Levantamiento de una plataforma de pruebas para sistemas multidrones con OptiTrack y Crazyswarm

Brandon Amisael Garrido Ramírez

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Levantamiento de una plataforma de pruebas para sistemas multidrones con OptiTrack y Crazyswarm

Trabajo de graduación presentado por Brandon Amisael Garrido Ramírez para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2024

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Levantamiento de una plataforma de pruebas para sistemas multidrones con OptiTrack y Crazyswarm

Trabajo de graduación presentado por Brandon Amisael Garrido Ramírez para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2024

Vo.Bo.:

(f) _____
MSc. Miguel Zea

Tribunal Examinador:

(f) _____
MSc. Miguel Zea

(f) _____
Ing. Jonathan Mansilla

(f) _____
Ing. Christopher Chiroy

Fecha de aprobación: Guatemala, 20 de enero de 2024.

Prefacio

Después de muchos sacrificios al llegar a la etapa final de este largo recorrido, deseo expresar mi sincero agradecimiento a tres pilares fundamentales en mi vida.

En primer lugar, a mi familia, cuyo amor y apoyo incondicional han sido un ancla que me ha mantenido en pie a lo largo de todas las etapas de mi vida.

A Dios, quien ha sido mi guía y fuente de fortaleza en momentos de dificultad y duda.

Y a mis amigos, con quienes he compartido risas, alegrías y desafíos a lo largo de esta travesía.

Este logro no habría sido posible sin cada uno de ustedes. Mi gratitud es eterna.

Índice

Lista de figuras

Lista de cuadros

Resumen

En este trabajo, se aborda la necesidad de establecer una infraestructura para experimentación y control de múltiples drones en la Universidad del Valle de Guatemala. Para esto se integró el sistema de captura de movimiento OptiTrack y la librería Crazyswarm 2, para el control de múltiples drones y llevar a cabo una variedad de experimentos. Durante este proceso, se realizaron pruebas de configuración y funcionamiento con los drones Crazyflie 2.1 disponibles en la universidad, y se determinó que la configuración óptima incluye el controlador Mellinger y el estimador de Kalman.

Además, se llevó a cabo la implementación de las librerías Mocap4ROS2 y NatNet, con lo cual se estableció la conexión con el programa Motive para obtener transmitir datos de la posición de los drones. Esto permitió obtener mediciones de posición tomadas por el OptiTrack para retroalimentar el control del vuelo de los drones. Dicha comunicación se estableció con una velocidad de 1.0 ms y una pérdida de paquetes de 1.0 %. Finalmente implementando la librería Crazyswarm 2, se realizó pruebas de vuelo de los Crazyflies 2.1 con trayectorias estáticas y dinámicas. Se obtuvo un porcentaje de error promedio de 0.3 % respecto a la trayectoria teórica esperada.

Abstract

This work addresses the need to establish an infrastructure for experimentation and control of multiple drones at the Universidad del Valle de Guatemala. To achieve this, the OptiTrack motion capture system and the Crazyswarm 2 library were integrated for the control of multiple drones to conduct a variety of experiments. Throughout this process, configuration and functionality tests were conducted using the Crazyflie 2.1 drones available at the university. It was determined that the optimal setup includes the Mellinger controller and the Kalman estimator.

Furthermore, the implementation of Mocap4ROS2 and NatNet libraries was carried out to establish connection with the Motive program, enabling the transmission of drone position data. This facilitated the acquisition of position measurements by OptiTrack to provide feedback for drone flight control. The communication was established at a speed of 1.0 ms with a packet loss of 1.0 %. Finally, by implementing the Crazyswarm 2 library, flight tests were conducted with the Crazyflies 2.1 using both static and dynamic trajectories. An average error percentage of 0.3 % was obtained concerning the expected theoretical trajectory.

CAPÍTULO 1

Introducción

El avance constante de la robótica y la creciente relevancia de los sistemas multi-agente han dado lugar a investigaciones y desarrollos significativos en este campo. Los sistemas multi-agente implican la colaboración y coordinación entre múltiples robots, y su aplicabilidad se ha expandido a diversas áreas. En este contexto, el presente trabajo se adentra en la exploración y desarrollo de un sistema especializado para la coordinación y formación de un enjambre de drones, centrándose específicamente en los drones Crazyflie 2.1. Esta iniciativa se apoya en el *framework* Crazyswarm 2, diseñado para el control multidrones, y se integra con el Sistema Operativo para Robots (ROS 2) para lograr un enfoque integral y altamente versátil.

El proceso de implementación se dividió en tres etapas fundamentales desarrolladas en cada uno de los capítulos de este trabajo, que en conjunto cumplen con cada uno de los objetivos específicos del trabajo de graduación. En primer lugar, se llevó a cabo una fase de experimentación con los sensores internos del dron. Esta implicó la exploración de una amplia gama de parámetros de configuración disponibles en el Crazyflie 2.1, entre los que destacan el controlador, el estimador y el *firmware*, con el objetivo de definir los parámetros óptimos.

Posteriormente, se procedió a la integración del sistema de captura de movimiento, para realizar la fusión de las mediciones de los sensores internos del dron con los datos del Mocap. Esto resultó fundamental para obtener mediciones precisas de la orientación y posición del dron, lo que contribuyó en la mejora de estabilidad y el control de vuelo.

Finalmente con una base sólida establecida a través de la integración del Mocap, se realizó experimentos con algoritmos de control de vuelo y trayectorias específicas para los drones, aprovechando las capacidades y recursos proporcionados por la librería Crazyswarm 2. Estos algoritmos permitieron una coordinación precisa de múltiples drones en un enjambre, añadiendo una capa adicional de versatilidad y adaptabilidad a la infraestructura.

CAPÍTULO 2

Antecedentes

Los vehículos aéreos no tripulados (UAV por sus siglas en inglés), o mejor conocidos como drones, son un tipo de robots que buscan imitar la biología de las aves con el fin de facilitar al ser humano realizar tareas que requieren surcar los aires [1]. Tienen distintas aplicaciones que van desde para usos militares, hasta vigilancia y transporte. Sin embargo, previo a poder utilizarlos es necesario el diseño y desarrollo de una plataforma para poder realizar el control de los mismos, principalmente cuando se trate de grandes grupos de drones. Estos entornos permiten el control de enjambres de drones para permitir el funcionamiento en conjunto de estos como una inteligencia colectiva, tanto en entornos cerrados como a la interperie.

2.1. Experimentación de CrazyFlie 2.0 con máquina virtual

En la Universidad del Valle de Guatemala, Francisco Sanabria [2] realizó una fase inicial previa al levantamiento de esta plataforma de pruebas, en donde validó el funcionamiento y control de los drones CrazyFlie 2.0 a través de una máquina virtual, logrando realizar la conexión del dron a la aplicación y realizando pruebas físicas preliminares con estos con una interfaz gráfica desarrollada en Python.

2.2. Plataformas para control multidrones en entornos cerrados y a la interperie

Se pueden mencionar además, trabajos como el realizado en el laboratorio GRASP de la Universidad de Pensylvania [3] E.E.U.U, que realizó una serie de experimentos para un enjambre de nanodrones dentro de un entorno cerrado utilizando un sistema de captura de movimiento VICON, que permitió mediante un sistema cerrado de cámaras y con marcadores, capturar la pose del sistema de drones y mediante el *framework* implementado simultáneamente realizar el control.

Uno de los *frameworks* disponibles para el control de enjambres de drones es SwarmSim el cual es una alternativa para la simulación de sistemas multidrones. Sin embargo, la opción utilizada para esta implementación fue Crazyswarm, la cual es una plataforma que permite controlar sincronizaciones de enjambres de drones en espacios cerrados mediante sistemas de localización como captura de movimiento, LPS y LigthHouse. Sin embargo, Crazyswarm está optimizado para integrarse con sistemas de captura de movimiento, tales como OptiTrack, VICON, NOKOV y Qualisys [4] siguiendo la arquitectura para la implementación del sistema como se muestra en la Figura 1.

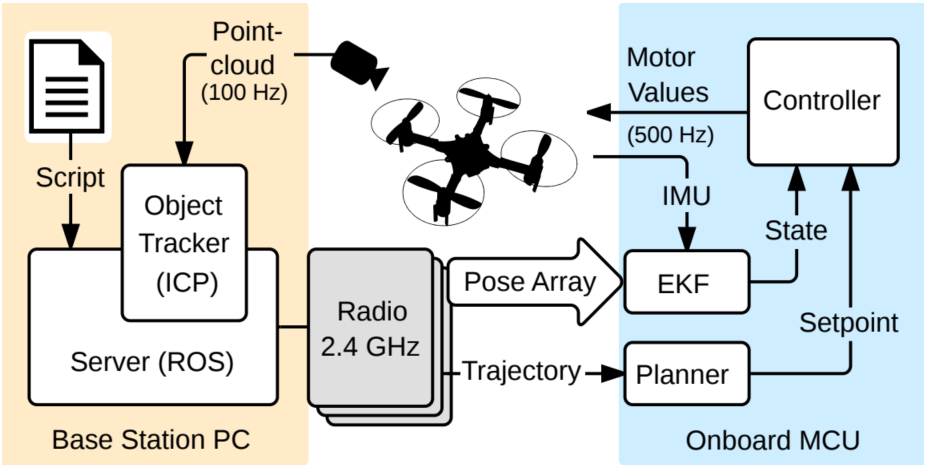


Figura 1: Arquitectura del sistema Crazyswarm [5].

Por otro lado, para entornos a la interperie se desarrolló un sistema para el control de multidrones basado en marcadores visuales que, en conjunto con algoritmos de rastreo y una red neuronal convolucional, se emplean para detectar agentes cercanos a los drones en tiempo real. Mediante un sistema de 4 cámaras se estiman las posiciones y velocidades relativas de los agentes cercanos para alimentar el algoritmo de control. Además, con GPS y los marcadores de cada dron, se permite configurar la cohesión de los sistemas de drones [6]. Se utiliza DarkNet que es un *framework* de red neuronal, que permite la detección de objetos en tiempo real a través de YOLO y clasificación de imágenes mediante ResNeXt para poder entrenar la red neuronal.

Sistemas como este se implementaron en experimentos como el realizado por la agencia estadounidense DARPA, que realizó pruebas con un conjunto de 250 drones con el propósito de implementarlos para soporte militar como parte del programa OFFSET [7]. Asi mismo, en proyectos como la coordinación de múltiples drones para explorar el bosque, generando trayectorias más eficientes y funcionando como una inteligencia de enjambres realizado por estudiantes de la universidad de Zhejian en China [8].

CAPÍTULO 3

Justificación

El propósito de este trabajo de graduación se fundamenta en la necesidad de aprovechar los recursos tecnológicos disponibles en la Universidad del Valle de Guatemala, tales como el sistema de captura de movimiento OptiTrack y el enjambre de drones Crazyflie 2.1. Estos recursos representan una oportunidad única para explorar y desarrollar una plataforma de control multidrones, un campo que aún no ha sido explorado en el país.

La combinación de estos elementos con el framework Crazyswarm 2, diseñado para configuraciones sincronizadas de drones, permitirá optimizar el control y la coordinación de múltiples drones de manera eficiente. En particular, el uso de OptiTrack para el rastreo de la posición de los drones mediante marcadores proporciona una ventaja significativa, para desarrollar un sistema altamente preciso y confiable.

Con este trabajo se busca sentar las bases para el desarrollo de una plataforma de control multidrones en Guatemala, similar a las utilizadas en otras instituciones de renombre, como la Universidad de Pensilvania en los Estados Unidos, que utiliza el sistema de captura VICON, para la experimentación con multidrones. Al lograrlo, se abrirán nuevas posibilidades para la investigación y la experimentación en el campo de los enjambres de drones en el país.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Adaptar Crazyswarm al sistema de captura de movimiento OptiTrack para el levantamiento de una plataforma que permita realizar distintos experimentos con múltiples drones.

4.2. Objetivos específicos

- Instalar las dependencias y configurar el entorno de desarrolllo para la comunicación y control de los Crazyflie 2.1 bajo los requerimientos del framework Crazyswarm.
- Implementar el framework Crazyswarm para el control de un conjunto de drones Crazyflie 2.1.
- Integrar al sistema de captura de movimiento OptiTrack el control de los multidrones.

CAPÍTULO 5

Alcance

El alcance de este proyecto engloba la configuración y el uso de varios drones Crazyflie 2.1, los cuales se encuentran disponibles dentro Universidad del Valle de Guatemala. Al realizar las pruebas con los drones, se determinó que utilizando las baterías disponibles, la duración del vuelo continuo fue de aproximadamente 8 minutos con 50 segundos, por lo cual se tuvo que tener a disposición una cantidad considerable de estas para poder realizar las pruebas a lo largo de la experimentación.

Además, cabe resaltar que dicha plataforma de pruebas se desarrolló utilizando las bibliotecas Mocap4ROS y NatNet, para obtener datos desde Motive. Con la disponibilidad de un conjunto de seis cámaras OptiTrack. La transmisión de datos se realizó entre sistemas operativos, con Motive funcionando en una computadora con Windows como servidor de datos, ya que se encontró la limítación que dicho software está disponible en dicho sistema operativo. Además, una computadora corriendo el sistema operativo basado en Linux Ubuntu, actuando como cliente y recibiendo los datos a través de un nodo implementado y configurado con NatNet en ROS2.

El desarrollo de la infraestructura de pruebas integrando ROS2 con Crazyswarm 2 se realizó en un *dongle* USB de 64 GB, donde se creó una partición del sistema operativo de Linux con persistencia de datos. Dentro de esta partición, se llevó a cabo la instalación de todas las librerías de desarrollo. Esta limitación surgió debido a la infraestructura de puertos disponible en la universidad en ese momento, lo que dificultó la adaptación de dicho sistema a las configuraciones existentes.

Adicionalmente, se adaptó diferentes nodos en ROS2 para Crazyswarm 2, lo que permitió establecer una comunicación bidireccional con el dron a través de una antena de radiofrecuencia CrazyRadio PA. Este enfoque posibilitó la integración de los datos de los sensores de estimación del Crazyflie con los datos del sistema de captura de movimiento.

CAPÍTULO 6

Marco teórico

6.1. Sistema OptiTrack

OptiTrack es un proveedor de Sistemas de Captura de Movimiento (Mocap), que proporciona sofware para la captura de movimiento, así como hardware de rastreo de alta calidad y velocidad del movimiento de un cuerpo rígido a través de marcadores [9].

El principio fundamental para obtener el posicionamiento del cuerpo es que las cámaras emiten luz infrarroja, que se refleja mediante marcadores reflectantes especiales. Esto permitirá que las cámaras infrarojas (IR) detecten la ubicación del marcador, desde el cual el *software* del Mocap en una computadora externa, calculará la posición real [10].

El sistema de captura de movimiento en los laboratorios de la Universidad del Valle de Guatemala consta de 6 cámaras de alta precisión con tecnología Primex41 compatibles con sofware de captura de movimiento Motive y Camera SDK. Dicho modelo proporciona las siguientes características [11]:

Errores máximos para marcadores activos y pasivos:

- Posición: +/-0.10mm.
- 0.5 Grados.

- Dimensiones: 12.6cm x 12.6cm x 13.2cm.
- Peso: 1.36kg.
- Resolución: 2048x2048.
- Velocidad: 180FPS (nativa) 250+FPS (máxima).
- Latencia: 5.5ms.

- Rango: 30.48 m.
- Puerto de datos: GigE (1000BASE-T).
- Sincronización de Cámara: Ethernet.
- Alimentación: PoE or PoE+1.

Figura 2: Cámara OptiTrack Prime x41 [9].

6.2. Crazyflie 2.1

El Crazyflie 2.1 es una plataforma de vuelo de código abierto desarrollada por Bitcraze, que puede observarse en la Figura 3. Tiene un peso de aproximadamente 27g y un tamaño de motor a motor, incluyendo los pies de montura, de 92mm x 92mm x 29mm. Utiliza el sistema operativo en tiempo real FreeRTOS y dos microcontroladores (MCU), un STM32F405 para el *handling* y el ciclo de vuelo así como un nRF51822 para el control de la comunicación por radiofrecuencia y la batería [12].

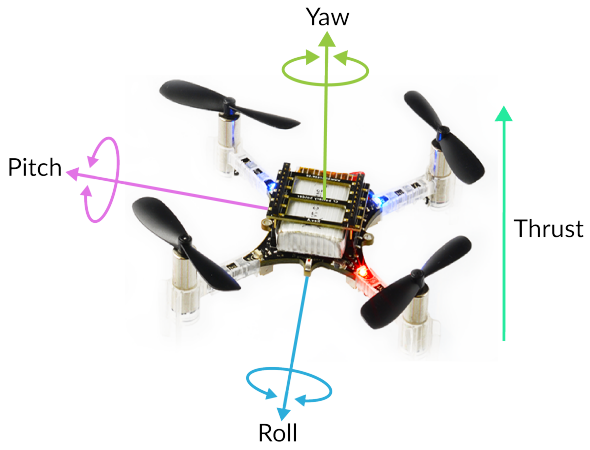


Figura 3: Drone Crazyflie 2.1[13].

Para la comunicación con el Crazyflie 2.1 se utiliza CrazyRadio 2.0 con la cual es posible transmitir los paquetes de información requeridos de forma bidireccional utilizando una

antena de radio frecuencia USB de 2.4GHz de largo alcance basada en el nRF52840. Funciona como medio amplificador de 20dBm con baja amplificación de ruido (LNA) teniendo un alcance de hasta 2km de distancia en óptimas condiciones [14].

Figura 4: CrazyRadio 2.0 [14].

6.3. Crazyswarm

Crazyswarm es una plataforma que permite volar un enjambre de drones Crazyflie 2.x en configuraciones sincronizadas, así como drones basados en Crazyflie Bolt. Este *framework* soporta diferentes sistemas de localización de posición tales como LightHouse, LPS y captura de movimiento. Para los sistemas de captura de movimiento permite trabajar con VICON, NOKOV, OptiTrack y Qualisys [15].

Esta plataforma se diferencia de la API (Python) Crazyflie desarrollada en un inicio por Bitcraze para el control de los drones, ya que el programa del servidor de Crazyswarm es un nodo de ROS que utiliza código de alto nivel de Python para el control de enjambres. Además Crazyswarm contiene controladores para sistemas comunes de captura de movimiento lo que permite, de forma nativa, rastrear múltiples drones con un sólo marcador cada uno. Crazyswarm también permite generar un modelo de simulación 3D que permite validar los algoritmos de control multidrones antes de ejecutarlos en *hardware* real[15].

6.4. Crazyflie *server*

Según se hace referencia documentación de Crazyswarm2 v1.0 [16], el nodo del servidor Crazyflie controla varios aspectos de comunicación de bajo nivel con los Crazyflies además de permitir conectar múltiples drones con uno o más dispositivos PA de CrazyRadio.

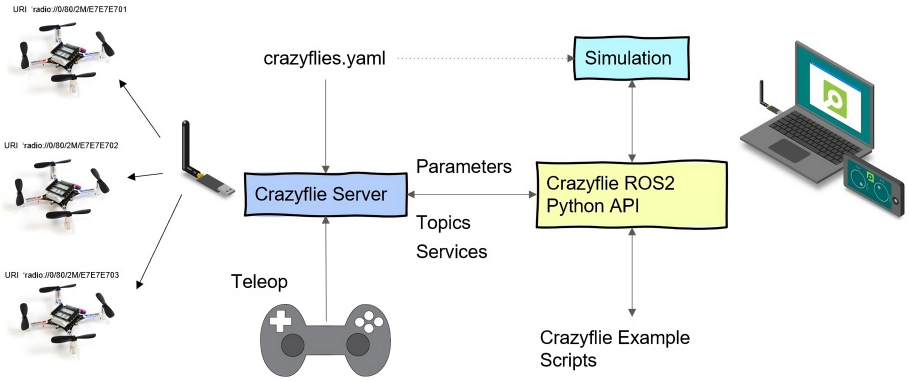


Figura 5: Nodos Crazyflie ROS 2 [16].

El servidor transforma los parámetros de la tabla de contenidos (ToC) del Crazyflie en parámetros de ROS2. Además, configura los bloques de registro para la transmisión de datos en Crazyflie y permite la transformación de las variables recibidas en datos o tópicos de ROS2. Tanto los parámetros como los bloques de registro se pueden configurar durante el tiempo de ejecución mientras el servidor está conectado a los Crazyflies.

También configura los servicios de comando de vuelo del Crazyflie tales como:

- *Takeoff / Land / GoTo*: Permite, con un solo comando al servidor, enviar una coordenada o altura específica hacia el cual los Crazyflie ejecturán la secuencia de despegue, se dirigirán al punto especificado y posteriormente aterrizarán.
- *Upload / Start trajectory*: Se podrá predefinir una trayectoria la cual los Crazyflies realizarán cuando se les indique.
- *Emergency*: Permite apagar los motores en caso de algún contratiempo durante el vuelo.
- /*all* o /cfx : Indica si el comando aplicará para todos los drones Crazyflie que respondan al broadcast o solo a alguno en específico.

6.5. CrazyRadio Real Time Protocol (CTRP)

El crazyflie recibe la información determinada para su control y movimiento de acuerdo a lo proporcionado a través de la antena de radio frecuencia CrazyRadio PA, que envía los datos a través de un protocolo de envío de paquetes en tiempo real (CTRP).

El CrazyRadio *Real Time Protocol* (CTRP) describe el protocolo de paquetes que se utiliza para comunicarse con los Crazyflies, cuyos datos proporcionados se enrutan a los diferentes subsistemas del dron. Dicho protocolo de comunicación se implementa en distintas capas independientes, como se muestra en la Figura 6.

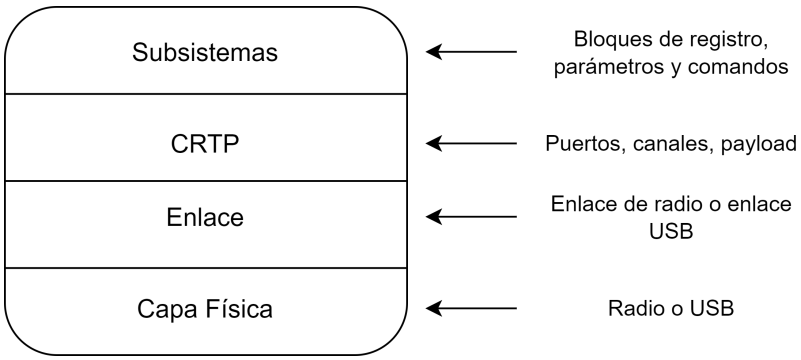


Figura 6: Protocolo CTRP[17] .

- Capa física: Es la responsable de transmitir los paquetes de forma bidireccional entre el servidor y los Crazyflies, puede ser tanto por USB o radio [17].
- Enlace: Esta capa es la responsable de la implementación segura y ordenada de los paquetes de información intercambiados con el Crazyflie. Abstrae el medio físico e implementa los canales apropiados para el intercambio de datos. El enlace garantiza un estricto pedido de paquetes dentro de un puerto así como la reorganización de estos y envíos fuera de servicio [17].
- CRTP: Es un protocolo diseñado para priorizar paquetes y ayudar al control del Crazyflie. Permite garantizar que ninguna instrucción quede sin procesar en un puerto, tal como el envío de alguna trayectoria de control al dron. En esta capa se implementa la información obtenida de los puertos y canales para enrutar los datos a los diferentes subsistemas. Los puertos funcionan de manera similar a los puertos UDP, cada uno de ellos asignado a los diferentes subsistemas del Crazyflie. Funciona con un sistema de prioridad en el cual se le asigna mayor valor a un número de puerto inferior [17].
- Subsistemas: Implementa las funcionalidades del Crazyflie, controlando el levantamiento, envío y desactivación de los paquetes de registro de datos desde el dron al cliente y los parámetros definidos utilizando un macro definido en el código fuente del Crazyflie, enviando *setpoints* instantáneos de bajo nivel para regular la orientación *roll/pitch/yaw/thrust* del dron [17].

Cada paquete carga 4 *bits* que indican el número de puerto a utilizar, 2 bits para el canal y una carga útil de 31 *bytes*. Los puertos CTRP se enumeran del 0 al 15 y cada uno se asigna a un subsistema del Crazyflie, lo cual permite una comunicación bidireccional entre un subsistema y el control del dron sobre el suelo[17].

Puerto	Objetivo
0	Consola
2	Parámetros
3	Comandos
4	Memoria
5	Registro de datos
6	Localización
7	Punto de ajuste genérico
13	Plataforma
14	Depuración del lado del cliente
15	Capa de enlace

Cuadro 1: Puertos CTRP[17].

Según la documentación de Bitcraze, respecto a la comunicación con paquetes CTRP [17] los distintos subsistemas que se implementan son:

- Consola: Este puerto se utiliza como consola para imprimir texto de forma unidireccional desde el Crazyflie con una longitud máxima de 31 *bytes*.
- Parámetros: Permite la lectura o modificación de parámetros que se encuentran en la tabla de contenido (ToC) del dron.
- Comandos: Se utiliza para enviar puntos de ajuste de control para *roll/pitch/yaw/thrust* desde el servidor al Crazyflie. Cada rotación se envía con una longitud de 4 *bytes*, mientras el *thrust* en 2 *bytes*. Una vez establecido el vínculo de la comunicación los paquetes se pueden enviar y son válidos hasta que se recibe el siguiente.
- Memoria: Permite la lectura y escritura de la memoria de datos del Crazyflie.
- Registro de datos: Permite la configuración de bloques de registro con variables que se enviarán de vuelta a Crazyflie en un período específico.
- Localización: Este puerto en el canal 0, se utiliza para enviar la posición de Crazyflie adquirida por un sistema externo. El uso principal es enviar la posición adquirida por un sistema de captura de movimiento para empujarlo en el filtro de Kalman extendido del Crazyflie para calcular una estimación y controlar su estado.

  En el canal 1, permite alojar paquetes útiles para la el subsistema de localización. Ha sido creado para servir a los paquetes del Sistema de Posicionamiento Loco en el cual utiliza el estimador de Kalman para estimar la posición. Sin embargo, se pueden usar para cosas más generales como GPS NMEA o transmisiones de paquetes binarios.
- Punto de ajuste genérico: Este puerto permite enviar puntos de ajuste instantáneos como velocidad, altura, posición y altitud.
- Plataforma: Se utiliza para la depuración de forma directa del Crazyflie.
- Depuración del lado del cliente: Depurar la interfaz de usuario y existe sólo en la API de Python y no en la propia Crazyflie.

- Capa de enlace: Puerto que trabaja con la capa de enlace de bajo nivel que permite hacer *ping* al Crazyflie.

Según lo descrito en la página oficial de Bitcraze, este protocolo garantiza un estricto pedido de paquetes dentro de un puerto en específico, y permitiendo la reorganización de paquetes de diferentes puertos o bien en su defecto ser enviados fuera de servicio. Los primeros 4 *bits* del paquete corresponde a el número de puerto, existiendo un número de 16, representados por valores de 0 al 15. A su vez con ello se implementa un sistema de priorización de paquetes, asignando una mayor prioridad desde el número de puerto inferior disminuyendo prioridad conforme un número de puerto más alto.

Esto permite casos de uso como cargar una trayectoria mientras se controla el Crazyflie en tiempo real para trabajar sin problemas siempre y cuando la trayectoria se cargue Utiliza un número de puerto más alto como puntos de ajuste en tiempo real. Por otro lado los siguientes 3 *bits* corresponden al número de canal, para definir la naturaleza del paquete. Los últimos 31 *bytes* de datos correspondencon un *payload* con la información enviada hacia el Crazyflie.

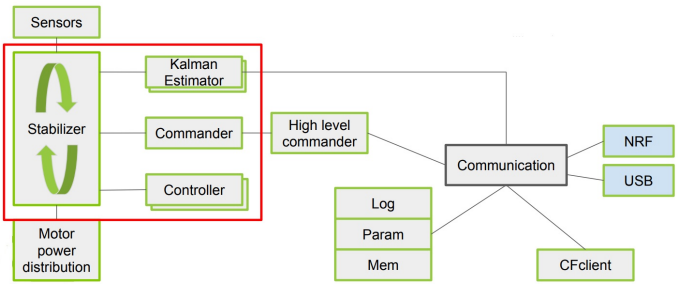


Figura 7: Modulo de *sofware* del Crazyflie[18].

6.6. Algoritmos de estimación

Los algoritmos de estimación son técnicas computacionales diseñadas para estimar o predecir el valor de una cantidad desconocida basándose en datos observados o mediciones. Estos algoritmos son fundamentales en una amplia gama de aplicaciones, desde el seguimiento de objetos en sistemas de radar hasta la predicción del clima y la navegación de vehículos autónomos [19]. A continuación, se presentan algunos de los algoritmos de estimación más comunes:

6.6.1. Estimador complementario

El estimador complementario es un enfoque de estimación de estado simple pero efectivo utilizado en el Crazyflie. Utiliza dos fuentes de información principales: la información del acelerómetro y la información del giroscopio. El acelerómetro proporciona información sobre la aceleración gravitatoria y lineal del Crazyflie, mientras que el giroscopio proporciona información sobre la velocidad angular [20].

El estimador complementario combina estas dos fuentes de información para estimar la orientación del Crazyflie en tiempo real. Es rápido y eficiente computacionalmente, lo que lo hace adecuado para sistemas de recursos limitados como el Crazyflie. Sin embargo, puede ser vulnerable a errores acumulativos en la estimación de la orientación a medida que el tiempo avanza .

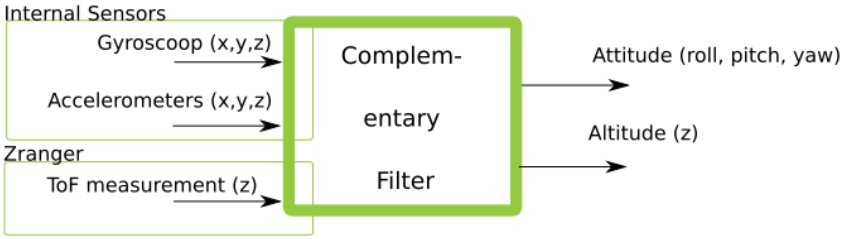


Figura 8: Diagrama de estimador complementario[20].

6.6.2. Estimador de Kalman

El Estimador de Kalman es un algoritmo de estimación de estado más avanzado que utiliza un modelo estadístico y las mediciones disponibles para estimar el estado del sistema. A diferencia del estimador complementario, el estimador de Kalman es capaz de estimar no solo la orientación sino también otros estados, como la velocidad y la posición[20].

Se basa en la implementación de un filtro de Kalman, el cual es un filtro recursivo que estima el estado actual del sistema basándose en las mediciones entrantes (en combinación con una desviación estándar prevista del ruido), el modelo de medición y el modelo del propio sistema[20].

Este es más robusto frente a ruido y errores de medición y puede manejar de manera efectiva la fusión de múltiples fuentes de información, como sensores inerciales, GPS y magnetómetros. Proporciona estimaciones más precisas y es menos propenso a errores acumulativos en comparación con el estimador complementario. Sin embargo, es más computacionalmente intensivo y puede requerir sensores adicionales, como un GPS, para obtener la máxima precisión en la estimación del estado.

6.6.3. Filtro de Kalman extendido

El filtro de Kalman extendido (EKF, por sus siglas en inglés, *Extended Kalman Filter*) es una extensión del filtro de Kalman clásico diseñada para estimar el estado de sistemas dinámicos no lineales. El EKF es una de las herramientas más utilizadas en la estimación y el seguimiento de sistemas no lineales en una variedad de campos, como la robótica, la navegación, la ingeniería de control y la procesamiento de señales[20].

A diferencia del filtro de Kalman estándar, que asume que las relaciones entre el estado y

las mediciones son lineales y que las distribuciones de probabilidad son gaussianas, el EKF es capaz de manejar sistemas no lineales y mediciones ruidosas mediante una aproximación local lineal. Esto se logra mediante una expansión en series de Taylor de las ecuaciones no lineales que describen la dinámica del sistema y las relaciones entre el estado y las mediciones[20].

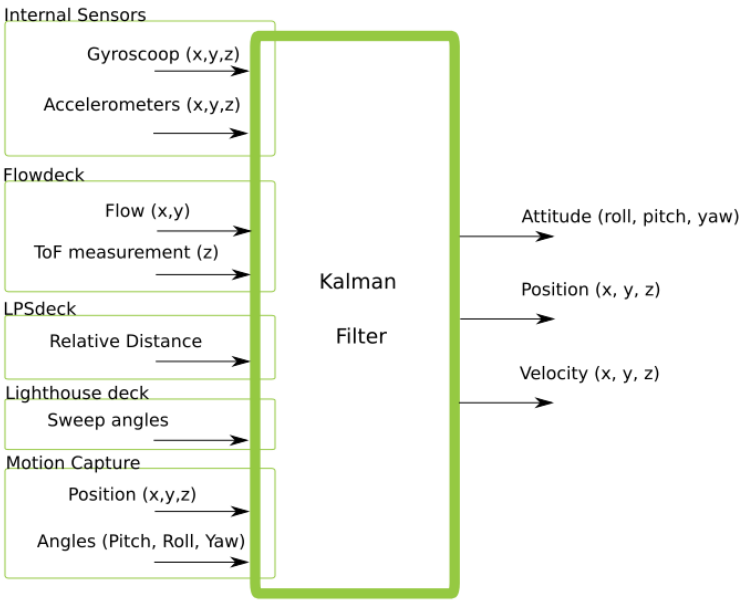


Figura 9: Diagrama de estimador de Kalman[20].

6.7. Controlador PID Crazyflie 2.1

El sistema de control del Crazyflie se encarga de mantener la posición, velocidad y actitud del dron en función de un *setpoint* deseado. Este sistema es fundamental para la estabilización y el control preciso del dron durante el vuelo. El sistema de control se divide en cuatro niveles, cada uno de los cuales se encarga de aspectos específicos del control del dron [21]:

Control de velocidad angular (*Attitude Rate*): Este nivel se encarga de controlar las tasas de cambio de la actitud del dron, como la velocidad de giro en los ejes de *pitch, roll y yaw*.

Control de Actitud Absoluta (*Attitude Absolute*): En este nivel, se controla la actitud absoluta del dron, es decir, su orientación en el espacio 3D.

Control de Velocidad (*Velocity*): Aquí se regula la velocidad del dron en términos de desplazamiento en el espacio.

Control de Posición (*Position*): Este nivel se encarga de mantener la posición espacial absoluta del dron en relación con un punto de referencia.

El sistema de control del Crazyflie utiliza controladores PID en cascada para mantener la estabilidad y controlar la posición y actitud del dron. Cada nivel de control se encarga de aspectos específicos y contribuye a lograr un vuelo estable y preciso. El controlador PID es una parte fundamental de este sistema, ya que proporciona las correcciones necesarias para seguir un objetivo deseado y mantener el dron en condiciones de vuelo seguras y controladas [21].

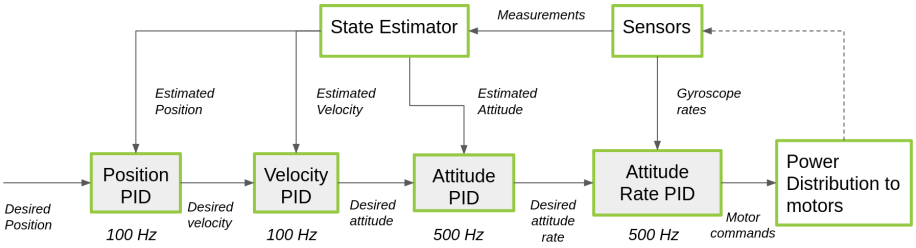


Figura 10: Sistema de controlador PID en cascada de Crazyflie[21].

6.8. Controlador Mellinger

El controlador Mellinger es un tipo de controlador utilizado en aplicaciones de control de vehículos aéreos no tripulados (UAVs o drones) y está diseñado para lograr un control de posición y actitud preciso y ágil. Este controlador se utiliza en sistemas de vuelo que requieren una alta agilidad y capacidad de maniobra, como drones utilizados en acrobacias o aplicaciones de investigación.

El controlador Mellinger se basa en algoritmos de control no lineales y utiliza técnicas de control de retroalimentación para lograr un seguimiento de trayectoria y estabilización precisa. A diferencia del controlador PID, el controlador Mellinger no se limita a sistemas lineales y puede manejar sistemas no lineales y complejas dinámicas de vuelo [22].

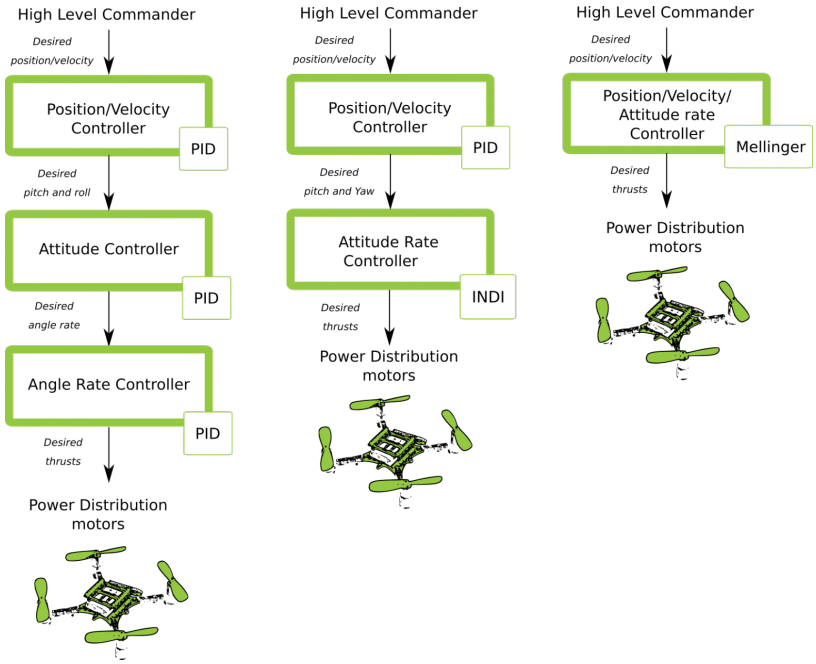


Figura 11: Sistema de controladores en Crazyflie[21].

6.9. ROS 2

El *Robot Operating System* (ROS) es un conjunto de bibliotecas de software y herramientas para crear aplicaciones de robots. ROS 2 se basa en el Servicio de Distribución de Datos (DDS), un estándar abierto para comunicaciones que se utiliza en infraestructura crítica como militar, naves espaciales y sistemas financieros. DDS habilita ROS 2 para obtener la mejor seguridad de su clase, soporte integrado y en tiempo real, comunicación multi-robot y operaciones en entornos de red no ideales [23].

ROS 2 proporcionan acceso a patrones de comunicación. Estos son, en particular, tópicos, servicios y acciones, que se organizan bajo el concepto de nodo. ROS 2 también proporciona una API para parámetros, temporizadores, inicio y otras herramientas auxiliares que se pueden utilizar para diseñar un sistema robótico [23].

- Tópicos: Son un elemento que actúa como un bus para que los nodos intercambien mensajes. Un nodo puede publicar datos en cualquier número de tópicos y simultáneamente tener suscripciones a cualquier número de tópicos. Son una de las principales formas en que los datos se mueven entre nodos y, por lo tanto, entre diferentes partes del sistema.

- Servicios: Son otro método de comunicación que se basan en un modelo de llamada y respuesta, frente al modelo de los tópicos de publicador-suscriptor. Mientras que los tópicos permiten a los nodos suscribirse a flujos de datos y obtener actualizaciones continuas, los servicios solo proporcionan datos cuando un cliente los llama específicamente.

- Acciones: Son un tipo de comunicación de ROS 2 que están destinadas a tareas de larga ejecución. Su funcionalidad, es similar a los servicios excepto que las acciones se pueden cancelar. También proporcionan retroalimentación constante, a diferencia de los servicios que devuelven una sola respuesta.

  Las acciones usan un modelo cliente-servidor, similar al modelo publicador-suscriptor de los tópicos. Un nodo cliente, envía un objetivo a un nodo servidor que reconoce el objetivo y devuelve un flujo de comentarios y un resultado.

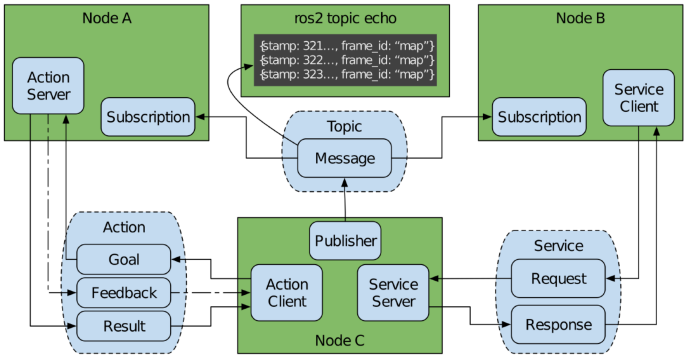


Figura 12: Arquitectura de nodos ROS2[23].

CAPÍTULO 7

Configuración y funcionamiento del crazyflie 2.1

En este capítulo, se presenta una descripción detallada de la configuración de los parámetros de control, estimación y funcionamiento del Crazyflie 2.1. Para llevar a cabo esta tarea, se utilizó el *software* de código abierto Crazyclient, el cual está integrado en la máquina virtual de Bitcraze. La instalación de esta máquina virtual se llevó a cabo siguiendo las instrucciones proporcionadas en el manual de instalación desarrollado por Francis Sanabria en su trabajo de graduación relacionado con los Crazyflie. Este proceso simplificó la capacidad de acceder y ajustar diversas características, como el tipo de controlador, las constantes de control utilizadas y el estimador, entre otros aspectos. Además, se incluye una descripción detallada de los resultados obtenidos durante la experimentación al modificar cada una de estas características.

7.1. Ensamblaje y compensación de hélices

Como primer paso, se procedió al ensamblaje de los Crazyflie 2.1 siguiendo el tutorial de armado y prueba que se encuentra en la página oficial de Bitcraze [24]. Se prestó especial atención a la instalación de las hélices del dron, asegurándose de revisar la compensación de estas para garantizar su equilibrio adecuado. Esta medida fue crucial para prevenir vibraciones excesivas que pudieran impactar negativamente en el rendimiento y estabilidad del dron durante el vuelo.

Para llevar a cabo este proceso, se utilizó como materiales un alfiler, las hélices correspondientes del Crazyflie y cinta adhesiva como se muestra en la figura 13. Se examinó minuciosamente las hélices para identificar posibles deformidades, imperfecciones o acumulaciones de suciedad en su superficie, y se procedió a limpiar o reemplazar las hélices según fuera el caso.

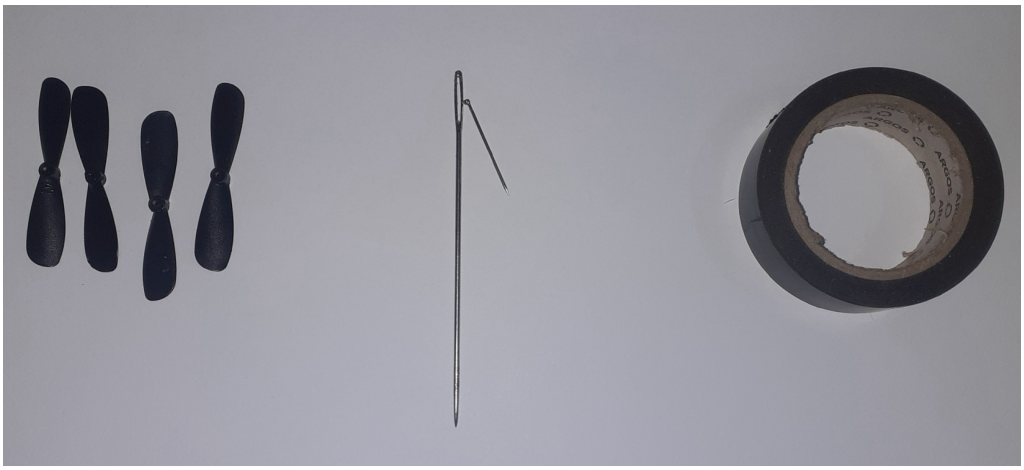


Figura 13: Materiales para proceso de compensación de hélices.

Posteriormente, se utilizó el alfiler para atravesar el centro de cada hélice, creando así un punto de equilibrio temporal. Con esto se verificó que la hélice se mantuviera en posición horizontal lo cual es indicador de un equilibrio adecuado. En su defecto, que la hélice se inclinara hacia un lado, era evidencia de un desequilibrio. Se giró suavemente la hélice alrededor del alfiler y se observó en qué posición se mantenía horizontal de manera más constante, identificando así el lado más pesado de la hélice.

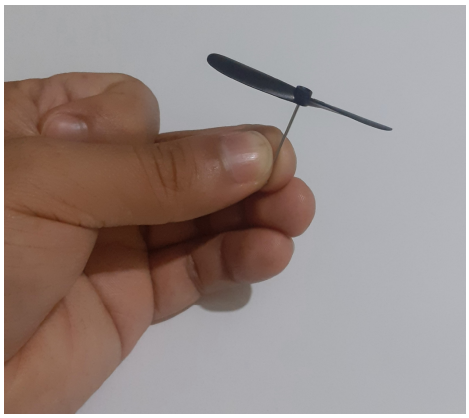


Figura 14: Prueba de equilibrio de la hélice.

Para corregir este desequilibrio, se aplicó tiras muy pequeñas de cinta adhesiva en el lado opuesto al más pesado de la hélice. Esto se hizo con el propósito de añadir un contrapeso a la hélice y lograr su equilibrio. Se verificó nuevamente el equilibrio deslizando la hélice a lo largo del alfiler, ajustando la cantidad de cinta adhesiva hasta que la hélice se mantuvo nivelada durante el giro sobre el alfiler.

Figura 15: Corrección de desequilibrio con contrapeso.

Finalmente, una vez que todas las hélices estuvieron equilibradas, se volvieron a montar en el dron y se llevó a cabo una prueba de vuelo suave utilizando el *software* Crazyclient en su versión móvil desarrollada para Android. Esto permitió asegurar que las vibraciones se redujeran al mínimo durante el vuelo del Crazyflie 2.1.

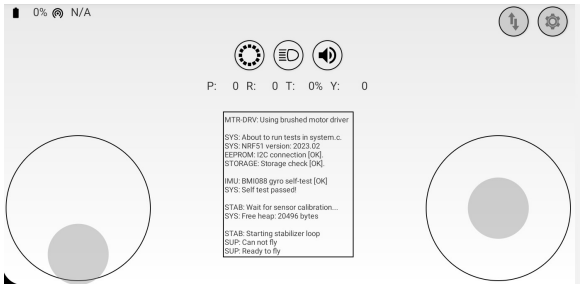


Figura 16: Crazyclient *mobile* para Android.

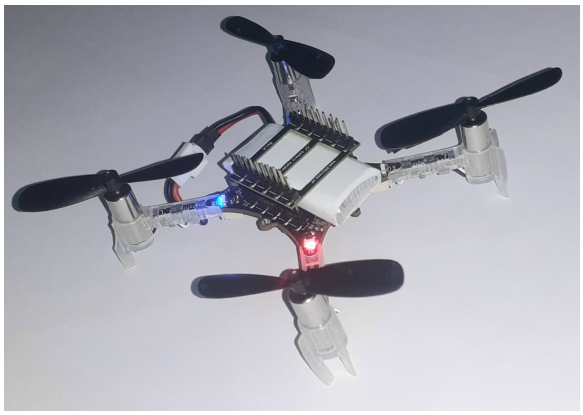


Figura 17: Crazyflie 2.1 ensamblado.

Figura 18: Prueba de vuelo preliminar con Crazyclient.

7.2. Crazyclient y configuración de parámetros

Se empleó el software Crazyclient (ver Figura 19) para acceder a los parámetros de configuración de los Crazyflie 2.1. Se configuró el parámetro *commander* con un valor de 1, lo que permitió el control y envío de comandos de forma remota al dron. Estos comandos se utilizan para recibir y procesar instrucciones de vuelo, lo que incluyó el control del despegue, aterrizaje, dirección, velocidad y otras funciones esenciales del Crazyflie. Sin este parámetro configurado, el dron no responderá a comandos provenientes de un controlador externo, como un control remoto, una aplicación móvil o una computadora.

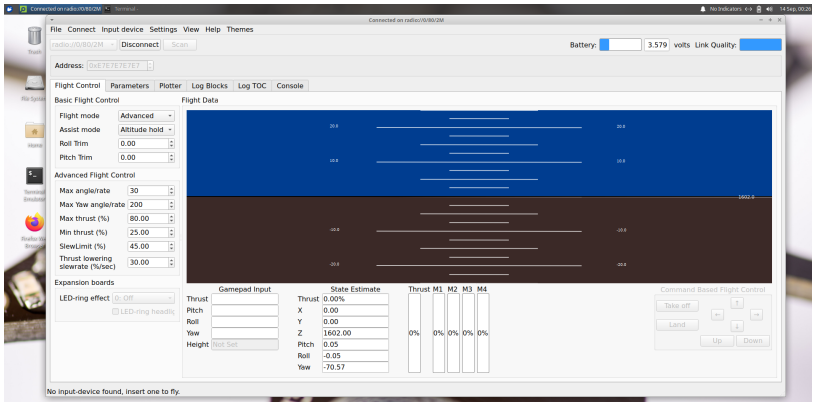


Figura 19: Conexión del Crazyflie 2.1 al Crazyclient.

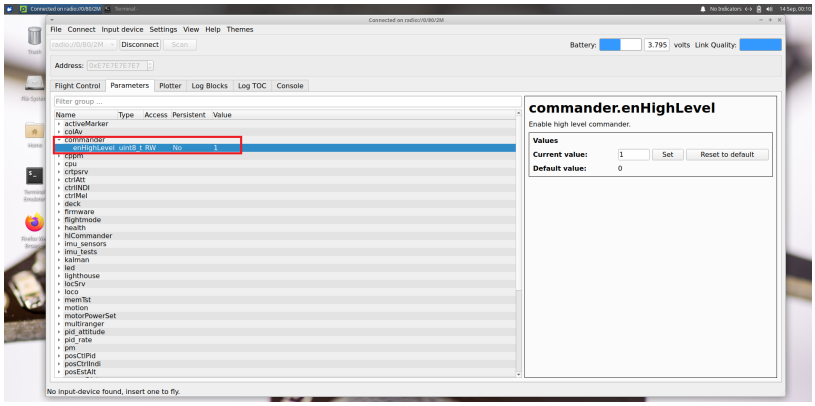


Figura 20: Configuración del parámetro *commander* en Crazyclient .

Posteriormente, se configuró el parámetro *stabilizer*, que corresponde a la configuración de vuelo que afecta cómo el dron responde a los comandos de control y cómo se estabiliza automáticamente en el aire. Esta característica dentro de Crazyclient permitió seleccionar y configurar tanto el controlador como el estimador que se utilizaron durante el vuelo del Crazyflie.

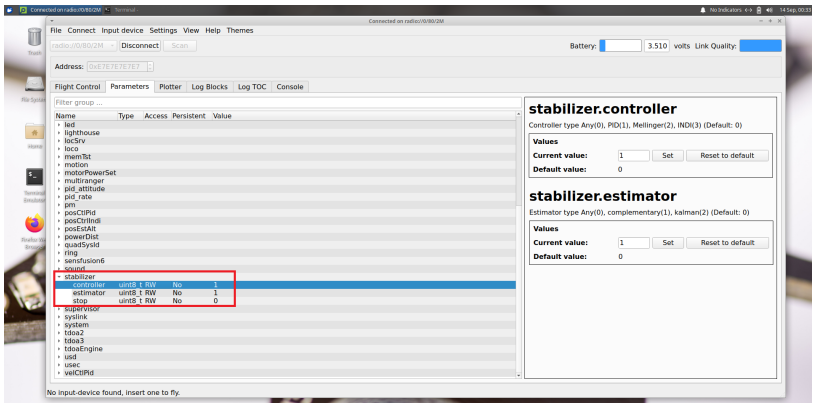


Figura 21: Configuración del parámetro *stabilizer* en Crazyclient.

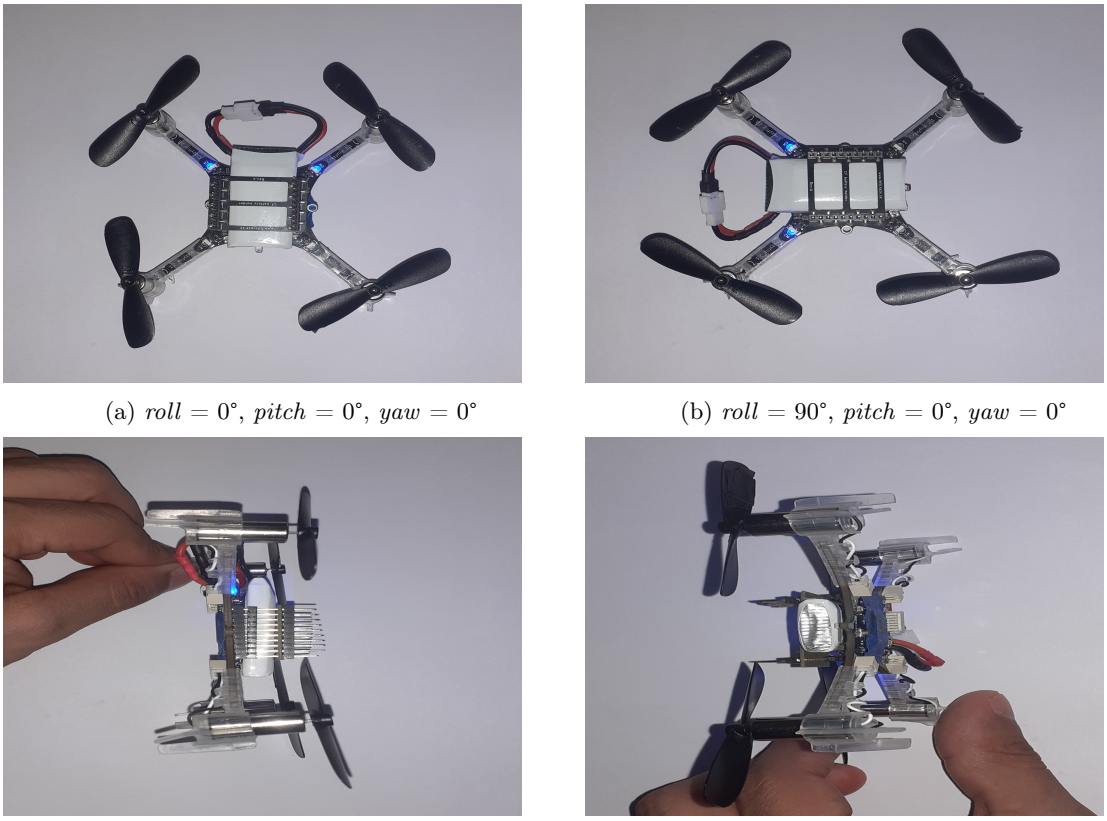
Es importante resaltar que diferentes combinaciones de controladores y estimadores proporcionaron diferentes niveles de estabilidad, capacidad de maniobra y respuesta a los comandos de vuelo.

7.2.1. Estimador

Se configuró el valor del estimador, considerando que el Crazyflie tiene disponibles dos opciones: el estimador complementario y el estimador de Kalman. Estos se utilizaron para estimar la orientación del dron y lograr una navegación precisa.

Estimador complementario

Al configurar el estimador en modo complementario, utilizando por defecto un controlador PID según se muestra en la Figura 21, se llevaron a cabo experimentos para verificar la precisión de la estimación. Estos experimentos involucraron la colocación del dron en posiciones de referencia conocidas. En particular, se posicionó el dron con valores de orientación: $roll = 0°$, $pitch = 0°$ y $yaw = 0°$, además de establecer cada uno de estos valores a 90°, mientras los otros dos se mantenían en 0°, tal y como se muestran en las configuraciones de la Figura 22. Esta disposición siguió la convención de orientación del dron y su representación de signos, como se ilustra en la Figura 3 en el capítulo del marco teórico.



(a) $roll = 0°$, $pitch = 0°$, $yaw = 0°$

(b) $roll = 90°$, $pitch = 0°$, $yaw = 0°$

(c) $roll = 0°$, $pitch = 90°$, $yaw = 0°$

(d) $roll = 0°$, $pitch = 0°$, $yaw = 90°$

Figura 22: Posiciones de orientación conocidas del dron

Los resultados de estos experimentos se encuentran detallados en el Cuadro 2. En este, se observa que el Error Cuadrático Medio (ECM), fue notablemente elevado. Además, al analizar las gráficas que representan los valores en las diferentes posiciones conocidas, se advierte que el estimador complementario proporciona mediciones continuas. Esto implica mantener una medición del estado actual del dron sin interrupciones que se actualiza de acuerdo a la orientación.

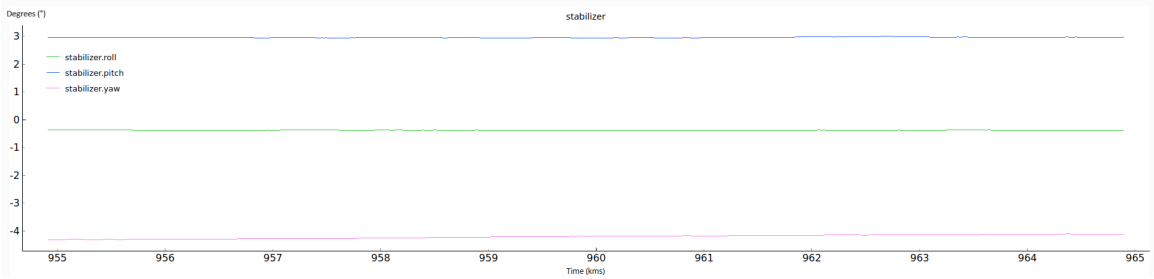


Figura 23: Gráfica orientación $roll = 0°$, $pitch = 0°$, $yaw = 0°$ con estimador complementario

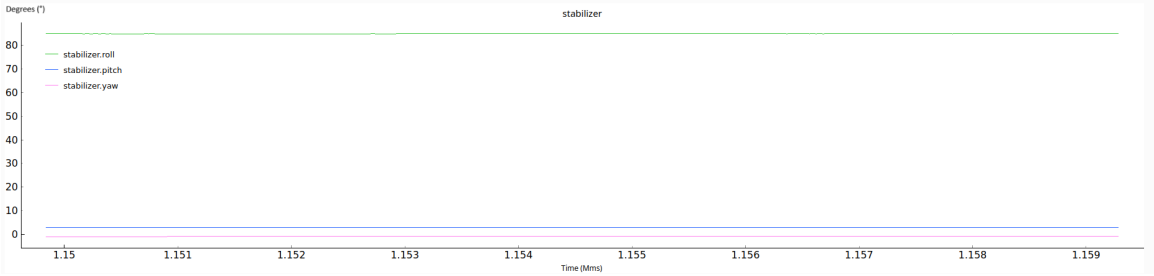


Figura 24: Gráfica orientación $roll = 90°$, $pitch = 0°$, $yaw = 0°$ con estimador complementario

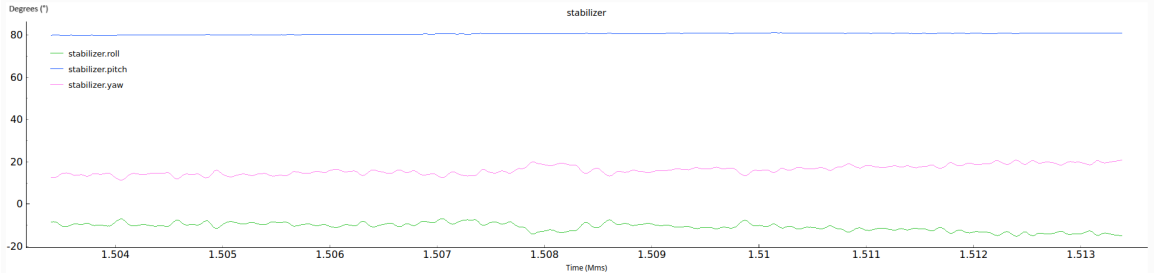


Figura 25: Gráfica orientación $roll = 0°$, $pitch = 90°$, $yaw = 0°$ con estimador complementario

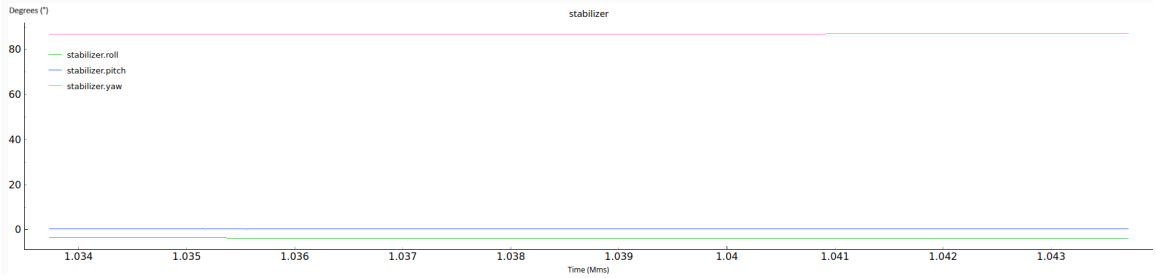


Figura 26: Gráfica orientación $roll = 0°$, $pitch = 0°$, $yaw = 90°$ con estimador complementario

Orientación real (°)	Orientación estimada (°)	ECM (deg^2)
$roll$ = 0.0, $pitch$ = 0.0, yaw = 0.0	$roll$ = -0.2, $pitch$ = 3.0, yaw = -4.2	8.89
$roll$ = 90.0, $pitch$ = 0.0, yaw = 0.0	$roll$ = 85.0, $pitch$ = 2.5, yaw = -0.8	10.63
$roll$ = 0.0, $pitch$ = 90.0, yaw = 0.0	$roll$ = -10.0, $pitch$ = 80.0, yaw = 15.0	141.67
$roll$ = 0.0, $pitch$ = 0.0, yaw = 90.0	$roll$ = -1.9, $pitch$ = 0.9, yaw = 87.0	4.47

Cuadro 2: Error cuadrático medio para la estimación con complementario para la orientación

Estimador de Kalman

Por otro lado, se configuró el dron utilizando el estimador de Kalman, de igual manera utilizando por defecto un controlador PID. Se repitió el mismo experimento que con el estimador complementario, colocando el dron en las cuatro posiciones conocidas.

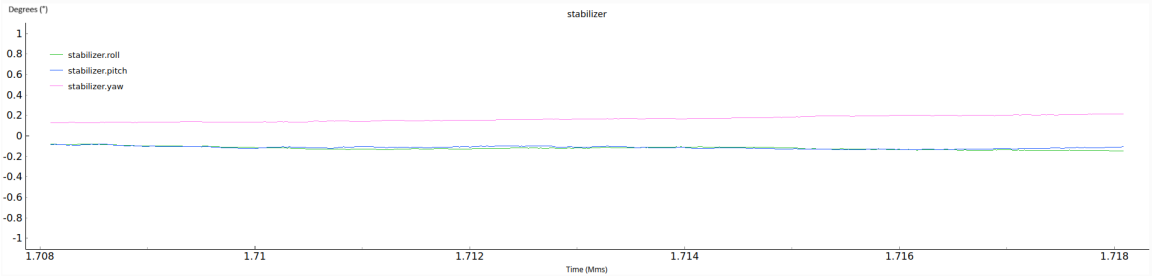


Figura 27: Gráfica orientación $roll$ = 0°, $pitch$ = 0°, yaw = 0° con Kalman

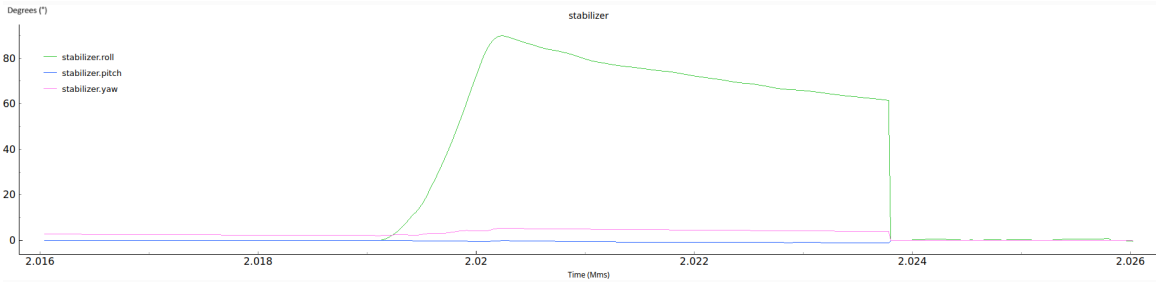


Figura 28: Gráfica orientación $roll$ = 90°, $pitch$ = 0°, yaw = 0° con Kalman

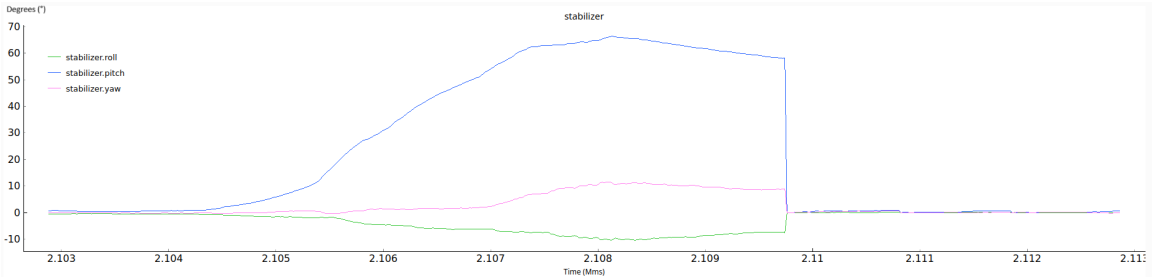


Figura 29: Gráfica orientación $roll$ = 0°, $pitch$ = 90°, yaw = 0° con Kalman

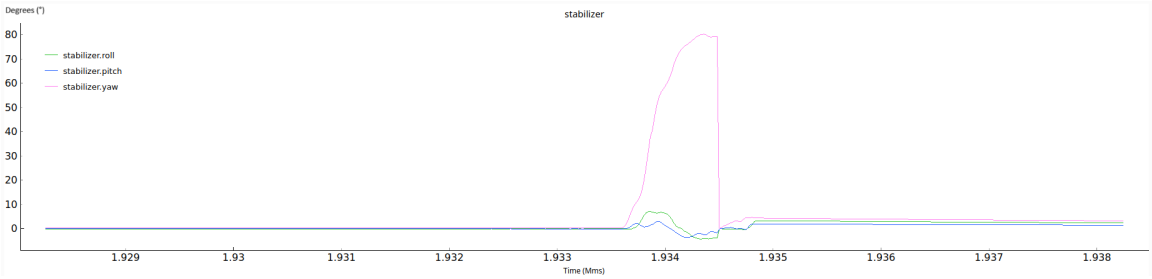


Figura 30: Gráfica orientación $roll = 0°$, $pitch = 0°$, $yaw = 90°$ con Kalman

Como se puede observar en los resultados tabulados en el Cuadro 3, los errores cuadráticos medios se redujeron considerablemente en los primeros dos casos, indicando que la precisión de la estimación al utilizar Kalman, que utiliza un modelo de filtro de Kalman, mejora significativamente respecto al complementario.

Orientación real (°)	**Orientación estimada (°)**	**ECM (deg^2)**
$roll = 0.0$, $pitch = 0.0$, $yaw = 0.0$	$roll$ = -0.6, $pitch$ = -0.6, $yaw = 1.4$	0.89
$roll = 90.0$, $pitch = 0.0$, $yaw = 0.0$	$roll = 85.6$, $pitch = 2.1$, yaw = -0.3	7.9
$roll = 0.0$, $pitch = 90.0$, $yaw = 0.0$	$roll$ = -9.5, $pitch = 70.0$, yaw= 9.5	193.5
$roll = 0.0$, $pitch = 0.0$, $yaw = 90.0$	$roll = 2.2$, $pitch = 1.1$, $yaw = 82.0$	23.35

Cuadro 3: Error cuadrático medio para la estimación con complementario para la orientación

Además, como se observa en las figuras, se puede notar que la estimación realizada por el filtro de Kalman no es continua, es decir, únicamente se muestra el cambio en los valores de orientación *roll*, *pitch* y *yaw*. Por esto mismo se puede notar que al realizar la estimación con Kalman, se ingresa ligeramente ruido a la medición de de cada uno de los ángulos de orientación, respecto a las estimaciones realizadas con el complementario.

La diferencia en el comportamiento de la orientación entre el estimador complementario y el estimador Kalman en el Crazyclient se debe a las diferencias fundamentales en la forma en que estos estimadores calculan y proporcionan datos de orientación.

Por un lado el estimador complementario combina datos de sensores inerciales (como el giroscopio) y datos de sensores de referencia (como el magnetómetro) para calcular la orientación del dron [20]. Debido a la naturaleza de su cálculo, el estimador complementario puede proporcionar una estimación de orientación continua y persistente incluso cuando el dron no está en movimiento. Esto significa que cuando graficas el roll, pitch y yaw en el Crazyclient, los valores se mantienen constantes incluso si el dron no se está moviendo.

Por otro lado, el estimador Kalman utiliza un modelo de filtro de Kalman para estimar la orientación del dron en función de múltiples fuentes de datos, como giroscopios, acelerómetros, magnetómetros, GPS [20]. El estimador Kalman está diseñado para proporcionar estimaciones más precisas de la orientación, especialmente cuando el dron está en movimiento. Como resultado, se centra en la detección de cambios en la orientación en lugar de proporcionar una orientación constante [5]. Por lo cual en las gráficas se observa que el *roll*, *pitch* y *yaw* en el Crazyclient con el estimador Kalman, los valores se actualizan

constantemente para reflejar cualquier cambio en la orientación del dron.

En resumen, la diferencia en el comportamiento de la visualización de la orientación entre el estimador complementario y el estimador Kalman se debe a sus métodos de cálculo y sus objetivos. El estimador complementario tiende a proporcionar una orientación persistente, mientras que Kalman se centra en la detección precisa de cambios en la orientación, lo que puede resultar en valores que se actualizan continuamente en la gráfica. Por lo cual elección hasta este punto, la elección del uso de uno u otro depende de las necesidades específicas y de qué aspecto de la orientación se desea visualizar en tiempo real.

7.2.2. Controlador

Se configuró el valor del controlador teniendo como opciones para Crazyflie que utiliza tanto el controlador PID y *Mellinger* para realizar el control de vuelo del dron.

Controlador PID

Al configurar el como controlador a utilizar en el Crazyflie como PID, se realizó el experimento de visualizar de que manera el controlador y el adecuado ajuste de las constantes del mismo, ayudó a mejorar la precisión de la estimación de posición y orientación que a su vez permite un mejor control del dron.

Se realizó las pruebas utilizando el estimador de Kalman, el cual además de proporcionar mediciones más exactas de acuerdo a lo descrito en la anterior sección, también permitió verificar la funcionalidad de los controladores. Este proporciona una medición de la posición y orientación del dron, de acuerdo con un cambio en esta, lo cual simula el comportamiento más realista del dron al momento de realizar una prueba de vuelo, y de la misma manera permite experimentar la robustes del controlador ante perturbaciones externas del dron durante el movimiento.

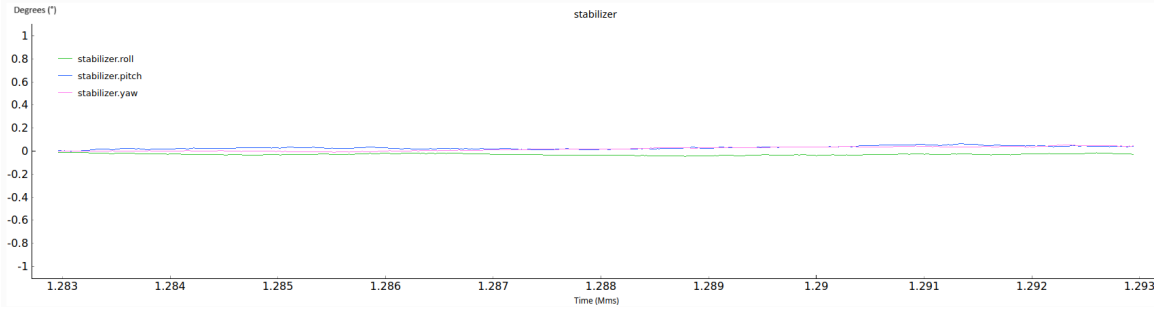


Figura 31: Gráfica de estimación con Kalman y controlador PID para orientación

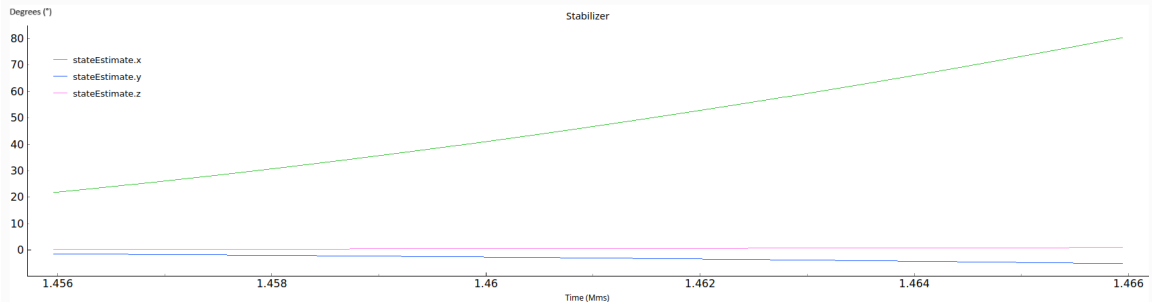


Figura 32: Gráfica de estimación con Kalman y controlador proporcional para posición

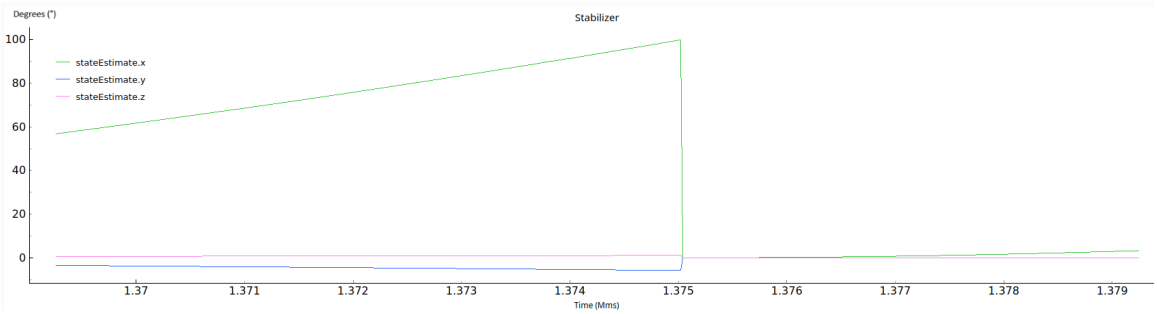


Figura 33: Gráfica de estimación con Kalman y controlador integral para posición

Como se puede observar en las Figuras 31-33, se realizaron pruebas con distintos valores de constantes del controlador. Se determinó que la mejor combinación fue con Kp = 2 para el control de posición en los ejes x, y y z, Kd = 0, para los tres y Ki = 0, para x e y, con Ki= 0.5 para z. Al utilizar el controlador PID para estimar los valores de posición del dron, no se obtuvo mediciones precisas, ya que como se observa en la imágen tal, al realizar la prueba en una posición conocida, con x, y y z en cero, la medición obtenida del dron marcaba un desfase en z de 89 metros. Inicialmente al utilizar el Ki = 0 para el eje z, se tuvo que el valor de desfase que se registraba en el eje z, no convergía a un punto específico, llegando a 80 m en algunos casos, y valores mayores en otros. Pero al agregar un poco de integrador al controlador en z se está utilizando un valor no nulo que permite si el dron tiene una tendencia constante a desviarse de la altitud deseada como lo es es este caso, el término integral con un valor pequeño corregió esta desviación o rebote de altitud, ya que tenerlo activado ayuda a la acumulación de errores en la altitud reducir la desviación en la medición del eje z.

Por otro lado, al verificar las orientaciones *roll*, *pitch*, *yaw*, mientras estas se mantienen en la posición conocida de 0° en las tres, se pudo observar que la exactitud de la estimación no se ver mayormente afectada, ya que mantiene una medición cercana a la esperada. Sin embargo, reducir o aumentar los valores de las constantes en cada uno de los valores de orientación, desvió ligeramente del valor esperado para *roll*, *pitch*, *yaw*, teniendo la medición más precisa con los valores de Kp = 6 en las tres orientaciones, Ki = 3, en *roll* y *pitch*, Ki = 1 en *yaw*, Kd = 0 en *roll* y *pitch*, y Kd = 0.35 en *yaw*.

Además, al probar el dron en roll = 0° pitch = 0° y yaw = 0° con x e y en 0 m y z = 0.3 m, se pudo observar que la estimación de la posición x, y, z, fue imprecisa como se observa

en la Figura 34.

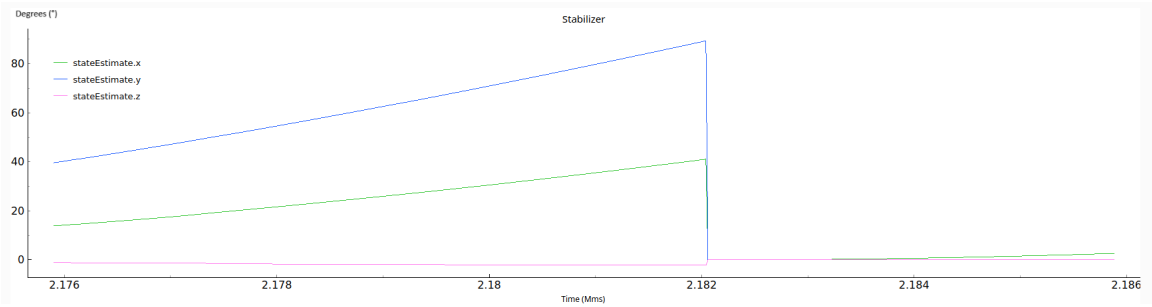


Figura 34: Gráfica de estimación con Kalman y controlador PID a z = 0.3 m

Controlador Mellinger

Se realizaron las mismas pruebas para validar el funcionamiento de la estimación con el controlador mellinger, utilizando las variables de control por defecto del controlador dentro del crazyflie. Se pudo observar como se muestra en la figura tal, que al estimar la orientación en las 4 posiciónes conocidas tomadas para el experimento con PID, con tal porcentaje de error, se determinó que la combinación de estimador de Kalman y Mellinger, proporcionó una mejor estimación para la orientación del dron. Sin embargo, de la misma manera que con el PID, al realizar las mediciones para la posición x, y, z, esta divergieron del valor esperado.

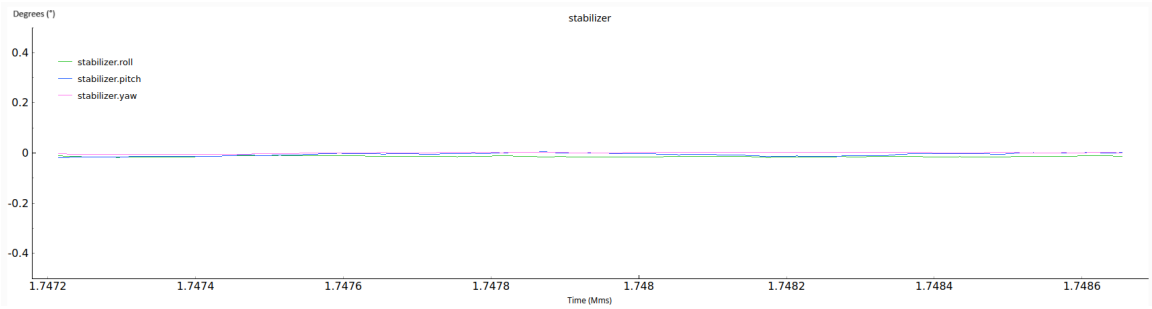


Figura 35: Gráfica estimación *roll* = 0°, *pitch* = 0°, *yaw* = 0° con Kalman y Mellinger

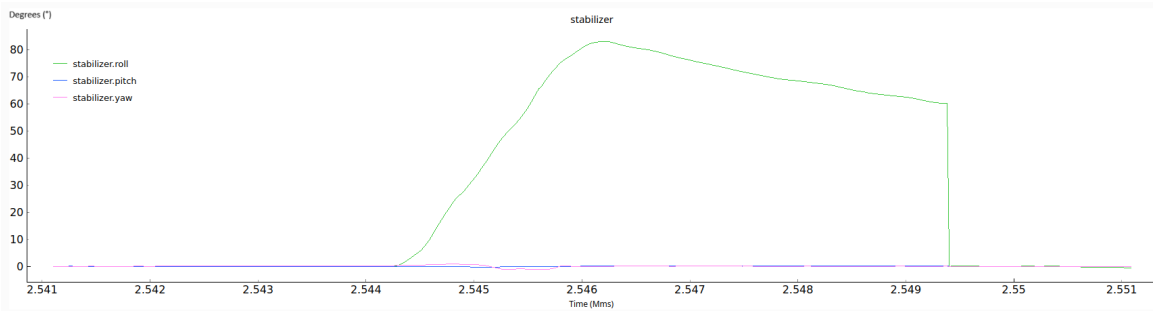


Figura 36: Gráfica estimación *roll* = 90°, *pitch* = 0°, *yaw* = 0° con Kalman y Mellinger

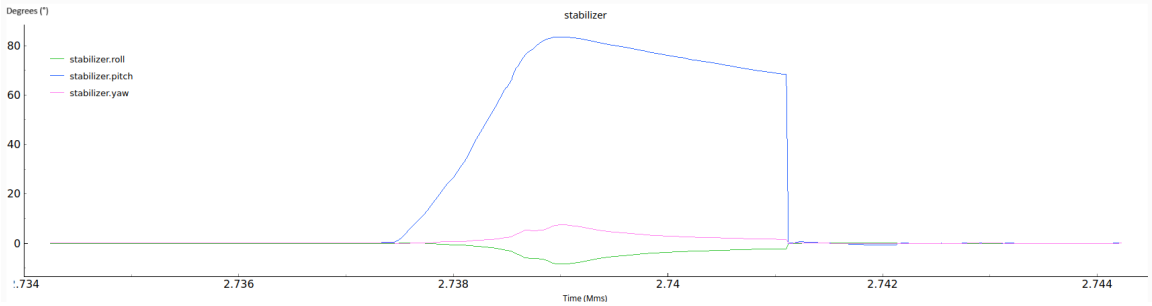


Figura 37: Gráfica estimación $roll = 0°$, $pitch = 90°$, $yaw = 0°$ con Kalman y Mellinger

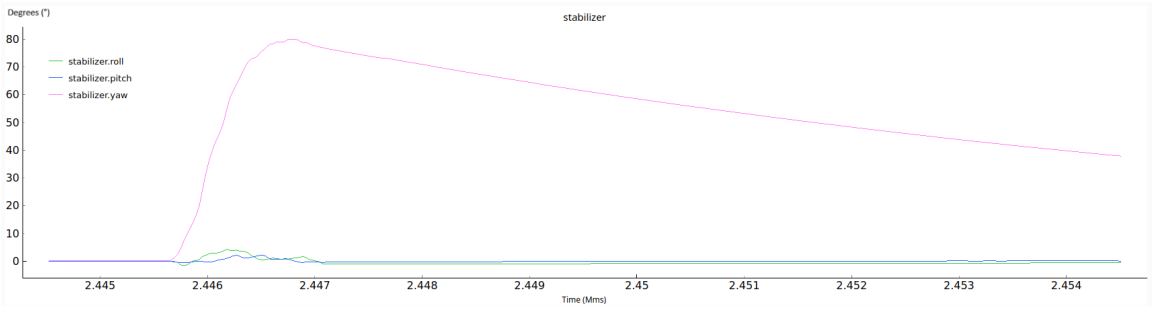


Figura 38: Gráfica estimación $roll = 0°$, $pitch = 0°$, $yaw = 90°$ con Kalman y Mellinger

Como se observa en las Figuras 35-38, la medición proporcionada por el giroscopio del dron, no fue suficiente para poder estimar la orientación. Sin embargo, para estimar la posición utilizando el acelerómetro que proporciona información sobre la aceleración gravitatoria y lineal del Crazyflie no fue necesaria. Por lo cual requiere implementar mediciones externas para implementar un filtro de Kalman extendido para la estimación que ayude a proporcionar una mejor información que estabilice de mejor manera el vuelo.

Por otro lado, al realizar pruebas de vuelo preliminares del crazyflie con el software crazyclient para móvil como se puede observar en las imágenes, al utilizar el estimador complementario, tanto con melinger como con pid, el dron cae al intentar estabilizarlo en el aire, mientras que al cambiar al estimador de Kalman al utilizar tanto PID, como Mellinger, se obtuvo resultados satisfactorios, ya que se logró mantener mejor estabilidad en el aire, aunque de igualmanera después de cierto tiempo se inestabilizaba, tal y como concuerda con el comportamiento mostrado por las gráficas que cada cierto tiempo la medición en el eje z y x, mostraba una perturbación extraña.

Los resultados de acuerdo con las gráficas obtenidas, concuerdan que si bien funciona adecuadamente el PID, y Mellinger con las variables tunneadas, se observó que el dron presentó menor *drift* durante el vuelo con el controlador Mellinger utilizado en las pruebas. Esto concuerda con el comportamiento más estable en las estimaciones obtenido al observar los valores graficados en las posiciones determinadas.

7.2.3. *Firmware*

El *firmware* de un dron Crazyflie desempeña un papel fundamental en su operación y rendimiento. Este es el núcleo de operación del dron, ya que controla su comportamiento en vuelo, su estabilidad y sus funciones avanzadas. El *firmware* está diseñado para funcionar en la plataforma *hardware* específica del Crazyflie, lo que hace que este sea altamente optimizado para garantizar un rendimiento eficiente y preciso de todos sus recursos.

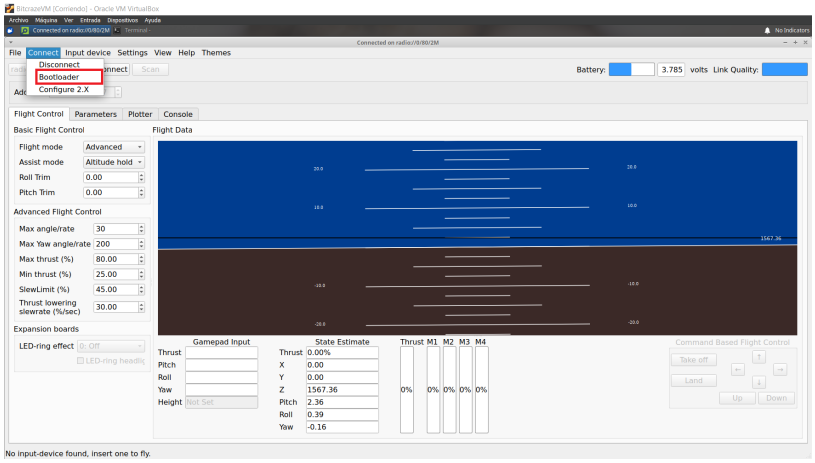


Figura 39: Selección de *bootloader* para actualizar *firmware* con Crazyclient

Durante las pruebas de vuelo preliminares realizadas, en donde se verificó la estabilidad del dron, se observó que habían momentos de desconexión del dron con el cliente en los cuales el dron se tuvo que reiniciar para poder conectar correctamente y comunicarse con el Crazyflie. Por lo mismo se realizó una actualización del dron para mejorar tanto los problemas de comunicación que se tenían como también de esa manera mejorar la estabilidad del dron. Se determinó la versión 2023.06, como la mejor elección de firmware con base a mejoras en el rendimiento, correcciones de errores, nuevas características e incluso una mayor compatibilidad con componentes o sensores actualizados.

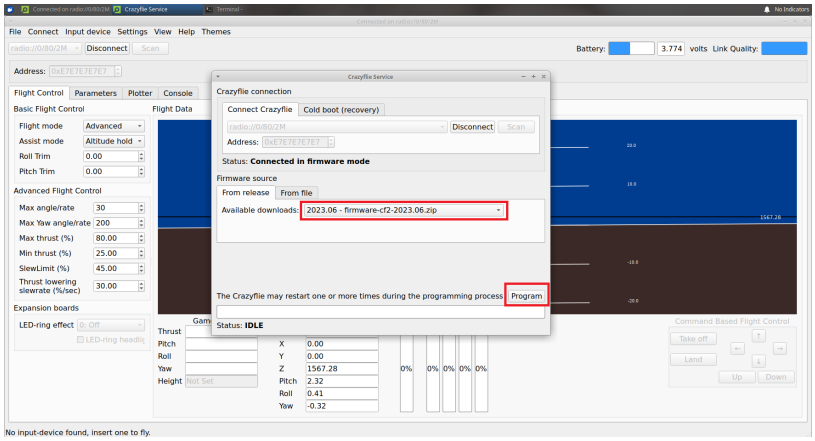


Figura 40: Selección de *firmware* 2023.06 para el Crazyflie 2.1

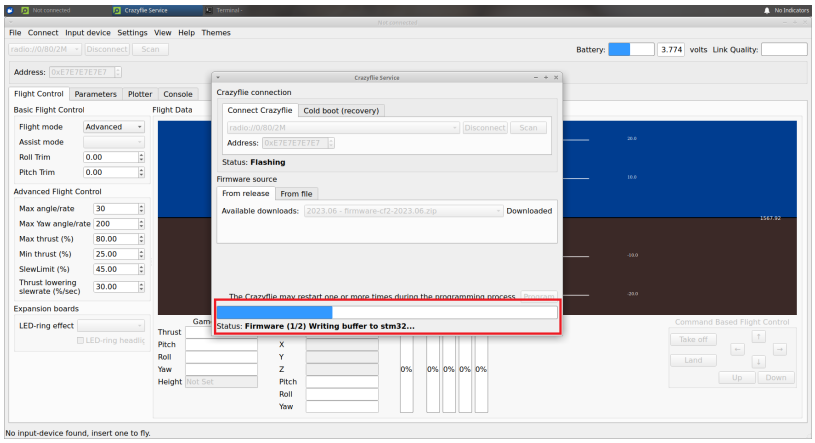


Figura 41: Actualización de *firmware*

Figura 42: Conexión de Crazyflie 2.1 mediante cable para actualización de *firmware*

Luego de realizar dichas configuraciones y actualizaciones de para el dron, se vio resuelto el problema de comunicación que se generaba tanto probar la conexión del Crazyflie mientras volaba y cuando se configuraba desde el Crazyclient. Además, se observó que el *drift* y la estabilidad del vuelo del dron mejoró considerablemente.

Figura 43: Prueba de *takeover* del Crazyflie con el Crazyclient

CAPÍTULO 8

Integración del Mocap OptiTrack al Sistema Operativo para Robots

Como parte del proceso de integración del *Mocap* OptiTrack al entorno de desarrollo para la infraestructura de control multidrones, se creó un servidor en Ubuntu. En este servidor, se procedió a la instalación del Sistema Operativo para Robots en su segunda versión (ROS 2) para permitir la interoperabilidad con OptiTrack. Este capítulo detalla el procedimiento de instalación, así como los pasos y configuraciones llevados a cabo para establecer la conexión que posibilita la recepción del *streaming* de datos de las posiciones de los marcadores a través del *software* Motive, conectado a las cámaras del sistema de captura de movimiento OptiTrack.

8.1. Instalación y configuración de Ubuntu LTS 22.04 y ROS2 Humble

Se realizó una instalación limpia del sistema operativo basado en Linux Ubuntu LTS 22.04. La elección de Ubuntu como sistema operativo se debió a su amplia compatibilidad y soporte para ROS 2, un marco de trabajo ampliamente utilizado en la investigación y desarrollo de robots, incluidos los drones. La instalación del sistema operativo se llevó a cabo en una unidad USB de 64 GB para favorecer la portabilidad del servidor en la infraestructura de pruebas con múltiples drones.

Se descargó el archivo ISO de Ubuntu LTS 22.04 desde la página oficial, seleccionando la arquitectura de 64 bits. Luego, se creó un medio de arranque *booteable* utilizando la herramienta Rufus en Windows. Esto permitió crear una unidad que se pudiera ejecutar desde la computadora para formatear la memoria USB y seleccionarla como disco de instalación, garantizando así la permanencia de los datos.

Se estableció un nombre de usuario y una contraseña durante el proceso de instalación, y

posteriormente se realizó una actualización del software. Para mantener el sistema Ubuntu y los paquetes de software actualizados a las versiones más recientes disponibles, se utilizó el administrador de paquetes de linux comandos en la terminal de usuario con las respectivas credenciales.

```
>> sudo apt update
>> sudo apt upgrade
```

Este paso fue fundamental como preparación para la instalación de ROS 2.

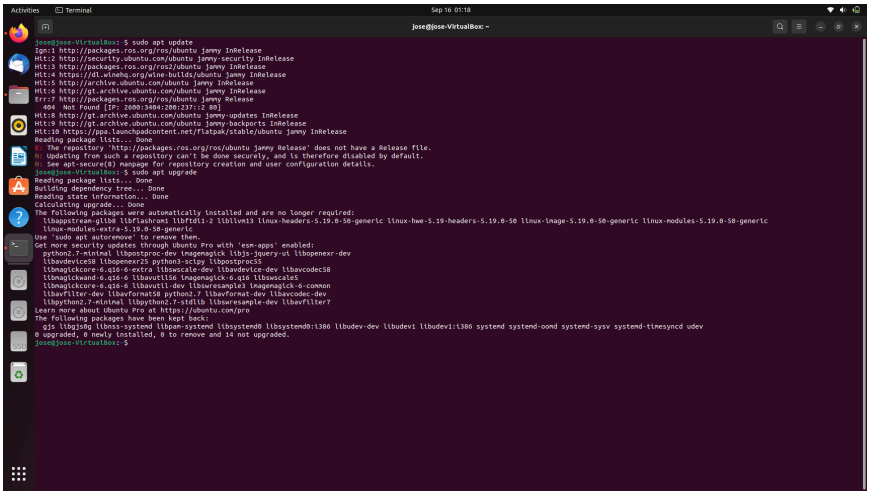


Figura 44: Actualización de software y paquetes de linux

Luego de completar estas tareas, se procedió a la instalación de ROS 2, específicamente la distribución LTS “Humble Hawksbill”, la cual es compatible con Ubuntu 22.04 y Crazyswarm 2. La instalación de ROS 2 Humble Hawksbill, siguió los pasos proporcionados en la página oficial [25] como se muestra en el diagrama de flujo de la Figura 45, en donde se muestra el procedimiento paso a paso realizado, además de posibles soluciones que se dieron a los errores más comunes encontrados en cada paso durante el proceso de instalación.

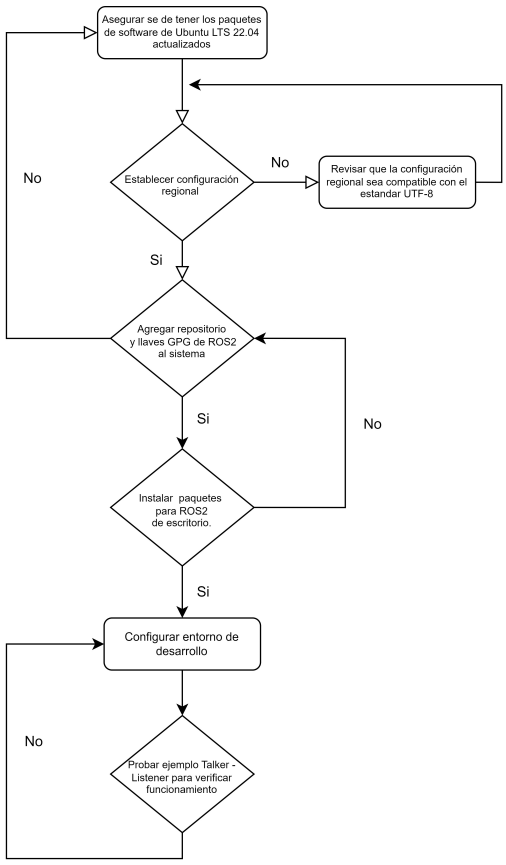


Figura 45: Diagrama de flujo de instalación de ROS 2

Este proceso detallado proporcionó un entorno de desarrollo sólido para la investigación y el desarrollo en robótica utilizando Ubuntu y ROS 2. La elección de Ubuntu como sistema operativo y la instalación de ROS 2 Humble Hawksbill permiten un desarrollo ágil y versátil de aplicaciones robóticas, brindando un marco sólido para la investigación y la innovación en el campo de la robótica. Cada paso cumplió un propósito específico, desde la elección del sistema operativo hasta la configuración del entorno de desarrollo, lo que resultó en un ambiente de trabajo eficiente y versátil para los investigadores y desarrolladores robóticos.

8.2. Integración de sistema de captura de movimiento

Se realizó la integración del sistema de captura de movimiento OptiTrack con el entorno de desarrollo para la infraestructura de control multidrones. Para esta integración, se utilizó la librería Mocap4ROS2, la cual permitió estandarizar el Mocap en el entorno de trabajo. Además, proporcionó una implementación en forma de paquetes ROS 2 que incluyen mensajes, formatos, herramientas y nodos. Para lograr esta integración, se utilizó NatNet, una herramienta de conexión que facilitó el establecimiento de comunicación con el *streaming* de datos de OptiTrack con el servidor. Además, se configuró el *software* Motive con los parámetros requeridos para garantizar dicha comunicación de manera efectiva.

8.2.1. Mocap4ROS2

Se instaló la librería Mocap4ROS en el servidor de Ubuntu, para ello se creó la el espacio de trabajo *mocap4ros2_ ws*, dentro de la cual se agrego el repositorio de la librería, asi mismo se agrego el repositorio de los paquetes de controladores para realizar la conexion con optitrack. Posteriormente, se instaló dichas dependencias y se procedió a construir el espacio de trabajo dentro de la carpeta raíz de la librería. La instalacion se realizo como una dependencia de ROS para permitir enlazar la libreria de MocapROS2 dentro del espacio de trabajo de ROS 2 para permitir trabajar con este según como se muestra en el Código 8.1 donde se describe el procedimiento realizado para crear el espacio de trabajo para establecer la comunicación con el Mocap OptiTrack y el servidor para la plataforma de pruebas con múltiples drones.

```

// Crear espacio de trabajo:
> mkdir -p mocap4ros2_ws/src && cd mocap4ros2_ws/src

// Descargar repositorio de MOCAP4ROS2 y controlador de OptiTrack:
> git clone https://github.com/MOCAP4ROS2-Project/mocap4ros2_optitrack.git
> git clone https://github.com/MOCAP4ROS2-Project/mocap.git


// Instalar dependencias:
> vcs import < mocap4ros2_optitrack/dependency_repos.repos
> cd .. && rosdep install --from-paths src --ignore-src -r -y

//Compilar espacio de trabajo:
> colcon build --symlink-install

//  Configurar espacio de trabajo:
> source install/setup.bash
```

Código 8.1: Configuración espacio de trabajo Mocap4ROS2.

Luego de completar el procedimiento de instalación del entorno de trabajo con Mocap4ROS2, se avanzó a la fase de configuración. En esta etapa, se realizó las configuraciones necesarias para lograr una integración adecuada entre ROS2 y OptiTrack. Parte esencial de esta configuración fue el nodo de comunicación con OptiTrack, donde se definió los valores de sus parámetros específicos. Estos ajustes se llevaron a cabo dentro del archivo *mocap_ optitrack_ driver_ params.yaml*, ubicado en la carpeta *mocap4ros2_ optitrack/optitrack_ driver* como se muestra en el Código 8.2 .

```
mocap_optitrack_driver_node:
  ros__parameters:
    connection_type: "Multicast" # Unicast / Multicast
    server_address: "192.168.50.200"
    local_address: "192.168.50.170"
    multicast_address: "239.255.42.99"
    server_command_port: 1510
    server_data_port: 1511
    ...
```

Código 8.2: Configuración de nodo de comunicación con OptiTrack.

Como se observa, dentro de la configuración se incluyó los parámetros clave, como el

tipo de conexión configurado como *multicast* para posibilitar la comunicación uno a muchos desde el servidor. También se especifican las direcciones del servidor, la local que se refiere a la ubicación del dispositivo ROS2 en la red y la dirección *multicast* utilizada para enviar datos de seguimiento a varios receptores.

Además, se estableció los puertos de comandos y datos. El puerto de datos transporta la información de seguimiento del OptiTrack, mientras que el de comandos permite enviar instrucciones al servidor de datos. Estos deben ser los mismos configurados en el panel de *"streaming data"* de Motive, y que para efectos de esta configuración se estableció los puertos utilizados por defecto para la comunicación con NatNet, 1510 para el puerto de comandos y 1511 para el puerto de datos. Es vital que ambos puertos estén disponibles y sin restricciones para lograr una comunicación de datos exitosa.

Este proceso de configuración fue esencial para establecer la conexión con el sistema de captura de movimiento de OptiTrack, permitiendo que ROS2 funcione como un nodo integrado. Esto se logró al configurar ROS2 de acuerdo con sus funcionalidades específicas.

8.2.2. NatNet

Se configuró NatNet para transmitir en tiempo real los datos de seguimiento de movimiento desde un sistema de captura de movimiento, como OptiTrack a través de Motive, hacia otras aplicaciones o dispositivos en una red. En este escenario, se estableció una comunicación entre una computadora con Windows ejecutando Motive y otra computadora con Ubuntu utilizando ROS 2. Esto permitió que la segunda computadora recibiera los datos de seguimiento de movimiento a través de NatNet de manera eficiente.

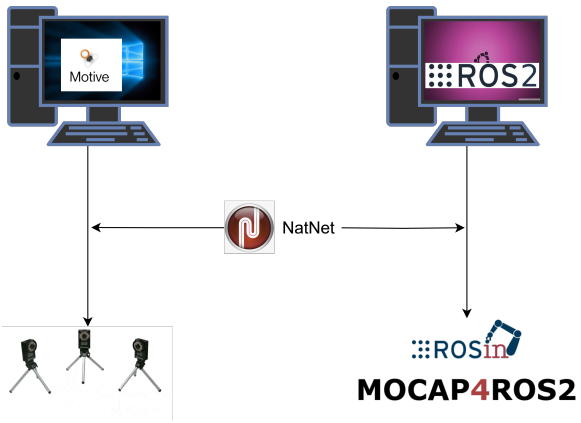


Figura 46: Diagrama de NatNet

Como se puede apreciar en el diagrama de la Figura 46, la computadora con sistema Windows ejecuta OptiTrack Motive, un programa que utiliza los datos proporcionados por el sistema de captura de movimiento que utiliza cámaras y marcadores para rastrear objetos en el espacio. Este programa, se configuró para transmitir datos de seguimiento a través del protocolo NatNet en la red local entre Motive y el servidor de ROS 2. La computadora con Windows actuó como servidor NatNet, mientras que la computadora con Ubuntu como cliente. El servidor de ROS 2, empleó una biblioteca, encargada de recibir los datos de

seguimiento de Motive a través de la red y convertirlos en mensajes o datos comprensibles, posibilitando la transferencia de datos de seguimiento de movimiento en tiempo real a través de la red local.

Para ello, se configuró la biblioteca que permite la conexión. Se estableció los siguientes parámetros al configurar el archivo *NatNetTypes.h*, definido dentro de la carpeta *src/mocap_ optitrack_ driver/NatNetSDK/include* del controlador de OptiTrack instalado junto con la librería Mocap4ROS2.

```
1
2  #define NATNET_DEFAULT_PORT_COMMAND         1510
3  #define NATNET_DEFAULT_PORT_DATA            1511
4  #define NATNET_DEFAULT_MULTICAST_ADDRESS    "239.255.42.99"
5
6  // limites de modelo
7  #define MAX_MODELS                  2000
8  #define MAX_MARKERSETS              1000
9  #define MAX_RIGIDBODIES             1000
10 ...
11 #define MAX_MARKERS                 200
12 ...
13 #define MAX_ANALOG_CHANNELS         32
14 #define MAX_ANALOG_SUBFRAMES        30
15 #define MAX_PACKETSIZE              65503
```

Código 8.3: Configuración biblioteca de NatNet.

Como se puede observar en el código, se definó las propiedades de la conexión, incluyendo elementos importantes como el puerto de comando y el de datos, a través de los cuales se estableció la comunicación de forma bidireccional para el *streaming* de datos. También se especificó, la dirección del servidor de datos de Motive configurado. Además, fue posible configurar el número máximo de modelos que pueden transmitirse, incluyendo un conjunto de marcadores individuales y cuerpos rígidos. Se estableció un límite máximo para la cantidad de subtramas analógicas por trama de captura de movimiento, es decir la cantidad de datos analógicos que se pueden capturar y transmitir en un solo marco de datos en el proceso de captura de movimiento. Lo cual fue útil para gestionar el ancho de banda y los recursos del sistema

Asimismo, se definió el tamaño máximo permitido para un paquete NatNet en *bytes*. Esto se utiliza para asegurar que los datos transmitidos a través del protocolo NatNet no superen dicho límite. Estos parámetros fueron configurables de acuerdo a las necesidades y requerimientos de rendimiento, permitiendo una comunicación efectiva entre los sistemas de captura de movimiento y las aplicaciones que hacen uso de estos datos.

8.2.3. OptiTrack Motive

Por otro lado, para establecer la conexión con OptiTrack y permitir el intercambio de datos, se llevó a cabo el proceso de encendido y calibración de las cámaras utilizando el programa Motive. Este proceso fue escencial para garantizar que las cámaras estén funcionando correctamente y puedan rastrear con precisión los marcadores utilizados en el sistema de captura de movimiento, como se ilustra en el diagrama de la Figura 47 .

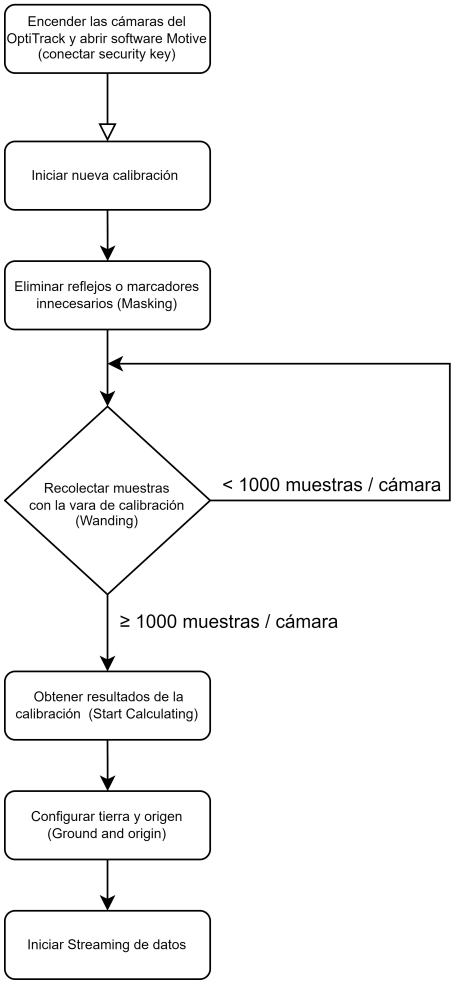


Figura 47: Diagrama de flujo para la calibración Mocap OptiTrack

Se realizó la configuración de Motive para permitir la transmisión de datos desde OptiTrack hacia Mocap4ROS2 a través NatNet. Para esto, en el panel de *Data Streaming* de Motive, se habilitó la opción *"Broadcast Frame Data"* para transmitir los datos de los fotogramas capturados por las cámaras.

A continuación, se configuró los puertos de comando y datos, que suelen ser 1510 y 1511 por defecto para establecer la comunicación y así mantener la congruencia con lo configurado en el dispositivo cliente con el que se busca comunicar. Sin embargo, estos puertos pueden variar según la disponibilidad en la red en la que se encuentren.

Además, se configuró la interfaz local con la dirección IP 192.168.50.200, que correspondió a la IP local del servidor del Robotat. Esto permitió el acceso y la obtención de los datos proporcionados por el sistema de captura de movimiento.

Dentro de la misma ventana de Data Streaming, se seleccionó los datos a transmitir a través de NatNet, incluyendo los *"Rigid Bodies"* para rastrear objetos rígidos, los *"Markers"* para incluir marcadores individuales o conjuntos (*single markers*), y los *"Skeletons"* en caso de trabajar con esqueletos humanos u objetos similares.

También se configuró un tipo de transmisión *multicast*, que resultó útil para permitir que múltiples receptores, como varias computadoras o dispositivos, reciban simultáneamente los mismos datos de seguimiento. Esto fue esencial para permitir la ejecución de la plataforma de control de los Crazyflies en paralelo con el ecosistema Robotat.

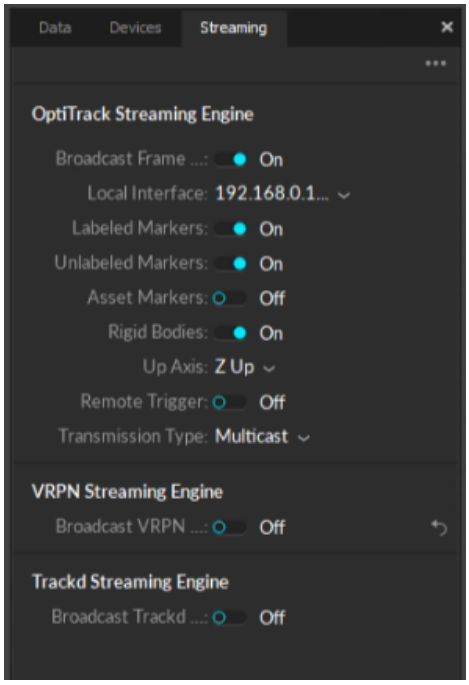


Figura 48: Configuración del panel de Data Streming para comunicación con NatNet

8.2.4. Pruebas preliminares con Single Markers

Se llevaron a cabo pruebas de conexión entre el servidor Ubuntu y el flujo de datos de Motive. El proceso comenzó con la inicialización del espacio de trabajo y la carga del script de configuración del entorno de ROS2. Esto se realizó en la carpeta raíz del espacio de trabajo, donde se ejecutaron los comandos correspondientes. También se activó el nodo de comunicación con OptiTrack utilizando el siguiente comando dentro del entorno de ROS2 tal y como se observa en el proceso de configuración del listado de comandos del Código 8.4.

```
1
2 //  Configurar espacio de trabajo MOCAP4ROS2:
3 > source install/setup.bash
4
5 // Cargar script de inicio de ROS 2
6 > source /opt/ros/humble/setup.bash
7
8 // Iniciar el sistema OptiTrack mediante un archivo de lanzamiento
9 > ros2 launch mocap_optitrack_driver optitrack2.launch.py
```

Código 8.4: Configuración biblioteca de NatNet.

Sin embargo, al ejecutar únicamente estos pasos, no se logró establecer una comunicación con el servidor de datos de OptiTrack como se muestra en la Figura 49, puesto que el servicio

de ROS2 que conecta al cliente con OptiTrack mediante NatNet no se encontró activo.

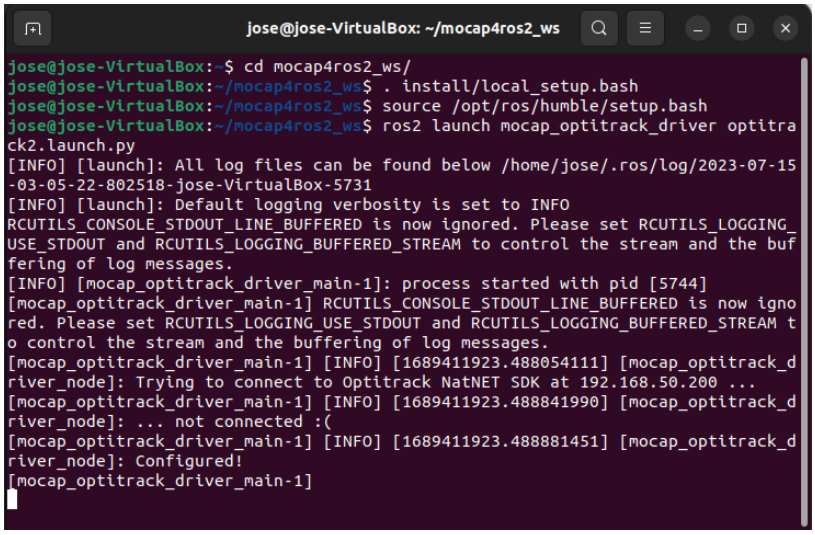


Figura 49: Conexión fallida con servidor NatNet

En una consola adicional, se abrió RQT Gui y se cargó el complemento MocapControl. Este complemento es utilizado en entornos de ROS para controlar y visualizar sistemas de captura de movimiento en Ubuntu. Esto permitió habilitar de manera manual los servicios y publicar los tópicos necesarios para establecer una conexión estable al ejecutar el nodo.

```
1
2 // Ejecutar la interfaz grafica de usuario (GUI) de RQT
3 > ros2 run rqt_gui rqt_gui --force-discover
```

Código 8.5: Ejecutar RQT GUI.

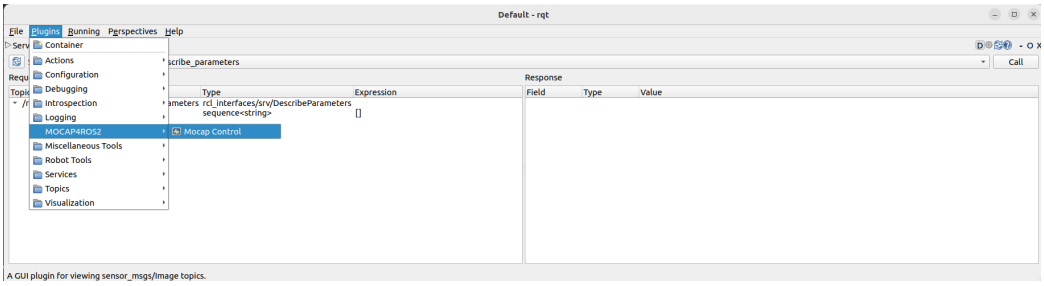


Figura 50: Carga de complemento para el control del Mocap

Figura 51: Selección de tópicos y servicios en RQT

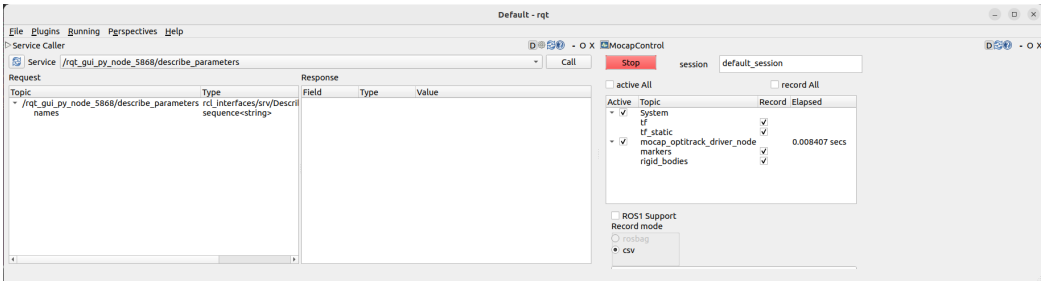


Figura 52: Publicación de tópicos y servicios con RQT

Además, se verificó que la configuración de OptiTrack funcionara correctamente y estuviera conectada. Dado que el nodo del controlador es un nodo de ciclo de vida, se realizó la transición para cambiar su estado y activarlo. Esto permitió la comunicación y la adquisición de datos desde el sistema de seguimiento de movimiento OptiTrack.

```
1
2 // Activar nodo controlador
3 > ros2 lifecycle set /mocap_optitrack_driver_node activate
```

Código 8.6: Activación de nodo de controlador de OptiTrack.

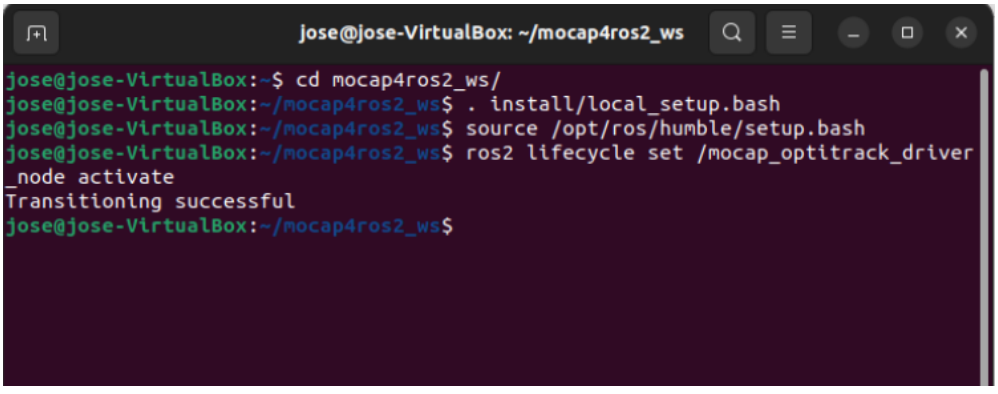


Figura 53: Transición a estado activo del nodo del controlador

La comunicación se estableció adecuadamente, actualizando los fotogramas a 120 Hz, como se configuró en Motive. Además, utilizando los tópicos publicados desde RQT GUI

como se muestra en el código 8.7, se pudo realizar un eco de los valores de las posiciones de los marcadores obtenidos del flujo de datos. Esto permitió observar cuántos marcadores estaban en la plataforma y conocer sus posiciones en los ejes x y z. En estas pruebas, se utilizaron solo marcadores individuales (*single markers*), obteniendo una tasa de actualización de pose de 1000Hz, es decir una transmisión de 1 paquete de puntos cada milisegundo desde Motive al servidor de ROS 2, con una pérdida de tan solo 1 % como se observa en la Figura 56 .

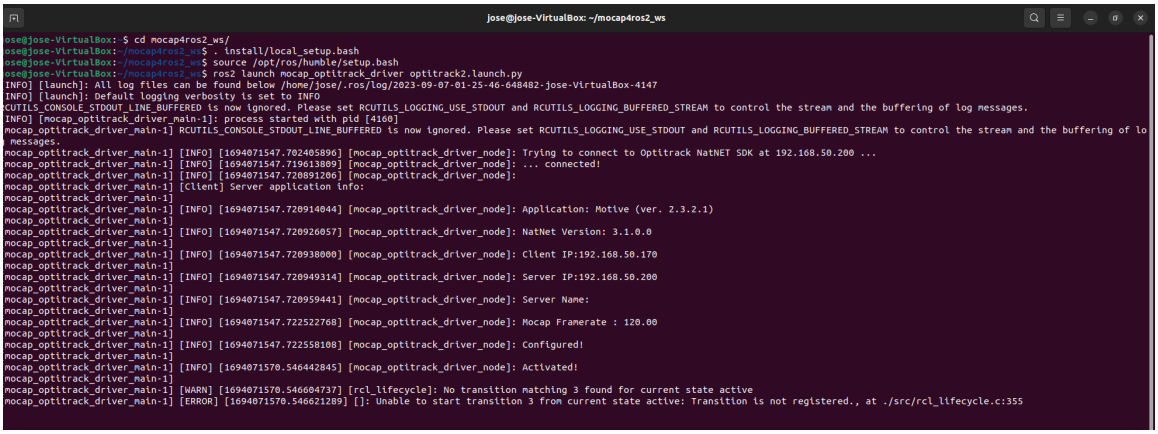


Figura 54: Conexión establecida entre cliente-servidor mediante NatNet

Para estas primeras pruebas se empleó un solo marcador reflectivo. Las pruebas incluyeron la verificación de la posición y la detección de marcadores. También se revisó la detección de marcadores de cuerpos rígidos. Durante las pruebas, se examinaron las posiciones en los ejes x y z de los *single markers*.

```
// Escuchar y mostrar mensajes de topico /markers
> ros2 topic echo /markers

// Escuchar y mostrar mensajes de topico /rigid_bodies
> ros2 topic echo /rigid_bodies
```

Código 8.7: Publicación de tópicos del nodo de OptiTrack.

Figura 55: Marcador reflectivo (*single marker*)

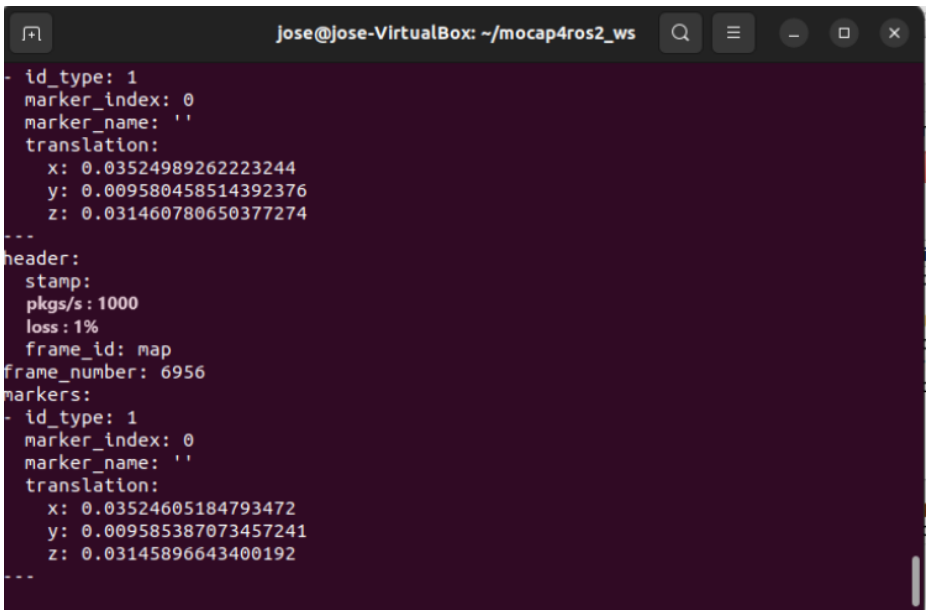


Figura 56: Publicación de posiciones de los marcadores

CAPÍTULO 9

Integración y configuración Crazyswarm 2

Se realizó la integración y configuración de Crazyswarm 2. En primer lugar, se instalaron las bibliotecas siguiendo las instrucciones de la página oficial del *framework*. Esto incluyó la instalación de las dependencias necesarias para Crazyswarm 2, así como la configuración del *backend* a través de *cflib*.

Posteriormente, se configuró el espacio de trabajo de ROS 2 para instalar Crazyswarm 2 como una extensión de ROS 2, y se construyó el espacio de trabajo de la librería. Finalmente, se actualizó el espacio de trabajo de manera local en la computadora.

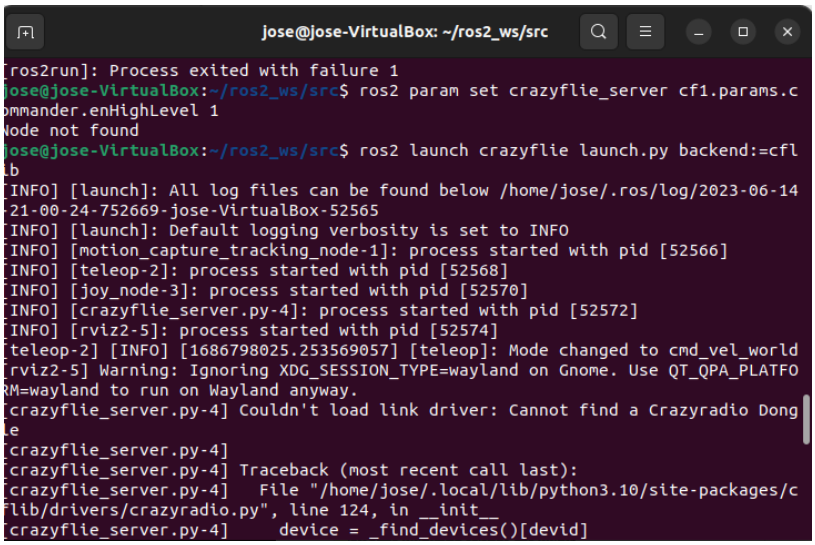


Figura 57: Instalación de Crazyswarm 2

9.0.1. Configuración y pruebas de comunicación ROS2

Se configuró el nodo *crazyflie_server* contenido en el archivo *server.yaml*, dentro espacio de trabajo instalado de Crazyswarm 2, ubicado dentro del espacio de trabajo *.../crazyswarm2/crazyflie/config*. A través de esta configuración, se definió parámetros esenciales que rigen el comportamiento del Crazyflie y el funcionamiento de este nodo en particular como se muestra a en el código 9.1.

```
/crazyflie_server:
  ros__parameters:
    warnings:
      frequency: 100.0 # comprobaciones por segundo
      ...
    sim:
      max_dt: 0 #0.1
      backend: np
      visualizations:
        rviz:
          enabled: true
       ...
      controller: mellinger # none, pid, mellinger
```

Código 9.1: Configuración nodo del servidor de Crazyflie.

Dentro de la sección *ros_parameters*, se encuentran diversas especificaciones que detallan cómo se deben llevar a cabo ciertas operaciones y qué configuraciones son necesarias para garantizar un rendimiento óptimo. En primer lugar, se definió el parámetro *warnings*, el cual establece la frecuencia con la que se notifican y verifican advertencias en el sistema. En este caso, se configuró para realizar estas comprobaciones una vez por segundo, a una frecuencia de 100.0 Hz.

El siguiente aspecto que se abordó fue la simulación, donde se observa dos parámetros que son de utilidad si se llegase a tomar en cuenta alguna simulación, pero principalmente el argumento que habilita la visualización en Rviz, fue fundamental para poder utilizar esta herramienta para poder visualizar la posición gráfica dentro del sistemas de coordenadas globales de la plataforma o infraestructura en donde se encuentra el dron. En las primeras pruebas, realizadas de manera aislada del sistema de seguimiento de movimiento (Mocap), esta característica no generó un impacto significativo.

Finalmente, el parámetro *controller* desempeñó un papel fundamental, ya que determinó qué algoritmo de control utilizó el dron. Las opciones disponibles incluyó *none* (sin controlador), *PID* y *Mellinger*. La elección del mismo dentro de esta configuración se tomó con base a los resultados observados en el primer capitulo en donde el controlador Mellinger mostró mejor comportamiento. En resumen, este fragmento de código fue esencial para configurar el nodo *crazyflie_server*, además del comportamiento y las funcionalidades del dron Crazyflie 2.1 al momento de realizar pruebas.

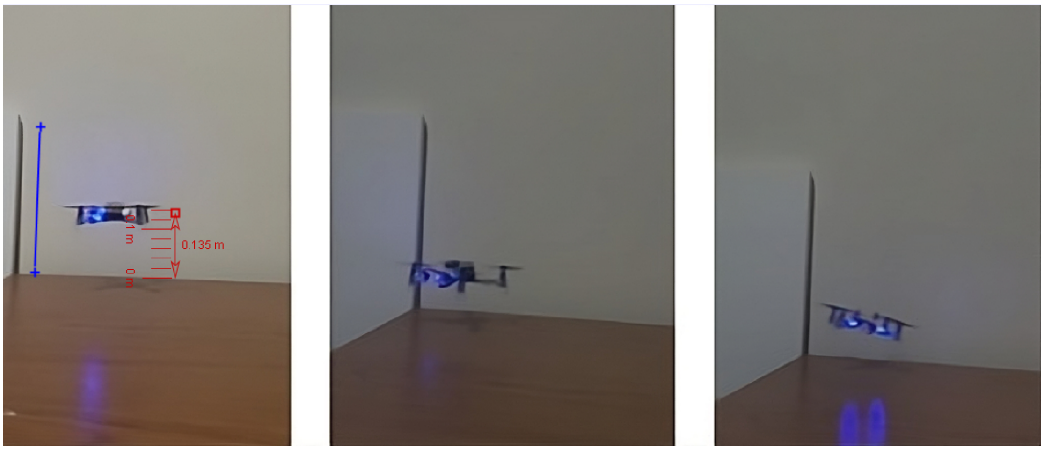


Figura 58: Secuencia de *takeover* con Crazyswarm sin Mocap

Con esto, se ejecutó el servidor de Crazyswarm 2 como un nodo de ROS 2 y se llevó a cabo un experimento físico para realizar un *takeover* a una altura de 0.2 metros. Se ejecutó el ejemplo *Hello World* incluido por defecto en la librería de instalación de Crazyswarm 2. El resultado mostró que el dron se elevó aproximadamente a 0.135 metros de altura, sin embargo, instantes después el comportamiento del Crazyflie 2.1 se volvió errático y terminó por caer. Este comportamiento era el esperado, ya que, aunque se mantuvo estable durante el despegue hasta la altura máxima alcanzada, al igual que en las gráficas de la estimación de posición proporcionadas en el primer capítulo, estas también divergieron. Por lo tanto, sin el apoyo de las mediciones del OptiTrack, no se esperaba que mantuviera estabilidad en el vuelo.

Así mismo como se muestra en la Figura 59, se graficó con ayuda de Motive colocando un *single marker* en el Crazyflie 2.1 para poder visualizar el funcionamiento del mismo sin la retroalimentación del sensor exteroceptivo (Mocap). Como se observa al ejecutar el comando para que el dron se eleve a una altura de 1 m, este realiza un movimiento buscando llegar punto estipulado. Sin embargo, termina por comportarse de manera errática y diverger del punto objetivo.

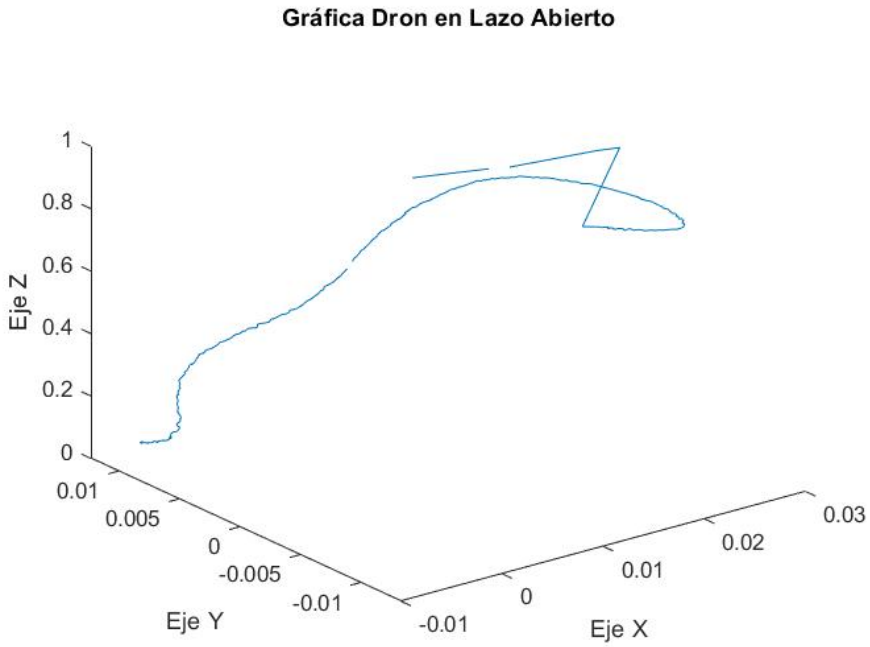


Figura 59: Comportamiento Crazyflie sin Mocap con Motive

9.0.2. Integración del Mocap y Crazyswarm2

Se procedió a realizar la configuración del nodo de conexión de Crazyswarm con el sistema de captura de movimiento OptiTrack. Como se puede observar en el Código 9.2, se realiza la configuración de los parámetros relacionados con el sistema de captura de movimiento dentro del contexto de ROS 2. Aquí se definen varios aspectos esenciales para establecer la comunicación y el funcionamiento adecuado del sistema.

```
1
2  /motion_capture_tracking:
3    ros__parameters:
4      type: "optitrack"
5      hostname: "192.168.50.200"
6      mode: "libobjecttracker" #motionCapture,libRigidBodyTracker
7      ...
8
9      numDynamicsConfigurations: 1
10       dynamicsConfigurations:
11         "0":
12           maxVelocity: [2.0, 2.0, 3.0]
13           maxRate: [20.0, 20.0, 10.0 ]
14           maxAngle: [1.4, 1.4, 1.4]
15           maxFitnessScore: 0.001
16     ...
```

Código 9.2: Configuración nodo del Crazyswarm.

En primer lugar, se configura el tipo de sistema de captura de movimiento que se utilizó, en este caso, se especifica que se empleará OptiTrack. Posteriormente, se estableció la direc-

ción IP del servidor del Mocap que realiza el *streaming* de datos, configurando la dirección como "192.168.50.200".

El siguiente parámetro, *mode*, determinó la biblioteca o modo de seguimiento que se utilizará. En el código se mencionan dos opciones: *libobjecttracker* y *motionCapture*. Esta configuración influirá en cómo se procesan y se obtienen los datos de seguimiento, principalmente porque *motionCapture* proporciona un seguimiento de cuerpo rígido completa. En este caso se realizaron pruebas con *single markers* por lo que se estableció *libobjecttracker* como biblioteca a utilizar.

Por último, se especificarón las configuraciones de la dinámicas de los drones, donde se definen límites máximos para la velocidad, la tasa de cambio, el ángulo y la puntuación de aptitud de los objetos que se están rastreando de acuerdo con las necesidades y requerimientos que se tengan.

Con el nodo de conexión con el Mocap configurado, se realizaron pruebas de comunicación con el streaming de datos de OptiTrack, sin ejecutar el servidor de Mocap4ROS. Esto se logró simplemente realizando la conexión mediante una versión del Cliente de NatNet integrada dentro de la librería de Crazyswarm 2, llamado *NatNetClient.py*.

```
...
class NatNetClient:
    def __init__( self ):

        self.serverIPAddress = "192.168.50.200"
        self.localIPAddress = "127.0.0.1"

        self.multicastAddress = "239.255.42.99"
        self.commandPort = 1510
        self.dataPort = 1511
        ...
    # Create a data socket to attach to the NatNet stream
    def __createDataSocket( self, port ):
        result = socket.socket( socket.AF_INET,     # Internet
                                socket.SOCK_DGRAM,
                                socket.IPPROTO_UDP)    # UDP

        result.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        result.setsockopt(socket.IPPROTO_IP, socket.IP_ADD_MEMBERSHIP,
        socket.inet_aton(self.multicastAddress) +
        socket.inet_aton(self.localIPAddress))
        result.bind( (self.localIPAddress, port) )
        return result

    # Create a command socket to attach to the NatNet stream
    def __createCommandSocket( self ):
        result = socket.socket( socket.AF_INET, socket.SOCK_DGRAM )
        result.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        result.bind( ('', 0) )
        result.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
        return result
...
```

Código 9.3: Configuración servicio de NatNet.

Como se puede observar en el Código 9.3, el cliente de NatNet desarrollado en Python

crea y configuró un *socket* en los puertos de datos y comandos para establecer una conexión con un servidor NatNet. En su constructor, se definen las direcciones IP del servidor y del cliente, así como la dirección multicast para la transmisión de datos. Los puertos de comando y datos también se especifican en este punto.

El cliente utiliza dos métodos privados para la creación de *sockets*. El primer método, *_createDataSocket*, se encarga de configurar el socket de datos. Este *socket* se configura para escuchar y recibir datos en el puerto especificado y se une a la dirección multicast para recibir los datos de seguimiento de NatNet.

El segundo método, *_createCommandSocket*, se utiliza para configurar el *socket* de comandos. Este se configura para transmitir comandos y se vincula a un puerto disponible en el cliente. En resumen, el código del cliente NatNet en Python establece la infraestructura necesaria para la comunicación con un servidor NatNet mediante la configuración de sockets en los puertos de datos y comandos, lo que permite la transmisión y recepción de datos de seguimiento en tiempo real.

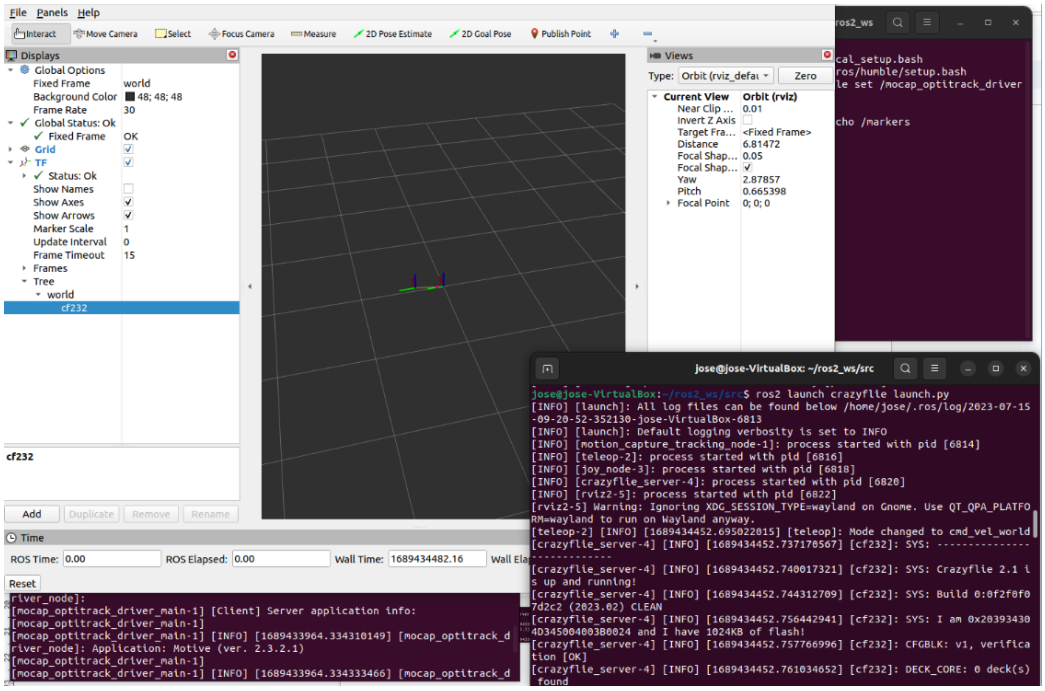


Figura 60: Prueba conexión NatNetClient de Crazyswarm 2

Sin embargo, al realizar la prueba únicamente estableciendo la conexión del streaming de datos con el cliente de NatNet de Crazyswarm 2, se observó que solo admitía la comunicación con un máximo de 2 drones, luego de esto el seguimiento desde el *software* Rviz para visualizar los drones divergió a partir del tercer Crazyflie agregado. Además la conexión era inestable pues se desconectaba por momentos. A partir de este punto, por lo tanto, fue necesario ejecutar el servidor de Mocap4ROS2 para permitir la comunicación con varios Crazyflies en la plataforma de Crazyswarm 2.

Pruebas preliminares Single marker

El enfoque de utilizar las mediciones de posición obtenidas a través de OptiTrack, se utilizó para mejorar las estimaciones y finalmente implementar un *Filtro de Kalman Extendido*. Inicialmente, como se mencionó en capítulos anteriores, se empleó el filtro de Kalman para estimar los valores de posición y orientación del Crazyflie mediante una fusión de sensores que integraba las mediciones del giroscopio y el acelerómetro del dron. Sin embargo, se determinó que, aunque la estimación de la orientación del dron utilizando el filtro de Kalman y el controlador Mellinger era suficientemente precisa, la estimación de las posiciones en los ejes x,y,z no convergía completamente.

Para abordar este problema, se extendió el filtro de Kalman para fusionar las mediciones proporcionadas por los sensores internos del Crazyflie 2.1, con el sistema de captura de movimiento. Esto agregó precisión a la estimación de posición, y el enfoque funcionó de manera satisfactoria. Se llevaron a cabo pruebas de vuelo, incluyendo un takeover en el cual se pudo mantener el Crazyflie a una altura específica sobre la plataforma de pruebas dentro del entorno Robotat.

Se realizó la configuracion de la dirección única de cada dron *URI* en donde se le estableció a cada uno a través del Crazyclient. Dicha dirección se utilizo para poder identificar el Crazyflie dentro del enjambre y poder controlar de manera aislada de los demas drones, asi como realizar también rutinas de *broadcast* que permitieran mandar comandos globales a los Crazyflies dentro de la plataforma.

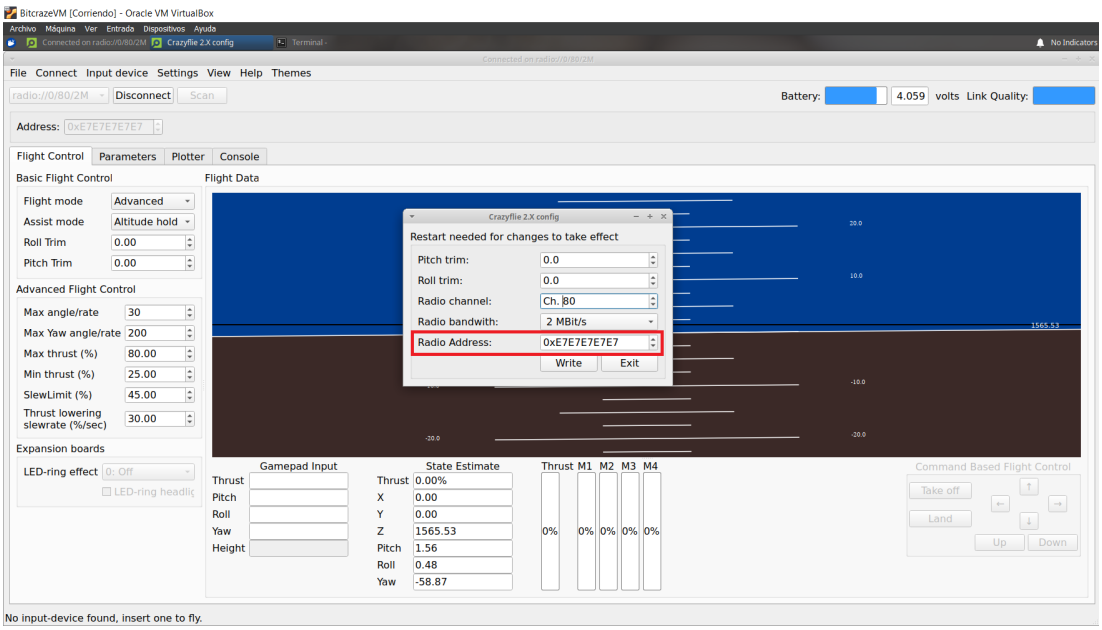


Figura 61: Configuración de URI para los Crazyflie

Se configuró el archivo *crazyflies.yaml*, en el cual se pudo agregar individualmente cada dron al enjambre utilizando su dirección URI, para con esto, asignarle una identificación y tipo que definió la clase de marker que se utilizó con el Crazyflie, siendo un *default_single_marker*, o bien *rigid_body*.

```
robots:
  cf231:
    enabled: true
    uri: radio://0/80/2M/E7E7E7E7E8
    initial_position: [0, 0, 0]
    type: cf21  # see robot_types
    ...

robot_types:
    cf21:
        motion_capture:
            enabled: true
            marker: default_single_marker
            ...
...
```

Código 9.4: Configuración parámetros de los drones.

Figura 62: Crazyflie 2.1 con marcador reflectivo integrado

Se realizó pruebas con el Crazyflie 2.1, colocando un *single marker* como se muestra en la Figura 61. Para ello se requirió correr el nodo y servicio de MOCAP4ROS2 para obtener los datos del sistema de captura de movimiento. Posteriormente, se ejecutó el nodo del servidor de Crazyswarm 2 para correr el archivo *helloword.py* que contenía el código a modificar. Dicho procedimiento para correr el servidor dentro del espacio de trabajo de ROS2, se muestra en el siguiente pseudocódigo.

```
//ros2_ws/src

//  Configurar espacio de trabajo MOCAP4ROS2:
> source install/setup.bash

// Cargar script de inicio de ROS 2
> source /opt/ros/humble/setup.bash

// Correr Servidor de Crazyswarm2
> ros2 launch crazyflie launch.py

// Ejecutar Hello Word
> ros2 run crazyflie_examples hello_world
```

Código 9.5: Ejecución de nodo de Crazyswarm.

Se desarrolló un algoritmo que permitó realizar un *takeover* utilizando la librería Crazyswarm2 tal como se observa en el Código 9.6, la cual se importó a un archivo de Python ubicado dentro del espacio de trabajo de ROS2. Dentro de la función *main* se definió el comportamiento del dron. Se creó un objeto Crazyswarm para acceder a todas las funcionalidades de la librería. En este proceso, se creó un objeto *timeHelper* con el propósito de gestionar y sincronizar el tiempo necesario para las acciones específicas de los drones. También se implementó un objeto *crazyflie* que estableció la conexión con cada uno de los drones asociados a la plataforma de pruebas.

Es relevante destacar que esta implementación permitió acceder a todos los drones con el fin de enviar un comando único mediante una transmisión en modo broadcast para controlar todos los drones disponibles. También fue posible acceder de manera individual a cada Crazyflie en forma de array accediendo a la propiedad *crazyflies* del objeto. La posición de cada array, se determinó según la dirección URI del Crazyflie. En otras palabras, la dirección con el valor más bajo ocupará la posición 0 en el *array* de *crazyflies*, y así sucesivamente.

```
from crazyflie_py import Crazyswarm

TAKEOFF_DURATION = 2
HOVER_DURATION = 2


def main():
    swarm = Crazyswarm()
    timeHelper = swarm.timeHelper
    crazyflie = swarm.allcfs
    cf1 = crazyflie.crazyflies[0]

    cf1.takeoff(targetHeight=1.0, duration=TAKEOFF_DURATION)
    timeHelper.sleep(TAKEOFF_DURATION + HOVER_DURATION)
    cf1.land(targetHeight=0.00, duration=2)
```

Código 9.6: Algoritmo de *takeover*.

En el algoritmo proporcionado, se observa que se accede a la función *takeoff* definida dentro de la librería de Crazyswarm. Esta función permite enviar el parámetro de altura deseada para el Crazyflie y la duración de la secuencia. En este caso, se configuró la altura objetivo en 1.0 metro y se asignó una duración de 2 segundos para que el Crazyflie alcance esa altura desde el inicio. Como se puede apreciar en la Figura 63, el dron alcanzó una altura de 0.966 metros, lo que resulta en un porcentaje de error del 3.4 % de los datos estimados con el programa Tracker.

Luego, se ejecutó la instrucción *sleep* contenida en *timeHelper*, lo que permitió al dron, mantenerse en la posición deseada durante 4 segundos. Como se muestra en la Figura 64, esta prueba se logró exitosamente, ya que el dron pudo mantenerse en su posición incluso cuando se aplicó una perturbación externa durante este tiempo, regresando a su posición original. Por último, se implementó la función *land*, que permitió al dron aterrizar de manera estable.

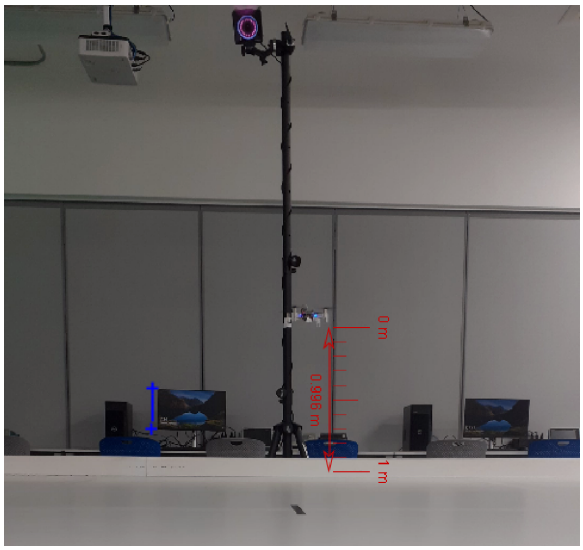


Figura 63: Prueba de Takeover a 1m de altura

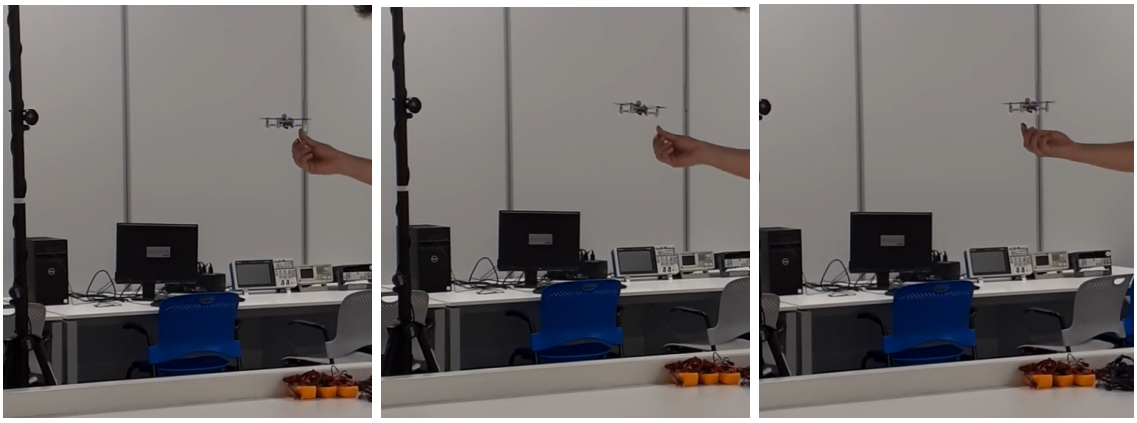


Figura 64: Prueba de Takeover a 1m de altura con perturbación

Además, para mejorar la precisión en la verificación de las mediciones del dron, se llevó a cabo una prueba de takeover a una altura de 1 metro. Esta prueba utilizó Motive para comparar con mayor precisión las diferencias entre los valores teóricos y experimentales. Como se puede apreciar en la Figura 65, se registró que el dron alcanzó una altura máxima de 0.997 metros, lo que representa un porcentaje de error del 0.3 %.

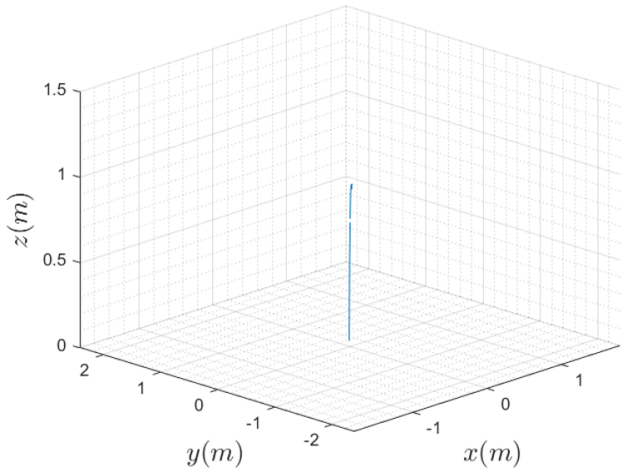


Figura 65: Prueba de Takeover a 1 m con Motive

9.0.3. Implementación de trayectorias para uno y múltiples drones

Se realizó la integración de un total de tres drones al enjambre para trabajar con la plataforma desarrollada, en donde se experimentó con la implementación de una trayectoria estática y una dinámica para validar el funcionamiento tanto de los algoritmos desarrollados con base a la librería Crazyswarm 2, como su integración con el Mocap OptiTrack.

Se desarrolló un algoritmo como el que se muestra en el Código 13.1, para coordinar el movimiento de tres drones utilizando el concepto de Movimiento Armónico Simple (MAS). El objetivo fue generar una trayectoria circular en tres dimensiones (x, y, z) con un desfase equidistante entre los drones, lo que resulta en un patrón de vuelo sincronizado y armonioso.

El algoritmo se implementó utilizando la librería Crazyswarm 2, que proporcionó las herramientas necesarias para controlar los drones Crazyflie. El proceso se dividió en varias etapas clave:

1. Conexión y preparación: En primer lugar, se estableció la conexión con la plataforma Crazyswarm, creando instancias para los tres drones (cf1, cf2 y cf3).

2. Definición de la trayectoria circular: Se definió una trayectoria circular en el espacio tridimensional. Las coordenadas x y y de cada dron se calcularon utilizando la fórmula del MAS:

$$x(t) = A \cdot \cos(\omega t + \phi) + x_{\text{centro}}$$
$$y(t) = A \cdot \sin(\omega t + \phi) + y_{\text{centro}}$$

Donde:
- A es el radio del círculo.
- ω es la frecuencia angular relacionada con el período T.

- ϕ es la fase inicial.
- x_{centro} y y_{centro} son las coordenadas del centro del círculo.

La coordenada z de cada dron se mantuvo constante durante el movimiento, ya que no se requirió movimiento vertical en esta aplicación, sin embargo fue un parámetro modificable para validar que se puede controlar cada dron individualmente.

3. Configuración de velocidad y duración: Se estableció la velocidad de vuelo deseada (*velocity*) para controlar la rapidez con la que los drones completarían el círculo. Además, se definió una duración específica para cada segmento de la trayectoria circular.

4. Ejecución de la trayectoria: El bucle principal del algoritmo se encargó de que cada dron siguiera la trayectoria circular. En cada iteración del bucle, se utilizó la función *cf.goTo()* para mover el dron a la posición especificada en los puntos de la trayectoria circular. La fase inicial (ϕ) de cada dron se ajustó para lograr un desfase equidistante entre ellos. Después de cada movimiento, se utilizó *timeHelper.sleep()* para asegurarse de que los drones se mantuvieran sincronizados.

5. Aterrizaje: Una vez que se completó la trayectoria, se programó el aterrizaje de los Crazyflies utilizando *cf.land()*. Se esperó un tiempo adicional para garantizar que los drones aterrizaran completamente antes de finalizar el algoritmo.

Este algoritmo permitió a los tres drones realizar un vuelo coordinado siguiendo una trayectoria circular con un desfase equidistante entre ellos. La sincronización y precisión en el movimiento se lograron mediante el uso de las fórmulas del MAS y el control proporcionado por la librería Crazyswarm.

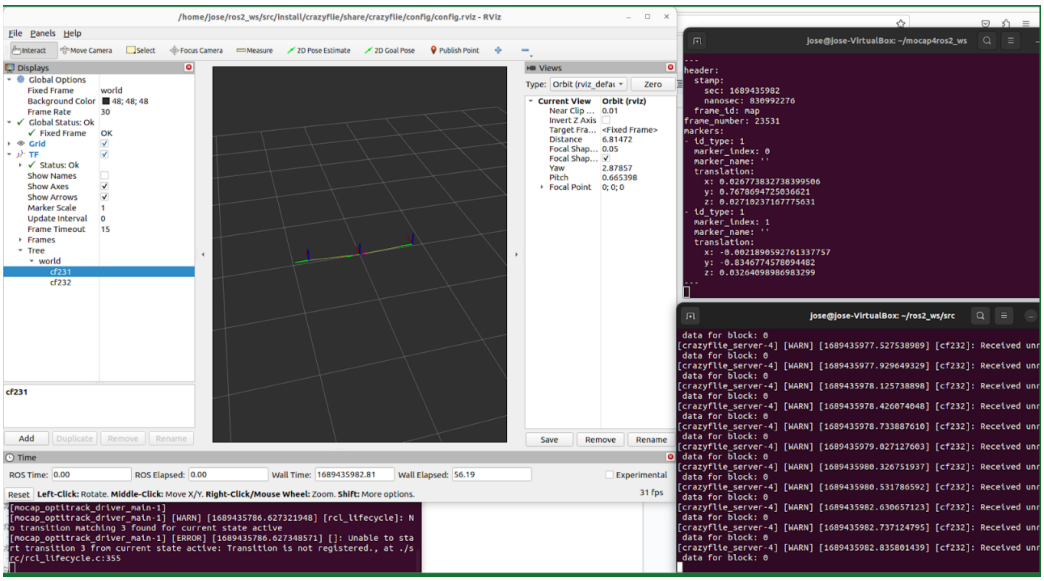


Figura 66: Integración a la plataforma de pruebas 3 drones

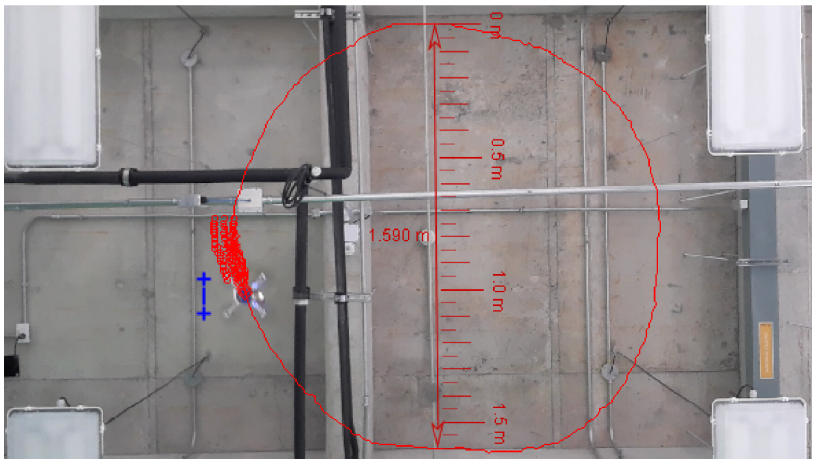


Figura 67: Trayectoria circular 1 dron

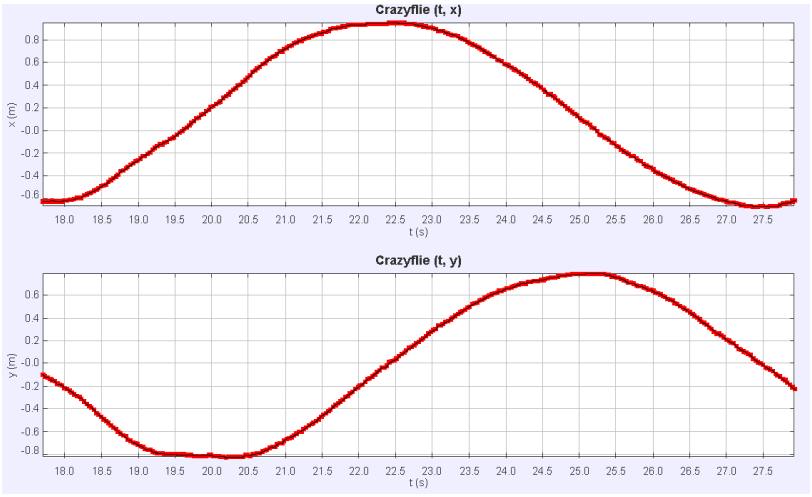


Figura 68: Gráfica de movimiento armónico simple del dron (Tracker)

Se validó el algoritmo mediante una prueba con un único dron, con el propósito de verificar la precisión de la trayectoria generada. Como se evidencia en la Figura 67, el dron ejecutó exitosamente la trayectoria circular. El radio estimado del círculo trazado en el aire, obtenido mediante el programa Tracker, fue de 1.59 metros. En comparación con los 1.60 metros esperados y configurados en el algoritmo, se obtiene un porcentaje de error del 0.625 %, lo cual confirma tanto la ejecución precisa de la trayectoria como la correcta integración del sistema de captura de movimiento en la infraestructura.

Además, se realizó la grabación de los resultados de la trayectoria mediante OptiTrack desde el programa Motive. Como se aprecia en las Figuras 67 y 68, el dron ejecutó la trayectoria de manera adecuada, partiendo desde el origen y completando un círculo a una altura de 1 metro con un radio de 0.8 metros, obteniendo un porcentaje de error promedio de 0.3 %, como se observa en el Cuadro 4.

Eje	Teórico (m)	Experimental (m)	error (%)
x	0.8	0.806	0.60
y	0.8	0.801	0.10
z	1.0	0.998	0.20
		EM	0.30

Cuadro 4: Porcentajes de error trayectoria circular

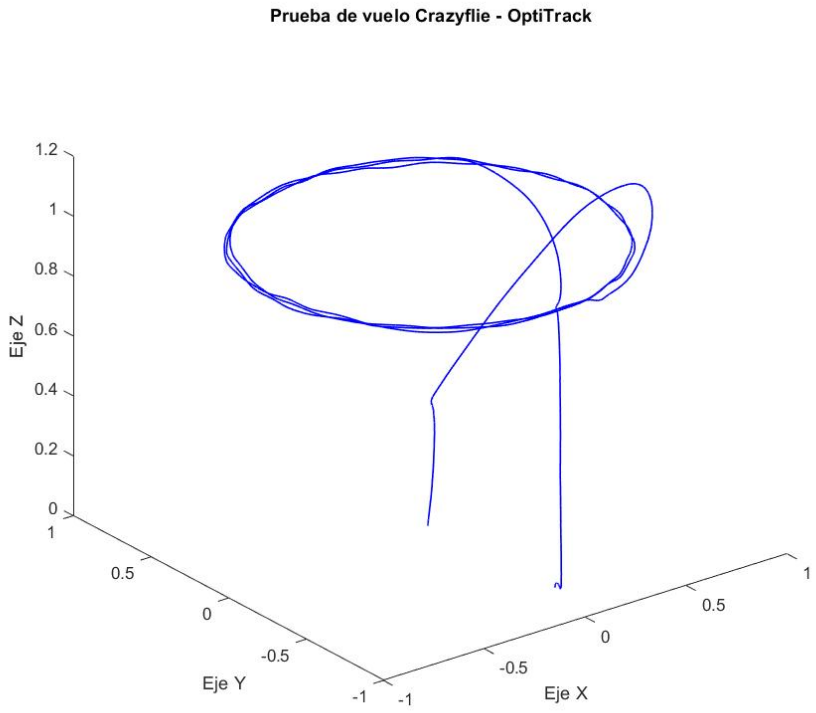


Figura 69: Gráfica datos en 3D grabados desde OptiTrack

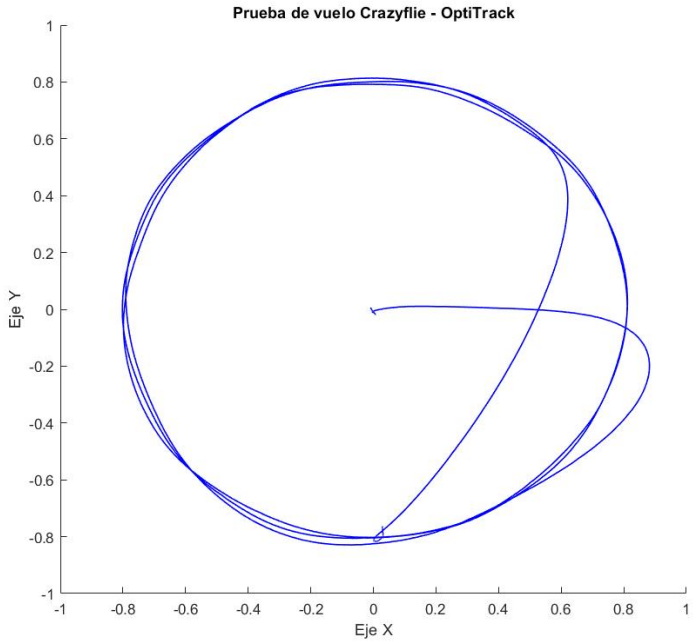


Figura 70: Gráfica datos en 2D grabados desde OptiTrack

Figura 71: Prueba de vuelo MAS con 2 drones

Figura 72: Prueba de vuelo MAS con 3 drones

Figura 73: Rutinas de movimiento con 4 drones

En la configuración de la plataforma de pruebas, se introdujeron hasta 5 drones de manera individual, como se puede apreciar en detalle en la Figura 74. Esta integración múltiple no generó inconvenientes durante la realización de las pruebas, lo que destacó la eficacia de las configuraciones individuales de cada dron. Es importante destacar que para la comunicación de estos 5 Crazyflies 2.1 se empleó una sola antena CrazyRadio PA. Esta implementación permitió un control coordinado y sin contratiempos, a pesar de la simultaneidad en la operación de múltiples drones.

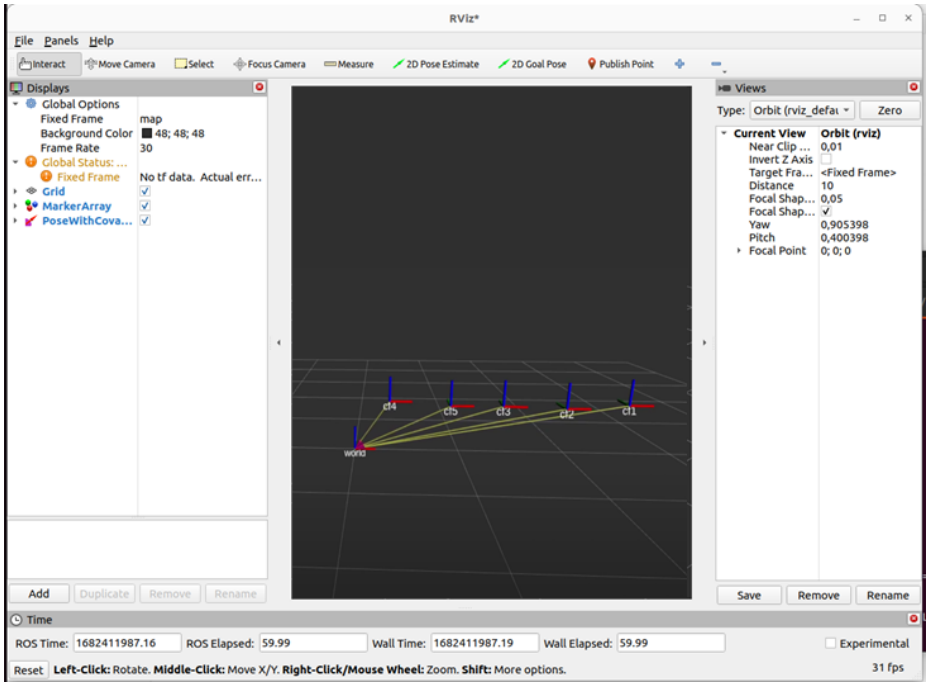


Figura 74: Integración a la infraestructura de 5 drones

Por otro lado, se implementó un algoritmo para trayectoria dinámica como se muestra en el Código 13.2, utilizando *uploadTrajectory*, donde se proporcionó un conjunto de puntos desde un archivo CSV, más separados entre sí. Se utilizó una interpolación mediante polinomios de séptimo grado para generar internamente una trayectoria óptima para el movimiento. Se realizó una prueba utilizando una figura en forma de ocho, como se muestra en la Figura 75.

Figura 75: Prueba de vuelo con 1 dron figura 8

Duración (s)	x^0 **(m)**	y^0 **(m)**	z^0**(m)**	yaw^0 **(°)**
1.0	0.000	0.000	0.500	0.0
1.0	0.396	-0.446	0.500	0.0
1.0	0.922	-0.291	0.500	0.0
1.0	0.923	0.290	0.500	0.0
1.0	0.405	0.451	0.500	0.0
1.0	0.001	0.002	0.500	0.0
1.0	-0.403	-0.450	0.500	0.0
1.0	-0.922	-0.292	0.500	0.0
1.0	-0.923	0.289	0.500	0.0
1.0	-0.399	0.447	0.500	0.0

Cuadro 5: Puntos cargados en el Crazyflie para trayectoria dinámica Crazyswarm

Es importante destacar que se observó diferencias visuales significativas entre las pruebas con los dos tipos de trayectorias. En el caso de la trayectoria estática, se determinó que el dron la ejecutaba ligeramente menos fluidamente, presentando una latencia de unos milisegundos para completar la trayectoria. Por otro lado, la trayectoria dinámica no solo permitía realizarla mucho más rápido en términos de milisegundos, sino que también se notó que el dron tenía cierta holgura al seguir la trayectoria, ajustando su orientación de la manera más eficiente posible. A diferencia de la prueba con la trayectoria dinámica, donde el dron siempre mantuvo una orientación de yaw=0, $roll$=0 y $pitch$=0.

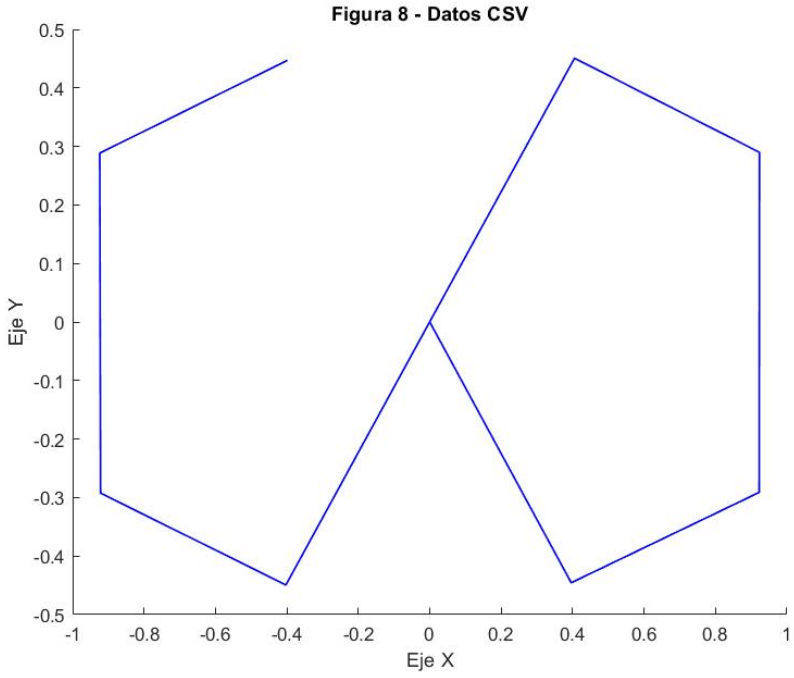


Figura 76: Gráfica figura 8 según CSV

Además, es posible notar la diferencia en la forma como se realiza la figura, en la gráfica generada utilizando la serie de puntos proporcionados en el archivo CSV para cargar la trayectoria. Se observa el trazo que dibuja el dron en el aire se siguiera simplemente la secuencia de puntos como una trayectoria estática, la figura resultante tiende a ser más angular. Por otro lado, al generar una trayectoria óptima de forma dinámica mediante la interpolación con polinomios de séptimo grado, se obtiene una trayectoria mucho más suave y visualmente atractiva.

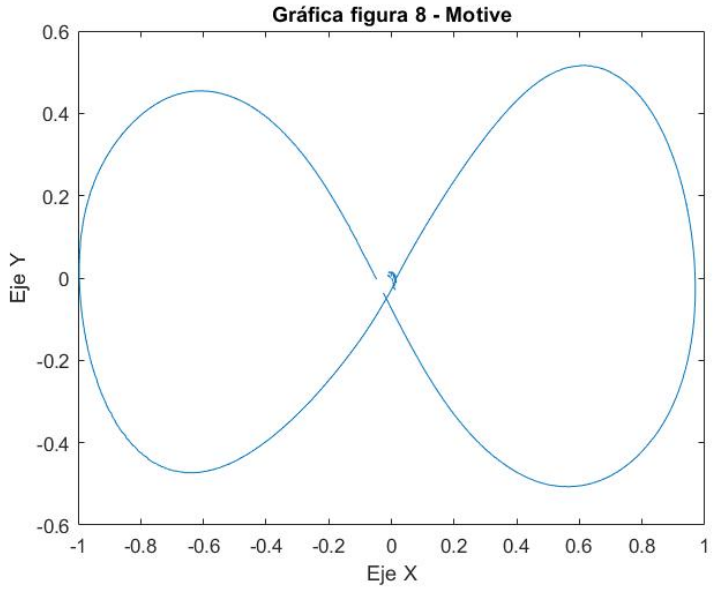


Figura 77: Gráfica figura 8 con interpolación de polinomios de séptimo grado

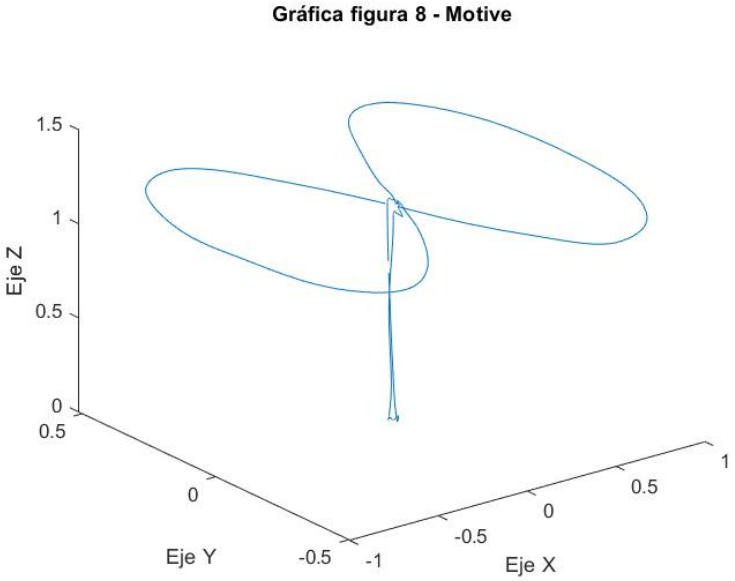


Figura 78: Grabación de rutina figura 8 con Motive

9.0.4. Integración de la infraestructura al Robotat

Como parte fundamental del objetivo de establecer una infraestructura robusta para la experimentación multidrones, se integró las funciones desarrolladas a lo largo de varias etapas en este proyecto basadas en la librería Crazyswarm 2, con el propósito específico de crear distintas formaciones con múltiples drones utilizando un servidor que a través del protocolo de comunicación TCP, implementa la infraestructura. Esto para permitir que varias computadoras con el *software* MatLab instalados, puedan acceder como clientes al nodo implementado en el servidor de ROS 2 que permite la comunicación con los drones Crazyflie 2.1.

El desarrollo en esta etapa, se centró en implementar y perfeccionar las funciones esenciales para el funcionamiento conjunto de estos drones. El servidor TCP desempeñó un papel como receptor de datos provenientes de la comunicación, especialmente puntos enviados desde una fuente externa. Estos puntos, que pueden representar una variedad de coordenadas, comandos o datos relevantes para el sistema, fueron procesados por la infraestructura.

Para lograr esto, se implementó un conjunto de comandos predefinidos basada en el framework Crazyswarm 2, los cuales se trabajó a lo largo de los diferentes capítulos de este trabajo. Este comportamiento se puede comprender mejor al revisar el Pseudocódigo 1, donde se detalla el proceso de mediante el cual se realiza una acción de acuerdo a los datos recibidos.

Algorithm 1 Procesamiento de puntos con la infraestructura

```
for cada punto en puntos3D do
    Realizar acciones con base en el punto recibido
    Obtener coordenadas x, y, z del punto
    RealizarAccionesSegunComando(comando, x, y, z)
end for

if comando es "despegar" then
    Ejecutar secuencia de despegue
else if comando es .ªterrizar" then
    Ejecutar secuencia de aterrizaje
else if comando es "mover" then
    Mover a (x, y, z)
end if
```

Además, se realizaron múltiples pruebas con varios drones, utilizando el servidor TCP como se muestra en la Figura 79 y 80. Estas corresponden a un elipsoide inclinado y un paraboloide hiperbólico, con los cuales se obtuvo un ECM de 0.0432 y 0.0223 respectivamente. Con esto, la integración permitió realizar diversas pruebas con los drones, demostrando su funcionamiento en conjunto sin conflictos y validando la solidez de la infraestructura que se desarrolló.

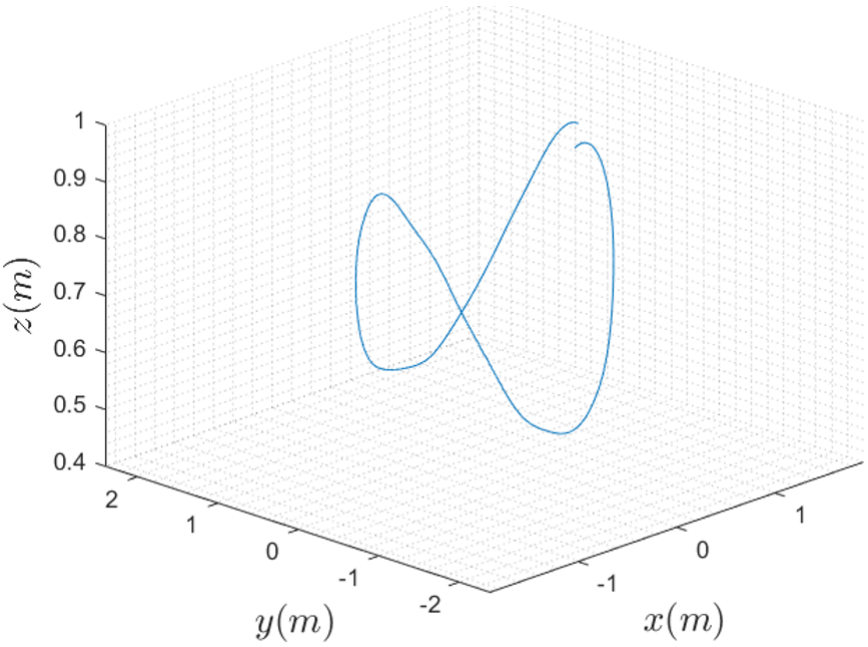


Figura 79: Paraboloide hiperbólico integrando Servidor TCP (Ávila J.)

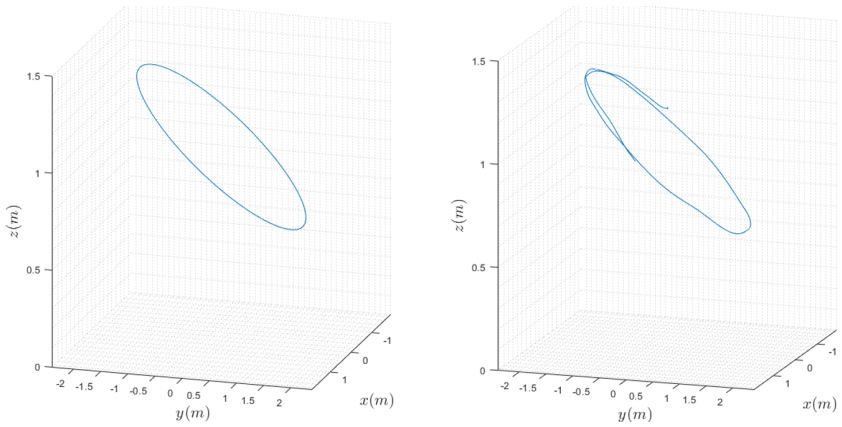


Figura 80: Elipsoide integrando Servidor TCP (Ávila J.)

CAPÍTULO 10

Conclusiones

- Se determinó que la combinación adecuada para la configuración del control y la estimacion para los drones Crazyflie 2.1 fue el estimador de Kalman y controlador Mellinger ya que con ellos se obtuvo mayor estabilidad y mejores resultados durante los experimentos.
- La implementación de nodos de comunicación de Crazyswarm 2 en ROS2 permitió desarrollar una plataforma de pruebas robusta permitió la comunicación con hasta 5 drones Crazyflie 2.1 realizando las trayectorias esperadas con un porcentaje de error menor a 3.5 %.
- La implementación de librerías de NatNet permitió la transmisión de datos desde OptiTrack a ROS 2 bajo una configuración *multicast*,que permitió utilizar el ecosistema Robotat a la par de la plataforma de pruebas siempre y cuando los marcadores reflectivos utilizados sean iguales es decir *single marker* o *rigid bodies*.

CAPÍTULO 11

Recomendaciones

- Desarrollar una interfaz que permita ejecutar de manera eficiente los comandos de inicialización de nodos y servicios, evitando la necesidad de ejecutar comandos individualmente en la consola, lo que puede resultar tedioso.
- Diseñar soportes que permitan una colocación más cómoda de los *headers*, la batería y los *markers* para evitar que estos se caigan.
- Integrar los distintos *decks* proporcionados por Bitcraze para mejorar la estabilidad y las funcionalidades de los drones, tanto dentro de la plataforma como en entornos exteriores.
- Llevar a cabo pruebas con marcadores de cuerpos rígidos para poder controlar tanto la orientación como la posición del dron.
- Implementar un sistema de redes en la plataforma para llevar a cabo pruebas con los Crazyflies y reducir los riesgos tanto para los drones como para las personas involucradas.

CAPÍTULO 12

Bibliografía

[1] M. Abdelkader, S. Güler, H. Jaleel y J. S. Shamma, "Aerial Swarms: Recent Applications and Challenges," *Current Robotics Reports*, vol. 2, n.º 3, págs. 309-320, sep. de 2021, ISSN: 2662-4087. DOI: 10.1007/s43154-021-00063-4. dirección: https://doi.org/10.1007/s43154-021-00063-4.

[2] F. J. Sanabria, "Diseño e implementación de una plataforma de pruebas para sistemas de control para el dron Crazyflie 2.0," Tesis doct., 2022.

[3] *UPenn's GRASP lab unleashes a swarm of Nano Quadrotors*, en-US, Section: Robotics, feb. de 2012. dirección: https://newatlas.com/grasp-nano-quadrotor-robots-swarm/21302/ (visitado 22-04-2023).

[4] J. A. Preiss, *Crazyswarm: A Powerful Framework for Aerial Swarms in Research and Education*, oct. de 2021.

[5] J. A. Preiss, W. Honig, G. S. Sukhatme y N. Ayanian, "Crazyswarm: A large nano-quadcopter swarm," en *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, págs. 3299-3304. DOI: 10.1109/ICRA.2017.7989376.

[6] F. Schilling, F. Schiano y D. Floreano, "Vision-Based Drone Flocking in Outdoor Environments," *IEEE Robotics and Automation Letters*, vol. 6, n.º 2, págs. 2954-2961, 2021. DOI: 10.1109/LRA.2021.3062298.

[7] *OFFensive Swarm-Enabled Tactics (OFFSET)*. dirección: https://www.darpa.mil/work-with-us/offensive-swarm-enabled-tactics (visitado 22-04-2023).

[8] X. Zhou, X. Wen, Z. Wang et al., "Swarm of micro flying robots in the wild," *Science Robotics*, vol. 7, n.º 66, eabm5954, 2022. DOI: 10.1126/scirobotics.abm5954. eprint: https://www.science.org/doi/pdf/10.1126/scirobotics.abm5954. dirección: https://www.science.org/doi/abs/10.1126/scirobotics.abm5954.

[9] *OptiTrack Documentation*, en. dirección: https://docs.optitrack.com/ (visitado 13-05-2023).

[10] *Positioning Systems Overview | Bitcraze*. dirección: https://www.bitcraze.io/documentation/system/positioning/ (visitado 21-05-2023).

[11] *Support - Prime x 41*, en. dirección: http://optitrack.com/support/hardware/primex-41.html (visitado 13-05-2023).

[12] J. Möllerström y M. Nyberg, "Emulation of the Crazyflie 2.1 Hardware; for Embedded Control System Testing," Tesis doct., Department of Automatic Control, 2021.

[13] *Getting started with the Crazyflie 2.X | Bitcraze*. dirección: https://www.bitcraze.io/documentation/tutorials/getting-started-with-crazyflie-2-x/ (visitado 13-05-2023).

[14] *Crazyradio 2.0 | Bitcraze*. dirección: https://www.bitcraze.io/products/crazyradio-2-0/ (visitado 20-05-2023).

[15] J. A. Preiss*, W. Hönig*, G. S. Sukhatme y N. Ayanian, "Crazyswarm: A large nano-quadcopter swarm," en *IEEE International Conference on Robotics and Automation (ICRA)*, Software available at https://github.com/USC-ACTLab/crazyswarm, IEEE, 2017, págs. 3299-3304. DOI: 10.1109/ICRA.2017.7989376. dirección: https://doi.org/10.1109/ICRA.2017.7989376.

[16] *Overview — Crazyswarm2 1.0a1 documentation*. dirección: https://imrclab.github.io/crazyswarm2/overview.html (visitado 21-05-2023).

[17] *CRTP - Communication with the Crazyflie | Bitcraze*. dirección: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/crtp/ (visitado 20-05-2023).

[18] "Guest Lecture at EPFL Aerial Robotics Course 2021 | Bitcraze," dirección: https://www.bitcraze.io/about/events/epfl2021/ (visitado 21-05-2023).

[19] D. Simon, *Optimal State Estimation: Kalman, H infinity, and nonlinear approaches*. John Wiley amp; Sons, Inc., 2006.

[20] *State estimation | Bitcraze*. dirección: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/sensor-to-control/state_estimators/#extended-kalman-filter (visitado 22-05-2023).

[21] Dirección: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/sensor-to-control/controllers/.

[22] D. Mellinger y V. Kumar, "Minimum snap trajectory generation and control for quadrotors," *2011 IEEE International Conference on Robotics and Automation*, 2011. DOI: 10.1109/icra.2011.5980409.

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

[24] Dirección: https://www.bitcraze.io/documentation/tutorials/getting-started-with-crazyflie-2-x/.

[25] Dirección: https://docs.ros.org/en/foxy/Installation/Ubuntu-Install-Debians.html.

CAPÍTULO 13

Anexos

13.1. Algoritmo para generar trayectoria estática con MAS

```
1
2  import numpy as np
3  import time
4  from crazyflie_py import Crazyswarm
5
6
7  def main():
8      # Inicializar la conexion con Crazyswarm
9      swarm = Crazyswarm()
10     cf = swarm.allcfs.crazyflies[0]
11     cf1 = swarm.allcfs.crazyflies[1]
12     cf2 = swarm.allcfs.crazyflies[2]
13
14     # Esperar a que los Crazyflies esten listos
15     timeHelper = swarm.timeHelper
16
17     cf.takeoff(targetHeight=0.5, duration=2.0)
18     cf1.takeoff(targetHeight=0.5, duration=2.0)
19     cf2.takeoff(targetHeight=0.5, duration=2.0)
20     timeHelper.sleep(2.5)
21
22     # Coordenadas del centro del circulo
23     center_x = 0.0
24     center_y = 0.0
25     center_z = 1.0
26
27     # Radio del circulo
28     radio = 0.8
29
30     # Numero de puntos en la trayectoria circulare
31     num_puntos = 20
32
33     # Generar puntos equidistantes en la trayectoria circular
34     waypoints = []
```

```
35
36      for i in range(num_puntos):
37          angle = i * 2.0 * np.pi / num_puntos
38          x = center_x + radio * np.cos(angle)
39          y = center_y + radio * np.sin(angle)
40          z = center_z
41
42          angle1 = (i * 2.0 * np.pi / num_puntos) + (2*np.pi/3)
43          x2 = center_x + radio * np.cos(angle1)
44          y2 = center_y + radio * np.sin(angle1)
45
46          angle1 = (i * 2.0 * np.pi / num_puntos) + (4*np.pi/3)
47          x3 = center_x + radio * np.cos(angle1)
48          y3 = center_y + radio * np.sin(angle1)
49
50          waypoints.append((x, y, z, x2, y2, x3, y3))
51
52      # Establecer la velocidad de vuelo
53      velocity = 0.2  # ajustar segun la velocidad deseada
54
55      # Ejecutar la trayectoria circular
56      while n<3:
57          n = n + 1
58          for waypoint in waypoints:
59           cf.goTo(goal=[waypoint[0],waypoint[1],waypoint[2]],yaw=0,
60           duration=2.0,relative=False)
61            cf1.goTo(goal=[waypoint[3],waypoint[4],waypoint[2]],yaw=0,
62            duration=2.0,relative=False)
63            cf2.goTo(goal=[waypoint[5],waypoint[6],waypoint[2]],yaw=0,
64            duration=2.0,relative=False)
65            timeHelper.sleep(1.0)
66
67      # Aterrizar los Crazyflies
68      cf.land(targetHeight=0.06, duration=2.0)
69      cf1.land(targetHeight=0.06, duration=2.0)
70      cf2.land(targetHeight=0.06, duration=2.0)
71      timeHelper.sleep(2.5)
72
73  if __name__ == "__main__":
74      main()
```

Código 13.1: Algoritmo Trayectoria Estática.

13.2. Algoritmo para generar trayectoria dinámica

```
1
2  #!/usr/bin/env python
3
4  import numpy as np
5  from pathlib import Path
6
7  from crazyflie_py import *
8  from crazyflie_py.uav_trajectory import Trajectory
9
10 def main():
11     swarm = Crazyswarm()
12     timeHelper = swarm.timeHelper
13     allcfs = swarm.allcfs
14
15     traj1 = Trajectory()
16     traj1.loadcsv(Path(__file__).parent / "data/figure8.csv")
17
18     TRIALS = 1
19     TIMESCALE = 1.0
20     for i in range(TRIALS):
21         for cf in allcfs.crazyflies:
22             cf.uploadTrajectory(0, 0, traj1)
23
24         allcfs.takeoff(targetHeight=1.0, duration=2.5)
25         timeHelper.sleep(2.5)
26         for cf in allcfs.crazyflies:
27             pos = np.array(cf.initialPosition) + np.array([0, 0, 1.2])
28             cf.goTo(pos, 0, 2.0)
29         timeHelper.sleep(2.5)
30
31         allcfs.startTrajectory(0, timescale=TIMESCALE)
32         timeHelper.sleep(traj1.duration * TIMESCALE + 2.0)
33
34         allcfs.land(targetHeight=0.06, duration=2.0)
35         timeHelper.sleep(3.0)
36
37 if __name__ == "__main__":
38     main()
```

Código 13.2: Algoritmo Trayectoria Dinámica.

13.3. Repositorio GitHub y Manual de usuario

Puedes encontrar las librerías que contienen el código y el manual de usuario en el siguiente repositorio de: GitHub

CAPÍTULO 14

Glosario

Estimador complementario: Un estimador complementario es un tipo de algoritmo utilizado en estimación y control para mejorar la precisión de las estimaciones al combinar múltiples fuentes de información o sensores de manera complementaria.

Estimador de Kalman: El estimador de Kalman es un método de estimación utilizado en control y procesamiento de señales para calcular de manera óptima el estado de un sistema dinámico a partir de una serie de mediciones, teniendo en cuenta el ruido y la incertidumbre en los datos.

Error cuadrático medio: El error cuadrático medio (ECM) es una métrica utilizada para evaluar la precisión de un modelo o estimación comparando los valores predichos con los valores reales. Se calcula como la media de las diferencias al cuadrado entre las predicciones y los valores verdaderos.

Mocap: Mocap es la abreviatura de *Motion Capture* en inglés, que se refiere a la captura de movimiento. Es una tecnología que se utiliza para registrar y seguir el movimiento de objetos o seres vivos en tiempo real, comúnmente utilizado en animación, videojuegos y biomecánica.

Socket: Es un componente de programación que se utiliza para establecer comunicación entre aplicaciones en una red. Permite que los programas envíen y reciban datos a través de una red, como Internet, utilizando protocolos de comunicación estándar.