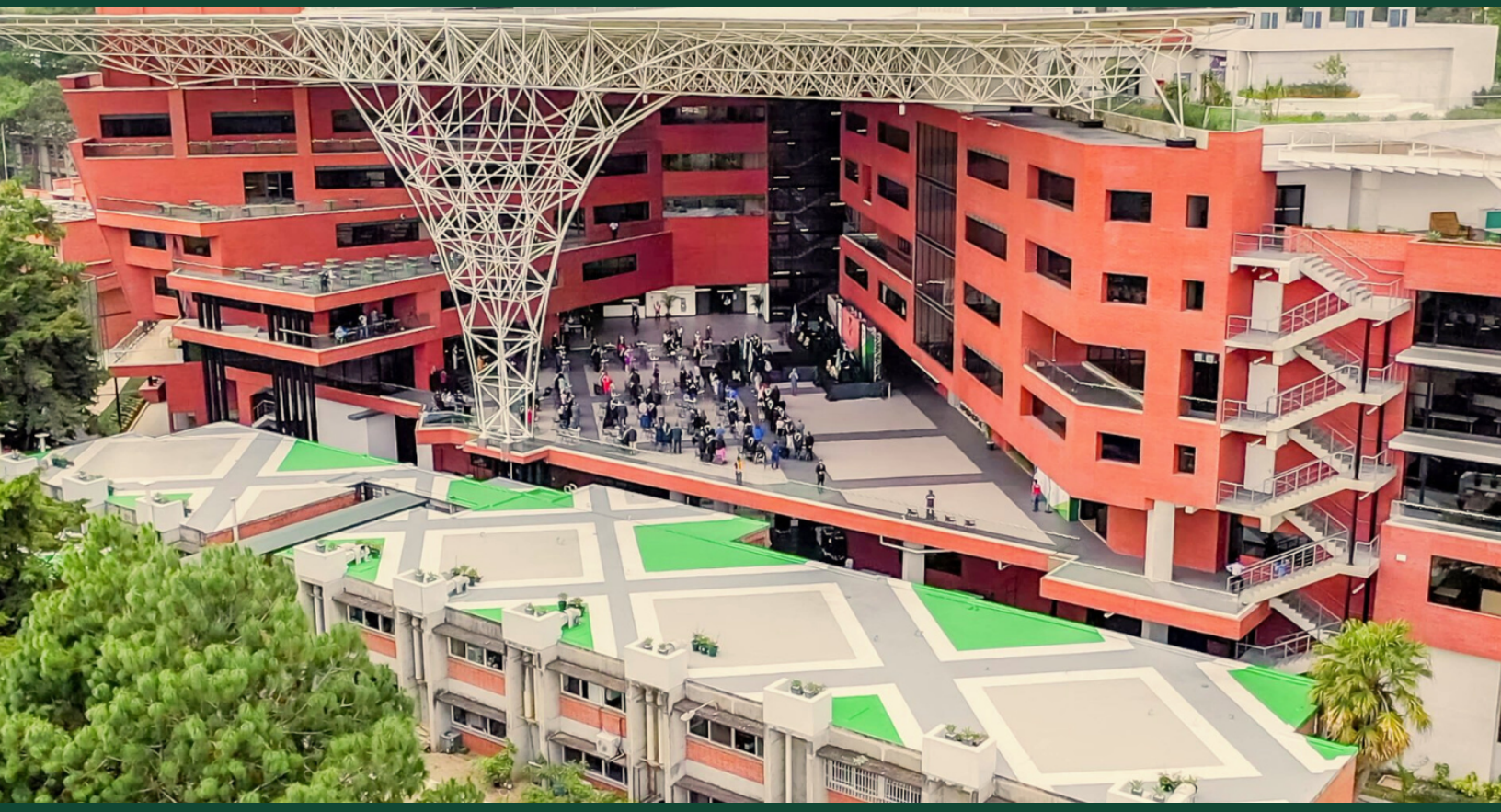
Implementación de infraestructura para el control de enjambres robóticos con ROS2 y captura de movimiento dentro del ecosistema Robotat

Thomas Jabes David Solis López



UVG | UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Implementación de infraestructura para el control de enjambres robóticos con ROS2 y captura de movimiento dentro del ecosistema Robotat

Trabajo de graduación presentado por Thomas Jabes David Solis López
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2025

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Implementación de infraestructura para el control de enjambres robóticos con ROS2 y captura de movimiento dentro del ecosistema Robotat

Trabajo de graduación presentado por Thomas Jabes David Solis López
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2025

Vo.Bo.:

(f) _____

M.Sc. Miguel Enrique Zea Arenales

(f) _____

M.Sc. Carlos Alberto Esquit Hernández

Fecha de aprobación: Guatemala, 28 de noviembre de 2025.

Prefacio

Agradezco profundamente a Dios por su guía y fortaleza en cada etapa de este proceso. Extiendo mi gratitud a mi familia, especialmente a mis padres, Jairo Solis y Elvia López, por su apoyo incondicional y por estar presentes en los momentos más significativos de mi vida. Asimismo, agradezco a mis hermanos, Diederich Solis y Katherin Solis, por su compañía, motivación y respaldo constante a lo largo de este camino académico.

Deseo también expresar mi sincero agradecimiento a mi asesor, MSc. Miguel Zea, por su acompañamiento, orientación y apoyo durante el desarrollo de este trabajo de graduación.

Finalmente, extiendo mi agradecimiento a mis amigos cercanos, quienes me acompañaron a lo largo de esta etapa tan importante de mi vida, brindándome su apoyo, comprensión y amistad.

Índice general

Índice de figuras

Índice de cuadros

Resumen

Ante la ausencia de una infraestructura moderna y robusta para experimentación multirrobot, este trabajo desarrolló una solución integrada de hardware y software basada en captura de movimiento y ROS2 dentro del ecosistema Robotat. Se configuró un servidor con Ubuntu y ROS2, se incorporó MOCAP4ROS2 para la transmisión directa de poses desde OptiTrack y se habilitó compatibilidad con CrazySwarm2 mediante entornos reproducibles en Docker. La evaluación comparativa evidenció una mejora sustancial en el desempeño: MOCAP4ROS2 alcanzó una latencia media de 37.9 ms frente a los 344.1 ms del flujo tradicional de Robotat, y sostuvo una frecuencia cercana a 100 Hz frente a los $\sim$ 3 Hz del sistema previo. Finalmente, se validó el control distribuido de un enjambre de nueve robots Pololu 3Pi+ siguiendo trayectorias hacia múltiples metas, demostrando que la infraestructura propuesta es adecuada para experimentos avanzados con enjambres de robots móviles y drones.

Palabras clave: captura de movimiento, ROS2, enjambres robóticos, Crazyswarm2, sistemas en tiempo real.

Abstract

In the absence of a modern and robust infrastructure for multi-robot experimentation, this work developed an integrated hardware and software solution based on motion capture and ROS2 within the Robotat ecosystem. A dedicated server running Ubuntu and ROS2 was configured, MOCAP4ROS2 was incorporated to enable direct pose streaming from the OptiTrack system, and compatibility with CrazySwarm2 was achieved through reproducible Docker-based environments. Comparative evaluation demonstrated a substantial performance improvement: MOCAP4ROS2 achieved an average latency of 37.9 ms compared to 344.1 ms in the traditional Robotat pipeline, and maintained a publication frequency close to 100 Hz versus the $\sim$ 3 Hz of the previous system. Finally, distributed control of a swarm of nine Pololu 3Pi+ robots was validated through multi-goal trajectory experiments, showing that the proposed infrastructure is suitable for advanced research involving mobile robot swarms and aerial drone systems.

Keywords: motion capture, ROS2, robotic swarms, Crazyswarm2, real-time systems.

CAPÍTULO 1

Introducción

La robótica moderna avanza hacia sistemas cada vez más autónomos, distribuidos y colaborativos, donde la captura de movimiento (*motion capture*) desempeña un papel fundamental al proporcionar retroalimentación precisa de posición y orientación para algoritmos de control en tiempo real. En este contexto, el presente trabajo se enmarca en el área de la robótica de enjambre y los sistemas basados en ROS2, aprovechando la infraestructura del laboratorio Robotat y su sistema de cámaras OptiTrack. El objetivo principal fue integrar el paquete MOCAP4ROS2 con el flujo de captura existente, habilitando la publicación directa de poses en ROS2 y preparando la infraestructura para su uso con plataformas como Crazyswarm2 y los robots móviles Pololu 3Pi+.

En cuanto a su delimitación, el proyecto se centró en la preparación del servidor y la infraestructura necesaria incluyendo la instalación de Ubuntu, ROS2 Humble y entornos reproducibles mediante Docker con el fin de habilitar un flujo de trabajo estable y compatible con Crazyswarm2. A partir de ello, se realizó una evaluación experimental del rendimiento de MOCAP4ROS2 frente al flujo tradicional de Robotat basado en MATLAB y JSON, analizando métricas críticas como la latencia y la frecuencia de publicación para determinar su idoneidad en aplicaciones de control en tiempo real. Finalmente, se validó la integración con agentes robóticos mediante pruebas de control punto a punto y experimentos multirrobot empleando plataformas Pololu 3Pi+, lo que permitió verificar la estabilidad del sistema al operar múltiples agentes de manera simultánea.

CAPÍTULO 2

Antecedentes

El desarrollo de plataformas para la experimentación con drones en la Universidad del Valle de Guatemala inició con el trabajo de Francis Sanabria[1], quien en su proyecto de graduación implementó una solución enfocada en el control de orientación del dron Crazyflie 2.0. Su aporte consistió en el diseño de una plataforma física de pruebas, una guía de instalación y configuración del entorno, así como el desarrollo de una interfaz gráfica en Python para el control del dron mediante la computadora. La interfaz permitía modificar parámetros de los controladores de actitud (PID y LQR), visualizar datos del codificador rotacional de la plataforma y cargar nuevos controladores al firmware del dron.

Además, Sanabria elaboró manuales de operación, recuperación del entorno de trabajo, y desarrolló dos guías de laboratorio aplicadas a cursos de Sistemas de Control, donde se evaluaron en la práctica los resultados del sistema implementado. Aunque el trabajo se centró en el uso individual del Crazyflie 2.0 y no en la coordinación de múltiples agentes, su proyecto fue clave para establecer las bases técnicas del uso de estos drones en entornos académicos, permitiendo su adopción posterior en investigaciones más complejas como los sistemas de enjambre.

Posteriormente, como parte del trabajo de Camilo Perafán[2], se consolidó el desarrollo del ecosistema Robotat, un entorno de experimentación robótica ubicado en el laboratorio de Robótica CIT-116 de la Universidad del Valle de Guatemala. Este ecosistema se diseñó para permitir la integración de múltiples agentes con sistemas de captura de movimiento, estableciendo una red Wi-Fi dedicada y una infraestructura basada en el sistema OptiTrack, que proporciona mediciones precisas de posición y orientación. El sistema se compone de seis cámaras de captura, un switch de red y un protocolo de comunicación basado en MQTT, que permite la transmisión eficiente de datos entre los agentes y el servidor de procesamiento. Esta infraestructura se convirtió en un componente clave para el desarrollo posterior de proyectos multiagente dentro del laboratorio.

En una etapa posterior, Diego Mencos desarrolló un proyecto complementario que buscó aprovechar la infraestructura del laboratorio Robotat para integrar ROS2 en un robot móvil terrestre. Su objetivo fue fusionar múltiples fuentes de posicionamiento, incluyendo módulos

DWM1001, sensores LiDAR y el sistema de captura de movimiento OptiTrack[3]. La implementación permitió establecer comunicación con el software Motive mediante el protocolo NatNet, validando el funcionamiento del sistema en ROS2 bajo condiciones reales. No obstante, el enfoque se limitó a un solo agente terrestre y no consideró aplicaciones multiagente o aéreas. Tampoco se empleó el paquete especializado MOCAP4ROS2, lo cual restringió la compatibilidad con herramientas dentro del ecosistema ROS2.

A partir de la necesidad de estandarizar la integración de sistemas de captura de movimiento dentro del ecosistema ROS2, surge el proyecto de código abierto MOCAP4ROS2, disponible en la plataforma GitHub[4]. Esta herramienta proporciona una interfaz común y modular para conectar diversos sistemas de captura, como OptiTrack, Vicon y Qualisys, mediante controladores dedicados compatibles con ROS2.

A diferencia de trabajos previos realizados en el laboratorio, como el de Diego Mencos que utilizó una conexión directa basada en NatNet, MOCAP4ROS2 ofrece una solución estructurada, mantenible y alineada con los estándares actuales del ecosistema ROS2. Además, incorpora funcionalidades para visualización, simulación en Gazebo y monitoreo en tiempo real, convirtiéndose en un recurso clave para futuras aplicaciones multiagente, tanto aéreas como terrestres.

Finalmente, Julio Ávila y Brandon Garrido desarrollaron dos proyectos complementarios orientados a implementar un sistema funcional de control de enjambres aéreos con drones Crazyflie 2.1[5], [6]. Trabajaron sobre Ubuntu 22.04 LTS con ROS2, integrando el framework CrazySwarm2 y el sistema de captura de movimiento OptiTrack dentro del ecosistema Robotat. Entre sus principales logros se encuentra la conexión entre el sistema de captura y los drones, así como la ejecución de vuelos físicos controlados. Sin embargo, enfrentaron varias limitaciones técnicas: el sistema solo logró controlar entre tres y cuatro agentes simultáneamente y toda la infraestructura se ejecutaba de forma local en una laptop portátil. Esta elección de hardware, en lugar de un servidor fijo, afectó la estabilidad, escalabilidad y reutilización del sistema. Asimismo, no se contó con una interfaz amigable ni una documentación modular que facilitara su uso por parte de otros usuarios dentro del laboratorio.

CAPÍTULO 3

Justificación

El uso de sistemas de captura de movimiento ha demostrado ser fundamental para el control preciso de agentes móviles en robótica. En el contexto del laboratorio Robotat de la Universidad del Valle de Guatemala, se cuenta con una infraestructura equipada con el sistema OptiTrack, el cual ofrece alta precisión en el seguimiento de cuerpos rígidos. Sin embargo, la integración completa de este sistema con ROS2 ha sido limitada en trabajos anteriores debido a la falta de herramientas compatibles con las últimas versiones del ecosistema.

Este proyecto busca consolidar una infraestructura estable, actualizada y reutilizable que permita el uso de OptiTrack dentro de ROS2, tanto para aplicaciones individuales como para sistemas de enjambres de agentes robóticos. A través de la exploración e implementación del paquete MOCAP4ROS2, se pretende establecer una conexión estandarizada entre el sistema de captura y cualquier agente que opere dentro del entorno ROS2, facilitando la recolección de datos y el control en tiempo real.

Además, no se limitará exclusivamente al uso de drones Crazyflie 2.1, sino que también contemplará la posibilidad de emplear otros agentes como los robots móviles Pololu 3PI+, permitiendo así que la plataforma sea flexible y adaptable a distintos tipos de investigaciones y prácticas académicas. La infraestructura quedará implementada en un servidor fijo del laboratorio, lo cual permitirá su reutilización por parte de estudiantes e investigadores sin necesidad de configuraciones adicionales, facilitando el desarrollo de futuras aplicaciones en robótica multiagente y control distribuido.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Establecer una infraestructura completa de hardware y software, basada en un sistema de captura de movimiento y el uso de ROS2, para el control de enjambres de agentes robóticos dentro del ecosistema Robotat.

4.2. Objetivos específicos

- Implementar el uso de Crazyswarm2 en ROS2 en un servidor fijo con Ubuntu 22.04, para su funcionamiento dentro del ecosistema Robotat.
- Explorar y evaluar el paquete MOCAP4ROS2 para su integración con el sistema de captura de movimiento y su aplicación en experimentos con agentes robóticos y enjambres de los mismos.
- Desarrollar y validar el control simultáneo de al menos diez agentes robóticos.

CAPÍTULO 5

Alcance

Este trabajo abarcó la instalación del servidor fijo del laboratorio, la integración de ROS2, Docker y MOCAP4ROS2, y la validación experimental del sistema de captura con robots Pololu 3Pi+. Se evaluó el rendimiento de MOCAP4ROS2 frente al flujo tradicional de Robotat y se preparó la infraestructura necesaria para desplegar Crazyswarm2 dentro del ecosistema del laboratorio.

El proyecto alcanzó la transmisión en tiempo real de poses, la ejecución de pruebas con hasta nueve agentes móviles y la preparación del entorno para futuros trabajos que involucren enjambres mixtos de robots terrestres y aéreos.

Entre las limitaciones encontradas se incluyen el retraso en la llegada del servidor y la dependencia directa de dos trabajos de graduación paralelos: *"Implementación de un enjambre de drones Crazyflie 2.1 mediante Crazyswarm2 y ROS2 dentro del ecosistema Robotat"* de Christian Cruz, y *"Desarrollo de herramientas de software para el control y monitoreo de agentes robóticos Pololu 3Pi+ mediante ROS2 dentro del ecosistema Robotat"* de Bryan Ortiz. Ambos proyectos eran necesarios para completar la integración aérea y el control total de los robots terrestres.

Finalmente, la necesidad de cambiar repetidamente el modo de operación del sistema Robotat para habilitar la transmisión directa vía NatNet restringió el tiempo efectivo de experimentación, limitando el alcance a pruebas preliminares y a la validación funcional de la infraestructura diseñada.

CAPÍTULO 6

Marco teórico

El presente marco teórico reúne los conceptos y tecnologías fundamentales que sustentan el desarrollo de este proyecto. La integración de sistemas de captura de movimiento con plataformas robóticas exige comprender tanto la infraestructura de comunicación del ecosistema ROS2 como los mecanismos que permiten ejecutar aplicaciones de forma portátil y reproducible mediante contenedores Docker. Asimismo, debido a que el flujo de datos depende del sistema OptiTrack disponible en el laboratorio, se describen sus componentes principales incluyendo el software *Motive* y el protocolo NatNet que intervienen directamente en la generación y transmisión de información de posición y orientación. Finalmente, se aborda MOCAP4ROS2 como la herramienta que permite estandarizar esta comunicación dentro de ROS2, facilitando su uso en aplicaciones de control robótico. En conjunto, estos elementos proporcionan el marco conceptual necesario para comprender las decisiones técnicas adoptadas y la metodología implementada durante el desarrollo del proyecto.

6.1. ROS2

El Robot Operating System (ROS) es un framework de código abierto diseñado para facilitar el desarrollo de aplicaciones robóticas. Aunque no es un sistema operativo en el sentido tradicional, provee una infraestructura de comunicación y herramientas que permiten la interacción entre distintos módulos de software y hardware en un sistema robótico.

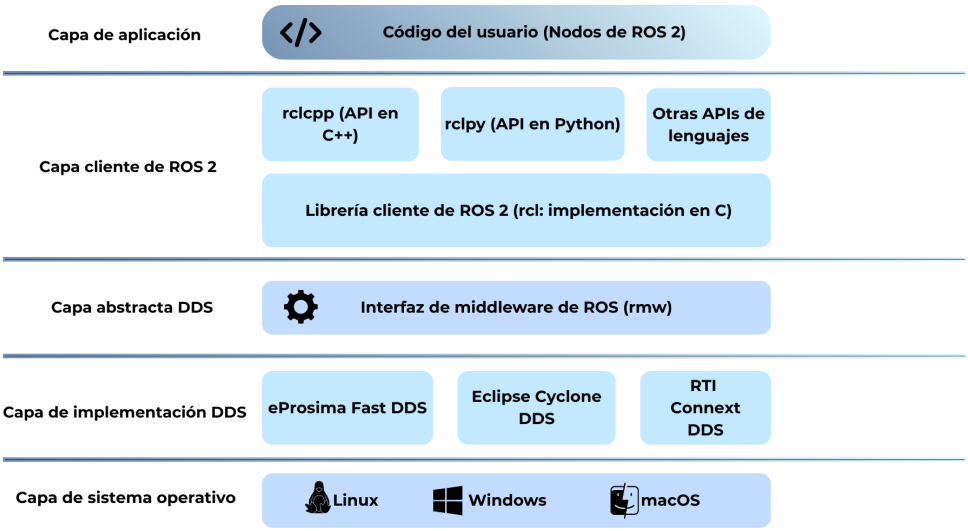
Desde su primera versión en 2007, conocida como ROS1, se consolidó como un estándar en investigación y desarrollo robótico. Sin embargo, con el tiempo surgieron limitaciones en cuanto a escalabilidad, seguridad, soporte en tiempo real y comunicación distribuida. Estas necesidades dieron lugar al desarrollo de ROS2, que introduce mejoras significativas como el uso del estándar DDS (Data Distribution Service) para la comunicación, mayor soporte para sistemas embebidos, capacidad de ejecución en entornos industriales y mejor integración con tecnologías modernas como contenedores y sistemas distribuidos [7].

ROS2 se basa en una arquitectura modular que facilita la comunicación y coordinación

entre diferentes componentes de un sistema robótico, como se muestra en la Figura 1. Esta arquitectura está organizada alrededor de conceptos principales:

- **Nodos** (*nodes*): es un proceso independiente que ejecuta tareas específicas dentro del sistema robótico. Se comunica con otros nodos mediante mensajes a través de tópicos, servicios y acciones y puede configurarse con parámetros propios. Son los bloques fundamentales de una aplicación en ROS2 y permiten organizar el sistema de forma modular y distribuida.
- **Tópicos** (*topics*): es un canal de comunicación que conecta nodos en ROS2 para el intercambio de información. Funciona bajo el modelo publicador–suscriptor, en el cual un nodo publicador envía mensajes a un tópico y uno o varios nodos suscriptores los reciben. Varios nodos pueden publicar en un mismo tópico y a la vez, varios pueden suscribirse a él. Los tópicos son especialmente útiles para transmitir datos que cambian de manera continua, como estados del robot o información proveniente de sensores.
- **Servicios** (*services*): proporcionan un modelo de comunicación sincrónica entre nodos bajo la lógica de cliente–servidor. A diferencia de los tópicos, que transmiten datos de forma continua, los servicios permiten realizar solicitudes puntuales y recibir respuestas, como encender un sensor o consultar el estado de un dispositivo.
- **Acciones** (*actions*): es un mecanismo de comunicación para tareas de larga duración, basado en el modelo cliente–servidor. Permite enviar una meta (goal), recibir retroalimentación durante la ejecución y obtener un resultado final al completarse.
- **Parámetros** (*Parameters*): son valores configurables que definen el comportamiento de un nodo sin necesidad de modificar su código. Se emplean, por ejemplo, para ajustar la ganancia de un controlador o definir la frecuencia de muestreo de un sensor.
- **Archivos de lanzamiento** (*Launch files*): permiten iniciar múltiples nodos y configurar sus parámetros de forma simultánea mediante un solo archivo. Esto simplifica la ejecución de sistemas complejos, donde se requiere poner en marcha varios módulos interconectados.

Figura 1. Arquitectura en capas de ROS2



Nota. Elaboración propia.

6.2. Docker

Docker es una plataforma de código abierto que permite desarrollar, empaquetar y ejecutar aplicaciones dentro de entornos aislados llamados contenedores. Su propósito central es garantizar que una aplicación se ejecute de forma uniforme sin importar la máquina o infraestructura donde se despliegue. Gracias a este aislamiento, Docker facilita la portabilidad, la escalabilidad y, especialmente, la reproducibilidad del software: si una aplicación funciona dentro de un contenedor, funcionará exactamente igual en cualquier otro equipo que ejecute Docker, independientemente de su configuración interna [8]. Esto resulta especialmente valioso en proyectos robóticos, donde la consistencia del entorno evita errores asociados a dependencias, versiones o configuraciones de sistema.

6.2.1. Virtualización vs. contenedores

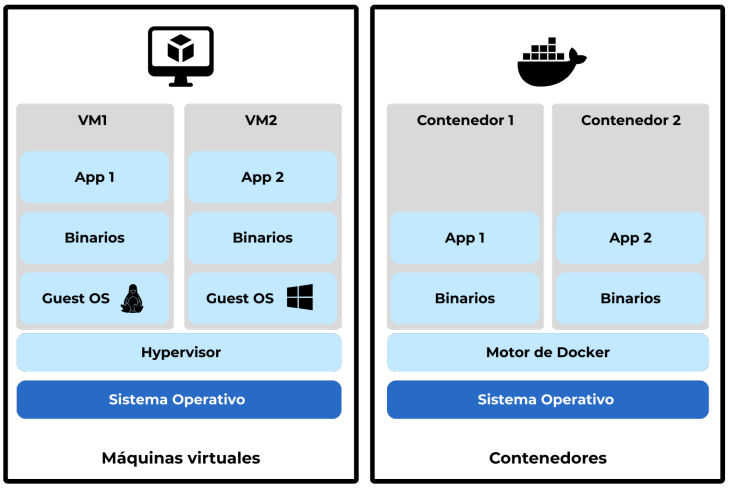
Las máquinas virtuales (*virtual machines*) son una representación virtual de un equipo físico, capaces de ejecutar sistemas operativos completos como invitados (*guest OS*). Su funcionamiento depende de un hipervisor encargado de gestionar la asignación de recursos entre el hardware real y las distintas máquinas virtuales. Este enfoque ofrece aislamiento robusto, pero implica un mayor consumo de memoria, almacenamiento y tiempo de arranque, ya que cada instancia requiere un sistema operativo completo [9].

En contraste, los contenedores son unidades ligeras que encapsulan únicamente el código de la aplicación, junto con sus dependencias y configuraciones esenciales. A diferencia de las máquinas virtuales, los contenedores no replican un sistema operativo completo, sino que

comparten el kernel del sistema anfitrión. Esta característica reduce significativamente el consumo de recursos y permite tiempos de despliegue y escalamiento mucho más rápidos. Al ser deterministas y portables, los contenedores se convierten en una herramienta ideal para garantizar la reproducibilidad en experimentos y desarrollos robóticos complejos.

La Figura 2 ilustra de forma conceptual las diferencias entre ambos paradigmas.

Figura 2. Comparación conceptual entre máquinas virtuales y contenedores



Nota. Elaboración propia.

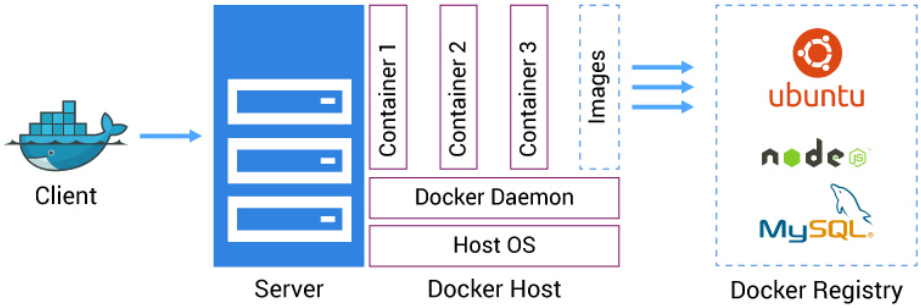
6.2.2. Arquitectura de Docker

Docker está compuesto por distintos elementos que trabajan de manera conjunta para crear, distribuir y ejecutar aplicaciones de forma ligera y reproducible. Como se muestra en la Figura 3, el sistema se basa en la interacción entre el cliente, el *Docker Daemon*, los registros de imágenes y los contenedores. Esta arquitectura permite encapsular aplicaciones en imágenes portables que pueden reconstruirse o ejecutarse en cualquier máquina sin alterar su comportamiento [10].

- **Imágenes** (*images*): son plantillas inmutables que contienen el código, las dependencias y el entorno de ejecución necesarios para desplegar una aplicación. Constituyen la base a partir de la cual se crean los contenedores.
- **Contenedores** (*containers*): son instancias en ejecución de una imagen. Se ejecutan de manera aislada pero comparten el kernel del sistema operativo anfitrión, lo que permite mayor eficiencia y menor consumo de recursos en comparación con las máquinas virtuales.
- ***Dockerfile***: es un archivo de texto declarativo que especifica las instrucciones para construir una imagen. Gracias a este mecanismo, se garantiza que las aplicaciones puedan desplegarse de manera consistente en cualquier entorno.

- ***Entrypoint***: define el proceso o comando que se ejecutará de manera predeterminada cuando se inicie un contenedor. Este elemento permite controlar el comportamiento inicial de las aplicaciones empaquetadas en Docker.
- ***Docker Daemon***: es el servicio que se ejecuta en segundo plano en el host y que se encarga de crear, ejecutar y gestionar los contenedores, imágenes, redes y volúmenes. Es el núcleo del funcionamiento de Docker [11].
- ***Docker Engine***: es la plataforma completa de Docker, que incluye el cliente de línea de comandos, el *Docker Daemon* y la API REST. Permite que los usuarios interactúen con los contenedores y servicios de Docker de forma integrada.

Figura 3. Arquitectura de Docker



Nota. Adaptado de [10].

Un aspecto fundamental de Docker dentro del contexto de este proyecto es su capacidad de asegurar la reproducibilidad del entorno de ejecución. Al encapsular dependencias, versiones y configuraciones en contenedores deterministas, Docker permite que los módulos desarrollados para MOCAP4ROS2, ROS2 y CrazySwarm2 se ejecuten de manera idéntica sin importar el equipo o la sesión de trabajo. Esta consistencia elimina problemas asociados a incompatibilidades del sistema operativo o variaciones en las librerías instaladas, garantizando que cada experimento robótico se lleve a cabo bajo condiciones controladas y replicables, requisito indispensable para validar resultados y comparar métricas de desempeño en sistemas multiagente.

6.3. OptiTrack

OptiTrack es un sistema de captura de movimiento de tipo óptico pasivo que emplea cámaras infrarrojas para detectar marcadores reflectivos adheridos a objetos o *rigid bodies* [12]. En el laboratorio Robotat se cuenta con seis cámaras OptiTrack PrimeX 41, diseñadas para ofrecer una cobertura amplia gracias a su ángulo de visión y capacidad de iluminación infrarroja. Estas cámaras permiten realizar seguimientos estables en todo el volumen de captura sin requerir un número elevado de dispositivos. La Figura 4 muestra el modelo utilizado.

Las PrimeX 41 pueden alcanzar hasta 250 Hz en configuraciones de baja resolución; sin embargo, para las pruebas de este proyecto se trabajó en un rango de 100–120 Hz, lo cual brinda un equilibrio adecuado entre estabilidad, ruido y carga computacional. En este flujo de trabajo, el software *Motive* se encarga de procesar en tiempo real la información registrada por las cámaras y de administrar la transmisión de datos hacia clientes externos mediante el protocolo NatNet.

En términos de comunicación en red, OptiTrack permite transmitir los datos generados por *Motive* mediante los modos *unicast* o *multicast*. En este proyecto se utilizó el modo *multicast*, ya que permite que múltiples clientes (ROS2, MATLAB u otras estaciones) reciban simultáneamente la información sin establecer conexiones individuales, lo cual reduce la carga del sistema y favorece la integración distribuida en aplicaciones robóticas.

Figura 4. Cámara OptiTrack PrimeX 41

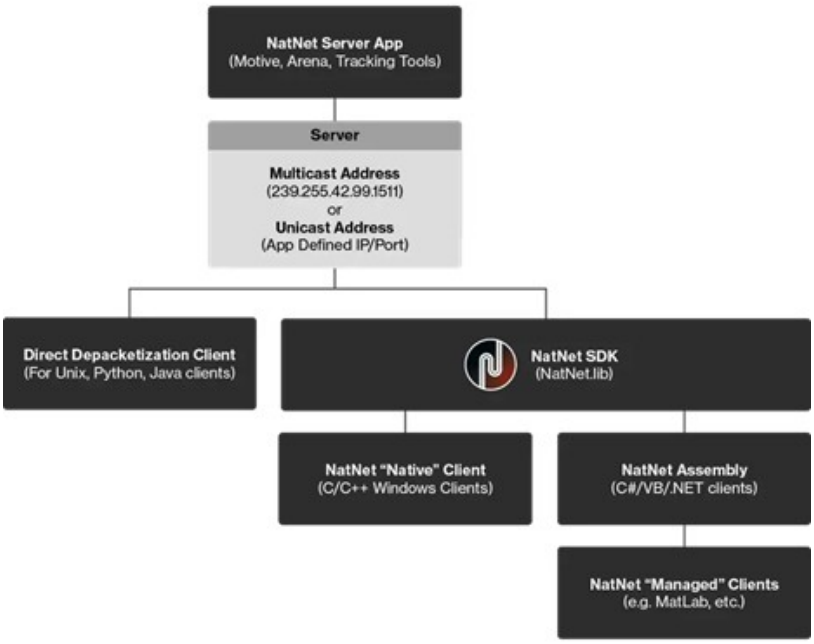
Nota. Imagen tomada de [13].

6.3.1. NatNet

NatNet es el protocolo oficial mediante el cual *Motive* transmite los datos de captura a través de la red. Opera utilizando dos puertos principales: 1510 (comandos) y 1511 (datos), y soporta tanto *unicast* como *multicast* [14]. La Figura 5 muestra un esquema general de la arquitectura cliente–servidor.

En modo *multicast* la configuración empleada en este proyecto el servidor envía la información a una dirección de grupo, permitiendo que múltiples clientes reciban simultáneamente las poses de marcadores y de *rigid bodies*. Este mecanismo reduce la sobrecarga del sistema y facilita la interoperabilidad con MOCAP4ROS2, que actúa como cliente NatNet dentro del servidor ROS2.

Figura 5. Arquitectura cliente/servidor de NatNet SDK



Nota. Imagen tomada de [14].

6.3.2. Motive

Motive es el software central de OptiTrack encargado de la calibración del sistema, la reconstrucción en tiempo real de los marcadores y la definición de *rigid bodies* [15]. Además, administra la transmisión de datos mediante NatNet, actuando como servidor dentro de la red de captura.

Para las experimentaciones realizadas en este proyecto, *Motive* se configuró como servidor NatNet en modo *multicast*, transmitiendo datos a frecuencias entre 100 y 120 Hz hacia el servidor ROS2. Esta configuración permitió que MOCAP4ROS2 recibiera las poses de manera directa y con menor latencia que el flujo tradicional utilizado previamente en el ecosistema Robotat.

6.4. MOCAP4ROS2

MOCAP4ROS2 es un paquete de código abierto (*open source*) desarrollado en el marco del proyecto europeo ROSIN, cuyo propósito es estandarizar la integración de sistemas de captura de movimiento en el ecosistema ROS2 mediante interfaces comunes y estructuradas.

Este paquete proporciona nodos, mensajes y herramientas que permiten recibir y procesar datos de mocap de forma unificada, independientemente del fabricante del sistema de captura [4].

Su relevancia dentro de este proyecto radica en que actúa como punto de conexión directo entre el software *Motive* y ROS2 mediante el protocolo NatNet, eliminando la necesidad de flujos intermedios basados en MATLAB o servidores JSON, como ocurre en la infraestructura tradicional del laboratorio Robotat. Gracias a esta integración nativa, MOCAP4ROS2 permite obtener datos de posición y orientación con menor latencia, mayor estabilidad en la frecuencia de actualización y una estructura de mensajes compatible con los sistemas de control empleados en ROS2.

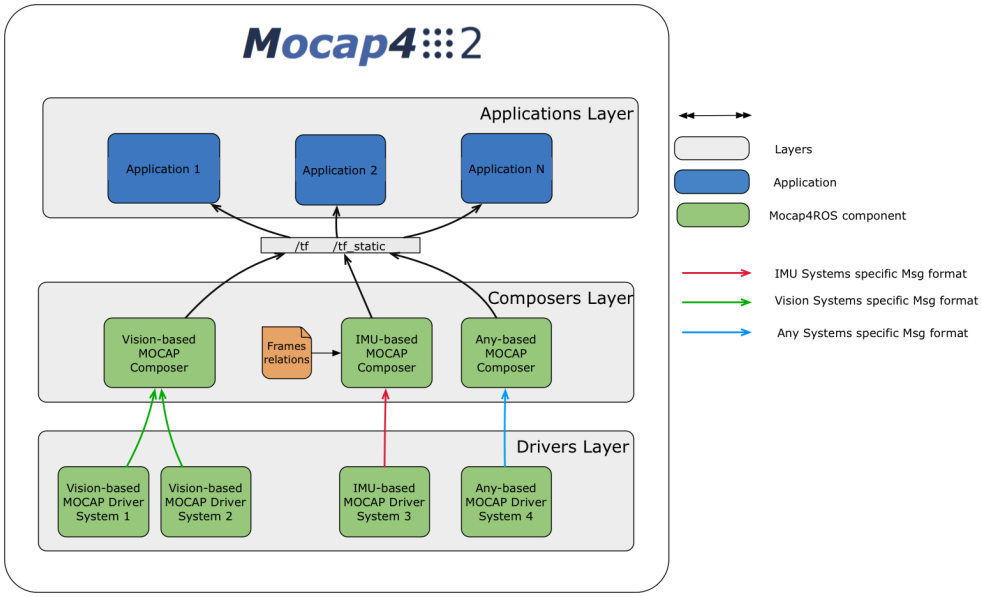
Además, el paquete incluye controladores específicos para sistemas OptiTrack, Vicon y Qualisys, lo que garantiza interoperabilidad y facilita la reutilización del código en distintos tipos de experimentos robóticos. En este proyecto, MOCAP4ROS2 resulta fundamental para habilitar el envío eficiente de la pose de un *rigid body* hacia agentes robóticos, permitiendo así la implementación de controles en lazo cerrado y futuros ensambles de enjambres robóticos.

6.4.1. Arquitectura MOCAP4ROS2

La arquitectura de MOCAP4ROS2 se organiza en un modelo por capas (Figura 6), lo que permite separar de manera modular las funciones de adquisición, procesamiento y uso de los datos de captura de movimiento. Cada capa cumple un rol específico dentro del flujo de información, desde la comunicación con los sistemas de captura hasta la utilización de los datos por aplicaciones externas [16].

- ***Drivers Layer***: contiene los controladores encargados de comunicarse directamente con los sistemas de captura de movimiento. Aquí se incluyen tanto sistemas ópticos (cámaras y marcadores) como sensores inerciales (IMU) u otros métodos de captura. Su función principal es estandarizar la salida de cada sistema a un formato común.
- ***Composer Layer***: agrupa y unifica la información proveniente de distintos tipos de sistemas de captura. Utiliza el sistema de transformaciones (*tf*) de ROS2 para alinear la información en un marco de referencia geométrico común, permitiendo la integración de datos heterogéneos.
- ***Application Layer***: corresponde a los programas y aplicaciones que consumen los datos procesados, ya sea para control, visualización o análisis. Gracias a esta capa, cualquier aplicación desarrollada bajo MOCAP4ROS2 puede ser reutilizada con diferentes sistemas de captura de movimiento.

Figura 6. Capas de MOCAP4ROS2



Nota. Imagen adaptada de [16].

CAPÍTULO 7

Configuración y funcionamiento del servidor

En este capítulo se describe la configuración inicial del servidor fijo utilizado en el ecosistema Robotat, empleando Ubuntu 22.04 y la distribución ROS2 Humble Hawksbill como base para la ejecución de aplicaciones robóticas. Asimismo, se integró Docker como herramienta de virtualización ligera, lo que facilita la gestión de contenedores y prepara el entorno para futuras implementaciones de CrazySwarm2.

Si bien la instalación de Ubuntu y ROS2 fue previamente documentada por Brandon Garrido [6], en esta ocasión se simplificaron y adaptaron procedimientos para optimizar la integración de recursos. Finalmente, se presentan las características técnicas del servidor y las pruebas preliminares realizadas.

7.1. Instalación del sistema base: Ubuntu 22.04 y ROS2 Humble Hawksbill

La instalación del sistema operativo Ubuntu 22.04 LTS se realizó de manera nativa en el servidor fijo, evitando el uso de máquinas virtuales con el fin de obtener mayor estabilidad y rendimiento en las pruebas experimentales. Para este proceso se siguieron los pasos habituales de instalación:

1. **Descarga de la ISO**: se descargó la imagen oficial de Ubuntu 22.04 LTS desde el sitio web de Ubuntu [17], seleccionando la versión de escritorio de 64 bits.
2. **Creación de medio de arranque**: se preparó una memoria USB de 32 GB como unidad booteable utilizando la herramienta Rufus en un equipo con Windows. El esquema de partición seleccionado fue GPT, con sistema de archivos por defecto.

3. **Configuración de arranque**: al reiniciar el equipo servidor, se accedió a la BIOS y se configuró la memoria USB como primera opción de arranque.

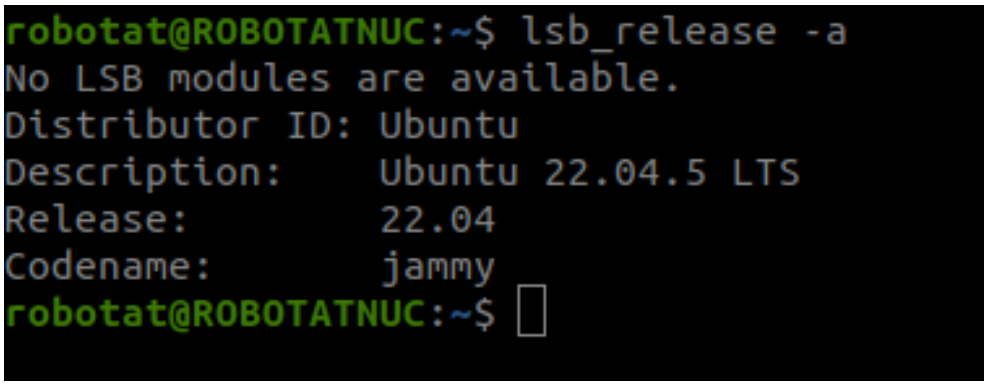
4. **Instalación del sistema**: una vez iniciado el instalador gráfico de Ubuntu, se optó por una instalación limpia, reemplazando cualquier configuración previa y estableciendo las credenciales de usuario.

5. **Actualización inicial**: tras finalizar la instalación, se ejecutó una actualización completa de paquetes mediante el administrador de paquetes `apt`, asegurando compatibilidad con ROS2 Humble Hawksbill.

La decisión de realizar una instalación local, en lugar de utilizar una máquina virtual, se fundamenta en que Ubuntu instalado de forma nativa en hardware físico ofrece un mejor aprovechamiento de los recursos, menor latencia y mayor estabilidad en comparación con entornos virtualizados, lo cual resulta crítico para experimentos con aplicaciones robóticas en tiempo real.

La Figura 7 ilustra la verificación de la correcta instalación del sistema operativo Ubuntu 22.04 LTS en el servidor.

Figura 7. Verificación de la versión instalada de Ubuntu 22.04 LTS



Nota. Elaboración propia.

Como base sobre el sistema operativo Ubuntu 22.04 LTS se optó por instalar la distribución ROS2 Humble Hawksbill, debido a la estabilidad que ofrece en sus dependencias y a su soporte oficial para esta versión de Ubuntu. Aunque existen versiones más recientes de ROS2, como Jazzy Jalisco o Kinetic Kaiju, se seleccionó Humble por su madurez y compatibilidad comprobada en entornos de investigación. El proceso de instalación se llevó a cabo siguiendo las instrucciones proporcionadas en la documentación oficial de la distribución [18].

Para comprobar la correcta instalación del entorno, se ejecutó una prueba básica utilizando los nodos de ejemplo `talker` y `listener`. En esta verificación, el nodo publicador `talker` transmite mensajes de texto, mientras que el nodo suscriptor `listener` los recibe e imprime en la consola. La comunicación exitosa entre ambos nodos valida que la instalación

fue realizada correctamente y que las APIs de C++ y Python se encuentran operativas en el sistema.

Para ello, en una primera terminal se cargó el entorno de ROS2 y se inició el nodo `talker` en C++ con los siguientes comandos:

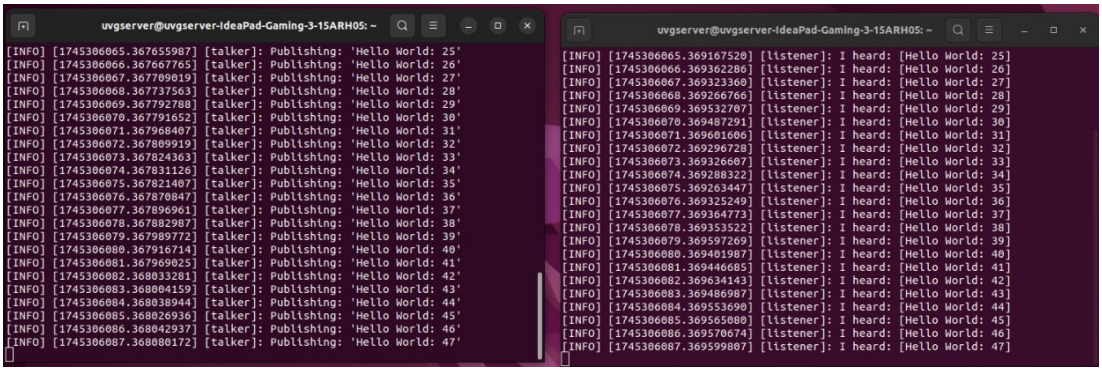
```
source /opt/ros/humble/setup.bash
ros2 run demo_nodes_cpp talker
```

Simultáneamente, en una segunda terminal se repitió el procedimiento de carga del entorno y se ejecutó el nodo `listener` en Python:

```
source /opt/ros/humble/setup.bash
ros2 run demo_nodes_py listener
```

La Figura 8 ilustra la interacción básica entre ambos nodos durante esta prueba de verificación.

Figura 8. Interacción básica entre los nodos `talker` y `listener` en ROS2



Nota. Elaboración propia.

7.2. Instalación de Docker

Con el fin de facilitar la portabilidad de los proyectos y la gestión de dependencias, se instaló Docker en el sistema operativo Ubuntu 22.04 LTS. Esta herramienta permite ejecutar aplicaciones en contenedores ligeros, garantizando un despliegue reproducible y aislado, lo cual resulta fundamental para la integración futura de CrazySwarm2. La instalación se realizó siguiendo las instrucciones oficiales de la documentación de Docker [19].

Para verificar la instalación, se ejecutó el comando:

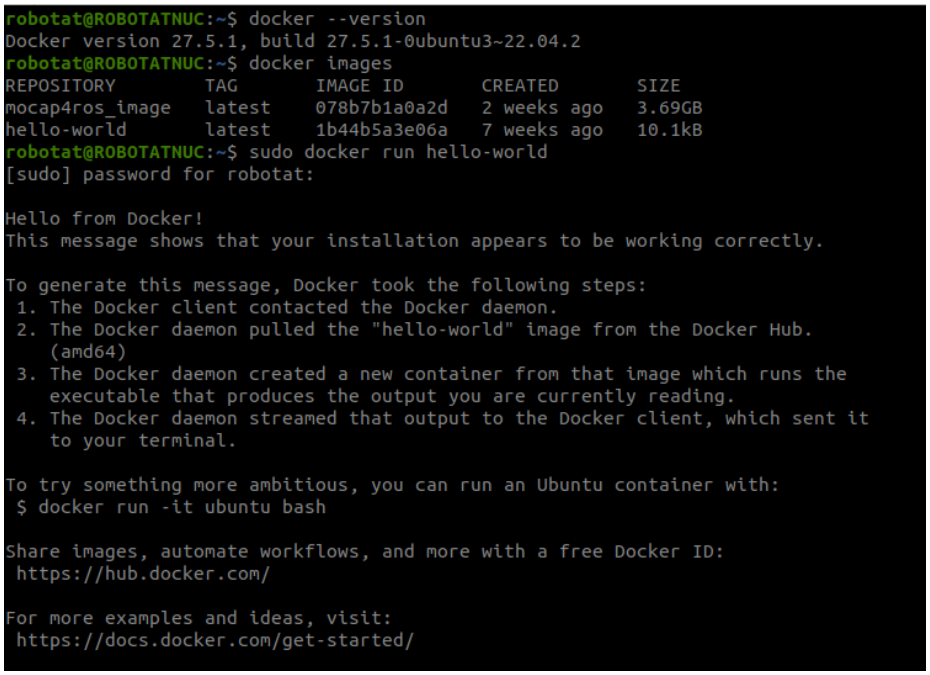
```
docker --version
```

El cual devolvió la versión instalada de Docker, confirmando que el sistema reconocía la herramienta. Posteriormente, se probó la ejecución de un contenedor de prueba con el comando:

```
sudo docker run hello-world
```

Este comando descarga y ejecuta una imagen mínima desde Docker Hub, mostrando un mensaje en consola que confirma el correcto funcionamiento del motor de contenedores. Las Figura 9 presenta la salida de estas verificaciones.

Figura 9. Verificación de la instalación de Docker



Nota. Elaboración propia.

7.2.1. Integración con Visual Studio Code

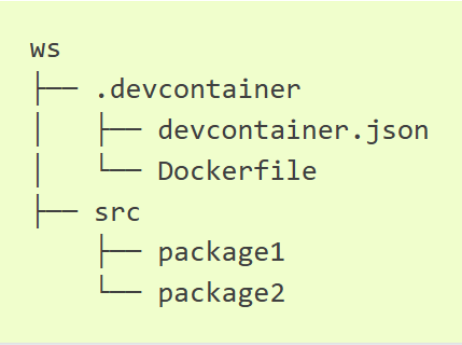
Adicionalmente, se instaló el editor de texto *Visual Studio Code* (VS Code) de forma local, siguiendo el procedimiento descrito en la guía *Setup ROS2 with VSCode and Docker [community-contributed]* [20]. Esta herramienta se complementó con la extensión *Remote Development*, la cual permite acceder y manipular directamente contenedores de Docker como entornos de desarrollo, facilitando así la depuración y el manejo de proyectos en ROS2.

7.2.2. Estructura del espacio de trabajo en ROS2

Inicialmente se probó el flujo recomendado por la comunidad de ROS2 para trabajar con contenedores y Visual Studio Code, empleando un directorio `.devcontainer/` con los archivos `devcontainer.json` y `Dockerfile`. Este enfoque permite reconstruir y abrir el espacio de trabajo dentro de un contenedor con la extensión *Remote Development*, gestionando de forma automática extensiones, variables de entorno y mapeo del *workspace*, donde la carpeta

`src` contiene los paquetes del proyecto. La Figura 10 muestra esta estructura recomendada por la documentación oficial.

Figura 10. Workspace en ROS2 con VS Code y Docker



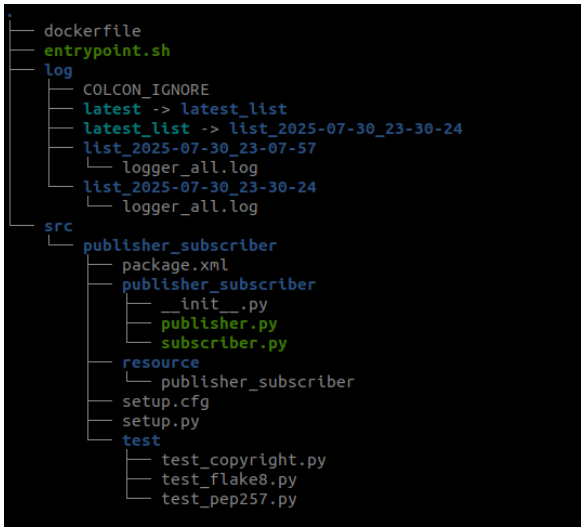
Nota. Imagen tomada de [20].

Durante las pruebas, este método presentó incidencias relacionadas con rutas y permisos (propiedad de archivos generados desde el contenedor y discrepancias de UID/GID), por lo que se optó por un flujo más controlado basado en un `Dockerfile` y un `entrypoint.sh` propios, ejecutados con comandos explícitos de `docker build/run`.

Este cambio permitió un mayor control del usuario y de los permisos dentro del contenedor, evitando la generación de archivos con propietario `root` en el host. Asimismo, se logró definir con precisión las variables de entorno y el *overlay* de ROS2 a cargar, lo que aportó claridad y flexibilidad en la configuración. Finalmente, se simplificó el *bind-mount* del *workspace* y la gestión de los directorios de *logs*, haciendo más eficiente el proceso de desarrollo.

La Figura 11 muestra la estructura manual implementada en el proyecto, donde la carpeta `src` concentra los paquetes de ROS2 desarrollados y la carpeta `log` almacena los registros de ejecución.

Figura 11. Estructura manual del proyecto con Docker y ROS2



Nota. Elaboración propia.

7.3. Integración de CrazySwarm2 y pruebas iniciales

Este capítulo presenta la integración de CrazySwarm2 en el servidor del laboratorio, utilizando el entorno basado en Docker configurado previamente. El objetivo principal fue asegurar que la plataforma pudiera ejecutarse de forma estable y reproducible sobre ROS2 Humble.

Además, se realizaron pruebas iniciales para verificar la comunicación entre los nodos principales del sistema y confirmar que la infraestructura del servidor es apta para futuras implementaciones con enjambres de drones. Estos resultados permiten establecer una base funcional para las etapas posteriores del proyecto.

7.3.1. Características del servidor

Para llevar a cabo una infraestructura funcional y eficiente se requiere un ordenador capaz de ejecutar el sistema operativo Ubuntu 22.04 LTS en su versión de escritorio, cumpliendo con los requisitos mínimos recomendados: una arquitectura de 64 bits, al menos 4 GB de memoria RAM, un procesador de doble núcleo a 2 GHz y 25 GB de almacenamiento libre [17].

Bajo estas consideraciones, se seleccionó como servidor central la computadora *MINISFORUM UM870*, con el objetivo de ejecutar diversos proyectos en ROS2 y principalmente, la plataforma CrazySwarm2. Este equipo destaca por su diseño compacto, que combina un reducido tamaño físico con componentes de alto rendimiento, lo que lo convierte en una opción adecuada para aplicaciones que requieren portabilidad sin sacrificar potencia de procesamiento. La Figura 12 muestra el dispositivo empleado como servidor central.

Figura 12. Mini PC *MINISFORUM UM870 Slim*

Nota. Imagen tomada de [21].

En el Cuadro 1 se presenta una tabla detallada con las principales características del servidor seleccionado *MINISFORUM UM870 Slim*[21]. Este equipo fue elegido por su capacidad de ejecución en proyectos complejos de ROS2 y la integración de CrazySwarm2.

Cuadro 1. Características técnicas del servidor MINISFORUM UM870 Slim

Característica	**Especificación**
Procesador	AMD Ryzen 7 8745HS con Radeon 780M (8 núcleos / 16 hilos)
Memoria RAM	32 GB DDR5
Almacenamiento	SSD NVMe 1 TB
Gráficos	Radeon 780M (integrado)
Sistema Operativo	Ubuntu 22.04.5 LTS (64 bits)
Alimentación	DC 19V
Ethernet	1 × RJ45 2.5G
USB	2 × USB 3.2 Gen2, 2 × USB 2.0, 1 × USB4 (Alt PD)
Video	1 × HDMI, 1 × DisplayPort
Audio	1 × Jack 3.5 mm
Otros	Clear CMOS ×1

Nota. Datos tomados de [21].

7.3.2. Preparación del entorno para CrazySwarm2

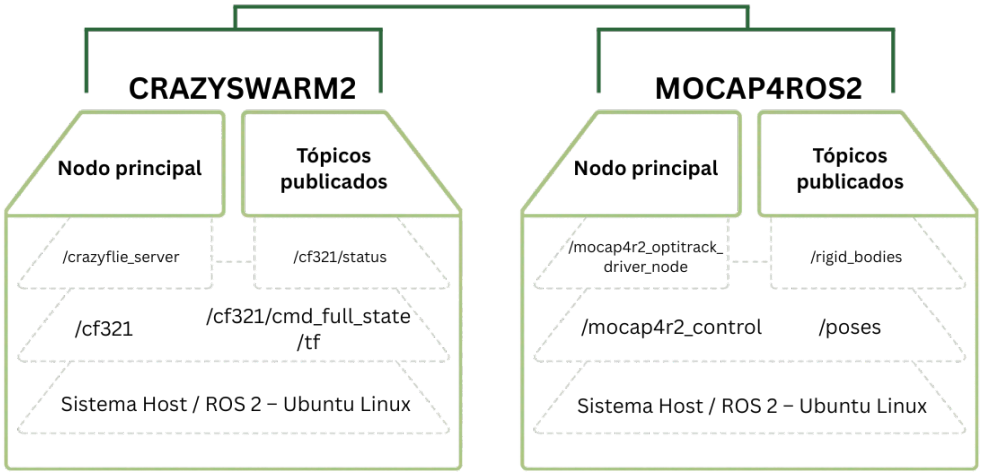
La integración de CrazySwarm2 requiere un entorno de ejecución estable, reproducible y aislado, capaz de gestionar dependencias complejas propias del ecosistema ROS2. Debido a que la instalación nativa de estas herramientas puede resultar propensa a conflictos de versiones, se optó por construir un entorno basado en contenedores Docker. Este enfoque permite mantener una configuración homogénea del sistema, independientemente del equipo físico desde el cual se ejecute, garantizando así la reproducibilidad de los experimentos.

Para ello, se desarrolló un `Dockerfile` específico que instala ROS2 Humble Hawksbill, las dependencias de CrazySwarm2, las herramientas de compilación y los controladores necesarios. Adicionalmente, se creó un script `entrypoint.sh` encargado de inicializar correctamente el entorno, cargar el `overlay` de ROS2 y configurar las variables de entorno asociadas al *workspace*. De esta forma, cada instancia del contenedor inicia con una configuración limpia y controlada, evitando inconsistencias entre sesiones.

Durante la implementación se elaboró también un manual de usuario destinado a facilitar la replicación del entorno por otros integrantes del laboratorio. Este documento detalla el proceso completo de construcción, ejecución y mantenimiento del contenedor, incluyendo la compilación del `workspace`, el uso del `entrypoint`, los comandos principales del sistema y las buenas prácticas de operación. El manual completo se incluye en la sección de Anexos de este trabajo.

Como parte de los resultados de esta etapa, la Figura 13 presenta una representación conceptual de la estructura interna del entorno en contenedores. Esta ilustración muestra cómo se organizan las capas de software y los servicios dentro de Docker, destacando la equivalencia entre contenedores con configuraciones idénticas. Este diseño permitió obtener un entorno modular, escalable y fácilmente reproducible, aspectos esenciales para futuros experimentos con enjambres robóticos.

Figura 13. Representación conceptual de la estructura en capas del entorno con contenedores Docker



Nota. Elaboración propia.

En conjunto, la preparación del entorno permitió establecer una base sólida para la ejecución de CrazySwarm2 dentro del servidor, garantizando que los controles, simulaciones y algoritmos posteriores puedan desarrollarse bajo un sistema confiable y replicable. Este entorno constituye el punto de partida para las pruebas iniciales y la integración con los agentes robóticos, temas abordados en la siguiente sección.

CAPÍTULO 8

MOCAP4ROS2

En este capítulo se aborda la integración del proyecto MOCAP4ROS2 dentro del marco de trabajo del sistema de captura de movimiento OptiTrack, con el objetivo de estandarizar la comunicación de este tipo de sistemas en ROS2. Para ello, se empleó la librería MOCAP4ROS2, la cual proporciona paquetes y nodos que facilitan la compatibilidad y el manejo de datos de captura en entornos robóticos. Asimismo se utilizó NatNet, herramienta que permite el envío y recepción de datos de seguimiento en tiempo real, junto con la configuración del software *Motive*, encargado de procesar y transmitir la información capturada por las cámaras. De esta manera, se establecen las bases para evaluar el desempeño de MOCAP4ROS2 en comparación con los flujos de trabajo previamente empleados en el ecosistema Robotat.

8.1. Instalación y configuración de MOCAP4ROS2

La instalación de MOCAP4ROS2 se realizó en el servidor con Ubuntu 22.04, creando un espacio de trabajo denominado `mocap4ros2_ws`. En este entorno se incorporaron los repositorios principales del proyecto, incluyendo el paquete `mocap4ros2_optitrack`, encargado de gestionar la comunicación con el sistema de captura OptiTrack utilizado en el ecosistema Robotat.

Una vez descargados los repositorios, se instalaron las dependencias necesarias y se procedió a compilar el espacio de trabajo mediante `colcon build`. Finalmente, se configuró el entorno con el comando `source`, habilitando el uso de los paquetes de MOCAP4ROS2 dentro de ROS2. Este procedimiento permitió establecer la comunicación entre OptiTrack y el servidor.

A continuación, se muestran los comandos necesarios para realizar este procedimiento:

```
# Crear el directorio de trabajo
mkdir -p mocap4ros2_ws/src && cd mocap4ros2_ws/src

# Clonar los repositorios principales
git clone https://github.com/MOCAP4ROS2-Project/mocap4ros2_optitrack
    .git
git clone https://github.com/MOCAP4ROS2-Project/mocap.git
git clone https://github.com/MOCAP4ROS2-Project/mocap_msgs.git

# Instalar dependencias
rosdep install --from-paths src --ignore-src -r -y
vcs import < mocap4ros2_optitrack/dependency_repos.repos

# Compilar el workspace
cd .. && colcon build --symlink-install

# Activar el entorno
source install/setup.bash
```

Luego de completar la instalación de MOCAP4ROS2 y compilar el espacio de trabajo, es útil comprender la organización del paquete `mocap4ros2_optitrack`. En particular, el subpaquete `mocap4r2_optitrack_driver` concentra los archivos de configuración y lanzamiento necesarios para establecer la comunicación con *Motive* mediante NatNet.

Para inspeccionar su contenido, desde la raíz del *workspace* se ejecutaron los siguientes comandos:

```
mocap4ros2_ws$ cd src/
mocap4ros2_ws/src$ cd mocap4ros2_optitrack/
mocap4ros2_ws/src/mocap4ros2_optitrack$ cd mocap4r2_optitrack_driver
```

En el Cuadro 2 resume las carpetas principales y su función.

Cuadro 2. Estructura principal del paquete `mocap4ros2_optitrack`

Directorio	**Descripción**
config	Contiene los archivos de configuración, incluyendo parámetros para el *driver* de OptiTrack.
launch	Incluye los *launch files* de ROS2, necesarios para iniciar nodos y configurar el sistema.
NatNetSDK	Librería de OptiTrack que provee las cabeceras y archivos compilados para la comunicación con Motive mediante el protocolo NatNet.
scripts	Contiene utilidades en Python para métricas, activación de nodos y conversión de datos.
src	Implementación en C++ del *driver* de OptiTrack y sus nodos principales.

Nota. Elaboración propia.

En la fase de configuración para integrar ROS2 con el sistema OptiTrack, se trabajó den-

tro de la carpeta `config` del paquete `mocap4ros2_optitrack`. En esta ubicación se encuentra el archivo `mocap4r2_optitrack_driver_params.yaml`, el cual concentra los parámetros necesarios para establecer la comunicación con OptiTrack. En el siguiente código se muestra un extracto del archivo utilizado durante la configuración:

```
mocap4r2_optitrack_driver_node:
  ros__parameters:
    connection_type: "Multicast" # Unicast / Multicast
    server_address: "198.168.50.200"
    local_address: "198.168.50.240"
    multicast_address: "239.255.42.99"
    server_command_port: 1510
    server_data_port: 1511
```

Como se observa en la configuración de parámetros, uno de los aspectos principales es el campo `connection_type`, el cual fue establecido como `multicast`. Esta modalidad permite que el servidor envíe datos de seguimiento a múltiples receptores de manera simultánea, en lugar de limitar la comunicación a un único destino. Para garantizar la compatibilidad, es necesario que esta misma opción esté activada en el software *Motive*, de modo que el envío de información se realice bajo el mismo esquema.

Otro parámetro relevante es `server_address`, que corresponde a la dirección IP del servidor que transmite los datos de OptiTrack hacia ROS2. A este se suma el campo `local_address`, el cual indica la dirección IP de la máquina que ejecuta los nodos de ROS2 y que recibirá los datos en la red local.

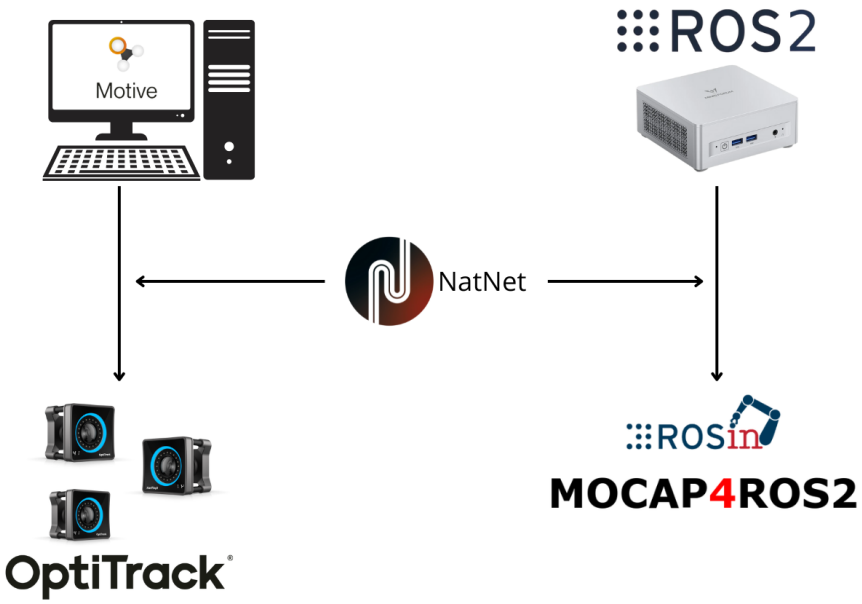
Finalmente, se configuraron los puertos de comunicación. El puerto `1510` se emplea para el envío de comandos al sistema OptiTrack, mientras que el puerto `1511` se destina a la transmisión de datos de seguimiento. Estos valores deben coincidir con los establecidos en el panel de *Motive*, asegurando que la comunicación fluya sin restricciones y que los nodos de ROS2 puedan procesar correctamente la información recibida.

NatNet

MOCAP4ROS2 proporciona las librerías necesarias de NatNet, las cuales permiten establecer la comunicación entre el sistema de captura de movimiento OptiTrack y el entorno de trabajo en ROS2. Gracias a estas librerías, es posible recibir en tiempo real los datos procesados por el software *Motive*, que actúa como servidor dentro de la red, y transferirlos hacia ROS2, configurado como cliente.

Como se ilustra en la Figura 14, este flujo de comunicación asegura que los datos de posición y orientación capturados por las cámaras de OptiTrack sean transmitidos de manera eficiente hacia ROS2. En el ecosistema Robotat, esta integración constituye la base para el trabajo con múltiples agentes robóticos.

Figura 14. Diagrama de comunicación NatNet entre Motive y ROS2



Nota. Elaboración propia.

Para llevar a cabo la configuración de NatNet, se accedió a la carpeta `NatNetSDK`, la cual contiene un subdirectorio denominado `include`. Dentro de este se encuentra el archivo `NatNetTypes.h`, encargado de definir los parámetros y estructuras necesarios para establecer la comunicación entre el sistema OptiTrack y ROS2. En el siguiente código se muestra un extracto del archivo utilizado durante la configuración:

```
#define NATNET_DEFAULT_PORT_COMMAND          1510
#define NATNET_DEFAULT_PORT_DATA             1511
#define NATNET_DEFAULT_MULTICAST_ADDRESS     "239.255.42.99"     // IANA,
    local network
```

Como se puede observar en el código, parte de la configuración, se definieron parámetros esenciales para garantizar la correcta comunicación entre OptiTrack y ROS2. Entre ellos destacan la asignación de los puertos de comando y datos, que permiten el envío de instrucciones y la transmisión de información de seguimiento, así como la dirección *multicast* utilizada para el flujo de datos en tiempo real.

8.1.1. Visualización de pose con MOCAP4ROS2

Una vez completada la fase de instalación y configuración, el siguiente paso consistió en verificar el correcto funcionamiento de la librería MOCAP4ROS2. Para ello, el procedimiento se dividió en dos etapas principales.

En primer lugar, se ejecutó la inicialización del espacio de trabajo de MOCAP4ROS2 y la carga del entorno de ROS2. Estos pasos son imprescindibles antes de lanzar cualquier nodo o archivo de configuración:

```
# Configurar espacio de trabajo MOCAP4ROS2
source install/setup.bash

# Cargar script de inicio de ROS2
source /opt/ros/humble/setup.bash
```

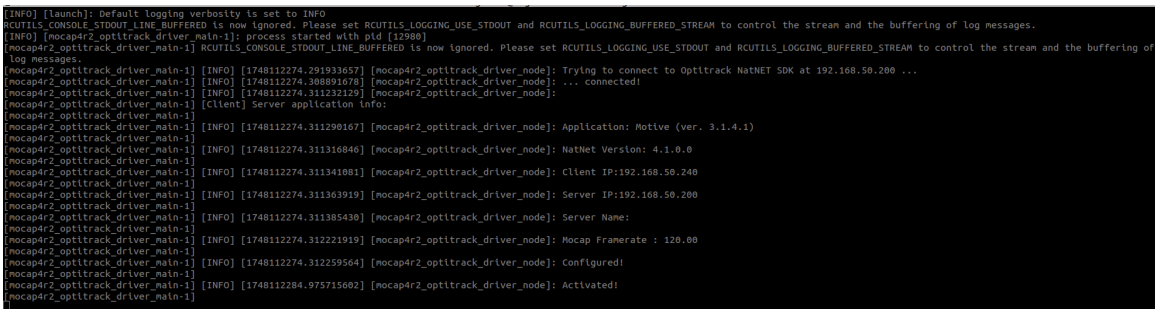
Una vez completada la carga del entorno, se procedió a iniciar el sistema OptiTrack mediante el archivo de lanzamiento, encargado de inicializar los nodos que gestionan la comunicación con el sistema de captura:

```
# Iniciar el sistema OptiTrack mediante un archivo de lanzamiento
ros2 launch mocap_optitrack_driver optitrack2.launch.py
```

Una vez inicializado el archivo de lanzamiento, es necesario activar el nodo del driver de MOCAP4ROS2, ya que este se ejecuta como un *lifecycle node*. Para ello, en una segunda terminal se ejecuto el siguiente comando, que permite la transición al estado activo y habilita la publicación de datos provenientes del sistema de captura:

```
ros2 lifecycle set /mocap4r2_optitrack_driver_node activate
```

Figura 15. Conexión establecida entre MOCAP4ROS2 y OptiTrack



Nota. Elaboración propia.

Como se puede observar en la Figura 15, la comunicación entre el cliente y el servidor se estableció correctamente. En la salida se aprecia tanto la dirección IP del cliente como la del servidor, así como la frecuencia de actualización de los fotogramas configurada en *Motive*, la cual corresponde a 120 Hz.

En la Figura 16 se muestra un marcador reflectivo individual (*single marker*) utilizado en las pruebas iniciales. Estos marcadores, al ser detectados por el sistema OptiTrack, permiten registrar su posición tridimensional en tiempo real y publicarla dentro del tópico `/markers` de ROS2.

Figura 16. Marcador reflectivo individual (*single marker*)

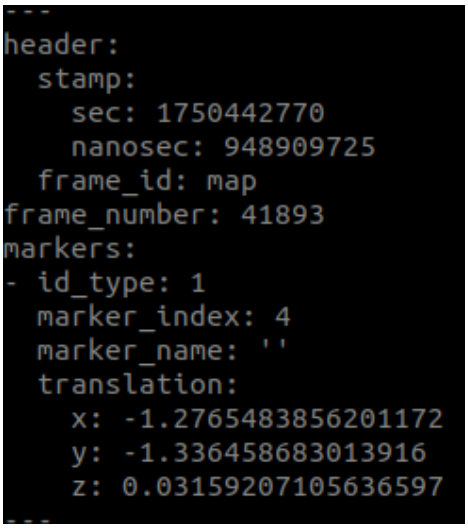
Nota. Elaboración propia.

Posteriormente, al ejecutar el comando:

```
ros2 topic echo /markers
```

La salida obtenida se muestra en la Figura 17, donde se aprecian los valores de posición del marcador en las coordenadas tridimensionales x, y y z. Estos datos corresponden a la localización del marcador en el sistema de referencia definido por el *frame* del sistema de captura, confirmando la correcta transmisión de información desde *Motive* hacia ROS2.

Figura 17. Detección de un marcador individual en ROS2



Nota. Elaboración propia.

En esta segunda prueba se empleó un *rigid body*, es decir, un conjunto de marcadores reflectivos dispuestos en una geometría fija que permite al sistema OptiTrack no solo identificar su posición, sino también calcular su orientación en el espacio tridimensional.

La Figura 18 muestra el *rigid body* utilizado en el experimento, mientras que en la Figura 19 se presenta el resultado obtenido en la terminal al ejecutar el comando:

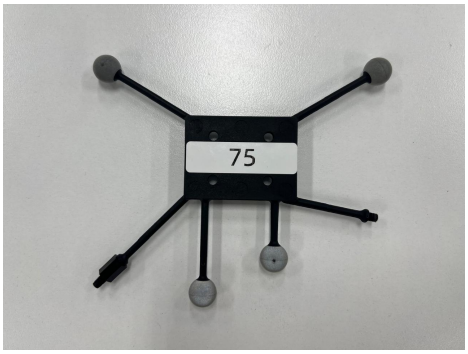
```
ros2 topic echo /rigid_bodies
```

Como se observa, además de la posición en los ejes x, y y z, se incluye la orientación representada mediante cuaterniones (x, y, z, w), lo que permite una descripción completa del movimiento del objeto en el espacio. Para lograr una visualización más clara de estos datos, se utilizó RViz mediante la ejecución del siguiente comando:

```
ros2 launch mocap4r2_marker_viz mocap4r2_marker_viz.launch.py
    mocap4r2_system:=optitrack
```

De esta manera, en la interfaz de RViz (Figura 20) se puede apreciar el sistema de referencia: la flecha verde ubicada en el centro de la cuadrícula corresponde al origen $(0, 0, 0)$, mientras que la flecha asociada al *rigid body* representa en tiempo real su posición y orientación dentro del entorno de ROS2.

Figura 18. Rigid body marker



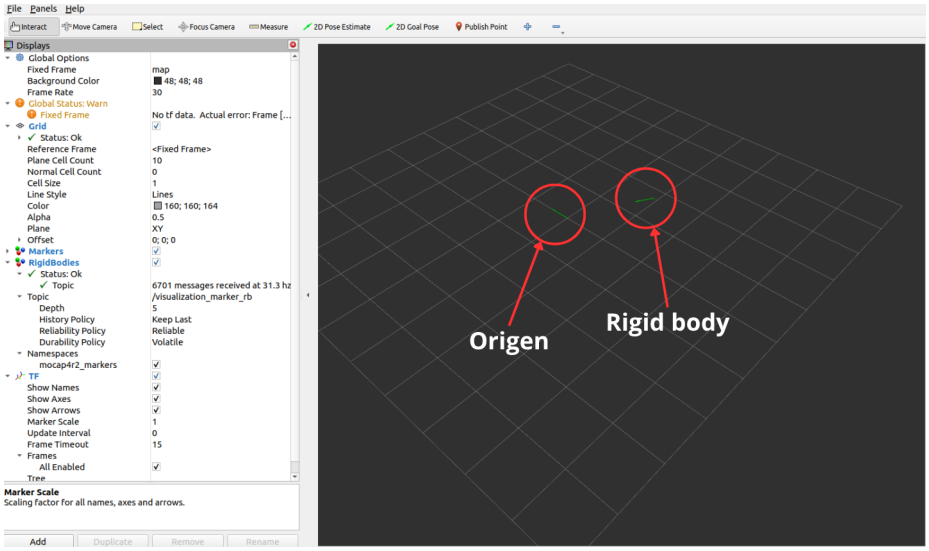
Nota. Elaboración propia.

Figura 19. Pose de un *rigid body* en ROS2

```
---
header:
  stamp:
    sec: 1754674532
    nanosec: 90516054
  frame_id: map
pose:
  position:
    x: -1.3363242149353027
    y: 1.0145357847213745
    z: -0.007144195493310 69
  orientation:
    x: -0.001472385716624558
    y: 0.003923092037439346
    z: 0.8662750124931335
    w: 0.49954986572265625
---
```

Nota. Elaboración propia.

Figura 20. Rigid body visualizado en RViz con MOCAP4ROS2



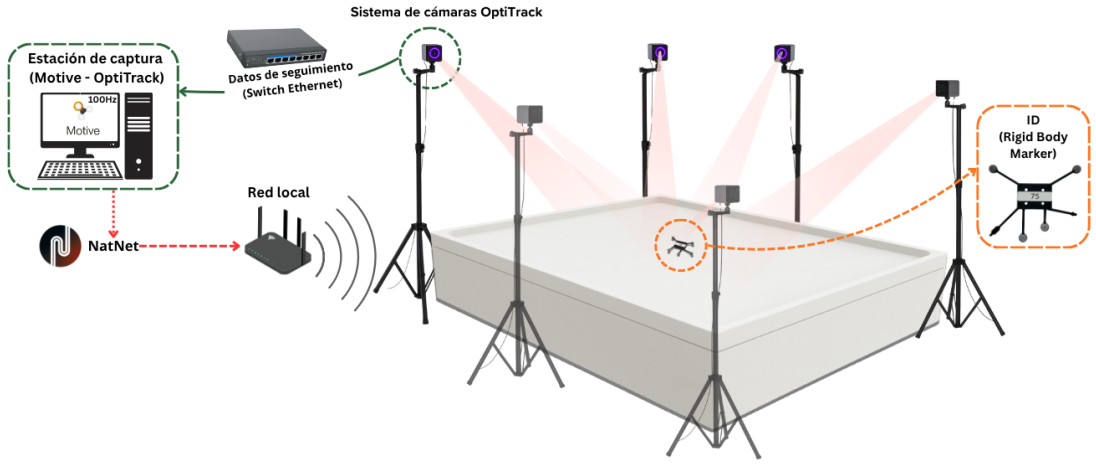
Nota. Elaboración propia.

8.2. Evaluación de rendimiento de MOCAP4ROS2 en el ecosistema Robotat

El objetivo de este apartado es evaluar el rendimiento del paquete MOCAP4ROS2 en la integración de sistemas de captura de movimiento, comparando sus resultados con el flujo tradicional del ecosistema Robotat. Para ello, se seleccionó un *rigid body* identificado mediante un marcador específico (*Marker ID*), el cual fue monitoreado tanto en el servidor con MOCAP4ROS2 como en la red de Robotat. En ambos casos se buscó medir dos métricas principales: la latencia (*Round Trip Time, RTT*) y la frecuencia de publicación de mensajes (Hz).

En el caso del ecosistema Robotat, los datos de captura generados por las cámaras OptiTrack son procesados en el software *Motive* y transmitidos a través de un servidor local utilizando el protocolo *UDP*. La información resultante se comparte en formato `JSON`, permitiendo su lectura desde aplicaciones externas en Python o *Matlab*, donde las poses se representan mediante cuaterniones o ángulos de Euler. Este flujo de procesamiento puede observarse en la Figura 21.

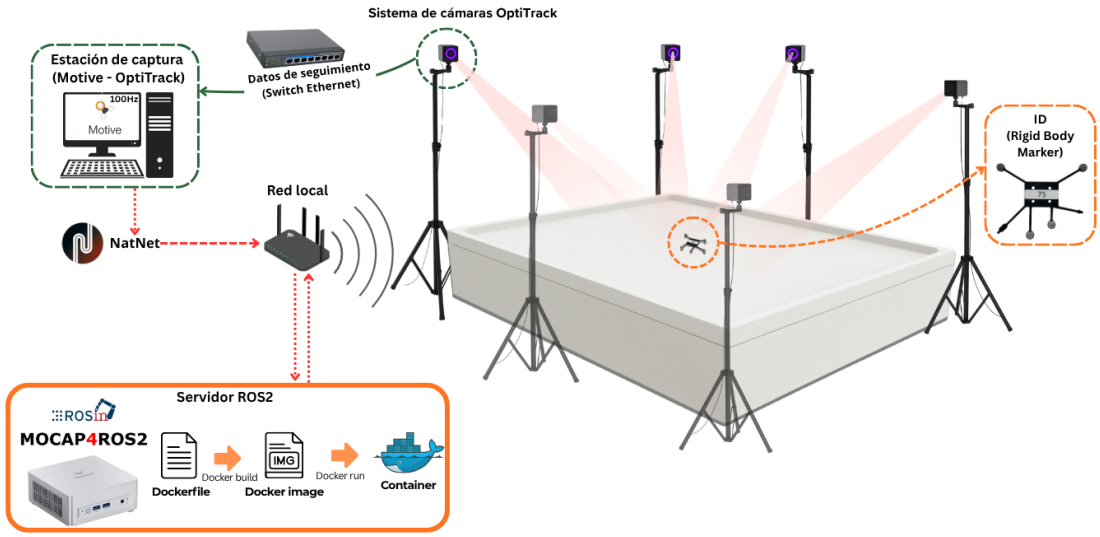
Figura 21. Flujo tradicional del ecosistema Robotat para la transmisión de datos de captura



Nota. Elaboración propia.

Por el contrario, MOCAP4ROS2 ofrece una integración más directa al emplear las librerías de NatNet para enlazar OptiTrack con ROS2, publicando automáticamente la información en tópicos nativos accesibles para cualquier nodo del sistema. Este enfoque elimina la necesidad de procesar datos `JSON`, habilitando un flujo más eficiente y mejor adaptado a aplicaciones de control en tiempo real. La Figura 22 ilustra el flujo empleado en este proyecto.

Figura 22. Integración de MOCAP4ROS2 con ROS2 mediante contenedores Docker



Nota. Elaboración propia.

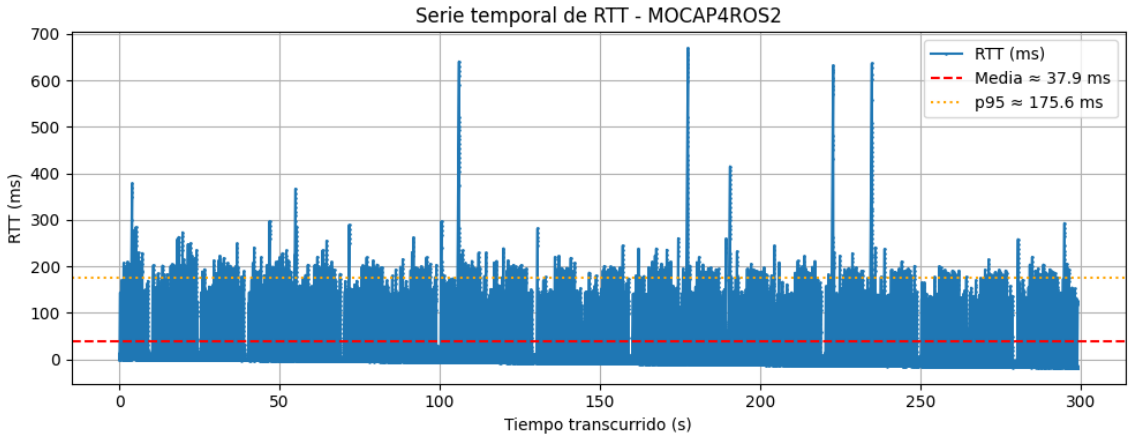
En ambos esquemas, el software *Motive* se configuró para transmitir datos a una tasa de 100 Hz, frecuencia que se tomó como referencia para evaluar el desempeño comparativo

entre ambos sistemas. A partir de esta configuración uniforme se analizan las métricas de latencia y frecuencia de publicación presentadas en las siguientes secciones.

8.2.1. Evaluación con MOCAP4ROS2

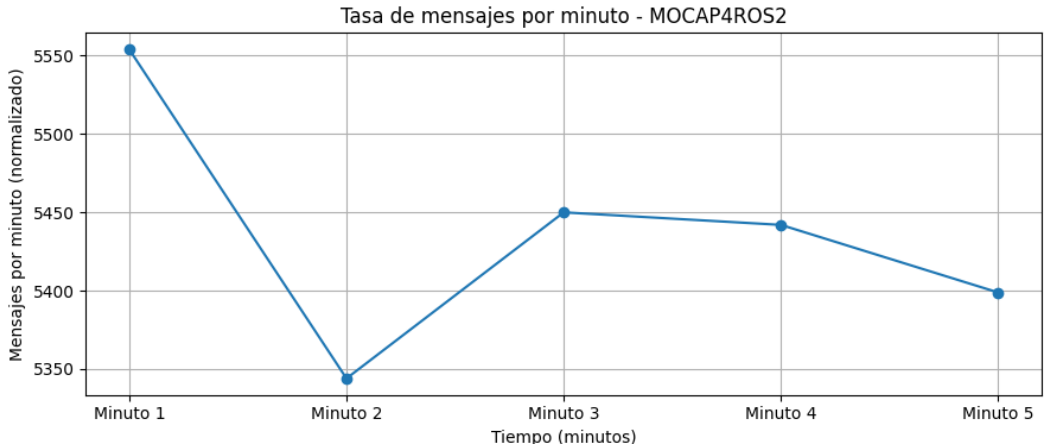
Para llevar a cabo la evaluación, se desarrolló un *script* en ROS2 encargado de suscribirse al tópico publicado por el servidor con Ubuntu, donde se encuentra instalado el paquete MOCAP4ROS2. Este servidor transmite en tiempo real la pose del *rigid body* seleccionado, permitiendo registrar y analizar las métricas de rendimiento. Los resultados obtenidos a partir de esta suscripción se presentan a continuación.

Figura 23. Serie temporal del RTT en el ecosistema Robotat



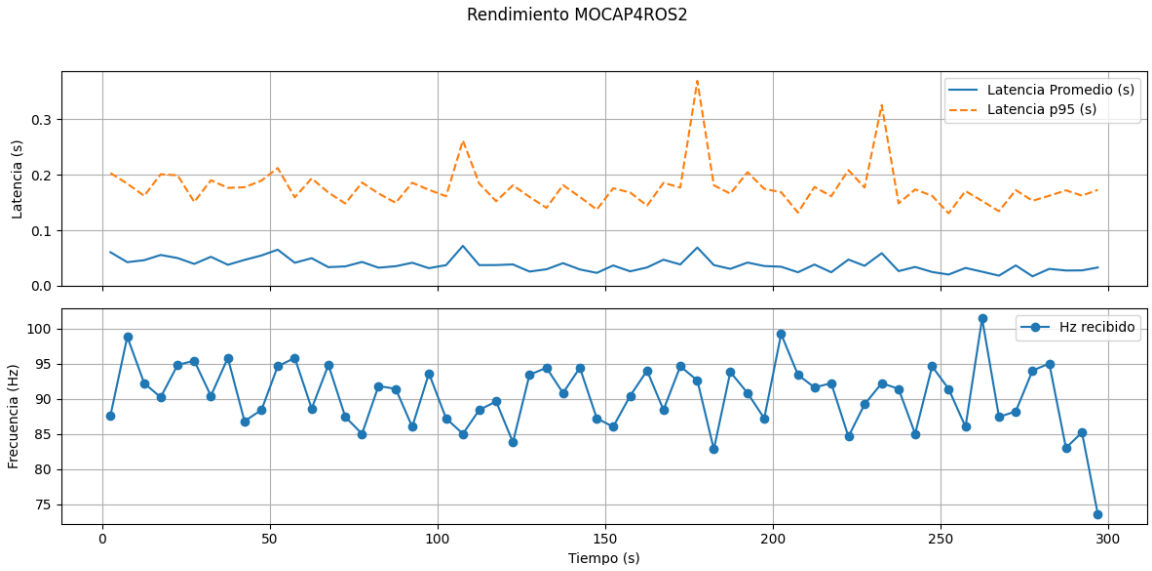
Nota. Elaboración propia.

Figura 24. Mensajes publicados por minuto en MOCAP4ROS2



Nota. Elaboración propia.

Figura 25. Latencia y frecuencia de publicación en MOCAP4ROS2



Nota. Elaboración propia.

Como se observa en la Figura 23, la serie temporal del *Round Trip Time* (RTT) permitió evaluar el tiempo de respuesta del sistema durante un período de 300 segundos. Los resultados muestran una latencia media de 37.9 ms, con un valor en el percentil 95 de aproximadamente 175.6 ms, lo que refleja un comportamiento estable con picos aislados, pero dentro de un rango aceptable para aplicaciones de localización en tiempo real.

Por otra parte, la Figura 24 presenta la tasa de publicación de mensajes por minuto. A lo largo de los cinco minutos de experimento, se observa que el sistema logró estabilizarse alrededor de un rango cercano a los 5400 mensajes por minuto, lo cual confirma la consistencia en el flujo de información y la robustez del canal de transmisión.

Finalmente, en la Figura 25 se detallan la latencia promedio y el percentil 95 junto con la frecuencia de publicación en Hz. Se evidencia que la latencia promedio se mantuvo baja y estable durante el periodo de observación, mientras que la frecuencia osciló alrededor de los 90 Hz, con ligeras variaciones. Estos resultados ponen de manifiesto que el sistema basado en MOCAP4ROS2 ofrece un desempeño confiable, garantizando tanto la baja latencia como la continuidad en la publicación de datos necesarios para la operación de plataformas robóticas.

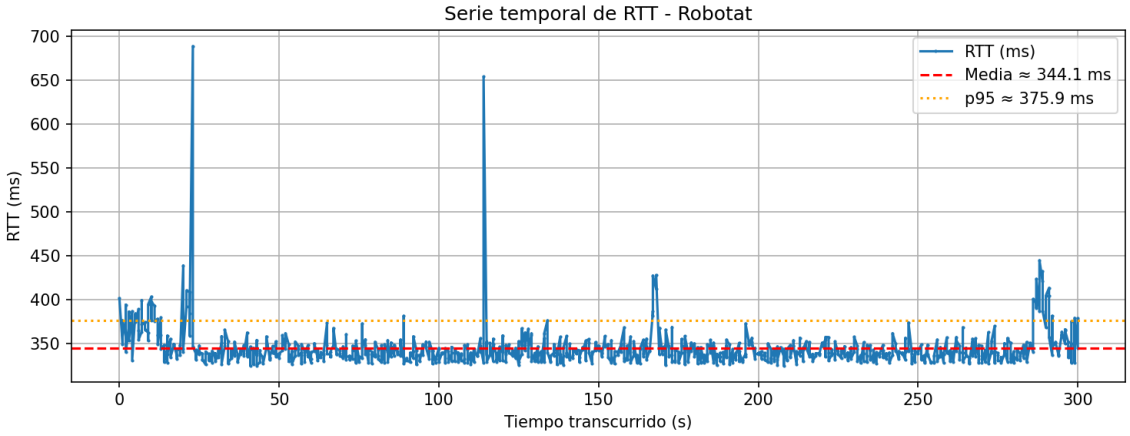
8.2.2. Evaluación con Robotat en MATLAB

Se realizaron las mismas pruebas de evaluación para el sistema de comunicación del ecosistema Robotat. En este caso, se empleó el software *MATLAB*, en el cual se desarrolló un programa encargado de suscribirse a la información publicada por el servidor de Robotat. Para ello, se configuró un *Marker ID* específico, junto con los parámetros de conexión correspondientes a la dirección IP local y al puerto de comunicación.

Además, se utilizaron funciones predefinidas que permiten establecer la conexión con el servidor y obtener la pose del marcador en tiempo real, expresada en coordenadas de posición (x, y, z) y orientación en cuaterniones (qx, qy, qz, qw).

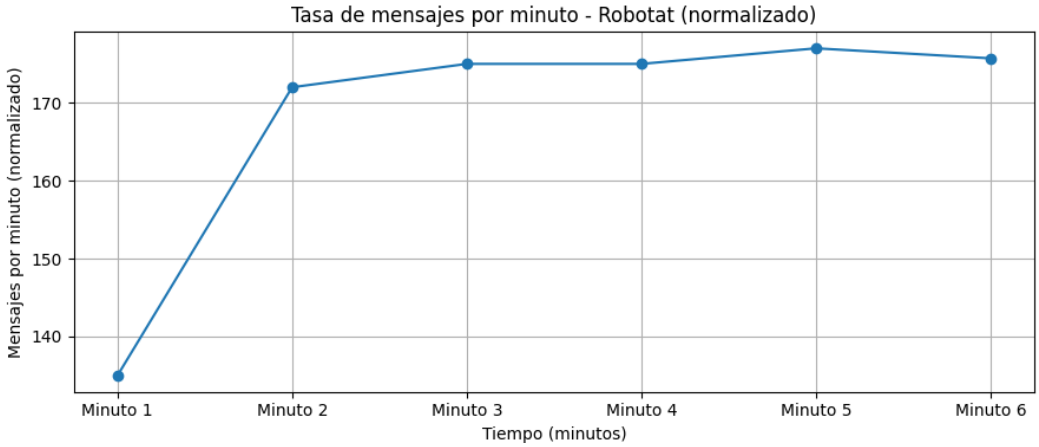
Los resultados obtenidos a partir de esta suscripción se procesaron para medir las métricas de latencia y frecuencia de publicación, los cuales se presentan en las figuras siguientes.

Figura 26. Serie temporal del RTT en el ecosistema Robotat



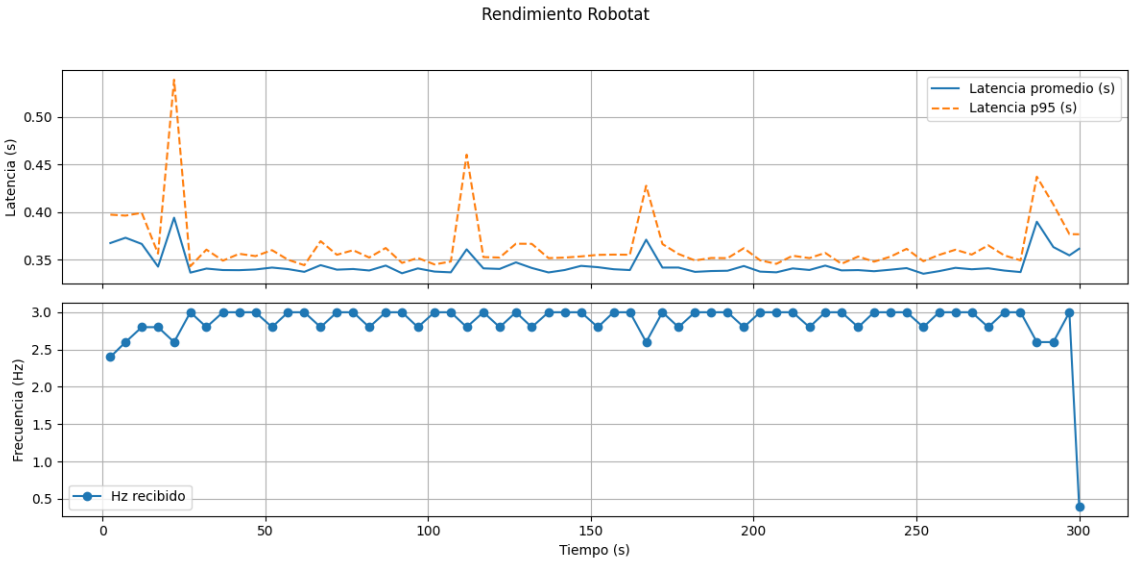
Nota. Elaboración propia.

Figura 27. Mensajes publicados por minuto en el ecosistema Robotat



Nota. Elaboración propia.

Figura 28. Latencia y frecuencia de publicación en el ecosistema Robotat



Nota. Elaboración propia.

En la Figura 26 se presenta la serie temporal del *Round Trip Time* (RTT) obtenida en Robotat durante un período de 300 segundos. Los resultados reflejan una latencia media de aproximadamente 344.1 ms, con un valor en el percentil 95 de 375.9 ms. Aunque se registraron picos aislados que alcanzaron los 680 ms, la tendencia general evidencia un comportamiento estable dentro de un rango elevado de latencia.

En relación con la frecuencia de publicación de mensajes, la Figura 27 muestra que el sistema alcanzó una estabilización progresiva alrededor de los 170 mensajes por minuto, tras un inicio con valores más bajos. Esto indica que, una vez establecida la conexión, el flujo de datos mantiene un comportamiento relativamente constante.

Finalmente, en la Figura 28 se ilustran de manera conjunta la latencia promedio, el percentil 95 y la frecuencia de publicación expresada en Hz. Los resultados reflejan que la frecuencia se mantuvo cercana a los 3 Hz durante la mayor parte de la prueba, lo cual pone de manifiesto las limitaciones del ecosistema Robotat en cuanto a la velocidad de actualización de datos para aplicaciones en tiempo real.

8.2.3. Comparación de resultados entre MOCAP4ROS2 y Robotat

Los resultados muestran una diferencia considerable entre ambos enfoques. El sistema basado en MOCAP4ROS2 alcanzó una latencia media de 37.9 ms, casi diez veces menor que la registrada en el ecosistema Robotat, cuyo valor promedio fue de 344.1 ms. De igual forma, en el percentil 95 se observa que MOCAP4ROS2 mantiene un desempeño más estable, mientras que Robotat presenta retardos superiores a los 370 ms, con picos aislados que superan los 600 ms.

En lo referente a la frecuencia de publicación, MOCAP4ROS2 sostuvo una tasa cercana a los 100 Hz, coherente con la configuración del sistema OptiTrack y adecuada para aplicaciones en tiempo real. Por el contrario, Robotat presentó una frecuencia promedio de apenas 3 Hz, lo cual limita su utilidad en escenarios que requieren actualizaciones rápidas, como el control de enjambres robóticos. Estos resultados, se resumen de forma comparativa en el Cuadro 3.

De esta forma la comparación evidencia que MOCAP4ROS2 ofrece un rendimiento significativamente superior tanto en latencia como en frecuencia de actualización, consolidándose como una alternativa más eficiente y adecuada para aplicaciones distribuidas en ROS2 frente al flujo tradicional empleado en Robotat.

Cuadro 3. Comparación de métricas de rendimiento entre MOCAP4ROS2 y Robotat

Métrica	**MOCAP4ROS2**	**Robotat**
Latencia media (ms)	37.9	344.1
Latencia p95 (ms)	175.6	375.9
Frecuencia (Hz)	∼90–100	∼3
Mensajes/minuto	∼5400	∼170

Nota. Elaboración propia.

En conjunto, estos resultados demuestran que MOCAP4ROS2 no solo mejora de manera drástica la latencia y la frecuencia de actualización, sino que también ofrece un flujo de datos más estable y adecuado para aplicaciones de control en tiempo real. Esta mejora en el rendimiento habilita nuevas posibilidades dentro del laboratorio, particularmente en escenarios donde la retroalimentación rápida es indispensable para el control seguro y coordinado de múltiples agentes.

A partir de este punto, y aprovechando las capacidades demostradas por MOCAP4ROS2, el siguiente capítulo aborda su integración directa con agentes robóticos dentro del ecosistema ROS2, evaluando cómo este flujo optimizado de información de movimiento puede emplearse en tareas de control punto a punto y experimentos iniciales con robots móviles.

CAPÍTULO 9

Implementación con agentes robóticos

La implementación con agentes robóticos representa la fase final del proyecto, en la cual se integró el sistema de captura basado en MOCAP4ROS2 directamente con plataformas móviles reales dentro del ecosistema Robotat. En esta etapa, el objetivo principal fue validar que los datos de pose provenientes de OptiTrack pueden ser utilizados de manera confiable por múltiples agentes en tiempo real, manteniendo coherencia temporal y estabilidad en la comunicación. Para ello, se emplearon robots Pololu 3Pi+ como unidades experimentales y se preparó la infraestructura necesaria para su futura interoperabilidad con CrazySwarm2, permitiendo así la transición hacia experimentos de enjambre mixto entre robots móviles y drones. Este capítulo presenta el proceso de integración, las pruebas preliminares de control y los experimentos multirrobot realizados para evaluar el comportamiento del sistema en escenarios colaborativos.

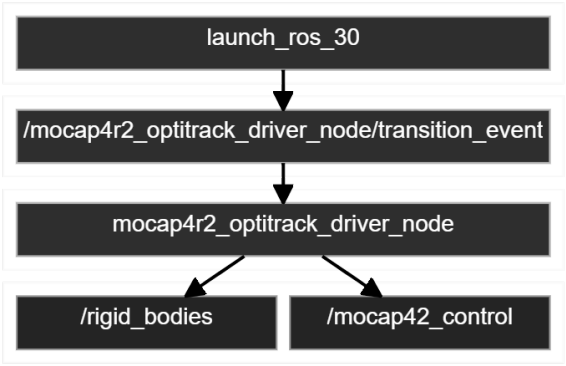
9.1. Integración de MOCAP4ROS2 con agentes robóticos

Para la validación del sistema de captura con MOCAP4ROS2, se seleccionó como plataforma experimental el robot móvil Pololu 3Pi+, empleado como agente robótico en las pruebas. Este robot cuenta con un sistema de control embebido y fue adaptado para recibir retroalimentación de su posición a partir de los datos publicados por MOCAP4ROS2. En particular, se desarrolló un nodo en ROS2 encargado de publicar la pose de un *rigid body* específico mediante un tópico, al cual el robot se suscribió para integrar esta información en su esquema de control. De esta manera, se estableció un flujo de comunicación en el que el sistema de captura proporciona en tiempo real la retroalimentación necesaria para la ejecución de tareas de navegación y control de movimiento.

Como parte de la integración, se generó el grafo de nodos y tópicos mediante la herramienta `rqt_graph`, lo cual permite visualizar la arquitectura de comunicación establecida por

el paquete MOCAP4ROS2. En la Figura 29 se aprecia que el nodo principal `/mocap4r2_optitrack_driver_node` publica la información de los cuerpos rígidos en el tópico `/rigid_bodies`, además de gestionar eventos de transición y comunicación con otros nodos auxiliares. Esta representación facilita comprender cómo se distribuyen los datos capturados desde OptiTrack hacia los diferentes nodos.

Figura 29. Visualización de nodos y tópicos de MOCAP4ROS2

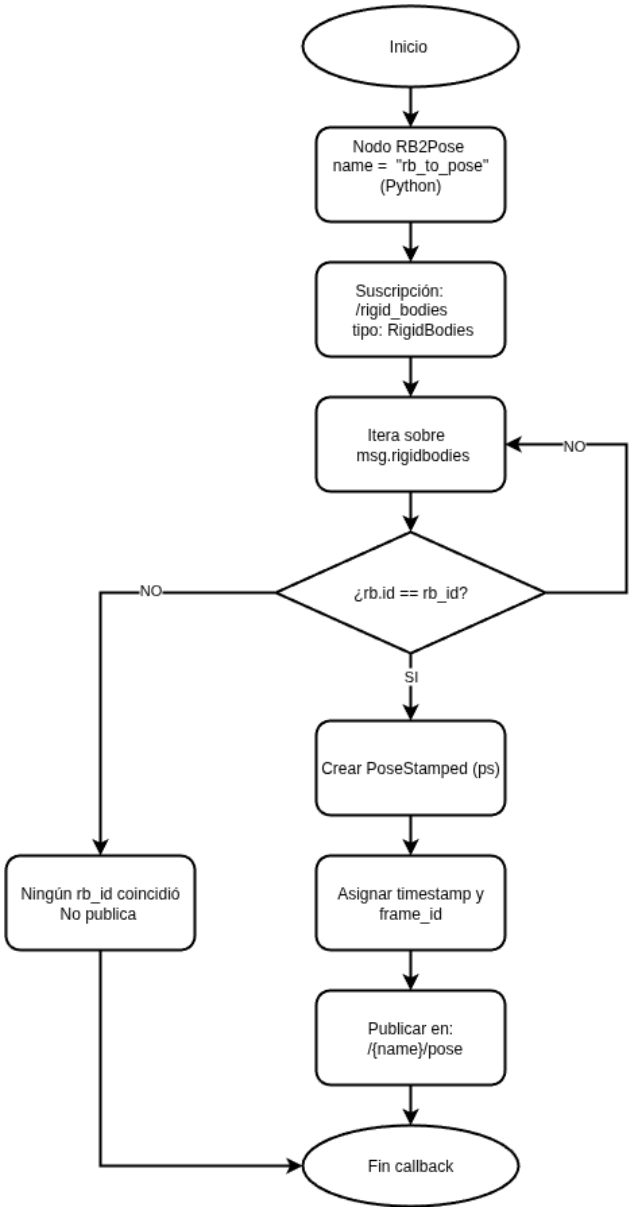


Nota. Elaboración propia.

9.2. Prueba preliminar de control punto a punto

Como etapa inicial se desarrolló un nodo denominado `rb_to_pose`, encargado de procesar la información proveniente del sistema de captura. En la Figura 30 se ilustra el diagrama de flujo de este nodo, donde se observa cómo se suscribe al tópico `/rigid_bodies`, itera sobre los identificadores de los *rigid bodies* y selecciona aquel correspondiente al *ID* configurado. Una vez identificado, se genera un mensaje de tipo `PoseStamped` que incluye la posición y orientación del marcador en el espacio, además de un `timestamp` y `frame_id` para garantizar la coherencia temporal. Dicho mensaje es posteriormente publicado en un tópico independiente con el formato `{name}/pose`, de manera que pueda ser consumido por otros nodos dentro del ecosistema ROS2.

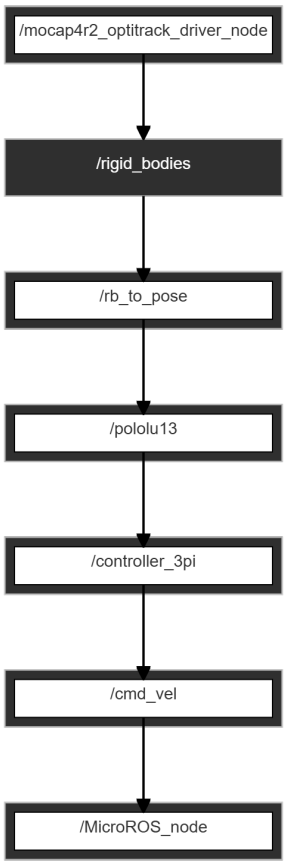
Figura 30. Diagrama de flujo del nodo `rb_to_pose`



Nota. Elaboración propia.

En la Figura 31 se presentan los resultados de esta integración, mostrando el grafo de nodos y tópicos generado en ROS2. Allí se visualiza la correcta publicación de la pose del *rigid body* en el tópico asignado al agente Pololu (`/pololu13/pose`), así como la existencia de un tópico adicional denominado `/Meta`, que agrupa la información de un conjunto de marcadores. Finalmente, se aprecia la conexión exitosa entre el tópico de control del Pololu y el nodo `/MicroROS_node`, encargado de ejecutar las instrucciones de movimiento sobre el agente físico.

Figura 31. Asignación de tópicos en ROS2



Nota. Elaboración propia.

9.3. Enjambre robótico

Una vez validada la integración individual entre MOCAP4ROS2 y el robot móvil Pololu 3Pi+, se procedió a evaluar el comportamiento del sistema en escenarios con múltiples agentes operando simultáneamente. El propósito de esta etapa fue analizar la estabilidad del flujo de datos, el seguimiento consistente de varios *rigid bodies* y la distribución correcta de la información hacia distintos nodos de control en ROS2, condiciones esenciales para la ejecución de comportamientos colaborativos en un enjambre robótico.

Aunque la retroalimentación principal de posición provino directamente de MOCAP4ROS2, se integró de forma auxiliar un canal basado en un puente *MQTT–ROS2*, utilizado únicamente para tareas secundarias de comunicación. No obstante, MOCAP4ROS2 permaneció como la fuente prioritaria y más confiable de pose, garantizando coherencia temporal y baja latencia en todos los agentes del enjambre.

Con el fin de evaluar la escalabilidad del sistema, se diseñaron tres experimentos en los que se modificó tanto el número de metas como la distribución de robots. Estos escena-

rios permitieron analizar el comportamiento del sistema de captura y de los controladores embarcados cuando varios agentes reciben retroalimentación en tiempo real.

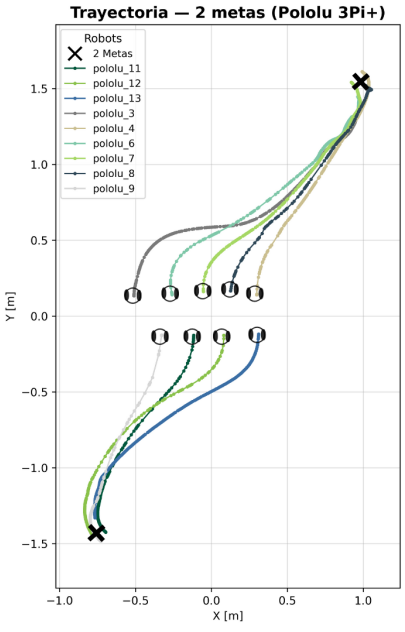
En el **primer experimento**, nueve Pololu 3Pi+ fueron divididos en dos grupos: cinco se dirigieron hacia una primera meta y cuatro hacia una segunda. Las trayectorias obtenidas, mostradas en la Figura 32, evidencian que los agentes lograron converger correctamente hacia sus objetivos asignados, manteniendo una navegación estable durante todo el recorrido.

En el **segundo experimento**, se establecieron tres metas distintas, distribuyendo a los nueve Pololu 3Pi+ en tres grupos de tres. La Figura 33 presenta las trayectorias de este escenario, donde se observa la separación espacial de los grupos y la ausencia de interferencias significativas entre ellos, lo cual refleja un procesamiento adecuado de múltiples *rigid bodies* en paralelo.

Finalmente, en el **tercer experimento** se consideraron cuatro metas, asignando tres agentes a una de ellas y dos a cada una de las restantes. Como se ilustra en la Figura 34, los robots alcanzaron sus destinos manteniendo trayectorias suaves y consistentes, demostrando que la carga adicional no deteriora el rendimiento de MOCAP4ROS2 ni su capacidad para alimentar simultáneamente a varios controladores distribuidos.

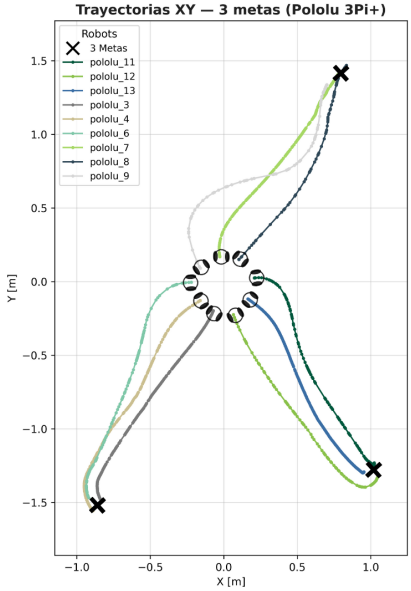
Estos ejercicios validan que MOCAP4ROS2 puede mantener un flujo de datos estable incluso cuando múltiples robots dependen simultáneamente de la retroalimentación de pose para ejecutar acciones coordinadas. En conjunto, los resultados obtenidos refuerzan el potencial del sistema para aplicaciones de enjambres robóticos en entornos controlados y sientan las bases para futuras pruebas con comportamientos más complejos.

Figura 32. Trayectorias de los robots Pololu 3Pi+ en el escenario de 2 metas



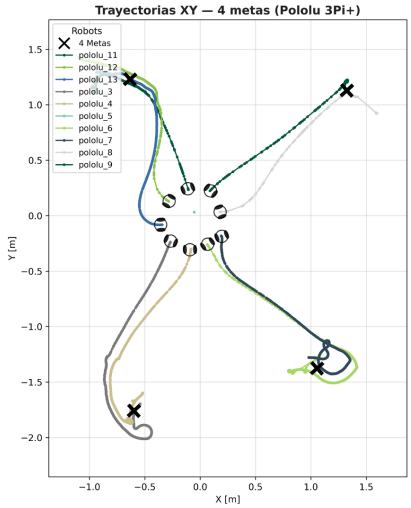
Nota. Elaboración propia.

Figura 33. Trayectorias de los robots Pololu 3Pi+ en el escenario de 3 metas



Nota. Elaboración propia.

Figura 34. Trayectorias de los robots Pololu 3Pi+ en el escenario de 4 metas

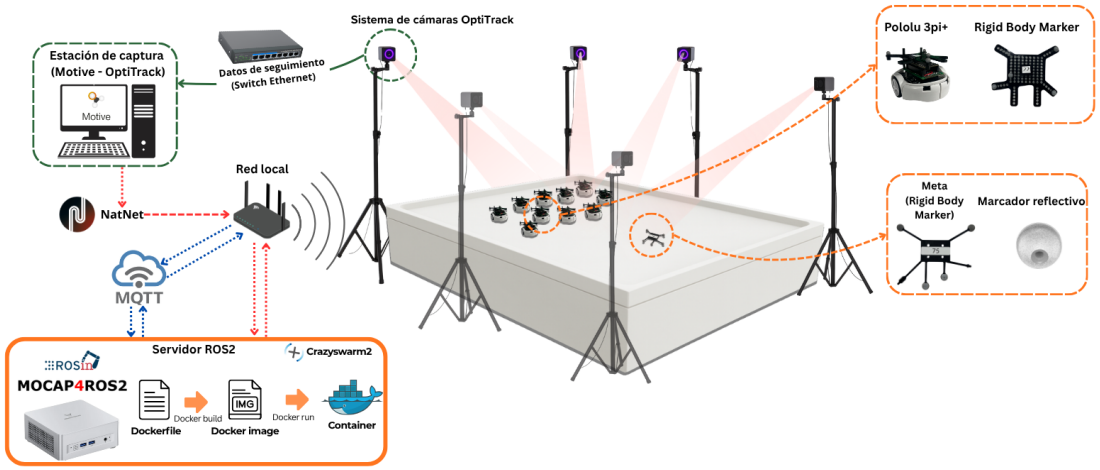


Nota. Elaboración propia.

La integración completa del sistema se resume en la Figura 35, donde se observa el flujo consolidado entre el sistema de captura OptiTrack, el protocolo NatNet, la infraestructura de ROS2 y el entorno de ejecución basado en contenedores Docker que aloja tanto MOCAP4ROS2 como CrazySwarm2. Este ecosistema unificado permitió establecer una comunicación estable y de baja latencia entre la captura de movimiento y los agentes robóticos, habilitando el control distribuido y la coordinación de múltiples Pololu 3Pi+ en tiempo real.

Con ello, se cumplió de manera efectiva cada uno de los objetivos planteados: (i) la configuración y despliegue de un servidor robusto para ROS2, (ii) la evaluación y comparación exhaustiva del rendimiento de MOCAP4ROS2 frente al flujo tradicional de Robotat, y (iii) la validación experimental mediante la integración de varios agentes robóticos operando en escenarios de enjambre. En conjunto, estos resultados demuestran la viabilidad técnica del nuevo ecosistema y sientan las bases para futuras extensiones hacia comportamientos cooperativos más complejos.

Figura 35. Ecosistema Robotat integrado con ROS2, MOCAP4ROS2 y CrazySwarm2



Nota. Elaboración propia.

CAPÍTULO 10

Conclusiones

- El servidor implementado permitió establecer una infraestructura robusta para la integración de ROS2 con sistemas de captura de movimiento, facilitando el desarrollo de aplicaciones orientadas al control y la robótica de enjambre. La incorporación de *Docker* aseguró la portabilidad del entorno, garantizando que la configuración pueda reproducirse en distintos escenarios sin pérdida de compatibilidad. Asimismo, la integración con Crazyswarm2 refuerza el potencial del laboratorio al habilitar experimentos colaborativos con enjambres de drones, ampliando las posibilidades de investigación y de aplicación en proyectos de sistemas autónomos.
- La evaluación comparativa demostró que MOCAP4ROS2 ofrece un rendimiento significativamente superior al flujo tradicional de Robotat. En las pruebas realizadas, MOCAP4ROS2 alcanzó una latencia media de 37.9 ms y un p95 de 175.6 ms, mientras que Robotat registró 344.1 ms en promedio y un p95 de 375.9 ms, con picos superiores a los 600 ms. Asimismo, el sistema basado en ROS2 mantuvo una frecuencia cercana a 90–100 Hz, en contraste con los $\sim$3 Hz de Robotat, limitados por su arquitectura en *MATLAB* y el uso de `JSON` sobre *UDP*. Estas métricas confirman que MOCAP4ROS2 reduce drásticamente la latencia y mejora la estabilidad del flujo de datos, requisitos esenciales para aplicaciones de control en tiempo real.
- La integración de MOCAP4ROS2 con los robots móviles Pololu 3Pi+ y la infraestructura preparada para CrazySwarm2 permitió validar un entorno distribuido, sincronizado y escalable para experimentos multirrobot. Durante las pruebas, el sistema sostuvo retroalimentación cercana a 100 Hz, posibilitando trayectorias estables con un enjambre de hasta nueve Pololus operando simultáneamente. Además, la compatibilidad con CrazySwarm2 confirma que el entorno puede extenderse para controlar hasta diez drones Crazyflie 2.1 con la coherencia temporal necesaria. Los experimentos con múltiples metas mostraron un flujo de datos estable sin degradación relevante, evidenciando que la arquitectura basada en ROS2 y MOCAP4ROS2 es adecuada para aplicaciones de enjambre y futuros escenarios con un mayor número de agentes autónomos.

CAPÍTULO 11

Recomendaciones

- Integrar un monitor secundario al servidor para visualizar en tiempo real las interfaces de RViz y los tópicos de los *rigid bodies*, evitando depender de una estación de trabajo externa y facilitando la supervisión de pruebas.
- Diseñar una interfaz gráfica ligera que permita observar métricas clave (latencia, frecuencia de publicación, pérdida de paquetes) durante las pruebas de integración con agentes robóticos.

CAPÍTULO 12

Referencias

[1] F. Sanabria, «Diseño e implementación de una plataforma de pruebas para sistemas de control para el dron Crazyflie 2.0,» Tesis de licenciatura, Universidad del Valle de Guatemala, 2022.

[2] C. Perafán, «Robotat: un ecosistema robótico de captura de movimiento y comunicación inalámbrica,» Tesis de licenciatura, Universidad del Valle de Guatemala, 2022.

[3] D. Mencos, «Integración de una computadora central en el Rover UVG para la ejecución de ROS,» Tesis de licenciatura, Universidad del Valle de Guatemala, 2023.

[4] M. Project, *MOCAP4ROS2 - Modular Framework for Motion Capture Integration in ROS2*, `https://github.com/MOCAP4ROS2-Project`, Repositorio GitHub. Último acceso: abril 2025, 2020.

[5] J. García, «Adaptación del sistema de drones Crazyswarm al ecosistema Robotat,» Tesis de licenciatura, Universidad del Valle de Guatemala, 2023.

[6] B. Garrido, «Levantamiento de una plataforma de pruebas para sistemas multidrones con OptiTrack y Crazyswarm,» Tesis de licenciatura, Universidad del Valle de Guatemala, 2024.

[7] S. Macenski, T. Foote, B. Gerkey, C. Lalancette y W. Woodall, «Robot Operating System 2: Design, architecture, and uses in the wild,» *Science Robotics*, vol. 7, n.º 66, eabm6074, 2022. DOI: `10.1126/scirobotics.abm6074`. dirección: `https://www.science.org/doi/abs/10.1126/scirobotics.abm6074`.

[8] S. Susnjara e I. Smalley. «¿Qué es Docker?» IBM Think. Consultado: 11 de septiembre de 2025. dirección: `https://www.ibm.com/mx-es/think/topics/docker`.

[9] S. Susnjara e I. Smalley. «Máquinas virtuales.» Consultado: 24 de septiembre de 2025. dirección: `https://www.ibm.com/mx-es/think/topics/virtual-machines`.

[10] N. S. Gill. «Docker container architecture and monitoring for enterprises.» Consultado: 24 de septiembre de 2025. dirección: `https://www.xenonstack.com/blog/docker-container`.

[11] Docker Inc. «Docker Overview.» Consultado: 24 de septiembre de 2025. dirección: `https://docs.docker.com/get-started/overview/`.

[12] OptiTrack. «OptiTrack Documentation.» Consultado: 11 de septiembre de 2025. dirección: https://docs.optitrack.com.

[13] OptiTrack. «PrimeX 41 Camera.» Consultado: 5 de junio de 2025. dirección: https://optitrack.com/cameras/primex-41.

[14] OptiTrack. «NatNet SDK.» Consultado: 22 de septiembre de 2025. dirección: https://optitrack.com/software/natnet-sdk.

[15] OptiTrack. «Motive: Software de captura de movimiento óptico.» Consultado: 20 de septiembre de 2025. dirección: https://optitrack-com.translate.goog/software/motive/?_x_tr_sl=en&_x_tr_tl=es&_x_tr_hl=es&_x_tr_pto=tc.

[16] F. Martin, J. M. Guerrero, A. A. Garcia, F. Rodriguez y V. Matellan, «MOCAP4ROS2: An Open Source Framework for Motion Capture Systems in Robotics,» en *Proceedings of the 18th International Symposium on Open Collaboration*, ép. OpenSym '22, Madrid, Spain: Association for Computing Machinery, 2022, ISBN: 9781450398459. DOI: 10.1145/3555051.3555076. dirección: https://doi.org/10.1145/3555051.3555076.

[17] Canonical Ltd. «Ubuntu - Alternative downloads.» Consultado: 11 de septiembre de 2025. dirección: https://ubuntu.com/download/alternative-downloads.

[18] Open Robotics, *Installing ROS2 Humble on Ubuntu (Official Guide)*, Accessed: 2025-06-06, 2023. dirección: https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debs.html.

[19] Docker Inc. «Get Docker.» Consultado: 26 de septiembre de 2025. dirección: https://docs.docker.com/get-docker/.

[20] Open Robotics. «Setup ROS2 with VS Code and Docker Container.» Consultado: 27 de septiembre de 2025. dirección: https://docs.ros.org/en/humble/How-To-Guides/Setup-ROS-2-with-VSCode-and-Docker-Container.html.

[21] *MINISFORUM UM870 Slim Mini PC*, https://minisforumpc.eu/products/um870-slim-mini-pc?srsltid=AfmBOop1CDrjSVhhcRwBIhg7DrmBZJPh30chg1zBn7YXGtcRkPxFbQBO&variant=51700059177326, Accedido: 29 de septiembre de 2025, 2025.

CAPÍTULO 13

Anexos

En este capítulo se reúne material complementario que apoya el desarrollo del proyecto y proporciona al lector los recursos necesarios para reproducir la infraestructura diseñada. Los anexos incluyen documentación técnica, archivos de configuración, contenedores y herramientas utilizadas durante la integración de MOCAP4ROS2, Docker y CrazySwarm2 dentro del ecosistema Robotat. Su finalidad es presentar esta información sin interrumpir la continuidad del cuerpo principal del informe.

El repositorio asociado al proyecto contiene:

- El manual de uso del entorno basado en Docker, incluyendo instrucciones de construcción, ejecución y depuración de contenedores para ROS2.
- Los archivos de configuración del espacio de trabajo ROS2 empleado para MOCAP4ROS2.
- El contenedor preparado para CrazySwarm2, junto con su `Dockerfile`, `entrypoint.sh` y dependencias asociadas.

13.1. Repositorio GitHub y Manual de usuario

Este anexo presenta el repositorio oficial del proyecto, el cual centraliza todo el material técnico empleado: manuales, contenedores, archivos de configuración y herramientas utilizadas en las etapas de instalación, integración y experimentación con agentes robóticos.

El repositorio está disponible en el siguiente enlace:

`https://github.com/ThomasJabes/robotat-infraestructura-ros2.git`