Adaptación del sistema de drones Crazyswarm al ecosistema Robotat

Julio Andrés Avila García-Salas

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Adaptación del sistema de drones Crazyswarm al ecosistema Robotat

Trabajo de graduación presentado por Julio Andrés Avila García-Salas
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2023

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Adaptación del sistema de drones Crazyswarm al ecosistema Robotat

Trabajo de graduación presentado por Julio Andrés Avila García-Salas
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2023

Vo.Bo.:

(f) 

Msc. Miguel Zea

Tribunal Examinador:

(f) 

Msc. Miguel Zea

(f) 

Ing. Christopher Chiroy

(f) 

Ing. Jonathan Mansilla

Fecha de aprobación: Guatemala, 13 de enero de 2024

Prefacio

El ámbito aeroespacial y los sistemas autónomos han sido por años mis áreas de interés, lo que me motivó a lograr una integración entre el laboratorio de robótica y los drones Crazyflie, para que futuras promociones de las carreras de Ingeniería Mecatrónica, Electrónica y Biomédica puedan utilizar dichos drones en sus cursos de la carrera. De esta forma, se estará revolucionando en el campo del control de drones y la robótica de enjambre en la Universidad del Valle de Guatemala y en el país.

Este proyecto fue asesorado por el Msc. Miguel Zea, a quien agradezco por darme la oportunidad de trabajar en el desarrollo del Robotat, por su asesoramiento y apoyo a lo largo del año y por motivarme a dedicarme a la investigación de sistemas autónomos. Junto a él, agradezco al PhD. Luis Rivera y al Ing. Kurt Kellner por su apoyo y retroalimentación en este proceso y por sus enseñanzas a lo largo de mi carrera.

Agradezco especialmente a mis padres y mi hermana por apoyar mi carrera todos estos años, motivarme a alcanzar la excelencia en todos los ámbitos y acompañarme en todos los aspectos de mi vida, llevándome a la culminación de mi carrera.

Índice

Lista de figuras

Lista de cuadros

Resumen

Para este trabajo se tenía como principal objetivo lograr una integración entre un conjunto de drones Crazyflie 2.1 y el ecosistema Robotat, de esta forma cualquier usuario puede desarrollar aplicaciones en el entorno. Se realizó esta integración mediante el sistema Crazyswarm, el cual funciona a través de ROS2 en Linux. Con el objetivo de que este sistema de drones pueda utilizarse en prácticas de laboratorio y otras líneas de investigación en la Universidad, se adaptó la infraestructura de este sistema al ecosistema Robotat a través de los paquetes y funciones de ROS2. Esto se realizó mediante la obtención de paquetes de información TCP/IP generados por el sistema de captura de movimiento del Robotat y la creación de nuevos paquetes para controlar los drones desde Matlab. Esto seguido del desarrollo de un servidor en Python que funciona como intermediario para la comunicación TCP/IP acoplándose al servidor existente del Robotat. Gracias al desarrollo de este servidor cualquier computadora con Matlab puede controlar los drones al conectarse a la red Robotat y diseñar trayectorias de una forma más eficiente. Para validar los resultados se realizaron análisis de error cuadrático medio para las trayectorias, donde se concluyó que pueden realizarse de forma física con un error cuadrático medio de 0.0437 metros si se utiliza un periodo de muestreo de 1 segundo.

Abstract

For this project, the main objective was to achieve integration between a set of Crazyflie 2.1 drones and the Robotat ecosystem, allowing any user to develop applications within this environment. This integration was accomplished using the Crazyswarm system, which operates through ROS2 on Linux. With the goal of making this drone system usable in laboratory practices and other research areas at the University, the infrastructure of this system was adapted to the Robotat ecosystem through ROS2 packages and functions. This was achieved by obtaining TCP/IP information packets generated by the Robotat motion capture system and creating new packets to control the drones from Matlab. This was followed by the development of a Python server that serves as an intermediary for TCP/IP communication, integrating with the existing Robotat server.

Thanks to the development of this server, any computer with Matlab can control the drones by connecting to the Robotat network and design trajectories more efficiently. To validate the results, mean square error analyses were conducted for the trajectories, which concluded that physical trajectories can be executed with a mean square error of 0.0437 meters when using a sampling period of 1 second.

CAPÍTULO 1

Introducción

La Universidad del Valle de Guatemala cuenta con un laboratorio de Robótica denominado Robotat, el cual cuenta con un sistema de captura de movimiento y se trabajan líneas de investigación relacionadas con sistemas de control y robótica, además de ser el espacio donde se realizan las prácticas de laboratorio de los cursos de robótica. Está diseñado para realizar pruebas con agentes autónomos, entre ellos robots humanoides, robots móviles con ruedas, brazos robóticos y drones. Entre los drones a utilizar se encuentran los drones Crazyflie 2.1, los cuales son de tamaño pequeño y pueden emplearse en conjunto para controlar enjambres de drones.

La siguiente tesis utiliza el ecosistema Robotat en la Universidad del Valle de Guatemala y su capacidad de implementar un grupo de múltiples nanocópteros para control de enjambre, siendo una gran oportunidad para que los estudiantes aprendan a controlar dichos drones. Se presentan inicialmente las pruebas de vuelo sin un sistema de captura de movimiento, seguido de la implementación de este junto con Crazyswarm con los resultados finales de la infraestructura, se muestra un análisis de error cuadrático medio para comparar las posiciones definidas y las obtenidas en las pruebas físicas. Posteriormente se muestra la creación y funcionamiento del servidor Crazyswarm-Robotat que permite la comunicación con los drones y Matlab junto con las pruebas realizadas y el análisis de trayectorias, de igual forma se muestran lo análisis basados en el error cuadrático medio y porcentajes de error para validar que las trayectorias realizadas se asemejan a las diseñadas en Matlab.

CAPÍTULO 2

Antecedentes

Los drones Crazyflie se han utilizado en líneas de investigación relacionadas con sistemas de control y robótica de enjambre debido a su extensa documentación y versatilidad para desarrollo de algoritmos. Para poder realizar un control eficiente de múltiples agentes, es necesario implementar un sistema que sea capaz de obtener y procesar información acerca del comportamiento de los agentes en tiempo real.

2.1 Crazyflie 2.1

Los drones a trabajar durante esta línea de investigación son los Crazyflies 2.1, los cuales son cuadricópteros de tamaño pequeño, que pueden ser controlados mediante un sistema de control de enjambre llamado Crazyswarm, el cual es utilizado mediante ROS en Ubuntu.

Anteriormente, se trabajó con estos drones en una línea de investigación pero no en forma de enjambre, sino con un drone únicamente. Se desarrollaron herramientas para el uso individual de estos drones. En primer lugar, se comprobó la compatibilidad de comunicación entre un drone y Python para el análisis de datos, estos se guardan en bruto y se recomendó graficarlos posteriormente mediante Matlab. Se desarrolló una interfaz gráfica para prácticas de laboratorio, en la cual, se pudo observar el comportamiento del drone ante las propiedades de control determinadas, en este caso, un controlador PID y sus respectivas constantes. Cabe destacar que para poder hacer esta herramienta fue necesario plantear un modelo matemático a través del identificador de sistemas de Matlab. Como resultado de estas herramientas, se llegaron a planificar 2 prácticas de laboratorio para los cursos de Sistemas de Control [1].

2.2 Ecosistema Robotat

El objetivo principal es utilizar este enjambre de drones en el ecosistema Robotat, el cual es un entorno tecnológico ubicado en el laboratorio de Robótica CIT-116 de la Universidad del Valle de Guatemala. [2]Se implementó un sistema de captura de movimiento mediante

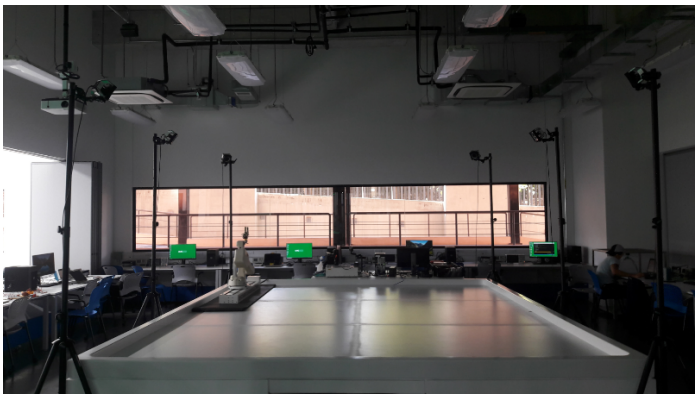

Figura 1: Ecosistema Robotat.
[2]

un set de 6 cámaras de la marca OptiTrack, las cuales están conectadas a un switch que se comunica con una computadora mediante UDP, también fue implementado un protocolo MQTT para establecer una comunicación Wi-Fi. La información recopilada es procesada mediante un algoritmo de Python y luego es enviada a un router que genera una red Wi-Fi local, a la cual es posible conectar diferentes dispositivos y obtener datos como posiciones lineales y angulares. Las cámaras están colocadas de tal forma que rodean una tarima hecha de concreto y acero, como se muestra en la Figura 1.

2.3 Crazyswarm y ROS2

ROS es un sistema operativo para sistemas robóticos, el cual provee funciones especiales que facilitan el análisis y control en proyectos de robótica. Este suele trabajarse en sistemas basados en Linux como Ubuntu. A través de ROS2 puede trabajarse Crazyswarm, el cual es un sistema para controlar enjambres de drones Crazyflie 2.1, además se cuenta con un cliente llamado Bitcraze, el cual está basado en Python y permite la comunicación entre el servidor y los drones mediante telecomunicación por radiofrecuencia, este es un sistema independiente de ROS. El entorno Crazyswarm está siendo reemplazado por la versión 2, a la cual se le han estado trabajando mejoras.

El posicionamiento de los drones se ha realizado de múltiples maneras en distintas líneas de investigación, entre la documentación oficial de Crazyswarm puede encontrarse información sobre pruebas realizadas en sistemas de captura de movimiento, tales como OptiTrack. Para poder realizar experimentos con OptiTrack es necesario implementar un módulo con marcadores reflectivos y este debe estar diseñado a la medida para que el drone pueda ser detectado por las cámaras. Otros métodos utilizados para el posicionamiento de drones es mediante un dispositivo Kinect de la marca Microsoft, utilizado en consolas Xbox [3].

Entre los centros de investigación que han utilizado Crazyswarm en forma de "enjambre", está el Laboratorio ACT de la *University of Southern California*, donde se implementó un control de 49 Crazyflies con un sistema de captura de movimiento, creando rutinas de trayectorias como en la Figura 2. Por otro lado, la Escuela en Ciencias de la Computación de la Universidad de McGill ha orientado el control de Crazyswarm para un solo drone de

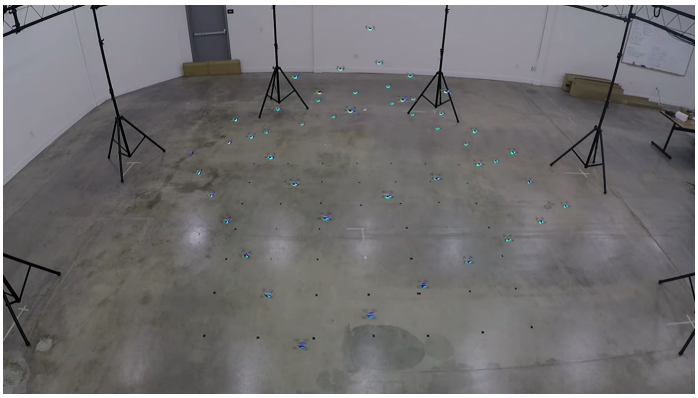

Figura 2: Investigación en ACT.
[4]

Figura 3: Investigación en McGill.
[5]

modo que pueda usarse el posicionamiento para realizar pinturas, como se ve en la Figura 3.

CAPÍTULO 3

Justificación

Los drones son dispositivos que han ganado popularidad en los últimos años por una característica especial, ya que al modelarlos como un sistema dinámico se presenta una dinámica no lineal, pero pueden ser controlados de forma satisfactoria con un controlador lineal, siendo este un control relativamente sencillo. Esto convierte a los drones en un excelente método de aprendizaje para diseñar y analizar modelos y algoritmos de control. Debido a esto, trabajar físicamente con un drone o inclusive, con un conjunto de estos, es una alternativa para poner en práctica los aprendizajes adquiridos a lo largo de una carrera universitaria relacionada con electrónica y para realizar investigación relacionada con sistemas de control clásico, moderno y robótica de enjambre. Para poder realizar esto, es necesario contar con un medio que le provea a los algoritmos de control la información necesaria para trabajar, incluyendo, pero no limitándose, a las posiciones espaciales y orientaciones de los respectivos drones. Debido a que se cuenta con un entorno que es capaz de capturar esta información física, el camino a tomar es la creación de un intermediario que permita la comunicación entre el sistema de captura y los algoritmos de control para drones.

El ecosistema Robotat fue creado como un entorno tecnológico que sirviera como un lugar de aprendizaje de sistemas robóticos y para desarrollar lineas de investigación relacionadas con sistemas de control y robótica. Para culminar el desarrollo de este entorno, es necesario integrar a todos los agentes autónomos que formarán parte de él, siendo los drones Crazyflie una parte importante del ecosistema.

CAPÍTULO 4

Objetivos

4.1 Objetivo general

Adaptar el sistema de control de drones Crazyswarm al ecosistema Robotat mediante ROS2 y sus respectivos paquetes y funciones.

4.2 Objetivos específicos

- Emplear la información recibida por las cámaras de captura OptiTrack en el ecosistema Robotat para definir algoritmos de control del sistema de drones.
- Crear algoritmos que conviertan información de origen TCP/UDP a radio frecuencia para establecer una comunicación directa con Crazyswarm.
- Levantar la infraestructura de Crazyswarm y desarrollar pruebas iniciales de vuelo.

CAPÍTULO 5

Alcance

Este proyecto tiene como alcance implementar la infraestructura de la versión 2 de Crazyswarm, utilizando las librerías y funciones disponibles de ROS2 y Crazyswarm 2 sin modificarlas o desarrollar nuevas. Además se buscó desarrollar un intermediario que permitiera la comunicación entre los drones y los usuarios del laboratorio, estableciendo TCP/IP como protocolo de comunicación a utilizar para acoplarse a las herramientas ya existentes en el laboratorio y expandir dichas herramientas mediante la mejora del servidor Robotat. Se aprovechó el espacio disponible en el Robotat considerando que no es factible movilizar las cámaras del sistema de captura de movimiento a otro espacio, por lo que las pruebas realizadas se limitaron a un espacio de $3.8 \times 4.8 \times 1.5$ metros. Por último se pretendía realizar pruebas con los distintos controladores disponibles para los Crazyflies 2.1, de los cuales se contaba únicamente con 10, esto se delimitó al uso de dichos controladores sin abordar directamente en el diseño o el análisis matemático de estos.

CAPÍTULO 6

Marco teórico

6.1. Crazyflie

Los drones Crazyflie son micro drones de dimensiones significativamente pequeñas (92x92x29 mm) y una masa de 27 g, cuentan con cuatro motores con un diseño simétrico (Figura 4).

El control directo de estos drones se realiza mediante radio frecuencia, específicamente a una frecuencia de 2.4 GHz, la cual pertenece a la banda de radios ISM. Cuenta con un amplificador de radio de 20 dB, con el cual se puede trabajar en un radio de hasta 1 km. Entre sus características eléctricas, se encuentra un microcontrolador STM32F405, el cual corre a 168 MHz. Este microcontrolador presenta una gran ventaja al trabajar a una velocidad alta ya que se puede garantizar un procesamiento de señales y una ejecución de algoritmos eficientes. Cuenta con un cargador LiPo con modos de 100 mA hasta 980 mA, el tiempo de vuelo estimado con la batería cargada es de 7 minutos, con un tiempo de carga de 40 minutos. Según Bitcraze, es recomendado que cualquier carga adicional que soporte el drone no supere los 15 g [6].

6.2. Control y estimación de estado

Un sistema de control es el conjunto de sub sistemas y procesos que coexisten con el propósito de cumplir con un comportamiento deseado, este comportamiento puede definirse a través de ciertos parámetros que, igualmente deben cumplir con un comportamiento establecido, dichos parámetros se denominan variables de estado. Dependiendo de la naturaleza del sistema a controlar, este puede modelarse mediante una ecuación diferencial basada en alguna ley de la naturaleza, para un sistema físico/mecánico el análisis se realiza basado en la segunda ley de Newton [7]. En el caso de un drone cuadricóptero, las variables de estado son los ángulos de rotación alrededor de sus ejes (roll:, pitch:, yaw:), su posición en el espacio y cómo estas variables cambian en el tiempo (sus derivadas). Para el caso de Crazyflie, las variables fundamentales a controlar son dichos ángulos y la altitud de este.

Figura 4: Crazyflie 2.1.
[6]

Se utilizan estimaciones de estado para convertir las señales de los sensores en variables de estado y así poder realizar el respectivo control. La IMU del Crazyflie 2.1 cuenta con un acelerómetro/giroscopio de 3 de ejes BMI080 y un sensor de presión BMP388.

6.2.1. Filtro de Kalman

Debido a que se utilizan métodos externos para determinar el estado de los drones, siendo un sistema de captura de movimiento para este caso, es necesario combinar las mediciones internas del drone con las externas por lo que se utiliza el filtro de Kalman como método para fusión de sensores y estimador/observador de estado. Los observadores de estado son algoritmos que utilizan el modelo matemático del sistema y mediciones del mismo para dar una estimación de las variables de estado. El filtro de Kalman es un observador de estado que toma en cuenta el ruido generado por perturbaciones en los actuadores o sensores del sistema, modelándolos como ruido blanco no correlacionado [8]. Además, puede utilizarse para fusión de sensores, permitiendo estimar el estado del sistema a través de múltiples mediciones.

Para pruebas de vuelo en lazo abierto puede utilizarse el filtro de Kalman Complementario, es liviano en cuanto a uso computacional y eficiente para posiciones angulares. Este filtro toma como valores de entrada las mediciones del giroscopio para los ángulos de Euler y el ToF (*Time of Flight*) para estimar la altura, sin embargo esta variación se recomienda únicamente para control manual. La otra variante es un filtro de Kalman Extendido como un filtro recursivo que estima el estado actual del drone basado en las mediciones internas y externas y el modelo matemático del mismo. Esta variación permite basar el estado tanto en las mediciones del giroscopio como en el FlowDeck de Bitcraze, el método de posicionamiento por LightHouse de Bitcraze o en un método de captura de movimiento especializado, de esta forma pueden obtenerse los ángulos de Euler, la posición espacial en los 3 ejes rectangulares

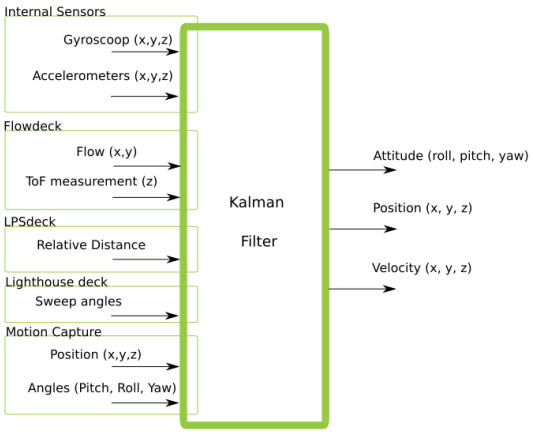

Figura 5: Filtro de Kalman Extendido.
[9]

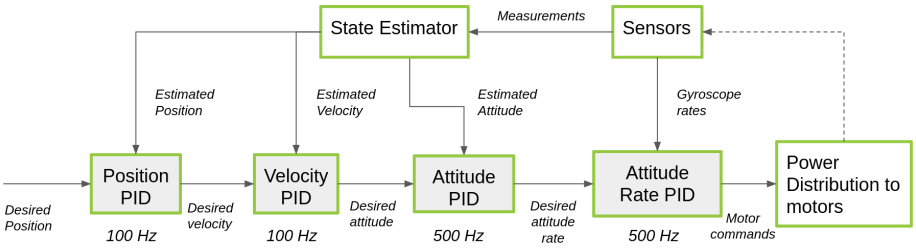

Figura 6: Arquitectura de control.
[9]

y sus velocidades, como se ve en la Figura 5 [9].

6.2.2. Método de control

Los drones Crazyflie cuentan con 3 tipos de controladores: PID, INDI y Mellinger. El controlador INDI (*Incremental Nonlinear Dynamic Inversion*) fue diseñado especialmente para sistemas aereos y permite un control eficiente sin necesidad de contar con un modelo detallado del sistema, utilizando las mediciones de las aceleraciones para generar estimaciones y sustituir al modelo [10]. El método de control de los drones por defecto se realiza mediante una arquitectura de controladores PID en cascada, donde la entrada de control es la posición y la velocidad esperada, que luego determina ángulos de pitch: y roll: deseados, seguido de una velocidad angular y finalmente determina el thrust: de los motores, como se ve en la Figura 6.

6.3. Sistemas de radio frecuencia

La comunicación por Radio Frecuencia (RF) es una tecnología para las telecomunicaciones que se da por un dispositivo que crea ondas electromagnéticas a altas frecuencias, donde las ondas se propagan en el espacio alcanzando a otro dispositivo receptor que procese la información a la misma frecuencia. Según la frecuencia de las ondas, las comunicación puede clasificarse en

- VLF: 3 kHz - 30 kHz
- LF: 30 kHz - 300 kHz
- HG: 3 MHz - 30 MHz
- VHF: 30 MHz - 300 MHz
- UHF: 300 MHz - 3 GHz
- SHF: 3 GHz - 30 GHz
- EHF: 30 GHz - 300 GHz

Las antenas convierten la información digital en señales eléctricas y luego en ondas electromagnéticas, o viceversa en el caso de la recepción de información. La ventaja que presenta este tipo de comunicación es que el dispositivo que procese tanto la información enviada como recibida puede ser un microcontrolador, siempre que este tenga la capacidad de conectarse a una antena y funcione a una frecuencia lo suficientemente alta como para procesar la señal de radio y digitalizar la información [11].

6.4. Multithreading

Un algoritmo de programación sigue una serie de instrucciones de forma secuencial, lo cual impide directamente que se realicen comandos de forma simultanea. El *multithreading* es un concepto utilizado en desarrollo de aplicaciones digitales para alcanzar un paralelismo en la ejecución de comandos en un algoritmo. Este paralelismo se logra mediante el procesamiento de "hilos" de forma simultánea, estos hilos corresponden a la unidad de procesamiento más pequeña de un sistema operativo. De esta forma un algoritmo puede ejecutar distintas funciones paralelamente sin inconvenientes de bloqueo por la ejecución de otras instrucciones, aumentando el rendimiento del núcleo sin tener que alterar la frecuencia del reloj del procesador [12].

El *multithreading* permite crear y gestionar estos hilos para asegurar que estos ejecuten comandos sin interferirse mutuamente, por lo que muchos lenguajes de programación implementan herramientas de bloqueo para evitar que dos o más hilos accedan y modifiquen información simultáneamente. Debido a esto último también es necesario definir secuencias y ejecuciones prioritarias, por lo que también suelen incluirse herramientas semáforo, las cuales permiten definir una jerarquía de prioridades entre instrucciones para asegurar que ciertos comandos se ejecuten antes que cualquier otros y el resto deban esperar para ejecutarse.

6.5. Protocolos de redes

Para crear una comunicación eficiente se debe trabajar en conjunto con protocolos de comunicación por redes, como TCP, el cual divide la información en paquetes y garantiza que estos lleguen correctamente a su destino. El protocolo TCP, que significa *Transmission Control Protocol*, es un protocolo de 3 vías donde primero se envía una solicitud inicial (SYN) desde el origen, luego el destino envía un paquete de confirmación (SYN-ACK) seguido de la información a transmitir, esta transmisión de datos puede hacerse en ambas vías y finalmente el origen envía un último paquete (ACK). Este protocolo se caracteriza por brindar una comunicación eficiente y segura debido a estos pasos mencionados que garantizan que los datos no se corrompan en el proceso. Este trabaja en conjunto con el protocolo IP, el cual se encarga únicamente de asegurar una conexión entre el servidor de origen y destino mediante el sistema de direcciones de Internet.

La ventaja de trabajar con este tipo de protocolos de comunicación es la versatilidad en la creación de herramientas con múltiples lenguajes de programación, como un servidor que permite la conexión de otros dispositivos conectados a la misma red en forma de clientes, permitiendo una comunicación directa entre los clientes conectados a dicho servidor. La creación de dichos servidores funciona mediante la creación de un *socket*, el cual permite la comunicación entre dispositivos a través de una dirección IP y un puerto, para posteriormente aceptar y escuchar conexiones, recibiendo y enviando datos. El método más utilizado para la creación de servidores es Python, ya que cuenta con librerías especializadas para la comunicación TCP, además permite trabajar con *multithreading*, lo que eficientiza la comunicación al permitir que múltiples clientes pidan o envíen datos al servidor simultáneamente [13].

El *User Datagram Protocol*, o UDP, es un protocolo de comunicación de Internet que, a diferencia del TCP, permite una comunicación rápida al no requerir que se establezca formalmente una conexión antes de iniciar con las transmisión de datos, sin embargo, esto también provoca que los paquetes puedan perderse en tránsito [14].

6.6. Formato JSON

Para la estructura de los datos transferidos entre los distintos medios de comunicación puede implementarse el formato *JavaScript Object Notation* (JSON), el cual es ligero computacionalmente y de fácil lectura y escritura. Adicionalmente, puede aplicarse a otros lenguajes de programación como C, C++, Python, etc. En el caso particular de Matlab, la codificación de información en formato JSON puede realizarse mediante la creación de una estructura con atributos nombrados como la estructura de datos que se desea manejar en un paquete, para posteriormente codificar dicha estructura a JSON y enviarlo mediante algún protocolo de comunicación especificado. Gracias a que los nombres de los atributos pueden ser especificados por el usuario, la codificación y decodificación de datos puede realizarse en distintos lenguajes de programación, permitiendo una comunicación más eficiente entre dispositivos [15].

Figura 7: Cámara de captura OptiTrack.
[16]

6.7. Sistemas de captura de movimiento

La captura de movimiento es una tecnología que ha ganado popularidad los últimos años al emplearse en aplicaciones como animación, videojuegos o investigación de sistemas mecánicos. Este proceso se da mediante cámaras de luz infrarroja, las cuales se apuntan hacia el sistema o conjunto de sistemas a capturar, los cuales deben contar con algún material que refleje la luz. Las cámaras de la marca Optitrack, las cuales son utilizadas en el Robotat en la Universidad del Valle de Guatemala (Figura 7), cuentan con un set de emisores de luz infrarroja y utiliza pequeñas bolas cubiertas de material reflectivo, conocidas como marcadores. Este tipo de captura de movimiento es conocido como óptica pasiva, ya que los marcadores reflejan la luz en lugar de generarla [2].

Los marcadores pueden usarse de forma individual para rastrear únicamente posiciones, o bien pueden trabajarse en grupos de 3 con una forma en específico para analizar también la orientación de los cuerpos. El rastreo de cuerpos rígidos se basa en la geometría de estos, ya que para que el sistema pueda diferenciar a cada cuerpo los marcadores deben estar en posiciones distintas de forma permanente para que siempre lo detecte de la misma forma.

6.7.1. MOCAP4ROS2

Pueden utilizarse distintos métodos de captura de movimiento para obtener las posiciones y orientaciones de los cuerpos rígidos a utilizar, independientemente del método utilizado se requiere de un intermediario entre dicho sistema y el sistema de control, el cual está definido por Crazyswarm en este caso. MOCAP4ROS2 es una herramienta que recibe la transmisión de datos de un sistema de captura y los envía a un nodo especificado, cabe resaltar que los datos transmitidos ya deben estar procesados, por lo que se requiere de otro ordenador que procese los datos en bruto mediante un programa especializado [17].

6.8. Ecosistema Robotat

El laboratorio de Robótica en el salón CIT-116 de la Univeridad del Valle de Guatemala, denominado Robotat, es un espacio conformado por una plataforma de acero con un tamaño de 5×4 metros, la cual es capaz de soportar cargas puntuales de hasta 2 toneladas. Este entorno de investigación fue inspirado en el laboratorio Robotarium del Instituto de Tecnología de Georgia.

Para transferir los datos obtenidos por el sistema de captura en el Robotat, las cámaras OptiTrack están conectadas a un Switch que se comunica con un servidor mediante un protocolo UDP. La información es procesada a través de un software llamado Motive de la marca OptiTrack y es enviada hacia una dirección IP local en el ordenador, posteriormente el servidor de Python Robotat accede a esta información a través de las librerías NatNetClient, estas contienen las funciones necesarias para convertir la información recopiladas para cada uno de los marcadores posicionados adentro del rango de visión de las cámaras y las envía en forma de arreglo en Python, dando las posiciones y las orientaciones en forma de cuaternión unitario. Posteriormente Motive se comunica con un router Wi-Fi, el cual permite la comunicación entre dispositivos externos y agentes autónomos en el Robotat. Esta comunicación entre el router y los dispositivos se logra mediante un protocolo TCP, de esta forma puede accederse directamente a la información mediante la decodificación de un documento de tipo JSON, permitiendo quue la información sea accesible en cualquier lenguaje de programación, actualmente se utilizan funciones en Matlab para conectarse al servidor y acceder a las poses de los marcadores en forma de arreglos, estas funciones también permiten convertir las orientaciones en forma de ángulos de Euler para mayor flexibilidad. [2].

6.9. Movimiento armónico con múltiples sistemas

Se define como movimiento armónico al movimiento periódico de un sistema que en cada ciclo pasa por una posición de equilibrio, además este es simple si dicho sistema no cuenta con un amortiguamiento notable ni fuerzas externas que lo perturben. Modelando a un cuerpo en movimiento periódico como un sistema masa resorte este puede analizarse con la siguiente ecuación diferencial 1

$$m\ddot{x} + b\dot{x} + kx = 0 \tag{1}$$

Donde x representa la posición de la partícula relativa a un marco de referencia. Al resolver esta ecuación asumiendo que no existe amortiguamiento y el sistema no está siendo perturbado, se obtiene la siguiente ecuación

$$x(t) = A\cos\omega t + \phi \tag{2}$$

Donde ω representa la frecuencia natural de oscilación del sistema y ϕ representa el desfase del movimiento. Al modelar un sistema como partícula, se obvian aspectos mecánicos como las dimensiones de este, a su vez se asume que dicho sistema no contará con una inercia por rotación alrededor se sus propios ejes. De esta forma puede modelarse un movimiento periódico para múltiples cuerpos donde estos se muevan a la misma frecuencia pero presenten un desfase entre sí. Para modelar un movimiento en dos o tres dimensiones puede definirse

que el movimiento en dos de las dimensiones corresponderá a un movimiento armónico y el tercero se definirá por una parametrización en función del respectivo movimiento.

En el caso de sistemas discretizados, debe tomarse en cuenta un periodo de muestreo. En general, puede trabajarse con el modelo de la Ecuaciones 3 y 4 [18], donde n es un vector de tiempo discretizado con el periodo de muestreo seleccionado, N es la cantidad de cuerpos y k es un número entero entre 0 y $N-1$ para crear el desfase simétrico entre estos.

$$x_k[n] = R\cos\left(n + \frac{2\pi k}{N}\right) \tag{3}$$

$$y_k[n] = R\sin\left(n + \frac{2\pi k}{N}\right) \tag{4}$$

6.10. Cliente de Crazyflie y ROS2

El *Robotics Operating System 2*, o ROS2 por sus siglas en ingles, es una plataforma complementaria al sistema operativo del ordenador que proporciona herramientas para desarrollo de sistemas robóticos. Esta plataforma funciona mediante nodos, los cuales son unidades independientes que envían y reciben información. Uno de los principales objetivos de ROS es permitir una comunicación flexible entre nodos. Esto funciona mediante el conceptos de publicación y suscripción, donde un nodo puede suscribirse a otro y así recibirá los datos publicados, datos que posteriormente pueden ser procesados para alguna tarea.

ROS2 permite que se puedan trabajar plataformas ya desarrolladas, tales como Crazyswarm. Crazyswarm es la infraestructura necesaria para trabajar con conjuntos de drones Crazyflie, esta utiliza la estructura de los nodos en ROS para establecer una comunicación entre el sistema de captura de movimiento a utilizar y los drones. ROS funciona con nodos, creando un servidor de Crazyflie que establece la comunicación directa entre los drones y la interfaz utilizada, que en este caso es ROS2 junto con la API de Python como se ve en la Figura 8.

El control de los drones se divide en cuatro capas independientes. La capa física transmite los paquetes de información desde y hacia el drone, donde puede implementarse una antena de radiofrecuencia para esto. El *link* implementa los canales para los paquetes, abstrayendo el medio físico e implementando un canal de transmisión y otro de recepción entre el Crazyflie. El CRTP, acrónimo de *Crazy Realtime Protocol*, implementa la información del puerto y el canal para crear la ruta del paquete a varios subsistemas. Finalmente, los subsistemas implementan las funcionalidades del Crazyflie que pueden ser controladas, habiendo sólo un puerto por subsistema.

El protocolo CRTP fue diseñado para permitir una priorización de paquetes y volver más eficiente la comunicación con el drone, permitiendo enviar una trayectoria mientras el drone se controla en tiempo real, siempre que el puerto de la trayectoria sea de mayor prioridad. Cada paquete CRTP contiene un puerto (4 bits), un canal (2 bits), y una carga de hasta 31 bytes. Para crear la conexión del protocolo se debe habilitar el *link* USB usando un paquete de control USB, luego el *link* Radio mantiene dos contadores de paquete para garantizar que no habrá pérdida de paquetes. Finalmente, el subsistema *log* mantiene un estado de todos

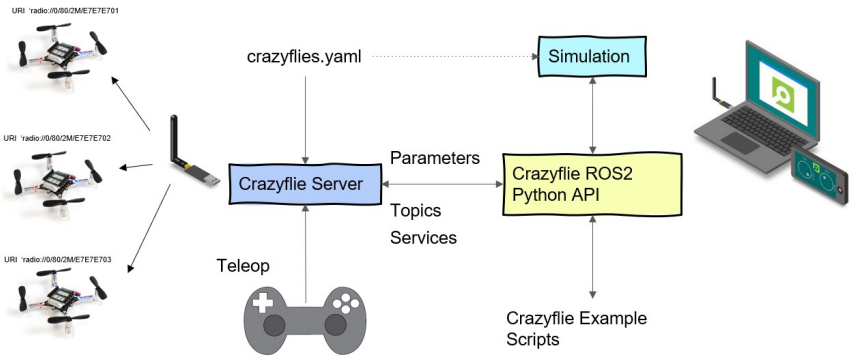

Figura 8: Estructura de control de Crazyswarm.
[19]

los bloques *log* y continua enviado información aún si se pierde el *link*[20]

Otro protocolo de comunicación para Crazyflie es el CPX (*Crazyflie Packet eXchange*), el cual fue diseñado para funcionar a bajo nivel y que se empleen otros protocolos por encima. Su función es solucionar el problema de distribuir paquetes a través de múltiples microcontroladores ya que este problema no existía cuando se diseñó el protocolo CRTP, por lo que es un protocolo complementario. Debido a que se utiliza entre microcontroladores, puede enviarse de múltiples formas, como Wi-Fi/TCP, SPI o UART dependiendo de los microcontroladores a comunicar y los módulos adicionales que se implementen en el drone. Este protocolo permite el manejo de paquetes grandes, donde paquetes por debajo de 30 bytes se entregan en un bloque y paquetes más grandes pueden separarse en bloques y un identificador especifica cual es el bloque final.

Para establecer la comunicación de Crazyswarm, el servidor de Crazyflie se conecta con múltiples drones mediante una o más antenas de radiofrecuencia. Se cuenta con dos *backends* a elegir, el "cpp" basado en la capa más baja y el "cflib", que funciona con Python a un nivel más alto. Este puede manejar aspectos de comunicación de bajo nivel, como recibir los parámetros del Crazyflie y convertirlos a parámetros de ROS2 para crear los parámetros de Crazyflie basado en la entrada.

Crazyflie cuenta con una máquina virtual, la cual funciona mediante Ubuntu y esta contiene todas las librerías para controlar los drones mediante una antena de radiofrecuencia USB. Estas librerías pueden visualizarse y editarse mediante Microsoft Visual Studio, que ya cuenta instalados los compiladores de C++ y Python. Desde Visual Studio, pueden iniciarse los códigos y, si la antena está conectada y un Crazyflie está encendido, pueden realizarse pruebas de movimiento y obtención de datos del drone. La máquina virtual también cuenta con el cliente de Crazyflie, desde el cual puede controlarse al Crazyflie mediante un control externo y es posible visualizar los ángulos de rotación del drone en tiempo real. En caso de que no se cuente con un control externo, puede utilizarse la aplicación para Android de Crazyflie y controlar al drone mediante *Bluetooth*.

Para poder realizar las pruebas del Crazyflie es necesario configurar el puerto de la antena durante la instalación de la máquina virtual, lo cual puede realizarse fácilmente mediante Oracle. También se deben configurar todos los dispositivos que se planeen utilizar en la

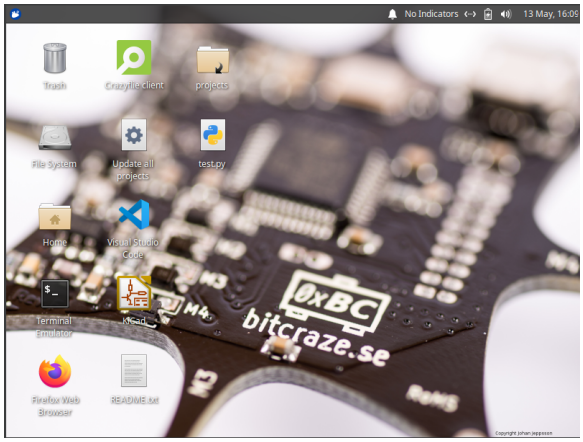

Figura 9: Máquina virtual de Crazyflie.

máquina virtual, de lo contrario no serán reconocidos. Una vez instalada la máquina virtual se debe abrir una terminal e instalar ciertas librerías mediante el gestor de paquetes pip3. Las librerías son:

- Matplotlib.
- Pandas.
- Canvas.
- pyinstaller.

Finalmente, es necesario actualizar las librerías de los controladores. Esto se logra mediante la opción incluída en el escritorio de la máquina virtual, como se ve en la Figura 9. Sin embargo, se presenta un problema al actualizar las librerías ya que luego de hacerlo deja de funcionar el cliente para controlar el drone con un control externo, por lo que si se desea trabajar con las librerías de Python y con el cliente se recomienda instalar dos máquinas virtuales y que una de estas no se actualice.

CAPÍTULO 7

Infraestructura de Crazyswarm

La implementación de Crazyswarm involucra el funcionamiento de múltiples funciones y estructuras de datos en ROS2, lo cual representa una infraestructura de control y comunicaciones compleja. Antes de llevar a cabo el levantamiento de dicha infraestructura es necesario verificar el funcionamiento de cada uno de los Crazyflies a utilizar, para posteriormente incluirlos en Crazyswarm. A continuación se presentan las pruebas realizadas para comprender el funcionamiento de los drones por sí solos y verificar que estos no presenten problemas en su vuelo, seguido de las pruebas realizadas durante la inicialización de Crazyswarm y su funcionamiento con múltiples Crazyflies.

7.1. Ensamblaje y verificación de funcionamiento

El empaque del Crazyflie 2.1 cuenta con el drone desensamblado como se aprecia en la Figura 10 e incluye:

- Placa con microcontrolador
- 6 bases para motores
- 5 motores
- 10 hélices para motores
- 4 pines para anclar módulos
- 1 cable micro USB
- 1 batería de litio con pegamento

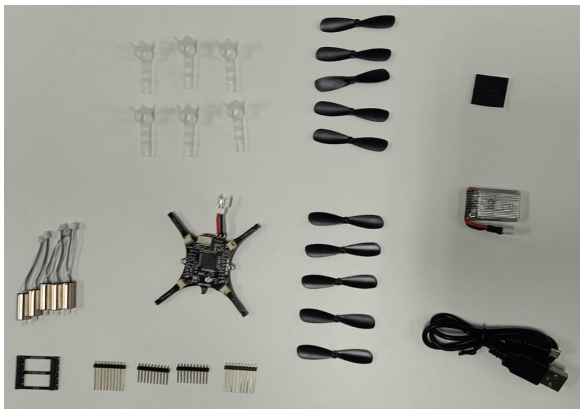

Figura 10: Piezas del Crazyflie 2.1.

Figura 11: Motor ensamblado en su base.

- 1 PCB para sostener la batería

Además, se debe contar con una antena *Crazydongle* para poder controlar a un Crazyflie o varios, la cual funciona como intermediario para la comunicación entre los drones y Crazyswarm. Esta antena envía las posiciones de destino para cada Crazyflie basado en el algoritmo de control en funcionamiento y las mediciones del sistema de captura de movimiento, esto a través del paquete de datos mencionado en la sección 6.9.

El primer paso es colocar los motores. Para esto, se recomienda enrollar los cables de cada motor para evitar que estos se enreden con las hélices, luego debe insertarse en los soportes y posicionar los cables de acuerdo a las guías de plástico de los soportes, quedando como en la Figura 11. Una vez que todos los motores estén montados en sus bases, deben colocarse en la placa principal. Estos se insertan hasta encajar en la placa para posteriormente conectarlos como en la Figura 12.

Finalmente, deben colocarse las hélices de forma cruzada para evitar una rotación natural alrededor del eje z. La forma en que se colocaron fue la siguiente, las hélices con la marca A fueron colocadas en la esquina inferior izquierda y en la esquina superior derecha, mientras

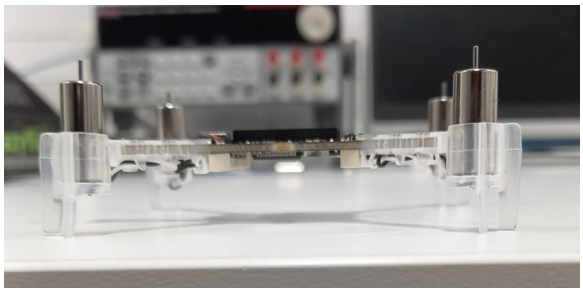

Figura 12: Motores conectados a la placa.

Figura 13: Crazyflie ensamblado.

que las hélices con la marca B se colocaron en las otras dos esquinas. La batería puede colocarse con los pines entre la placa y los headers:. Luego del ensamblaje debe verse como en la Figura 13.

7.1.1. Pruebas de vuelo en Android

Bitcraze cuenta con una aplicación compatible con Android que permite una conexión con un Crazyflie mediante Bluetooth, esta aplicación cuenta con dos controles que se asemejan a un *joystick* que controlan al drone. El control izquierdo aumenta el thrust: de los motores y el derecho controla los ángulos de roll: y pitch: (Figura 14) [21].

El principal problema que se presentó al volar el Crazyflie mediante la aplicación es que los drones presentan una descompensación en la distribución de thrust:, ya que al elevar el Crazyflie este se movía hacia la derecha naturalmente.

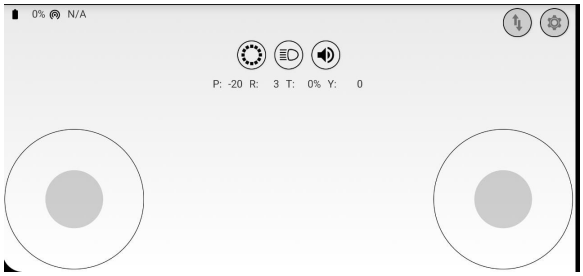

Figura 14: Aplicación de control en Android.

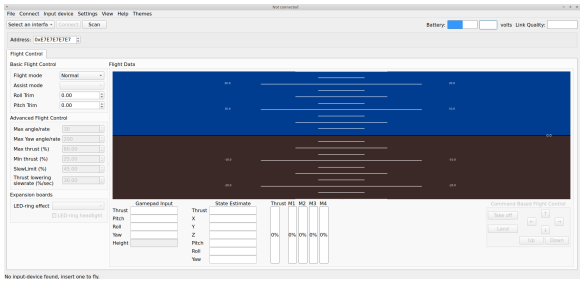

Figura 15: Cliente de Bitcraze

7.2. Máquina virtual y cliente de Crazyflie

Bitcraze proporciona una máquina virtual en formato ISO descargable, esta puede utilizarse en cualquier programa que corra máquinas virtuales. Esta máquina funciona basada en Ubuntu y cuenta con las herramientas necesarias para realizar pruebas iniciales de vuelo y desarrollar códigos de control para un Crazyflie (Figura 15).

El cliente de Bitcraze permite modificar la dirección de los Crazyflies, esto es de vital importancia ya que al momento de realizar un control de múltiples drones estos deben tener distintas direcciones. Este cliente también permite actualizar el firmware: de cada uno de los Crazyflies para asegurar que funcionen con los parámetros más recientes documentados por Bitcraze. Una vez que se ha configurado el firmware: y la dirección deseada, pueden realizarse pruebas de estimación de estado y control con un mando externo.

7.3. Pruebas iniciales de vuelo en Python

Los códigos de Python incluidos en la máquina virtual funcionan con las librerías provistas por la misma.

7.3.1. Códigos de control en Python

El proyecto cuenta con una serie de códigos de control, estos códigos reciben atributos que suelen ser ángulos de rotación, tasas de rotación (ambos en grados), posiciones espaciales (en metros) o porcentaje de thrust: en forma de PWM:, el cual va de 0x0000 a 0xFFFF. La

principal funcionalidad utilizada de Bitcraze consiste en una estructura de funciones llamada *commander*. Para las pruebas de vuelo, se hizo uso especial de la función send_setpoint() la cual recibe como parámetros los ángulos de roll:, pitch:, yawrate: y el thrust: deseado. Dichos parámetros son concatenados en un objeto que corresponde a la DATA del paquete, que posteriormente es enviado al puerto determinado automáticamente por la función.

El primer código probado fue el de ramp.py, el cual enciende los motores con un thrust: mínimo durante unos segundos. Este código fue modificado para aumentar el empuje de los motores y analizar cómo podía elevarse al Crazyflie.

7.3.2. Prueba de vuelo inicial y verificación

Para elevar el drone se implementó un código basado en la primera (Ecuaciones 5 y 6) y segunda ley de Newton (Ecuaciones 7 a 9), donde se le dio un thrust: suficiente para sacar al Crazyflie de su equilibrio estático y acelerarlo verticalmente hacia arriba.

$$\sum F_y = 0, \tag{5}$$

$$T = mg, \tag{6}$$

$$\sum F_y = ma_y, \tag{7}$$

$$T - mg = ma_y, \tag{8}$$

$$T = m(g + a_y). \tag{9}$$

De esta forma, se encontró que una forma experimental de encontrar la fuerza equivalente de los motores en función del thrust: fue mediante la primera Ley de Newton, como se ve en la Ecuación 10, esto claramente asumiendo una relación lineal.

$$F(PWM) = \frac{mg}{44500} PWM \tag{10}$$

Primero, se aplicó un PWM: de 47500, lo cual representa un 72 % del valor máximo permitido por los motores, este nivel en los motores se mantiene durante 350 ms hasta que alcanza una altura aproximada de medio metro. Luego de esto se llevó a un valor de 44500, aproximadamente el 68 % del valor máximo, este valor es suficiente como para que los motores generen un empuje equivalente al peso del drone. A pesar de la eficiencia de este método para elevar al drone, la descompensación mencionada en los motores no permite una estabilidad estática planar, ya que los Crazyflies tienden a moverse a la derecha y hacia el frente, esto puede explicarse debido a errores en la calibración del giroscopio, provocando que la IMU tome ángulos de pitch: y roll: distintos de cero como valores de equilibrio 16.

Figura 16: Descompensación de vuelo.

A continuación se muestra una prueba de vuelo con thrust: constante grabada con el programa Motive, del cual se hablará más adelante. Al aplicar un empuje constante en los motores con ángulos de pitch: y roll: iguales a cero el Crazyflie debería mantenerse en una posición planar constante basado en la primera Ley de Newton, por lo que solo debería moverse verticalmente. Sin embargo, como ya se explicó para la Figura 16 y como se muestra ahora en la Figura 17 el Crazyflie no se mantiene en una posición (x, y) constante, por el contrario se mueve a lo largo del espacio del Robotat descontroladamente.

7.4. Consideraciones físicas del modelo dinámico

Al realizar las pruebas de vuelo con los códigos de Python se notó una falta de estabilidad por parte del drone, ya que al dar un thrust: constante el Crazyflie se eleva pero tiende a moverse hacia la derecha y adelante. Esto continuó sucediendo a pesar de establecer ángulos de roll: y pitch: iguales a cero, lo cual mostró que el control de los Crazyflies sin la utilización de un sistema de captura es en lazo abierto. El modelo dinámico de un drone se establece basado en las leyes de Newton, para el análisis de torques se puede notar que no se toma en cuenta un torque provocado por el peso del drone alrededor de los ejes x y y, como se ve en la Ecuación 12.

$$\sum \tau = I\ddot{\eta}, \tag{11}$$

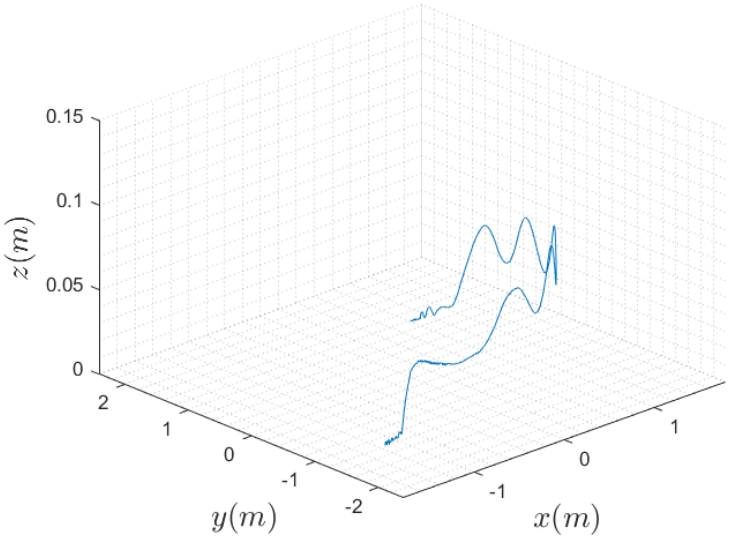

Figura 17: Descompensación de vuelo.

$$I\dot{v} + v \times (Iv) + \Gamma = \tau, \tag{12}$$

Donde I representa a la matriz de inercia, v es el vector de velocidades angulares, Γ el vector de fuerzas giroscópicas y τ los torques externos [22]. Debido a esto, cualquier variación en cuanto a la ubicación del centro de masa altera la ecuación fundamental y por ende el modelo del sistema dinámico. Cada vez que se hacía un cambio de batería se debía procurar que esta quedara lo más centrada posible para que la alteración al modelo fuera lo más pequeña posible, lo cual representa una notable fuente de error para las pruebas en lazo abierto. Además se tuvo que tomar en cuenta que cada batería utilizada tenía una masa distinta, por lo que cada vez que se hacía un cambio de batería era necesario alterar tanto el thrust: de arranque como el estable.

7.5. Instalación de Crazyswarm y sus dependencias

ROS2 funciona en Ubuntu, esta versión dicta utilizar la versión más reciente y que esta sea LTS, la versión de ROS2 utilizada fue la *Humble Hawksbill*. La instalación de ROS2 y Crazyswarm se realizó en una USB que permite iniciar Ubuntu en una computadora como sistema operativo nativo, lo que da flexibilidad de trabajar en distintos ordenadores pero a su vez limita las capacidades del sistema.

La instalación de Crazyswarm consiste en la preparación del espacio de trabajo de ROS2 y la inclusión de los repositorios de Github: especificados por la documentación de Crazyswarm.

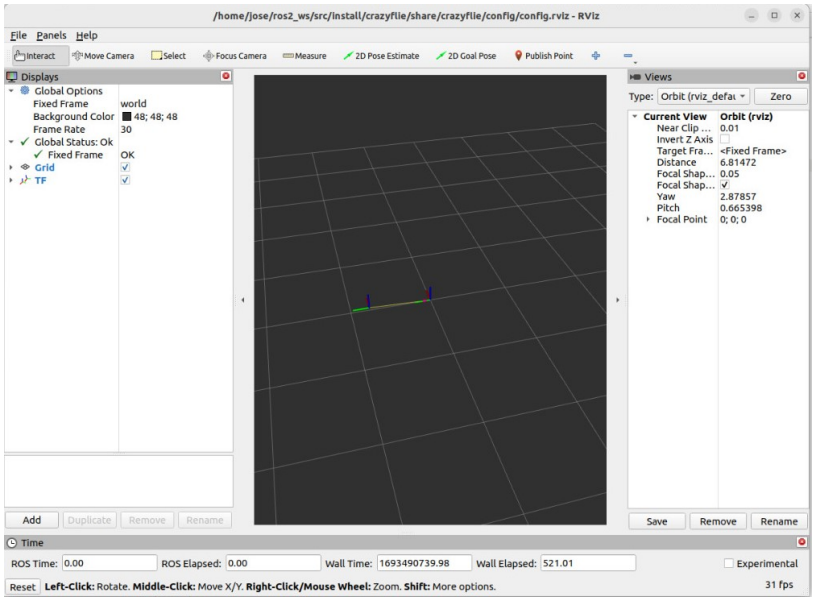

Figura 18: Intefaz de RViz.

7.6. Inicialización de Crazyswarm y pruebas iniciales

7.6.1. Comandos de iniciación

Una vez que se haya instalado Crazyswarm exitosamente, se puede proceder a las pruebas de control. Los archivos de configuración deben ser modificados para definir las características del entorno en el que se trabajará, como la cantidad de drones a utilizar y las direcciones de cada uno, sus posiciones iniciales y el método de captura de movimiento.

Luego de realizar la configuración de los drones, se deben inicializar los entornos de ROS2 en las carpetas raíz (*src*) donde se instalaron Crazyswarm y las dependencias de MOCAP4ROS2, posteriormente las librerías deben activarse y configurar la captura de datos de OptiTrack. Las librerías de MOCAP4ROS2 permiten seleccionar si se desea trabajar únicamente con *single markers* o con cuerpos rígidos que ya se hayan creado en Motive. Si la configuración y activación se hizo correctamente, se debería de apreciar la recepción de datos en una terminal de Ubuntu mediante el tópico de ROS2 que se encarga de procesar dichos datos, mostrando las coordenadas de los marcadores presentes en el Robotat.

Una vez que se inicia Crazyswarm, se inicializa MOCAP4ROS2 junto con el programa RViz el cual funciona como una interfaz gráfica para visualizaciones 3D y se ve como en la Figura 18. Si tanto el sistema de captura de movimiento como el sistema operativo funcionan correctamente y los Crazyflies a utilizar están al alcance de las cámaras, RViz debería mostrar la posición en tiempo real de los drones y la orientación en caso de que se usen múltiples marcadores. Al mover el Crazyflie encendido y conectado a la antena adentro del espacio de visibilidad de las cámaras se debería visualizar este movimiento en RViz, lo que valida el funcionamiento del Filtro de Kalman Extendido al dar una estimación lo suficientemente precisa sobre la pose del drone. Cuando la estimación de estado diverge puede observarse en la interfaz que los marcadores se alejan del área de visibilidad, lo que implica reiniciar

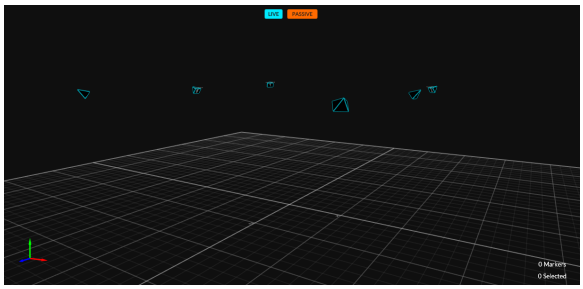

Figura 19: Espacio de trabajo en OptiTrack.

Crazyswarm y/o los Crazyflies para volver a inicializar el Filtro. Una forma de verificar que existe una comunicación entre Crazyswarm y los Crazyflies es verificar que todos los drones tengan una luz amarilla intermitente, evidenciando una transferencia de datos.

7.6.2. Calibración del sistema de captura

MOCAP4ROS2 requiere que los datos sean procesados anteriormente, por lo que debe utilizarse el programa Motive. Motive permite la calibración de cámaras y definir el origen $(0, 0, 0)$ en el espacio a trabajar. Las cámaras se encienden mediante un comando por Python, luego de abrir Motive deben calibrarse las cámaras a través de la vara de calibración hasta tener una cantidad de muestras aceptable, luego se emplea la escuadra que define el origen y al hacerlo correctamente debe verse como en la Figura 19. Una vez que la calibración se hizo de manera exitosa, puede iniciarse la transmisión de datos a través de una red local especificada, en el caso del Robotat la dirección IP es la 192.168.50.200.

7.6.3. Pruebas físicas

Para llevar a cabo pruebas con los drones, fue necesario colocar un marcador del sistema de captura sobre cada uno. Como método inicial se colocaron los marcadores entre la batería y el PCB que la sostiene, por lo que se mantiene relativamente firme sobre el Crazyflie como se observa en la Figura 23. Cabe destacar que este es un método poco eficiente ya que al no estar sujeto sobre el drone, este tiende a moverse debido que las vibraciones de los motores generan de igual forma una vibración en el marcador, provocando que los pines que sostienen la PCB sobre la batería comiencen a moverse al punto en que el marcador se afloja, esto a su vez induce aún más vibraciones en el drone, afectando su rendimiento.

Luego de inicializar Crazyswarm en ROS deben de mostrarse las posiciones de los Crazyflies que se encuentren en el espacio de trabajo, esto se aprecia en la terminal que muestra el nodo suscrito a la información del sistema de captura de movimiento como se ve en la Figura 20. Luego de verificar esta información pueden ejecutarse códigos de prueba. El código "hello-world.py" toma a un Crazyflie al azar de los que se encuentren conectados y dentro del sistema de captura y lo eleva 1 metro durante unos segundos. Se realizó una prueba de estabilidad al perturbar al drone una vez que este alcanzaba la altura especificada y este corregía el error inmediatamente con un *Overshoot* lo suficientemente bajo como para no ser notable a simple vista, este rechazo a perturbaciones funciona con la posición en los 3 ejes.

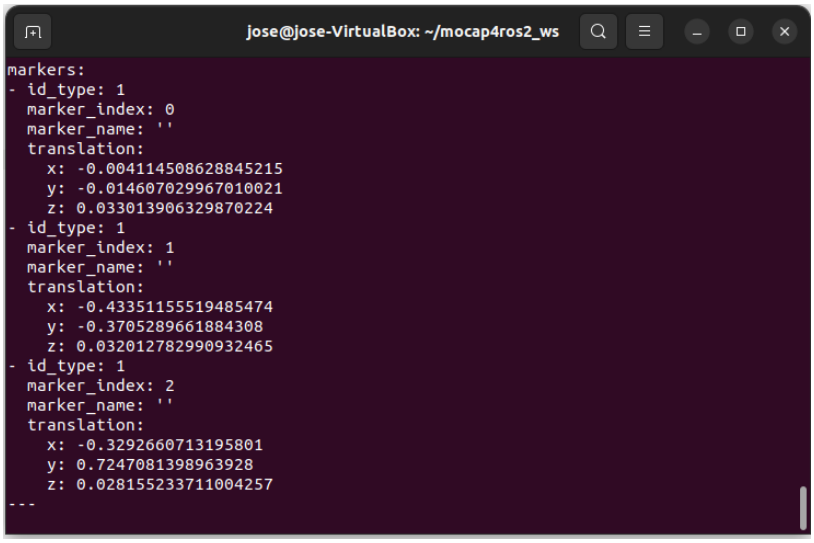

Figura 20: Despliegue de las mediciones del sistema de captura de movimiento.

La altura máxima alcanzada por el Crazyflie durante esta prueba de despegue fue de 0.9531 metros, manteniéndose a una altura promedio de 0.9471 metros con una desviación estándar de 0.0051, esta prueba tuvo una duración de 2.5 segundos, pero el estado de equilibrio se mantiene hasta que la batería se agote. Este movimiento puede apreciarse en las Figuras 21 y 22. De esta forma puede asegurarse que el controlador para estabilización de posiciones espaciales funciona correctamente con un error cuadrático medio de 0.007 metros.

Luego de analizar las funcionas provistas por Crazyswarm, se realizó una prueba con la función *goto*, a la cual se le envía un vector de posición absoluta con respecto al sistema de captura de movimiento. La altura especificada en la función se alcanzó y mantuvo un estado de equilibrio en el tiempo establecido, también se le dio perturbaciones externas empujando al Crazyflie en todas las direcciones y aún así corrigió su posición, verificando el funcionamiento del controlador.

Una vez que se han verificado las pruebas de control con un solo drone, puede procederse a las pruebas con múltiples Crazyflies, se recomienda agregarlos uno por uno para asegurar que cada uno funcione correctamente. Cabe destacar que las pruebas con múltiples Crazyflies requieren que las direcciones de la comunicación por radiofrecuencia con la antena ya hayan sido modificadas, la librería de Crazyswarm cuenta con una función que detecta a todos los Crazyflies encendidos cercanos a la antena y crea un vector que permite indexar a los drones y referirse a cada uno como la posición de dicho vector, de esta forma se puede controlar al Crazyflie número 1 refiriéndose a él como cf[0]. Luego de verificar el funcionamiento de múltiples drones logrando que estos se eleven a cierta altura durante unos segundos, puede procederse al diseño de rutinas.

7.6.4. Implementación inicial de trayectorias

Las primeras pruebas con trayectorias se realizaron utilizando el controlador PID de los Crazyflies, donde la primera trayectoria de prueba fue un movimiento armónico simple. Los

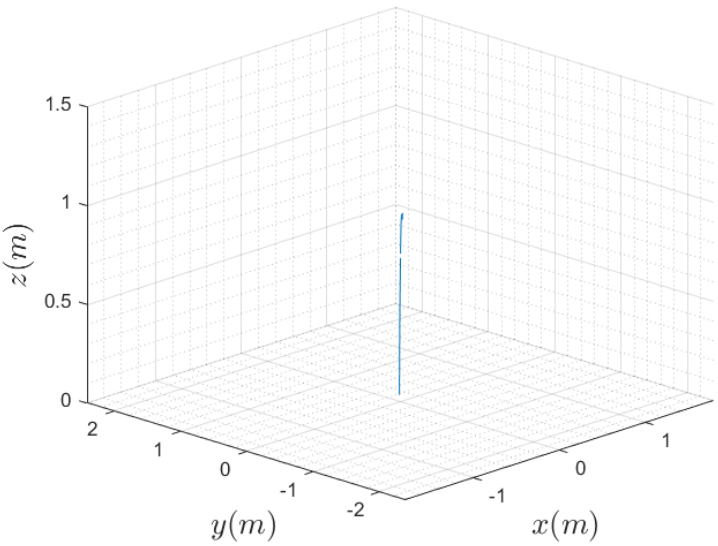

Figura 21: Prueba de despegue inicial.

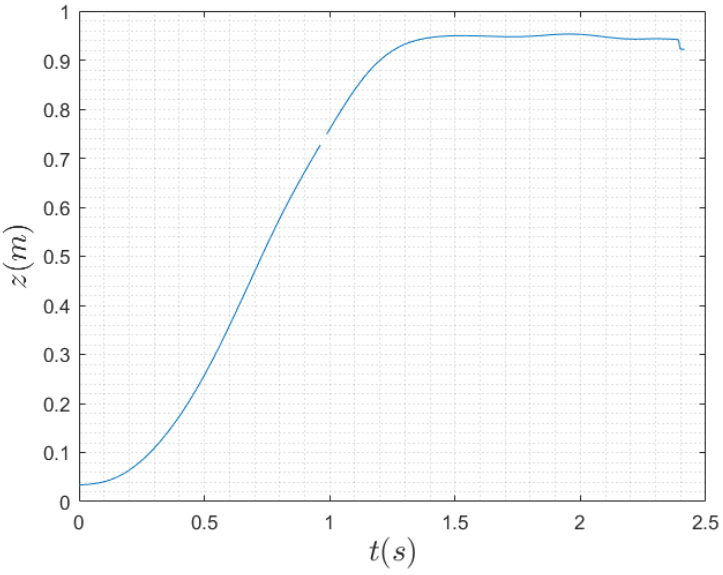

Figura 22: Prueba de despegue inicial.

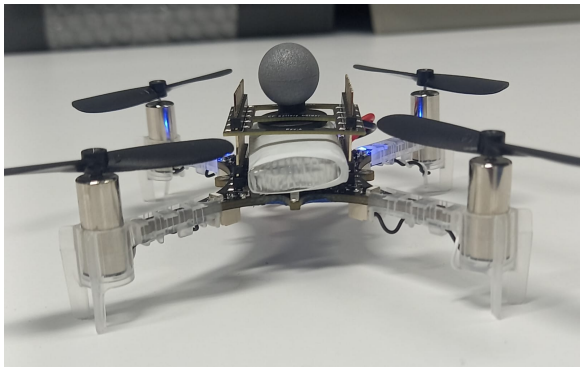

Figura 23: Marcador reflectivo instalado sobre un Crazyflie.

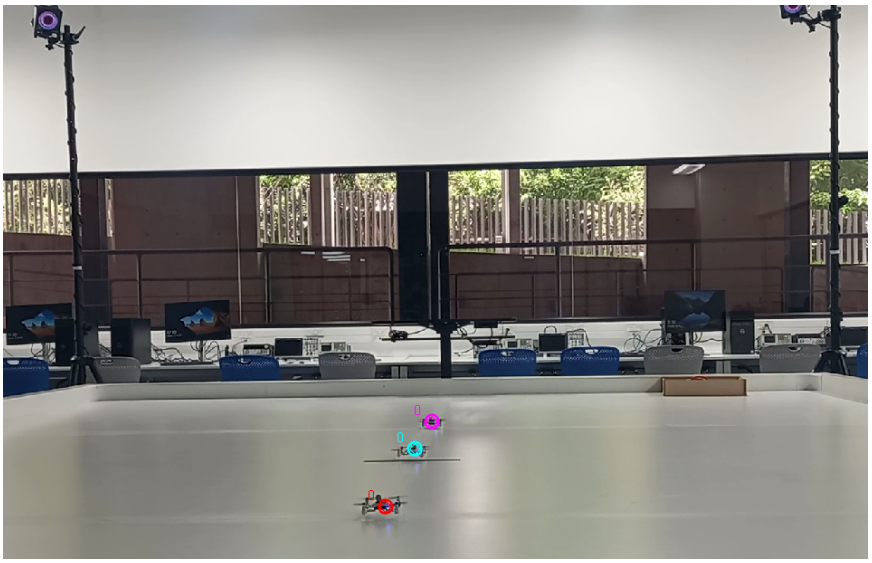

Figura 24: Prueba con 3 drones.

resultados pueden apreciarse de la Figuras 24 a la 27.

Debido a que este es un controlador para estabilización y no para rastreo de referencias, puede llegar a apreciarse un movimiento pausado durante las trayectorias si se utiliza un periodo de muestreo relativamente alto para actualizar el punto espacial de referencia para el controlador. Dependiendo de la velocidad de la trayectoria, o la frecuencia del movimiento en caso de que sea periódico, puede alcanzarse un movimiento más fluido en los Crazyflies. Para la implementación de una trayectoria circular con un periodo de muestreo de 1 segundo se obtuvo un RMSE de 0.0437 metros, por lo que la discretización no afecta significativamente en la fluidez de la trayectoria, la similitud se presenta en la Figura 28. Por otro lado se evaluó la misma trayectoria con un periodo de 2 segundos, resultando en un error cuadrático medio de 0.2466 metros, siendo casi 6 veces mayor al caso anterior, esta diferencia es notable visualmente y se muestra en la Figura 29. Este comportamiento puede explicarse a través del concepto del *Gain Scheduling*, en donde las ganancias de un controlador son actualizadas antes de que el sistema llegue a su objetivo para que este comience a rastrear la siguiente referencia, dando una mayor fluidez en la trayectoria al no permitir que el sistema se detenga al llegar a dicha referencia. Este comportamiento se refleja en las pruebas realizadas con

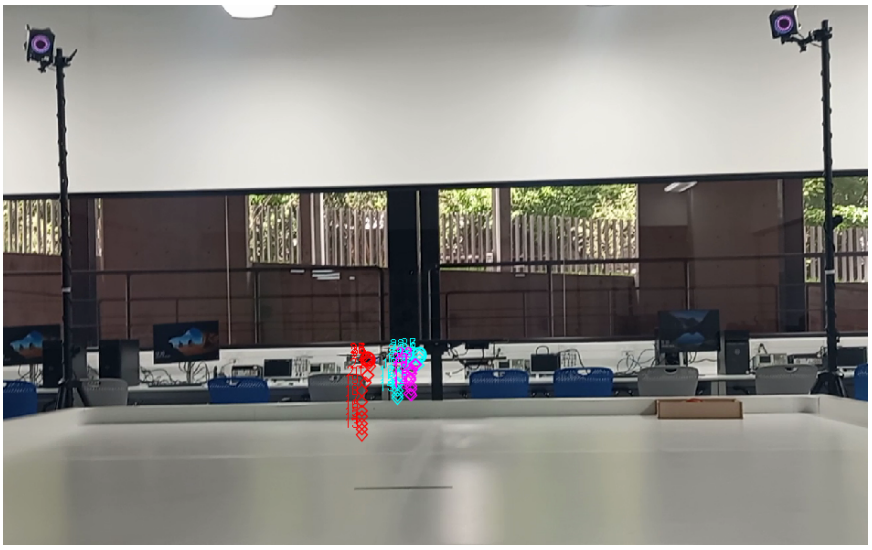

Figura 25: Prueba con 3 drones.

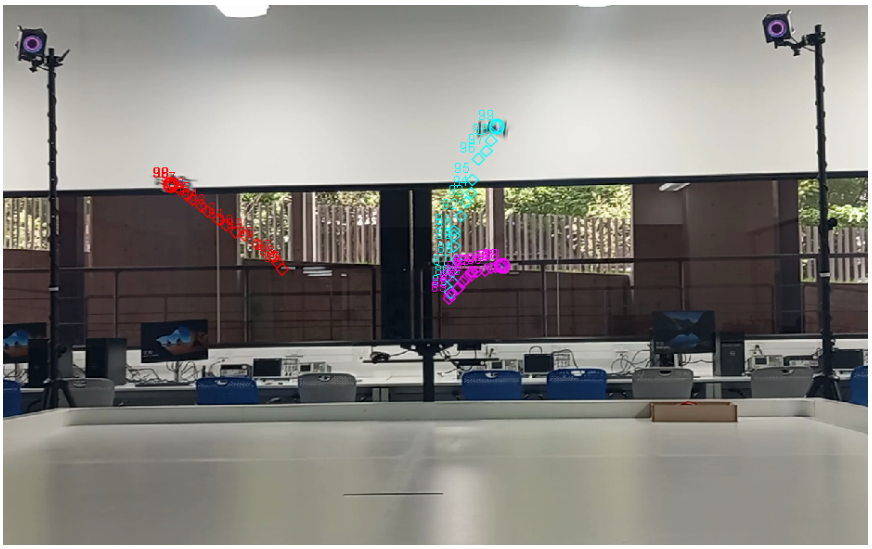

Figura 26: Prueba con 3 drones.

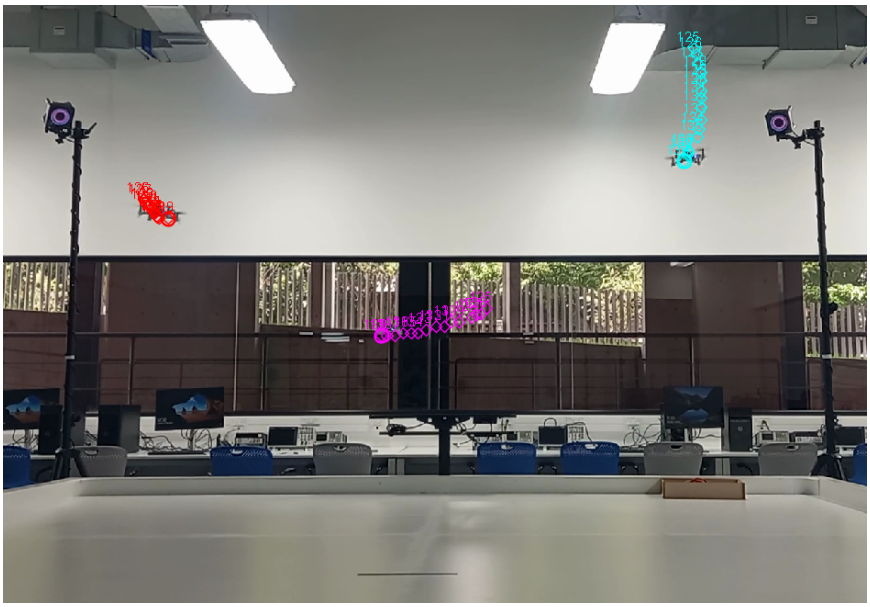

Figura 27: Prueba con 3 drones.

un periodo de muestreo de 1 segundo, ya que el comando *goto()* tiene especificado que la duración del movimiento sea de 2 segundos, de forma que al actualizar la posición objetivo cada segundo el controlador no alcanza esta nueva referencia y continua su movimiento con la fluidez mostrada.

Entre las observaciones realizadas durante estas pruebas se determinó que Crazyswarm no tiene un bloqueo en la ejecución de sus comandos en caso de que los drones no alcancen las posiciones establecidas, por lo que si uno de los Crazyflies llegara a quedarse sin batería durante una rutina y cayera, esto no afectaría en las trayectorias de los demás. De esta forma se concluye que el ordenador sólo envía el paquete de datos con la posición recopilada por el sistema de captura y las posiciones deseadas, el lazo de control se cierra internamente en el Crazyflie.

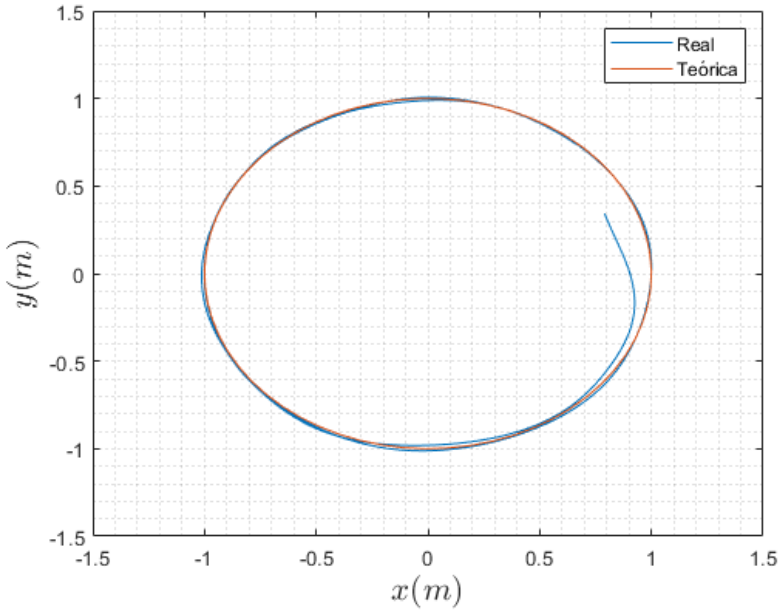

Figura 28: Trayectoria circular con periodo de muestreo de 1 segundo.

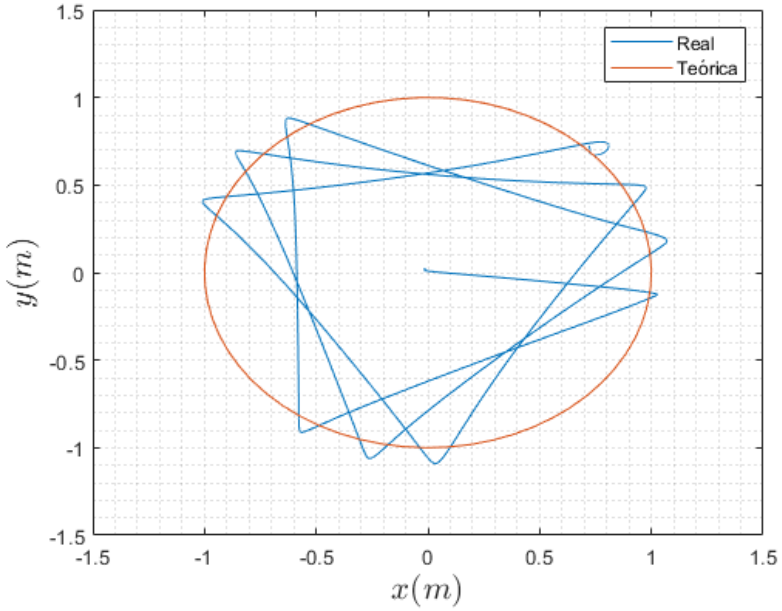

Figura 29: Trayectoria circular con periodo de muestreo de 2 segundo.

CAPÍTULO 8

Integración de Crazyswarm con el ecosistema Robotat

Como se ha mencionado a lo largo de este trabajo, la meta principal es alcanzar una fusión exitosa entre la comunicación interna del ecosistema Robotat y el sistema de control de Crazyswarm, por lo que dicha fusión debe realizarse mediante los métodos de comunicación ya establecidos y adaptar la infraestructura de Crazyswarm. Para esto, se creó una variante del servidor de comunicación del Robotat especialmente para establecer una comunicación con Crazyswarm, de esta forma es posible que cualquier computadora con la capacidad de crear clientes TCP puede comunicarse con los drones sin necesidad de utilizar Ubuntu.

8.1. Comunicación en Robotat

Como concepto inicial, se pretendía utilizar el paquete de herramientas de Matlab que permite la comunicación directa con los tópicos de ROS2, sin embargo esto presentó problemas debido a que los puertos de las computadoras del Robotat están bloqueados por motivos de seguridad. Por otro lado, este método requería que todos los estudiantes que quisieran desarrollar pruebas con Crazyswarm debían comprar dicho paquete de Matlab, lo cual no es factible. Debido a esto, se buscó otra alternativa para establecer una comunicación entre Crazyswarm y Matlab, por lo que se procedió a trabajar en un paquete especializado en ROS con un nodo que envíe y reciba datos mediante una comunicación TCP. Sin embargo, la comunicación entre Matlab y ROS requería de un servidor el cual podría realizar las mismas tareas que dicho nodo, por lo que finalmente se decidió implementar únicamente el servidor y que este ejecutara las funciones de Crazyswarm internamente basado en los comandos recibidos. La estructura final implementada en el Robotat se muestra en la Figura 30.

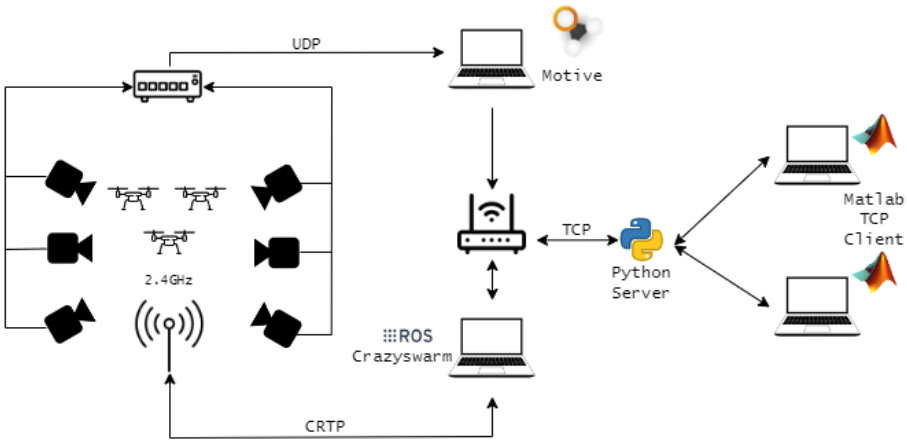

Figura 30: Infraestructura de comunicación Crazyswarm-Robotat.

	src	**len**	**pty**	**pld**
BYTES	1	1	1	8520
	Origen del paquete	Tamaño del array	Tipo de paquete	Concatenación de las trayectorias
	205-Matlab	Cantidad de CF o largo de la trayectoria	0-127 DATA 128-255 CMD	3 matrices de posiciones espaciales

Cuadro 1: Estructura del paquete de datos para la comunicación Crazyswarm-Robotat.

8.1.1. Creación del servidor Crazyswarm

Los dispositivos a conectar son

- Ordenador que corre Crazyswarm.
- Ordenador que corre OptiTrack.
- Ordenadores que deseen comunicarse con Crazyswarm.
- Crazyflies a utilizar.

Los usuarios del Robotat pueden conectarse al router que genera la red Wi-Fi Robotat mediante el servidor con el mismo nombre y pueden realizar dicha conexión mediante antenas Wi-Fi externas. Se replicó este concepto creando un servidor en Python usando la dirección IP 192.168.50.170 y el puerto 64183, a diferencia del servidor original del Robotat que tiene una dirección 192.168.50.200, este cambio se debe a que ya se está utilizando este intermediario para enviar datos del sistema de captura de movimiento desde OptiTrack a Crazyswarm, por lo que podría haber un problema de saturación en el puerto. El servidor recibe información codificada en formato JSON y la estructura de los datos se presenta en el Cuadro 1.

Las instrucciones base de control utilizan las funciones de Crazyswarm dentro del ciclo del servidor, enviando puntos mediante la función *goto* para posiciones específicas o para

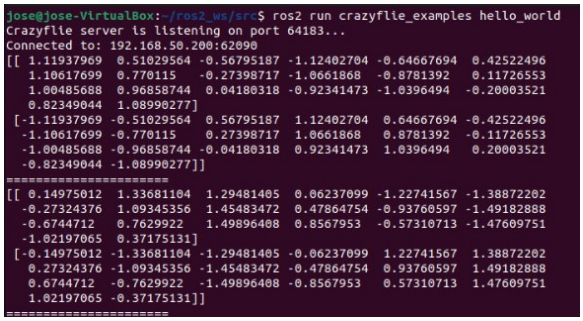

Figura 31: Servidor en funcionamiento.

trayectorias, que envía a una cantidad de Crazyflies especificada hacia las posiciones de una trayectoria generada previamente [23].

El servidor se conecta a la dirección IP y puerto especificados comenzando a recibir datos, el paquete se lee y decodifica siguiendo el formato JSON, luego los datos se procesan mediante un match (equivalente a switch en Python), el cual funciona como condicional en función de los comandos (CMD) recibidos. Luego de clasificar el comando, se procede a leer el set de datos (pld) y, dependiendo del comando, se re agrupan los datos. Esto se debe a que la decodificación funciona mejor si las matrices generadas en Matlab se envían como arreglos de una sola dimensión, haciendo necesario que dicho arreglo vuelva a agruparse como matriz. Además, el pld también se aprovecha para enviar datos como el número de Crazyflies a utilizar o en el caso de un *goto*, la dirección del Crazyflie a utilizar. Una vez que se extraen los datos del pld, estos se reorganizan de forma que se crean 3 arreglos en Python y se ponen en forma de matriz según el número de Crazyflies a utilizar y la cantidad de puntos en la trayectoria. De esta forma, el servidor se inicia como un código de rutinas normal de Crazyswarm y este se ejecuta directamente como comando de ROS2 luego de inicializar Crazyswarm. La lógica del algoritmo implementado en el servidor se muestra en la Figura 32

Una vez que se verificó el funcionamiento de la comunicación TCP, se procedió a evaluar la eficiencia de la comunicación. El objetivo inicial era generar las trayectorias para múltiples drones en Matlab y enviar los puntos en forma de arreglo. El problema que presentó este concepto es la cantidad de datos a enviar, ya que para generar las trayectorias se genera un punto cada 0.1 segundos, por lo que una trayectoria de 10 segundos representa 100 posiciones espaciales por Crazyflie, resultando en 3,000 datos de tipo *double* si se desea trabajar con 10 Crazyflies. Se redujo el periodo de duración de cada punto de las trayectorias a 1 segundo para evitar este problema, por lo que una trayectoria de 10 segundos con 10 Crazyflies genera 300 datos. Este cambio no genera inconvenientes con la implementación de las trayectorias en Crazyswarm, ya que esperar 1 segundo entre puntos no representa un problema de continuidad para los drones, siendo visualmente eficiente. Para asegurar que el paquete de datos puede ser procesado por el servidor, se definió adentro de este que cada paquete tiene un largo de 8525 bytes, dando espacio a que se trabajen con trayectorias de hasta 10 Crazyflies con una duración de 20 segundos.

Luego de múltiples pruebas con distintas rutinas de vuelo se observó un problema recurrente que se da cada vez que desde Matlab se envía un comando que genere un error en Crazyswarm, como por ejemplo enviar un comando para un Crazyflie que no existe o que no

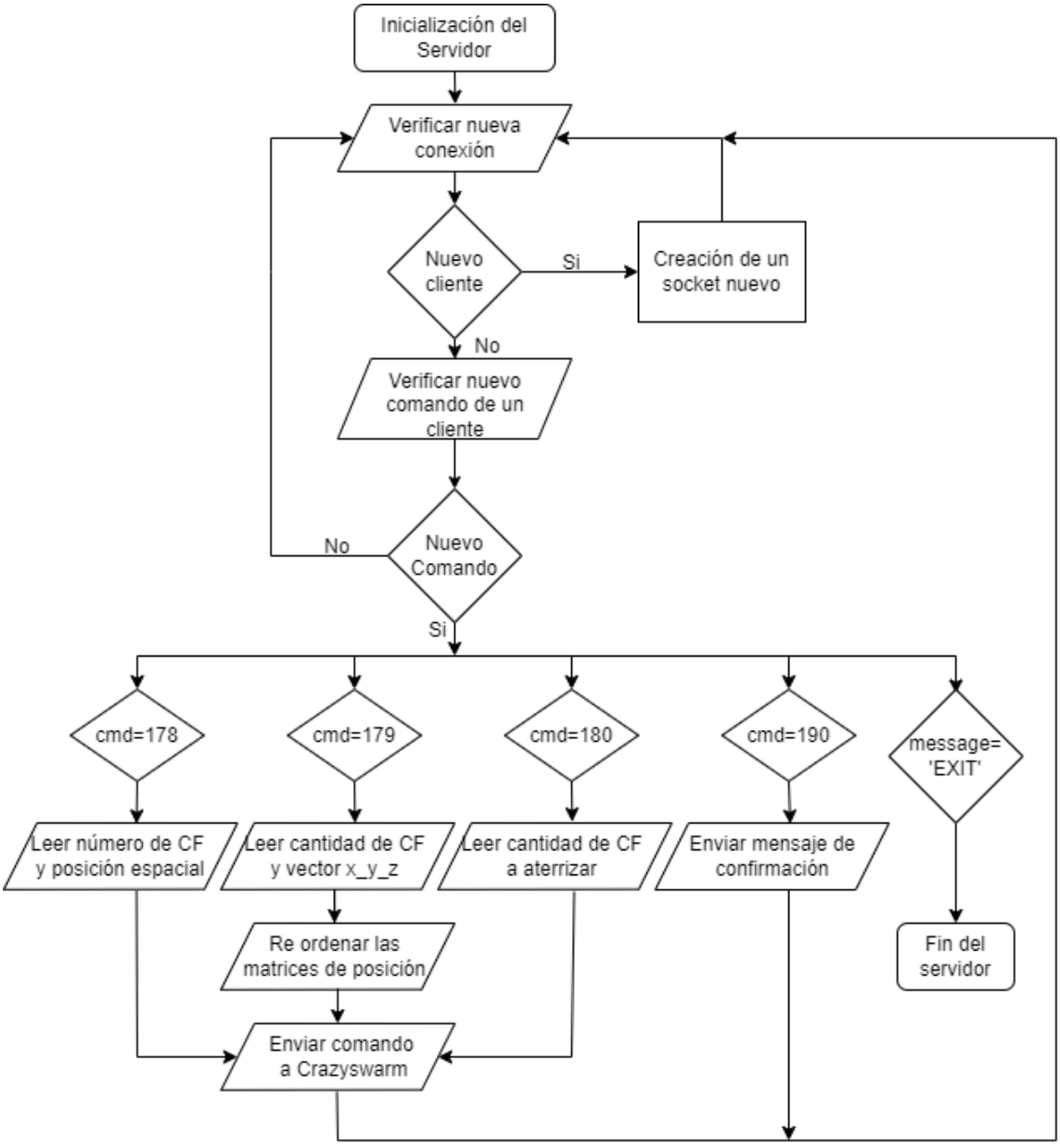

Figura 32: Diagrama de flujo del funcionamiento del servidor.

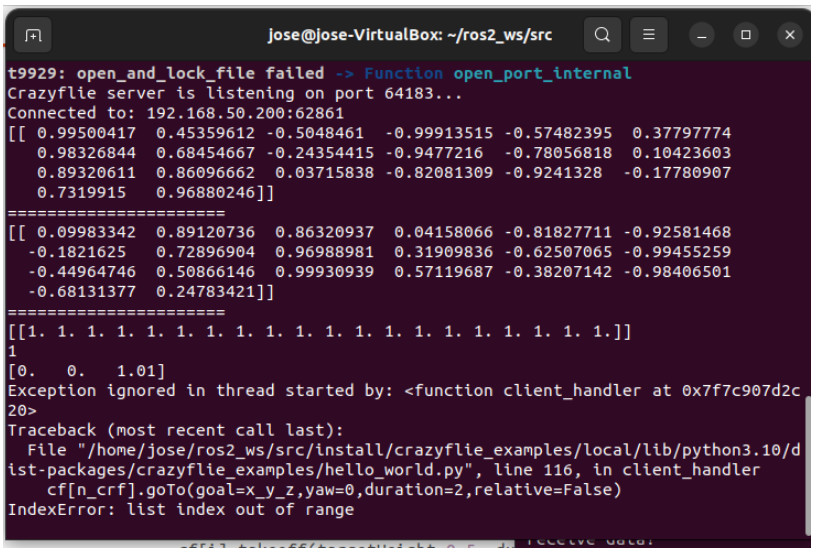

Figura 33: Error en el servidor.

fue inicializado. El problema que se presenta cuando se da este tipo de error es la pérdida en la conexión entre Matlab y el servidor, lo que requiere que el cliente vuelva a crearse. Esto puede volverse problemático ya que el servidor no indica si se perdió la conexión, además que pueden generarse errores en el servidor si se vuelve a crear un cliente sin que la conexión se haya perdido. Esto último obliga al usuario a reiniciar el servidor y por ende todas las conexiones en Matlab deben reiniciarse de igual forma. Reiniciar la conexión no presenta problemas a gran escala ni interfiere con el funcionamiento de Crazyswarm, sin embargo es necesario esperar un minuto o dos para volver a iniciar el servidor ya que luego de apagarlo este aun detecta que el puerto está en uso y no permite aceptar nuevas conexiones. Otra alternativa es cerrar la terminal de Ubuntu y abrir una nueva para volver a iniciar el servidor, tomando en cuenta que debe de volver a activarse la configuración de ROS. La Figura 33 muestra el error desplegado en la terminal de Ubuntu cuando desde uno de los clientes se envía un comando a un Crazyflie que no está inicializado, como se mencionó anteriormente esto puede provocar que el servidor deba reiniciarse en algunas ocasiones.

8.2. Códigos de comunicación por TCP en Matlab

Matlab provee una serie de herramientas que permite la comunicación por TCP con otros dispositivos conectados a la misma red mediante la función tcpclient(), la cual requiere de una dirección IP y un puerto, dicha función crea una variable de tipo *tcpclient* [24]. Una vez que se ha creado el cliente TCP en el espacio de trabajo, puede procederse a enviar y recibir datos del servidor mediante las funciones *write* y *read*, tomando en consideración que antes de escribir al cliente se debe crear la estructura del paquete con el formato mencionado anteriormente. Luego debe codificarse en formato JSON y convertirse en datos tipo uint8. Cabe destacar que, como se mencionó anteriormente, la comunicación se vuelve más eficiente cuando las matrices se envían como arreglos, por lo que las matrices de posición espacial se convirtieron en vectores y fueron concatenados en el pld. Cada función creada utiliza parámetros distintos y los manipula según la necesidad, pero todos presentan una estructura

similar a la siguiente

```
s.len = length(x);
s.cmd = 179;
s.pld = [N; x(:);y(:);z(:)];
write(tcp_obj, uint8(jsonencode(s)));
```

donde N representa el número de Crazyflies a utilizar y x, y y z son las matrices con las posiciones espaciales respectivas. Debido a que es preferible enviar las matrices como un arreglo en forma de vector de una sola dimensión con los datos concatenados, se envían las matrices en forma de vector. De esta forma 3 matrices de 10×10 se convierten en un vector con 300 dimensiones. Para alcanzar una integración eficiente y accesible para todos los usuarios del Robotat, se diseñaron funciones en Matlab que únicamente requieren de la especificación del o de los Crazyflies a utilizar y los distintos puntos de las trayectorias. Tomando esto en cuenta, dichas funciones cuentan con programación defensiva para evitar percances, como que se envíe un comando a un Crazyflie que no existe lo que podría causar fallas en Crazyswarm o como que se envíen posiciones espaciales que sobrepasen los límites del Robotat, lo cual puede incidir en daños a la Crazyflies, al equipo de laboratorio o a los usuarios. Las funciones creadas se muestran en el Cuadro 2.

Todas las funciones reciben como parámetro un objeto de tipo *tcpclient* de Matlab, a excepción de crazyswarm_connect() que genera dicho objeto.

8.3. Diseño de trayectorias en Matlab

Una de las ventajas más notables de utilizar Matlab como cliente principal, es la versatilidad que el programa presenta en el desarrollo de modelos matemáticos. El objetivo principal del algoritmo generador de trayectorias es que sea eficiente con cualquier cantidad de drones requerida. Para esto, se realizó un código que genera distintas trayectorias con las consideraciones del usuario, cada sección del código genera una trayectoria distinta a la que se le pueden modificar parámetros como duración, radio y/o amplitud de oscilación, altura máxima y cantidad de drones.

Todas las trayectorias se basaron en el modelo de un oscilador armónico y una parametrización polar para obtener una parametrización estándar. El movimiento circular de los drones puede modelarse con un movimiento armónico en ambas dimensiones. El resto de trayectorias se basaron en este mismo concepto variando la parametrización en función de cada geometría deseada, por ejemplo una trayectoria circular en el plano XY y una parábola en los otros planos se haría como en las Ecuaciones 3 y 4 junto la Ecuación 13, donde A y h representan las alturas máxima y mínima respectivamente.

$$z_k[n] = Ax[n]^2 + h \tag{13}$$

Este movimiento se ve como en la Figura 34 cuando se utilizan 10 Crazyflies.

Funciones de Matlab		
cmd	**Función**	**Descripción**
-	crazyswarm_connect	Crea el objeto TCP para conectar el cliente al servidor.
178	crazyswarm_goto	Envía a un Crazyflie especificado a una posición espacial.
179	crazyswarm_traj	Envía las trayectorias discretizadas para los Crazyflies deseados.
180	crazyswarm_land	Aterriza al mismo tiempo a todos los Crazyflies especificados.
190	crazyswarm_test	Verifica la comunicación con el servidor Crazyswarm.
-	crazyswarm_disconnect	Elimina el objeto TCP y corta la comunicación con el servidor.

Cuadro 2: Funciones de Matlab.

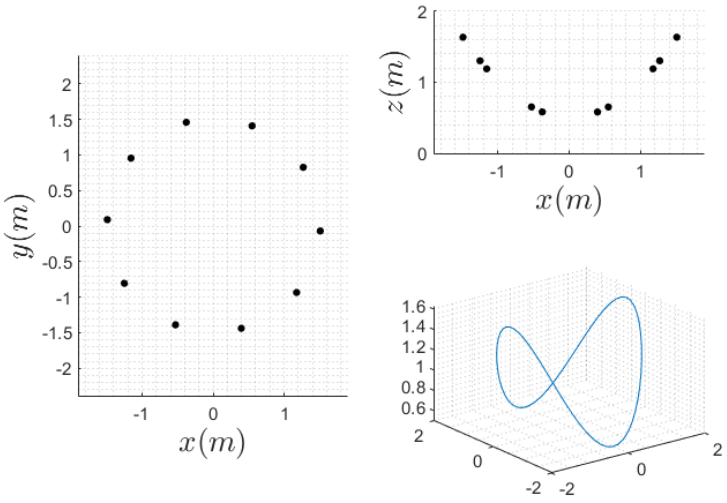

Figura 34: Trayectoria parabólica

8.4. Interfaz gráfica del Usuario en Matlab

Entre las principales metas de este proyecto es permitir que los estudiantes experimenten de forma sencilla y eficiente con Crazyswarm sin necesidad de contar con un conocimiento avanzado en sistemas de control, por lo que se cuenta con las funciones de Matlab ya mencionadas. Sin embargo, también se espera que estudiantes sin conocimiento en Matlab puedan realizar pruebas en el ecosistema y conocer sobre este, por lo que se diseñó una versión preliminar para una interfaz gráfica en Matlab que permite enviar posiciones espaciales a los Crazyflies especificados. La Figura 35 muestra el diseño actual para esta GUI. Esta interfaz utiliza las funciones creadas anteriormente tanto para generar las trayectorias deseadas como para enviar comandos a ROS2. Uno de los detalles más importantes a resaltar es que las funciones de Matlab requieren como argumento un objeto de tipo *TCPClient*, el cual corresponde al cliente del servidor Crazyswarm-Robotat, esto implica que el objeto debe ser global en el entorno de Matlab. Sin embargo esto no basta para un correcto funcionamiento de la interfaz, por lo que fue necesario establecer dicho objeto *TCPClient* como una propiedad especial de la aplicación, de esta forma puede establecerse la conexión sin inconvenientes y puede ser utilizada por las demás funciones para enviar comandos y trayectorias.

Entre las características implementadas en esta aplicación, se incluyeron:

- Botón para establecer la comunicación con el servidor Crazyswarm-Robotat.
- Botón desplegable para la selección de la trayectoria a ejecutar.
- Casilla para especificar la cantidad de Crazyflies a utilizar.
- Botón para iniciar una simulación de la trayectoria y gráfica que la muestra.
- Casilla para especificar el tiempo de duración de dicha trayectoria.

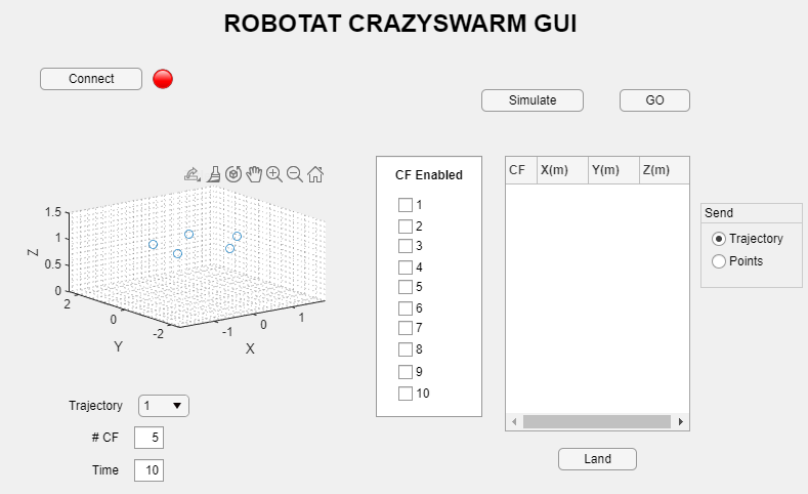

Figura 35: Interfaz gráfica para el uso de Crazyswarm.

- Casillas para especificar los Crazyflies que se desea enviar a una posición especificada.
- Casillas para especificar la posición espacial a la que se desea enviar a cada Crazyflie.
- Casilla para especificar si se desea enviar la trayectoria seleccionada o las posiciones puntuales especificadas.
- Botón para enviar el respectivo comando de ejecución basado en las opciones seleccionadas.
- Botón para aterrizar todos los Crazyflies que se encuentren en el aire.

8.5. Integración de Crazyswarm con otros agentes autónomos

Debido que la integración con el Robotat se realizó mediante un servidor local, este puede utilizarse también para manejar y redirigir la información recopilada por el sistema de captura de movimiento. OptiTrack proporciona un servidor llamado NatNetClient, el cual funciona de igual forma en Python mediante un protocolo TCP y permite obtener directamente la posición y la orientación de un cuerpo rígido, donde la orientación puede proporcionarse en forma de ángulos de Euler con una secuencia especificada o de cuaternión unitario. Las funciones de este cliente pueden importarse en forma de librería al servidor creado de Crazyswarm, permitiendo trabajar en conjunto con los Crazyflies y con el sistema de captura, de forma que otros agentes autónomos que requieran de un posicionamiento en tiempo real pueden implementarse en el Robotat junto con Crazyswarm.

Para realizar esta integración se intentó configurar el nodo en ROS2 que se encarga de publicar las mediciones del sistema de captura, de forma que tanto MOCAP4ROS2 como Crazyswarm fueran capaces de recibir y procesar el estado de marcadores en forma de cuerpos rígidos y no solo como *single markers*, como se estuvo trabajando desde un inicio. Una vez que se permitió el procesamiento de estos datos y que se incluyeron las funciones

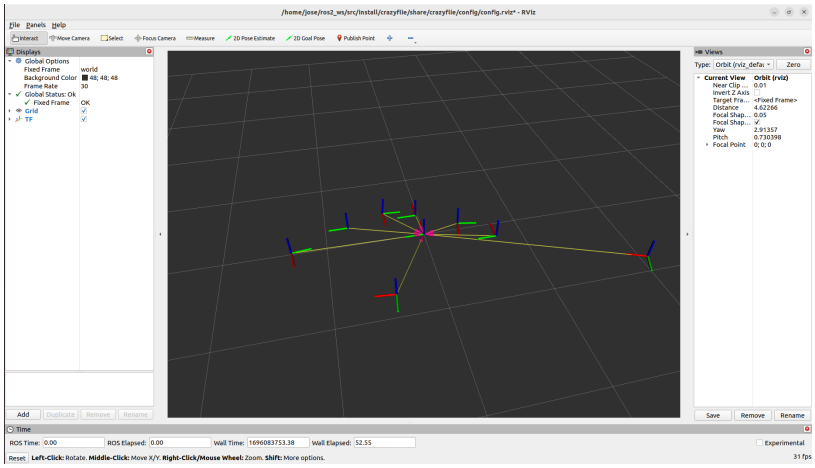

Figura 36: Interfaz de RViz con cuerpos rígidos junto con Crazyflies.

de NatNetClient en el servidor, se importaron las funciones existentes del Robotat para acceder en tiempo real a la posición y a la orientación de los cuerpos rígidos detectados por las cámaras. Cabe destacar la importancia de la utilización de *multi threading* para evitar un bloqueo en el funcionamiento del servidor mientras se realizan otras operaciones, de esta forma es posible solicitar el estado de un marcador al sistema de captura mientras se realiza una trayectoria con los Crazyflies sin que el servidor entre en conflicto.

La Figura 36 muestra como Crazyswarm es capaz de procesar la información del sistema de captura de movimiento tanto para *single markers* como para cuerpos rígidos sin que haya confusión en cuanto a la posición de los Crazyflies, cabe destacar que estos cuerpos rígidos deben ser activados previamente en Motive para que puedan visualizarse en Ubuntu.

Sin embargo este método no es factible para la integración con otros agentes, ya que el nodo encargado de tomar la información del sistema de captura en ROS no permite que otras funciones accedan a esta información. Para esto se aprovechó el servidor ya existente del Robotat y se le hicieron algunas modificaciones para que pudiera funcionar de forma simultanea con el servidor de Crazyswarm. Motive permite seleccionar hacia qué dirección IP enviar los datos procesados por el sistema de captura, entre estas opciones se encuentra la de enviar los datos de nuevo a la misma computadora para que el servidor Robotat pueda acceder a la información y enviarla a los clientes de este. Sin embargo esto funciona de una forma distinta para Crazyswarm, ya que son computadoras diferentes las que procesan la información del sistema de captura y albergan al servidor de Crazyswarm, por lo que en esta aplicación se seleccionó la dirección 192.168.50.200 para que toda la información sea enviada a Crazyswarm en Ubuntu. Para solucionar esto se hizo una modificación a la librería de NatNetClient, de forma que ahora toma la dirección 192.168.50.200 como dirección local de forma que, a pesar de que la información se envía a dicha dirección, el servidor Robotat es capaz de acceder de igual forma a la información. Esto se comprobó al inicializar Crazyswarm y solicitar la posición de un marcador en Matlab, lo cual funcionó correctamente.

Figura 37: Robot Pololu 3π

8.5.1. Pruebas con Robot Pololu

Una vez que se realizaron estas modificaciones se realizaron pruebas para verificar la factibilidad de utilizar la información del sistema de captura de forma simultanea con los Crazyflies. Para esto se crearon dos objetos tipo *tcpclient* en Matlab, uno para cada servidor, luego se le ordenó a un Crazyflie posicionarse medio metro por encima de la posición planar de uno de los marcadores y este lo realizó inmediatamente. Cabe resaltar que antes de cada movimiento del Crazyflie se le dio una pausa de 1 segundo al programa para evitar que este cambiara su dirección de forma tan abrupta y posteriormente cayera. Habiendo confirmado que se puede acceder a las posiciones de forma simultanea que se mueve un Crazyflie, se llevó a cabo una prueba en la que un Robot con ruedas Pololu 3π, como el que se muestra en la Figura 37 se puso en funcionamiento con un movimiento circular uniforme y se le colocó un marcador por encima para acceder a su posición en tiempo real, seguidamente se le ordenó desde Matlab a un Crazyflie que cada segundo actualizara su posición a la obtenida por el servidor de Robotat, dando como resultado un movimiento en el que el drone sigue de forma continua al robot Pololu.

Durante esta prueba de seguimiento, se observó que el movimiento circular del Crazyflie era pausado y no mostró ser eficiente, cabe resaltar que cada una de las posiciones espaciales era enviada en un paquete de datos diferente con un segundo de diferencia. Este movimiento no era tan fluido como en las trayectorias diseñadas en el código descrito anteriormente, el cual envía todas las posiciones en un solo paquete de datos hacia el servidor. A su vez es necesario considerar que la función utilizada para enviar las posiciones era *goto()*, la cual tiene asociada un movimiento espacial con una duración de 1 segundo adentro del servidor, lo cual no permite el movimiento fluido que se observó en las trayectorias implementadas con la función *traj()*.

8.5.2. Traslado del sistema de captura

El sistema de captura Optitrack funcionaba adentro del laboratorio CIT-116 originalmente, de forma que una de las computadoras del laboratorio inicializaba el programa Motive y contaba con las respectivas conexiones para comunicarse con las cámaras y crear el servidor

Figura 38: Servidor Robotat funcionando en un salón apartado.

Robotat. Sin embargo, uno de los objetivos principales de este laboratorio era procesar la información del sistema de captura en un ordenador aislado en un cuarto de servidores para conservar la estructura del Robotat lo más estandarizada y limpia posible. Esto conllevaba la tarea de realizar una conexión debajo del piso del laboratorio para comunicar el Switch que conecta las cámaras con el ordenador que se encontraría en el cuarto de servidores, lo cual presentaba un gran inconveniente con la utilización de Crazyswarm debido a que la computadora a utilizar para el sistema de captura debía contar con sus puertos habilitados para poder acceder a la información como se mencionó anteriormente. Este problema de puertos junto el inconveniente que se presentó inicialmente con no poder utilizar el sistema de captura simultáneamente con Crazyswarm impedía que pudiera realizarse el traslado del ordenador, debido a que no sería factible conectar una computadora al Switch cada vez que se quisiera utilizar Crazyswarm ya que ahora se tendría una conexión subterránea.

Una vez que se realizaron las pruebas mencionadas con el Robot Pololu y se validó el funcionamiento del sistema de captura paralelamente con el servidor Crazyswarm-Robotat, se realizó la respectiva gestión con el personal de IT de la Universidad del Valle de Guatemala para habilitar los puertos de la computadora ubicada en el cuarto de servidores (Figura 38). Una vez que los puertos fueron habilitados, fue posible establecer una comunicación entre la computadora con ROS y dicho ordenador, de forma que cada vez que se desee utilizar Crazyswarm basta con conectar la computadora con ROS al Router del Robotat (Figura 39) con un cable de red para acceder a la información del sistema de captura y esto no interferirá con el acceso a los usuarios hacia la misma información, por lo que pueden trabajarse distintos proyectos de forma simultanea.

8.6. Pruebas finales y validación

Para validar los resultados obtenidos en la experimentación de la integración, se pensó en utilizar una función propia de Crazyswarm para obtener la posición de los Crazyflies en tiempo real y verificar que las trayectorias seguidas fueran similares a las rutinas creadas en Matlab. Sin embargo, este tipo de funciones que acceden al estimador de los drones aún son propias de Crazyswarm 1 y no tienen soporte en Crazyswarm 2, por lo que se recurrió

Figura 39: Router de la red Robotat.

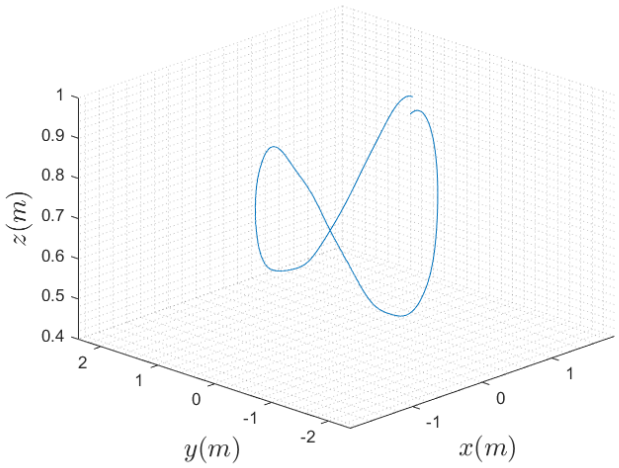

Figura 40: Trayectoria experimental 1.

a utilizar las herramientas de Motive para grabar los datos de las trayectorias.

Motive permite registrar datos de posiciones y orientaciones para los marcadores y cuerpos rígidos que se encuentren al alcance de las cámaras, pudiendo exportar la información en un archivo de tipo csv. Para validar los resultados y comprobar que los Crazyflies realizaban trayectorias similares a la diseñadas en Matlab, primero se compararon las trayectorias con el paraboloide elíptico. Luego de filtrar los datos y graficarlos se llegó a un resultado como el de la Figura 40. Como puede apreciarse, la trayectoria seguida es visualmente idéntica a la diseñada en Matlab (Figura 34), para validar los datos de forma cuantitativa se obtuvieron las distancias mínimas y máximas en los ejes en la trayectoria diseñada y en la real. Como puede observarse en el Cuadro 3, los errores en el posicionamiento de los extremos de la trayectoria no superan el 8.5 %, resultando en un error promedio del 3.85 %. Por otro lado, se realizó un análisis del error cuadrático medio resultando en un RMSE de 0.0432, por lo que se puede concluir que se cuenta con un error medio de 4.32 centímetros.

También se realizó la validación para la trayectoria conocida como elipsoide inclinado, de la misma forma en que se hizo con la trayectoria anterior. Los resultados se muestran en la Figura 41 y en el Cuadro 4. Como puede apreciarse en los porcentajes de error, esta trayectoria presentó un error máximo 2 veces menor al máximo de la trayectoria anterior y

Porcentajes de Error				
		Teórica(m)	Real(m)	Error(%)
Z	Máximo	0.9991	0.9374	6.18
	Mínimo	0.5297	0.4993	5.73
X	Máximo	0.9950	0.9897	0.53
	Mínimo	-0.9991	-1.0096	1.05
Y	Máximo	0.9699	0.9825	1.30
	Mínimo	-0.9258	-1.0028	8.32

Cuadro 3: Resultados de posiciones para trayectoria experimental 1.

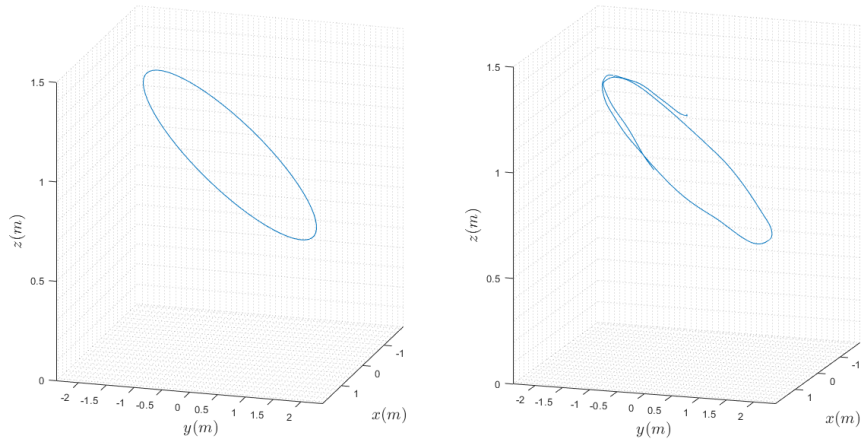

Figura 41: Validación de un Elipsoide inclinado

un error mínimo de 0 %, resultando en un error promedio de 1.3950 % y un error cuadrático medio de 2.23 cm, siendo este casi la mitad del RMSE de la trayectoria parabólica. Esto se debe a que el tipo de trayectoria presenta una continuidad suave y sus cambios con respecto al tiempo en los 3 ejes no presentan máximos o valores ascendentes de forma simultánea. A diferencia de dicha trayectoria, el paraboloide hiperbólico sí presenta valores ascendentes y máximos en los mismos instantes de tiempo, provocando movimientos y giros más agresivos como se aprecia en la Figura 42, donde se grafican las derivadas de las posiciones con respecto al tiempo y puede observarse que entre la medición 100 y 200 se presenta una velocidad en x con magnitud máxima y velocidades en y y z ascendentes. De esta forma, puede afirmarse que trayectorias suaves y continuas presentaran mayor eficiencia y fidelidad a los datos teóricos.

Porcentajes de Error				
		Teórica(m)	Real(m)	Error(%)
Z	Máximo	1.3750	1.3250	3.63
	Mínimo	0.6252	0.6084	2.67
X	Máximo	1.1248	1.1177	0.64
	Mínimo	-1.1247	-1.1139	0.96
Y	Máximo	1.4993	1.4994	0.00
	Mínimo	-1.4928	-1.4999	0.47

Cuadro 4: Resultados de posiciones para trayectoria elíptica.

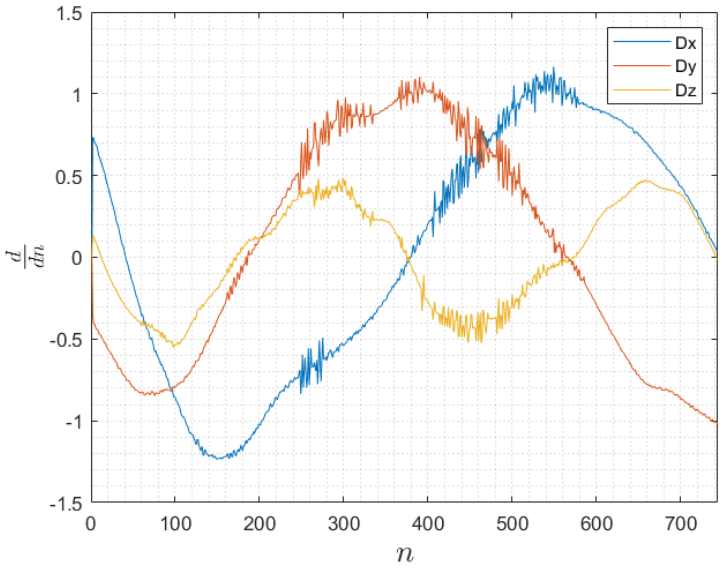

Figura 42: Derivadas del movimiento parabólico en 3 ejes.

Tanto en trayectorias circulares simples como en trayectorias complejas que combinan distintos movimientos (Figuras 26 y 34 respectivamente) la variación en los datos reales y los teóricos creados en Matlab fue relativamente pequeña, tal y como se discutió anteriormente. De esta forma puede apreciarse como la información transmitida mediante el servidor de Python no corrompió los datos y tanto las posiciones de las trayectorias como las direcciones de los Crazyflies y los comandos se mantuvieron intactos.

CAPÍTULO 9

Conclusiones

- Se levantó la infraestructura de Crazyswarm, implementando las cámaras de captura de movimiento OptiTrack y utilizando el método de *single marker* para determinar la posición espacial de los drones Crazyflie y posteriormente realizar códigos de control en Matlab y diseño de trayectorias, obteniendo una estabilidad notable con un error cuadrático medio de 0.007 metros para estabilización de posiciones espaciales.
- Se alcanzó una integración exitosa entre el sistema Crazyswarm y el Robotat a través de una comunicación TCP entre Matlab y Python, habiendo creado una comunicación cliente-servidor que permite el intercambio de datos entre la información del sistema de captura de movimiento y el comando de control *goto()*, para enviar a los Crazyflies a una posición deseada o implementar una trayectoria, esto resultando en un error cuadrático medio de 0.0437 metros para pruebas físicas con un periodo de muestreo de 1 segundo.
- El servidor implementado permite el desarrollo de proyectos relacionados con sistemas de control y robótica de enjambre, esto a través de la infraestructura de comunicación Crazyswarm-Robotat desarrollada y las librerías de NatNet para la obtención de poses en el Robotat, permitiendo expandir las capacidades del laboratorio Robotat y los campos de estudio con sistemas autónomos de distinta naturaleza.
- El laboratorio Robotat es ahora capaz de realizar experimentos con agentes robóticos de distinta naturaleza mediante la interacción entre el servidor Robotat y el servidor Crazyswarm-Robotat.

CAPÍTULO 10

Recomendaciones

- Se recomienda trabajar Crazyswarm en un ordenador que cuente con una partición de Ubuntu en lugar de una máquina virtual o desde una memoria externa, ya que esto limita las capacidades del sistema operativo y puede volver lenta la ejecución de Crazyswarm.
- Por cuestiones de seguridad de los usuarios del Robotat, se recomienda que durante la etapa de desarrollo e investigación del funcionamiento del sistema operativo se coloque una red o manta alrededor del espacio de trabajo del Robotat, para evitar accidentes que dañen a los usuarios, a los drones o al equipo del laboratorio.
- Se recomienda expandir las funcionalidades del servidor Crazyswarm desarrollado para que la comunicación sea más eficiente y con ello se facilite el desarrollo de aplicaciones, junto con la expansión en la gama de sistemas autónomos en el Robotat.
- Debido a que Crazyswarm2 aún está en desarrolló y no todas las funcionalidades están disponibles o documentadas, se recomienda utilizar Crazyswarm en su primera versión para expandir las funcionalidades del sistema y así poder implementarlas en un servidor similar al realizado en este trabajo.

CAPÍTULO 11

Bibliografía

[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. Perafan, "Robotat: un ecosistema robótico de captura de movimiento y comunicación inalámbrica," Tesis de licenciatura, Universidad Del Valle de Guatemala, 2022.

[3] "Crazyswarm Documentation." (), dirección: `https://crazyswarm.readthedocs.io/en/latest/`. (Accedido el: 21/4/2023).

[4] A. Lab. "Crazyswarm: ACT Lab." (). (Accedido el: 24/9/2023).

[5] K. P. Galea. B, "Tetheredflightcontrolofasmallquadrotorrobotforstippling," *McGill University*, S.f.

[6] "Crazyflie 2.1 | Bitcraze." (), dirección: `https://www.bitcraze.io/products/crazyflie-2-1/`. (Accedido el: 13/5/2023).

[7] N. Nise, *Control Systems Engineering, 7th edition*. John Wiley Sons, Inc., 2015.

[8] "Kalman Filter." (), dirección: `https://www.sciencedirect.com/topics/earth-and-planetary-sciences/kalman-filter`. (Accedido el: 18/8/2023).

[9] "State estimation." (), dirección: `https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/sensor-to-control/state_estimators/`. (Accedido el: 20/5/2023).

[10] G. EwoudJ.J.Smeur QipingChu, "Adaptive Incremental Nonlinear Dynamic Inversion for Attitude Control of Micro Air Vehicles," *Journal of Guidance, Control, and Dynamics: devoted to the technology of dynamics and control*, pág. 1, 2016.

[11] J. Murillo, *Fundamentos de Radiación y Radiocomunicación, 2da edición*. Universidad de Sevilla., 2012.

[12] "Multithreading, más potencia para los procesadores." (), dirección: `https://www.ionos.es/digitalguide/servidores/know-how/explicacion-del-multithreading/`. (Accedido el: 9/9/2023).

[13] "Socket Programming in Python." (), dirección: https://realpython.com/python-sockets/. (Accedido el: 18/8/2023).

[14] "¿Qué es el UDP?" (), dirección: https://www.cloudflare.com/es-es/learning/ddos/glossary/user-datagram-protocol-udp/. (Accedido el: 13/5/2023).

[15] "Formato JSON (JavaScript Object Notation)." (), dirección: https://www.ibm.com/docs/es/baw/20.x?topic=formats-javascript-object-notation-json-format. (Accedido el: 13/5/2023).

[16] "Primex-40-Specs." (), dirección: https://optitrack.com/cameras/primex-41/specs.html. (Accedido el: 21/4/2023).

[17] "MOCAP4ROS2." (), dirección: https://mocap4ros2-project.github.io/index.html. (Accedido el: 18/8/2023).

[18] S. J, *Cálculo de varias variables. 7ma edición. Trascendentes tempranas*. Cengage Learning, 2012.

[19] "Overview-Crazyswarm2." (), dirección: https://imrclab.github.io/crazyswarm2/overview.html. (Accedido el: 22/4/2023).

[20] "CRTP-Communication with the Crazyflie." (), dirección: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/crtp/. (Accedido el: 6/5/2023).

[21] "Crazyflie Android client." (), dirección: https://www.bitcraze.io/documentation/repository/crazyflie-android-client/master/. (Accedido el: 18/8/2023).

[22] M. C. Falconi. R, "Dynamic Model and Control of an Over-actuated Quadrotor UAV," *International Federation of Automatic Control*, pág. 1, 2012.

[23] "Python API Reference." (), dirección: https://crazyswarm.readthedocs.io/en/latest/api.html. (Accedido el: 18/8/2023).

[24] "Comunicación TCP/IP." (), dirección: https://la.mathworks.com/help/matlab/tcpip-communication.html. (Accedido el: 18/8/2023).

CAPÍTULO 12

Anexos

- Lista de reproducción de Youtube con vídeos que muestran el funcionamiento de Crazyswarm: `https://www.youtube.com/playlistlist=PL3x4dGAXnVd2yz4xZyO93wSEfZuOLzN6-`
- Repositorio de GitHub con la documentación `https://github.com/JulioAv/CrazyswarmServer`

CAPÍTULO 13

Glosario

firmware: Programa interno de un dispositivo eléctrico que controla su funcionamiento, no suele modificarse.. 21

Github: Plataforma para almacenamiento de proyectos mediante el control de versiones git.. 24

headers: Conectores hembra para la conexión de pines eléctricos.. 20

pitch: Ángulo de desplazamiento angular de un sistema con respecto al eje y de su propio marco de referencia, también denominado ángulo de ataque.. 8, 10, 20, 22, 23

PWM: Señal eléctrica cuadrara y periódica que es modulada mediante la modificación del ciclo de trabajo.. 21, 22

roll: Ángulo de desplazamiento angular de un sistema con respecto al eje x de su propio marco de referencia.. 8, 10, 20, 22, 23

thrust: Fuerza de empuje que genera uno o más motores de un sistema , siendo este vertical para drones.. 10, 20–24

yaw: Ángulo de desplazamiento angular de un sistema con respecto al eje z de su propio marco de referencia.. 8

yawrate: Velocidad con la que cambia el ángulo de rotación alrededor del eje z de un sistema.. 22