Uso del cuadricóptero Crazyflie dentro del ecosistema Robotat para la generación de trayectorias y la evasión de obstáculos

Denny Steve Otzoy Chex

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Uso del cuadricóptero Crazyflie dentro del ecosistema Robotat para la generación de trayectorias y la evasión de obstáculos

Trabajo de graduación presentado por Denny Steve Otzoy Chex para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2023

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Uso del cuadricóptero Crazyflie dentro del ecosistema Robotat para la generación de trayectorias y la evasión de obstáculos

Trabajo de graduación presentado por Denny Steve Otzoy Chex para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2023

Vo.Bo.:

(f) _____

MSc. Miguel Zea

Tribunal Examinador:

(f) _____

MSc. Miguel Zea

(f) _____

Ing. Yosemite Meléndez

(f) _____

MSc. Pedro Iván Castillo

Fecha de aprobación: Guatemala, 4 de enero de 2023.

Prefacio

A Dios, fuente de tanta gracia en mi vida.

A mi familia. Apoyo, soporte y motivación de toda mi vida hasta ahora.

A quien con cariño conozco como doña Isabelita, que me regaló una oportunidad para soñar, divertirme aprendiendo y conocer a personas invaluables e inolvidables.

A mis mentoras, que brindaron sabiduría y crecimiento a mi vida académica y personal.

A las personas que conocí durante este trayecto y a las que me acompañaron hasta ahora, brindándome alegría, soporte y un fuerte deseo de aprender.

A todos ellos, ¡gracias por estar en mi vida!

Índice

Prefacio III

Lista de figuras VII

Resumen VIII

Abstract IX

1. Introducción 1

2. Antecedentes 2
- 2.1. Cuadricóptero Crazyflie 2.0 . . . 2
- 2.2. Diseño e implementación de una plataforma de pruebas para sistemas de control para el dron Crazyflie 2.0 . . . 2
- 2.3. Control del cuadricóptero Crazyflie usando Simulink . . . 3
- 2.4. Vuelo autónomo del cuadricóptero Crazyflie 2.1 a través de una pista de obstáculos . . . 3
- 2.5. Generación de trayectorias y control para cuadricópteros . . . 4
- 2.6. Robotat: Un ecosistema robótico de captura de movimiento y comunicación inalámbrica . . . 5
- 2.7. Otros trabajos integrados al Robotat . . . 5

3. Justificación 6

4. Objetivos 7
- 4.1. Objetivo general . . . 7
- 4.2. Objetivos específicos . . . 7

5. Alcance 8

6. Marco teórico 10
- 6.1. Cuadricóptero Crazyflie . . . 10
- 6.2. Características del dron . . . 11
- 6.3. Programación del firmware . . . 12

6.4. Orientación del dron . . . 12
6.5. Crazyradio PA . . . 12
6.6. Modelo del dron . . . 13
6.7. Generación de trayectorias . . . 14
6.8. Sistemas de captura de movimiento . . . 15
6.9. Sistema Optitrack . . . 16
6.10. NatNet . . . 17
6.11. Curvas de Bézier y el dron Crazyflie . . . 18

7. Conexión y pruebas de vuelo usando Crazyflie y Robotat 19
7.1. Metodología . . . 19
7.2. Ensamblaje . . . 19
7.3. Instalación de la máquina virtual . . . 20
7.4. Instalación en máquina física . . . 21
7.5. Pruebas de funcionamiento de los repositorios . . . 21
7.6. Prueba de vuelo . . . 22
7.7. Uso de la librería *Thread* . . . 22
7.8. Uso de la librería *Event* . . . 23
7.9. Resultados . . . 24

8. Obstáculos en el Optitrack 27
8.1. Metodología . . . 27
8.2. Identificación y uso de cuerpos rígidos en Motive . . . 27
8.3. Uso de librería *Socket* . . . 28
8.4. Conexión y procesamiento de datos recibidos . . . 29
8.5. Datos de posición enviados al dron . . . 30
8.6. Resultados . . . 31

9. Control del dron en una trayectoria 34
9.1. Metodología . . . 34
9.2. Creación de trayectorias en Python . . . 34
9.3. Uso de un marcador más preciso. . . . 35
9.4. Herramientas para ejecutar trayectorias en el dron . . . 36
9.5. Pruebas de las subfunciones del módulo *commander* . . . 37
9.6. Pruebas de control mediante el código principal . . . 38
9.7. Resultados . . . 39

10. Conclusiones 44

11. Recomendaciones 45

12. Bibliografía 46

13. Anexos 48
13.1. Programa de Python usando *Event* . . . 48
13.2. Conexión a red Robotat . . . 49
13.3. Enlace a lista de reproducción con los resultados . . . 49

14. Glosario 50

Lista de figuras

1. Plataforma de pruebas para dron Crazyflie [2] . . . 3
2. Dron trasportando una carga útil. [3] . . . 4
3. Distintas pruebas de adaptación de trayectoria drones [3] . . . 4
4. Vista frontal del ecosistema Robotat [6] . . . 5

5. Dron Crazyflie [1]. . . . 10
6. Sistema de coordenadas interno del dron [14]. . . . 12
7. Modulo Crazyradio [15]. . . . 13
8. Dron crazyflie [9] . . . 13
9. Diagrama de control de trayectorias [5]. . . . 15
10. Un dron en un Sistema de captura de movimiento óptico pasivo [21]. . . . 16
11. Cámaras Optitrack *Prime*$^{\times}$ *41* [23]. . . . 16
12. Ejemplo de código de librería NatNet en Python [6]. . . . 17

13. Partes del dron Crazyflie 2.0 (a) y su montaje final (b) [2]. . . . 20
14. Escritorio de la máquina virtual. . . . 20
15. Cliente del Crazyflie. . . . 21
16. Cliente conectado al dron. . . . 24
17. Salida de la consola del programa *basiclog.py*. . . . 24
18. Evolución de la ejecución del programa *ramp.py*. . . . 25
19. Evolución de la ejecución de la rutina de emergencia. . . . 26

20. Montaje de dron en el marcador. . . . 28
21. Comparativa del dron en la realidad (a) con la imagen virtual en Motive (b). 28
22. Dron en posición de prueba 1 (a) y datos del estimador interno (b). . . . 31
23. Dron en posición de prueba 2 (a) y datos del estimador interno (b). . . . 31
24. Dron en posición de prueba 3 (a) y datos del estimador interno (b). . . . 32
25. Dron en posición de prueba 4 (a) y datos del estimador interno (b). . . . 32
26. Control del dron mediante la información de orientación de un marcador secundario. . . . 33

27. Trayectorias obtenidas de las curvas de Bézier lineal (a) curva (b). . . . 35
28. Trayectorias obtenidas de las curvas de Bézier lineal (a) curva (b). . . . 35
29. Área confiable del marcador en el Robotat. . . . 36

30. Línea de tiempo de ejecución del código *autonomous_sequence_high_level.py*. 40
31. Recorrido del dron al ejecutar el subcomando *zdistance*. 40
32. Recorrido del don al ejecutar el código de control en una sola coordenada. . . 41
33. Recorrido del don al realizar control en una sola coordenada, partiendo desde el punto final del intento anterior. 41
34. Ejecución del código de control con dos coordenadas. 42
35. Ejecución del código de control para un punto cercano. 43
36. Ejecución de control de altura. 43

Resumen

Los cuadricópteros son una herramienta cada vez más utilizada en el mundo moderno, en distintos ámbitos, como investigación, transporte de productos y como herramienta multimedia. Y dada su versatilidad, es necesario involucrarse en esta investigación para mantenerse a la vanguardia. En este trabajo se presenta la investigación realizada en torno al cuadricóptero Crazyflie 2.0, y cómo se combinó esta herramienta con el sistema de captura de movimiento Optitrack, incluido en el ecosistema Robotat, dentro de la Universidad del Valle de Guatemala. Se realizaron pruebas con el dron, de modo que se pueda validar el funcionamiento de los programas instalados para controlarlo. Así mismo, se realizaron pruebas con el ecosistema Robotat, para definir cuerpos rígidos que posteriormente se emplearon para darle un sentido de posición al dron, y poder ejecutar experimentos basados en esa información. Se logró realizar una conexión adecuada entre una máquina virtual y una máquina física con el dron. También fue posible hacer un vuelo simple del dron, mediante la activación de los motores. Inclusive, se implementó un botón de emergencia que apaga el dron de forma forzada. En cuanto a la conexión con el ecosistema Robotat, se logró crear cuerpos rígidos, y recibir sus datos usando Python, para luego enviarlos al dron. Con esto, se logró que el dron estimara su posición mediante sus herramientas internas. También fue posible controlar el giro del dron empleando la información de un marcador secundario.

Abstract

Quadcopters are a tool increasingly used in the modern world, in different fields, such as research, product transport and as a multimedia tool. And given their versatility, it is necessary to get involved in this research to stay ahead of the curve. This paper presents the research conducted around the Crazyflie 2.0 quadcopter, and how this tool was combined with the Optitrack motion capture system, included in the Robotat ecosystem, within the Universidad del Valle de Guatemala. Tests were performed with the drone, in order to validate the operation of the programs installed to control it. Likewise, tests were performed with the Robotat ecosystem, to define rigid bodies that were later used to give a sense of position to the drone, and to run experiments based on that information. A proper connection between a virtual machine and a physical machine with the drone was achieved. It was also possible to make a simple flight of the drone, by activating the motors. Even, an emergency button was implemented to turn off the drone in a forced way. As for the connection with the Robotat ecosystem, it was possible to create rigid bodies and receive their data using Python, and then send them to the drone. With this, the drone was able to estimate its position using its internal tools. It was also possible to control the rotation of the drone using information from a secondary marker.

CAPÍTULO 1

Introducción

Los vehículos aéreos no tripulados son una herramienta empleada cada vez más en distintos sectores industriales, sociales e individuales, por lo que es normal encontrar más trabajos de investigación relacionados a este campo. Con este trabajo se pretende continuar con una serie de trabajos realizados anteriormente en la Universidad del Valle de Guatemala que involucran el uso de drones, en concreto, el dron Crazyflie 2.0 y del sistema de cámaras que forma parte del ecosistema Robotat:. Esta combinación pretende realizar un control de trayectorias para el dron y, de manera proyectiva, permitirá establecer bases para el control de drones y la elaboración de mejores experimentos de control de posición.

En el capítulo 7 se profundiza en el dron y sus dependencias, como las librerías para controlarlo, el cliente de Python y otras herramientas, algunas desarrolladas por Bitcraze, empleadas para facilitar el control del dron sin necesidad de controlarlo a alto nivel. Se amplía en el funcionamiento del cuadricóptero junto con estas herramientas y sobre las pruebas realizadas para recibir datos desde el dron y cómo hacer que este se eleve.

En el capítulo 8 se explica la conexión entre el Robotat y el dron, ampliando detalles sobre el software Motive, explicando el uso de las librerías *Socket* y *threading* en el programa principal, y cómo la información era actualizada en el dron. También se explica cómo se decodifica la información recibida desde el Robotat al emplear estas librerías y se realizan unas pruebas para validar la recepción de datos en el dron.

CAPÍTULO 2

Antecedentes

La integración de múltiples tecnologías ha abierto paso a la elaboración de experimentos cada vez más complejos, que se suelen realizar para probar límites y mejorar otras tecnologías o métodos existentes. En este caso particular y para facilitar la comprensión del trabajo, se abordan los proyectos realizados en torno al sistema de captura de movimiento Optitrack y al cuadricóptero Crazyflie 2.0, en distintas universidades y especialmente en la Universidad del Valle de Guatemala.

2.1. Cuadricóptero Crazyflie 2.0

En [1], se propone al dron Crazyflie como una alternativa útil para la educación e investigación. Comenzando por el hecho de que los drones comerciales tienen la limitación de ser de software propietario, y por tanto no se puede tener control de sus parámetros, los Crazyflie son drones de software abierto, que permiten un mejor control de sus propiedades. Además, cuentan con variedad de componentes para el control del dron, como el MPU-9250 y el Crazyradio PA, para facilitar el control remoto e interno. También tienen herramientas que facilitan el manejo del dron en múltiples plataformas y lenguajes de programación, como en C, Java, Matlab y Simulink.

2.2. Diseño e implementación de una plataforma de pruebas para sistemas de control para el dron Crazyflie 2.0

En [2], se continuó el trabajo sobre una plataforma de pruebas con un grado de libertad para drones, mostrado en la Figura 1 , para el cual se emplearon los drones Crazyflie 2.0 de la Universidad del Valle de Guatemala. Se generó una guía de uso de estos drones y de instalación de los recursos para su uso y control. También se desarrolló una librería de Python para facilitar el control y el proceso de datos. Esto junto con la interfaz

gráfica de Matlab, que permitió visualizar en tiempo real los datos recibidos del dron en la plataforma de pruebas. Adicionalmente, diseñó guías de laboratorio para los cursos de Sistemas de control 1 y 2 que utilizan tanto el dron como la plataforma de pruebas. Se logró obtener un programa funcional con el cual recolectó los datos del dron y generó una planta basada en las mediciones del dron. A su vez, el análisis de estos datos permitió generar las guías de laboratorio para los cursos mencionados. Estas guías fueron probadas por estudiantes de los cursos de Sistemas de control 1 y 2, de la Universidad del Valle de Guatemala.

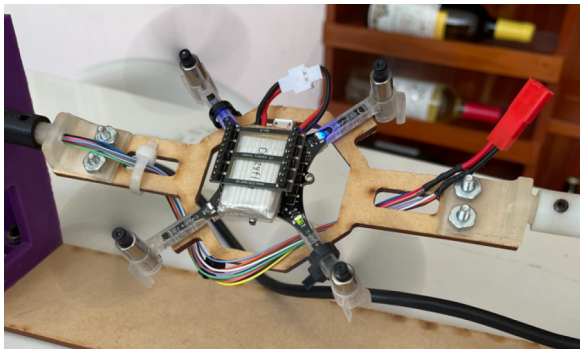


Figura 1: Plataforma de pruebas para dron Crazyflie [2]

.

2.3. Control del cuadricóptero Crazyflie usando Simulink

En [3], se desarrolló un modelo de Simulink empleado para el control de un dron Crazyflie, usando un modelo no lineal como aproximación. Con este trabajo se desarrolló una simulación muy precisa del comportamiento del dron, usando además controladores PID para vigilar el comportamiento de sus parámetros. Este programa fue usado para controlar tanto los parámetros simulados como los parámetros reales del dron, logrando con éxito estabilizar los tres ángulos de inclinación del dron a una posición estable, además de controlar su altura.

2.4. Vuelo autónomo del cuadricóptero Crazyflie 2.1 a través de una pista de obstáculos

En [4],se experimentó con el dron Crazyflie 2.1 para realizar un vuelo autónomo alrededor de un circuito de obstáculos en el menor tiempo posible. En estas pruebas, se usaron obstáculos previamente desconocidos, los cuales se detectaron mediante los sensores incorporados al dron. Además, se usaron los sistemas de posicionamiento Loco Positioning System y Flow Deck para proveer información adicional al dron. Con esto se logró que el dron recorriera exitosamente la mayoría de las trayectorias propuestas. Los desperfectos, como la precisión de los sensores de posición o la susceptibilidad a la luz del sol de los sensores incorporados, contribuyeron a un vuelo inestable e incluso a colisiones.

2.5. Generación de trayectorias y control para cuadricópteros

En [5] y las figuras 2 y 3, se presenta un trabajo completo sobre el control de cuadricópteros, presentado a lo largo de ocho capítulos. Se trabajó en la creación de controladores de alta velocidad, que permitió obtener latencias bajas en las respuestas de los motores y error menor al 5 % en posiciones y direcciones. También se trabajó un método para el transporte de carga por múltiples drones, con el cual se pudo generar 4 configuraciones usadas para cargar al menos 3 kg con una velocidad de 0.5 m/s. Más adelante, se abordó un método de estimación de parámetros para adaptación durante el vuelo con carga, el cual fue empleado para el transporte de masas a distintas posiciones, estimando en cada viaje el efecto de la carga transportada. Continua con un método de generación de trayectorias para espacios cerrados e inclinaciones, el cual funcionó con éxito para obstáculos complejos, superando una curva cerrada a través de una ventana estrecha, y posarse en ventanas con una inclinación cercana a los 90°. Completa el trabajo con la generación de trayectorias optimizadas y un método de reorganización de un enjambre de drones para la evasión de obstáculos.

Figura 2: Dron trasportando una carga útil. [3]

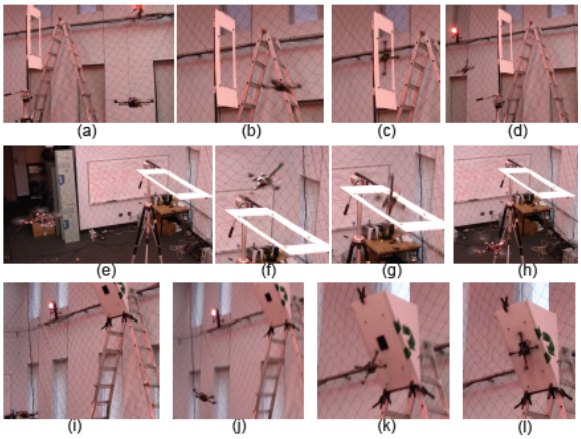


Figura 3: Distintas pruebas de adaptación de trayectoria drones [3]

.

2.6. Robotat: Un ecosistema robótico de captura de movimiento y comunicación inalámbrica

En [6], se explica el desarrollo de una red de comunicación WiFi que se usó en combinación con el sistema Optitrack, dando paso al sistema de experimentación robótica denominado como Robotat, observado en la Figura 4. Con dicho sistema, múltiples robots fueron capaces de conectarse al sistema y obtener los datos de sus poses. Además de este sistema, se desarrolló una librería en C que funciona en conjunto con un microcontrolador ESP32 para transferir los datos mediante el protocolo de red MQTT y el uso de un bróker, desde el cual se publicaban las poses de los robots. Adicional a esto, se diseñó una antena de comunicación WiFi, para que los diversos agentes robóticos se comunicaran con el sistema Optitrack. Con este proyecto fue posible enviar los datos a los distintos robots conectados a la red, y también determinar un límite máximo de 11 dispositivos conectados a la red.

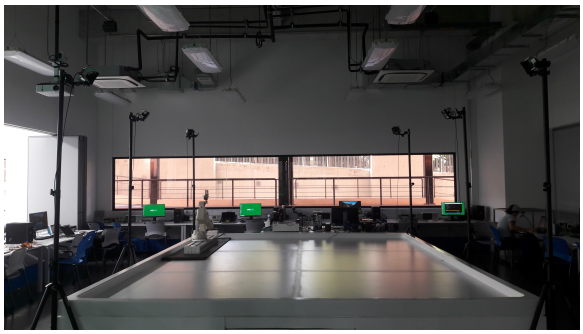


Figura 4: Vista frontal del ecosistema Robotat [6]

.

2.7. Otros trabajos integrados al Robotat

En [7], se desarrolló un controlador de cuadricóptero capaz de integrarse al ecosistema Robotat mediante la placa de desarrollo ESP32, mientras que en [8], se usó este controlador para la creación de un cuadricóptero y su posterior integración al Robotat mediante las herramientas desarrolladas anteriormente. La recepción de valores desde el Robotat hacia el cuadricóptero fue exitosa, obteniendo diferencias en la posición de un 5 %, pero presentó algunos fallos relacionados a la frecuencia de muestreo del microcontrolador y del sistema Optitrack, que se vieron reflejados en la velocidad de reacción del cuadricóptero.

CAPÍTULO 3

Justificación

Durante la experimentación realizada en [7] y [8], los proyectos que usaron el Robotat únicamente se centraron en lograr una conexión exitosa a un cuadricóptero y en cambiar la coordenada de rotación *yaw* de este. Mientras que en [2], los experimentos realizados con el dron Crazyflie mostraron su efectivo uso como herramienta didáctica mediante la generación de guías de laboratorio, limitando sus pruebas al uso de un grado de libertad en la plataforma de pruebas.

Los experimentos realizados en otras universidades [4] y [3] mostraron el alcance y las posibilidades del uso de drones en el sector educativo y de investigación, además de reforzar la versatilidad del uso de los drones Crazyflie en experimentos a gran escala, proponiéndose como una puerta al uso de drones de distintas maneras cotidianas.

En este trabajo se propone el uso de los drones Crazyflie dentro del ecosistema Robotat para la realización de las primeras pruebas de generación de trayectorias y evasión de obstáculos dentro de la Universidad del Valle de Guatemala. Considerando que con este sea posible expandirse a nuevos experimentos de mayor escala o complejidad en cuanto al control de posición de los drones y la integración de los drones Crazyflie al Robotat. También permitirá comenzar a nivel local la replicación de los resultados observados en otros trabajos y poder generar más y mejor conocimiento, para que luego sea posible emplearlos en mejoras a las herramientas educativas o de investigación disponibles en la universidad.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Usar el cuadricóptero Crazyflie dentro del ecosistema Robotat para ejecutar métodos de generación de trayectorias en un circuito de obstáculos.

4.2. Objetivos específicos

- Establecer conexión y hacer pruebas de vuelo del cuadricóptero Crazyflie dentro del ecosistema Robotat.
- Ejecutar métodos de generación de trayectorias usando el dron Crazyflie dentro del ecosistema Robotat.
- Crear obstáculos simples y reconocibles por el sistema de captura de movimiento Optitrack.

CAPÍTULO 5

Alcance

Este trabajo se centra principalmente en combinar las herramientas existentes para el control de vuelo del dron Crazyflie 2.0 y la información del sistema de cámaras Optitrack del ecosistema Robotat, para otorgarle sentido de posición y permitir vuelos controlados. Se emplearon diversas librerías de Python, algunas incorporadas y otras desarrolladas por Bitcraze, además de las cámaras del ecosistema Robotat y sus marcadores de posición. Dado que se están combinando dos herramientas por primera vez en la universidad, se descarta el control de trayectoria en vivo, es decir, que todas las trayectorias probadas han sido programadas en la memoria interna del dron o en el código principal.

Se descartó la manipulación de los parámetros internos del dron, dado que el objetivo del proyecto no es modificar el *firmware*. Parámetros como el Modulación de Ancho de pulso: de los controladores de motores o valores del giroscopio/acelerómetro no se analizaron, exceptuando los resultados de operaciones internas del dron que sí utilizan estos datos. Quedando como opción a futuro desarrollar proyectos que si los empleen, mejorando de alguna manera los resultados obtenidos.

Originalmente se planificó usar Matlab para la recepción de datos y el control del dron, pero se descartó dado que ya existían herramientas desarrolladas en Python que facilitaron el trabajo posterior. Además, la integración entre Matlab: y Motive Software: es lenta, al mismo tiempo que el uso de su librería de comunicación TCP por sockets.

Un factor que afectó al avance del proyecto fue la documentación de los drones Crazyflie, dado que al ser un sistema de software abierto y complejo el aprendizaje de las herramientas necesarias para controlarlo se torna lento y algunas partes de la documentación son muy generales, requiriendo realizar múltiples experimentos hasta comprender su funcionamiento. Además, la instalación de herramientas, como la máquina virtual o el cliente del dron, suelen depender de otras librerías de Python por lo que fue necesario hacer instalaciones y configuraciones adicionales antes de tener todo funcionando correctamente.

También hay que considerar que la universidad contaba con pocos drones y algunos tenían problemas de funcionamiento. Las nuevas unidades que solicitaron no llegaron

durante el tiempo de elaboración del trabajo.

Otro factor influyente fue la duración de las baterías incluidas con el dron, que brindaban 7 minutos máximo de vuelo del dron para luego esperar 40 minutos para volver a cargarse, por lo que fue necesario emplear un kit secundario de baterías para compensar estos tiempos de espera.

Finalmente, otra limitante fue el horario de uso del laboratorio. Debido a las interferencias que provoca la luz artificial con el sistema de captura Optitrack, únicamente era posible usar esta herramienta por las mañanas en la Universidad de modo que fue necesario realizar largos avances en código asumiendo su correcto funcionamiento, y la depuración constante también fue un motivo de atrasos.

CAPÍTULO 6

Marco teórico

6.1. Cuadricóptero Crazyflie

Un cuadricóptero o dron es un vehículo aéreo no tripulado usado para múltiples propósitos, que son hoy día una tecnología muy popular, aunque en muchos casos no tan accesible, ya sea por su costo o complejidad de uso. En este caso particular, el dron Crazyflie 5 es un cuadricóptero pensado como alternativa accesible y modificable entre el catálogo de drones disponibles en el mercado y orientado a investigación o para entretenimiento [1].

Figura 5: Dron Crazyflie [1].

La ventaja de este dron radica en sus características limitadas, entre ellas el hecho de ser de software abierto, por lo que cualquier usuario puede interactuar y modificar los

parámetros de control del dron mediante las herramientas de programación. Adicionalmente, la plataforma cuenta con hardware dedicado que facilita las comunicaciones y el control del dron, y también es posible añadir hardware adicional para incrementar su funcionalidad. Una lista detallada de las características del dron se encuentra en [9], y se muestran algunas en el siguiente título.

6.2. Características del dron

Dimensiones: 92 mm × 92 mm × 29 mm (Distancia de rotor a rotor).

Peso: 27 g.

Control y conexiones:

- STM32F405 Cortex-M4, 168MHz, 192kb SRAM, 1Mb flash como controlador principal.
- nRF51822 Cortex-M0, 32Mhz, 16kb SRAM, 128kb flash para gestión de radio y energía.
- Conector micro USB.
- Interfaz USB de alta velocidad.
- Capacidad parcial de USB OTG.
- Cargador de baterías de polímero de litio (LiPo) incorporado, modos de 100 mA, 500 mA y 980 mA.
- 8kB de EEPROM.

Radio:

- Banda ISG de 2.4 GHz.
- Amplificador de radio de 20 dBm, con un alcance de al menos 1 km, probado con Crazyflie PA.
- Bluetooth de baja energía, compatible con clientes iOS y Android.
- Retro compatible con los modelos originales Crazyflie Nano y Crazyradio.

Sensores principales:

- Unidad de medición inercial (*IMU* MPU-9250 y medidor de presión de precisión LPS25H.
- Giroscopio de 3 ejes.
- Acelerómetro de 3 ejes.
- Magnetómetro de 3 ejes.

Especificaciones de vuelo:

- Tiempo máximo de vuelo: 7 minutos.
- Tiempo de recarga: 40 minutos.
- Carga máxima recomendada: 15 g.

6.3. Programación del firmware

Para la programación del dron, se emplea una máquina virtual, que es un simulador de una computadora real y todos sus componentes, pero dentro de otro sistema físico. La máquina virtual empleada fue modificada por Bitcraze, documentada en [10]. Esta máquina virtual basada en Xubuntu posee repositorios de código abierto publicados en GitHub y contiene todos los recursos necesarios para la comunicación con el hardware del dron, como un entorno de desarrollo integrado (*IDE*), drivers para el módulo de radio usado para enviar datos al dron y el firmware de los dispositivos, tanto del dron como de la radio. Sanabria [2] explica su uso e instalación mediante una guía que desarrolló en su trabajo.

Por otro lado, estos recursos están disponibles para su descarga e instalación fuera de la máquina virtual. Mediante la descarga en GitHub, es posible la instalación de drivers y librerías para la programación del firmware y control del dron en una máquina física y en distintos sistemas operativos. [11] [12][13]

6.4. Orientación del dron

El dron tiene su propio sistema de coordenadas, que se ajusta automáticamente al encender el dron, mostrado en la Figura 6. Al colocar módulos externos o incluir sistemas de posicionamiento al dron, es necesario considerar la orientación del dron o modificar los datos recibidos para que los estimadores hagan cálculos correctos. Más información de esto se puede encontrar en la documentación del dron [14].

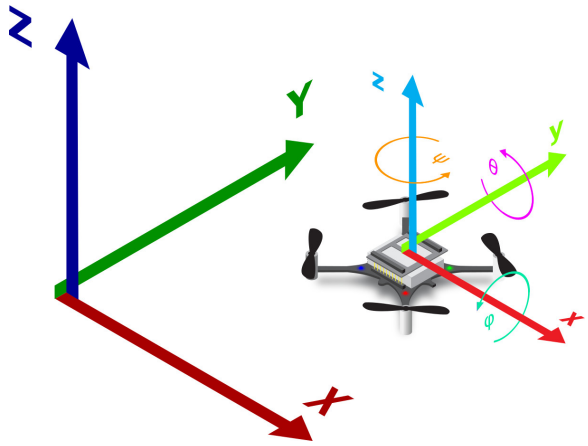


Figura 6: Sistema de coordenadas interno del dron [14].

6.5. Crazyradio PA

El módulo Crazyradio es un controlador de radio de software abierto creado por Bitcraze, mostrado en la Figura 7. Es utilizado para actualizar el firmware y enviar comandos a los drones para controlarlos o recibir parámetros. En combinación con el repositorio de

Python, este módulo permite enviar y recibir información del dron mediante el uso de sus direcciones, pudiendo controlar más de un dron en simultaneo. El uso de este módulo en Linux puede hacerse de forma directa, mientras que en Windows es necesario instalarlo usando un programa de terceros, de acuerdo con su documentación [11].

Figura 7: Modulo Crazyradio [15].

6.6. Modelo del dron

La caracterización del dron se ha abordado en el trabajo de Guerra [16], Mellinger [5] y Sanabria [2]. Este último compila y profundiza en detalles de estos trabajos y otros similares. Se parte de un cuadricóptero simplificado dentro de un marco de referencia inercial. Mediante el uso de los ángulos de Euler ZXY de alabeo, cabeceo y guiñada (*roll, pitch & yaw*), se puede describir la rotación del dron mostrada en la Figura 8.

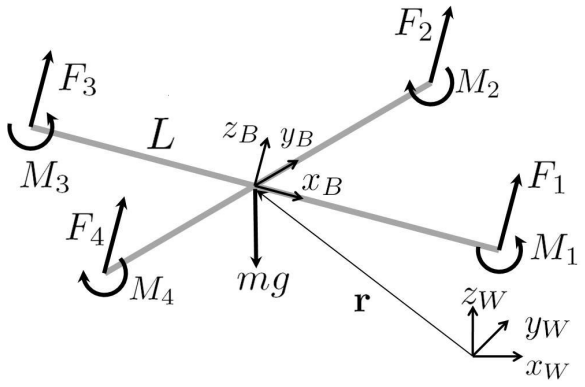


Figura 8: Dron crazyflie [9]

Obteniendo la matriz de rotación del cuerpo rígido,

$$\mathbf{R} = \begin{bmatrix} C_\psi C_\theta - S_\phi S_\psi S_\theta & -C_\phi S_\psi & C_\psi S_\theta + C_\theta S_\phi S_\psi \\ C_\theta S_\psi + C_\psi S_\phi S_\theta & C_\phi C_\psi & S_\psi S_\theta - C_\psi C_\theta S_\phi \\ -C_\phi S_\theta & S_\phi & C_\phi C_\theta \end{bmatrix}.$$

Luego, se hallan las ecuaciones que gobiernan la aceleración del centro de masa, donde r representa el vector de posición del centro de masa del cuadricóptero respecto al marco de referencia inercial, F_i las fuerzas que influyen en el dron, y m su masa,

$$m\ddot{\mathbf{r}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ \sum F_i \end{bmatrix}.$$

Las componentes de velocidad angular en el marco de referencia del dron son mapeadas desde el marco de referencia inercial usando una matriz de transformación.

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} C_\theta & 0 & -C_\phi S_\theta \\ 0 & 1 & S_\phi \\ S_\theta & 0 & C_\phi C_\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}.$$

También es posible obtener las velocidades angulares respecto al marco inercial transformando las velocidades angulares del dron [17]:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & S_\phi t_\theta & C_\phi t_\theta \\ 0 & C_\phi & -S_\phi \\ 0 & \frac{S_\phi}{C_\theta} & \frac{C_\phi}{C_\theta} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}.$$

Además, las fuerzas de los motores provocan un momento de rotación en distintas direcciones, aunque cada par de motores genera un momento opuesto al otro par. El efecto de estos momentos se considera en la ecuación para la aceleración angular.

$$I \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} L(F_2 - F_4) \\ L(F_3 - F_1) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times I \begin{bmatrix} p \\ q \\ r \end{bmatrix}.$$

6.7. Generación de trayectorias

Para que un dron pueda movilizarse en su ambiente debe obtener información del entorno para poder decidir a donde moverse. Esta información puede ser provista, o bien por un usuario interactuando con el dispositivo, o bien mediante sensores incorporados al dron, o bien con un sistema de captura de movimiento [5]. Incluso es posible proveer un archivo con coordenadas relativas a un sistema de posicionamiento [4]. Sin esta información, el dron únicamente permanecerá estático en el aire o se desplazará sin control hasta colisionar con un obstáculo. Aunque incluso con la información necesaria, dependerá de la calidad de los datos entregados al dron o de la precisión del operador el que el dron no colisione.

Para no depender de un operario en todo momento, ni de información estática, se pueden implementar algoritmos de control en el dron, que se encarguen de manejar los datos para dirigirlo a donde sea necesario en todo momento. Esto se logra aplicando control en el dron a distintos parámetros, que en concreto son:

- Control de altitud.
- Control de desplazamiento.
- Control de ruta tridimensional.

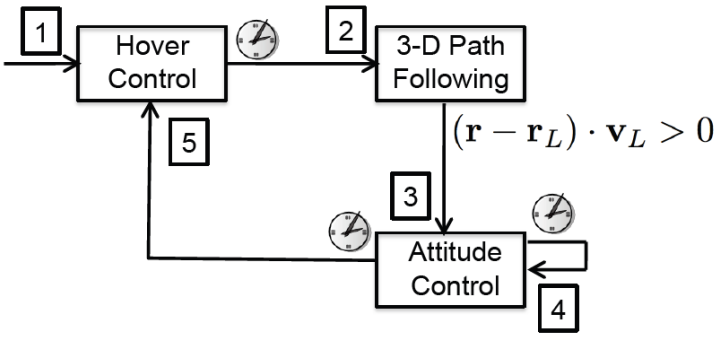


Figura 9: Diagrama de control de trayectorias [5].

Mellinger [5] propone un método de control de trayectorias agresivo para el dron, mediante la generación de secuencias. Esto consiste en usar un distinto modelo de control para distintos eventos del dispositivo. Se cita como ejemplo el uso de un control más eficaz para un modo de suspensión (Hover) luego de salir de un modo de vuelo hacia una nueva posición. Esto implica una adaptación dinámica de los parámetros de control del dron, que se estiman durante el vuelo. El diagrama de control se muestra en la siguiente figura. Cabe mencionar que este método de control fue implementado en el dron Crazyflie, y se puede acceder a el mediante la programación de parámetros, y que luego se actualiza recibiendo parámetros y mediciones de se nsores.

6.8. Sistemas de captura de movimiento

Estos sistemas se encargan de registrar el movimiento que realiza un elemento dentro de su rango de captura, para luego procesar esta información, ya sea en tiempo real o con otras técnicas fuera de linea. Estos datos posteriormente pueden ser usados en una variedad de actividades, como videos musicales [18], películas [19], e investigación [5] [20].

Las técnicas de captura de movimiento más usadas son:

- Captura óptica activa.
- Captura óptica pasiva.
- Captura sin marcadores.
- Captura inercial.

Figura 10: Un dron en un Sistema de captura de movimiento óptico pasivo [21].

Para las técnicas ópticas se emplean marcadores de dos tipos, retroreflectivos para la captura óptica pasiva, y emisores de luz para la captura óptica activa. En ambos casos, se detecta la luz que emite o refleja cada uno de los marcadores en el espacio de trabajo de las cámaras del sistema, y luego se calcula la posición de este en el espacio tridimensional. Por otro lado, las técnicas sin marcadores emplean sistemas de análisis de video junto con cámaras capaces de detectar profundidad. Y las técnicas de captura inercial emplean unidades de medición inercial (*IMUs*) para calcular movimiento y posición de los objetos.

6.9. Sistema Optitrack

La compañía Optitrack ofrece productos relacionados a la captura de movimiento. La Universidad del valle de Guatemala cuenta con 6 cámaras *Prime*$^{\times}$ *41*, mostradas en la Figura 11, y el sistema *Motive*, adquiridos de esta empresa [22]. Para registrar el movimiento de un objeto, los sistemas de captura de movimiento generalmente utilizan múltiples cámaras colocadas en diferentes poses, de modo que brinden información significativa del objeto en el espacio. También emplean marcadores reflectivos que son observados a través de las cámaras como puntos en el espacio, con los que se construye un cuerpo en el sistema, aunque puede funcionar con marcadores que emiten luz.

Estas cámaras requieren una calibración previa a su uso, para asegurar la mayor precisión posible. Para ello se realiza un abanicado con una herramienta de calibración de dimensiones concretas en el área de trabajo de las cámaras. Durante la calibración, el software almacena la información de lo observado por las cámaras y calibra automáticamente las cámaras.

Los detalles del uso de todas las herramientas anteriores en el ecosistema Robotat se amplían en [6].

Figura 11: Cámaras Optitrack *Prime*$^{\times}$ *41* [23].

Las cámaras *Prime*$^{\times}$ *41* cuentan con las siguientes especificaciones técnicas.

- **Dimensiones:** 12.6 cm × 12.6 cm × 13.2 cm
- **Peso:** 13.2 g
- **Resolución:** 2048 × 2048
- **Velocidad de fotogramas:** 180 Hz
- **Latencia:** 5.5 ms
- **Precisión 3D:** +/- 0.10 mm
- **Rango de rastreo de marcadores activos:** 45 m
- **Rango de rastreo de marcadores pasivos:** 30 m
- **Aro Led:** 20 leds infrarrojos de 850 nm

6.10. NatNet

Este es un sistema empleado por el software Motive. Es una arquitectura de software empleada para la transferencia de información en tiempo real, siendo capaz de funcionar en distintas interfaces de programación de aplicaciones (APIs) y protocolos de red [24]. Esta librería se emplea en el ecosistema Robotat para la transferencia de información mediante su red de comunicaciones, funcionando en Python [6].

El componente principal de NatNet es la librería *Socket* de Python. Esta se encarga de establecer una conexión de red mediante una dirección IP y un puerto en la red.

```python
# Uses the Python NatNetClient.py library to establish a connection
# (by creating a NatNetClient), and receive data via a NatNet connection
#and decode it using the NatNetClient library.
from NatNetClient import NatNetClient

# This is a callback function that gets connected to the NatNet client and
#called once per mocap frame.
def receiveNewFrame( frameNumber, markerSetCount, unlabeledMarkersCount,
                     rigidBodyCount, skeletonCount, labeledMarkerCount,
                     timecode, timecodeSub, timestamp, isRecording,
                     trackedModelsChanged ):
    print( "Received frame", frameNumber )

# This is a callback function that gets connected to the NatNet client. It is
#called once per rigid body per frame
def receiveRigidBodyFrame( id, position, rotation ):
    print( "Received frame for rigid body", id )
    print( "Received position for rigid body", position )
    print( "Received rotation for rigid body", rotation )

# This will create a new NatNet client
streamingClient = NatNetClient()

# Configure the streaming client to call our rigid body handler on the
#emulator to send data out.
streamingClient.newFrameListener = receiveNewFrame
streamingClient.rigidBodyListener = receiveRigidBodyFrame

# Start up the streaming client now that the callbacks are set up.
# This will run perpetually, and operate on a separate thread.
streamingClient.run()
```

Figura 12: Ejemplo de código de librería NatNet en Python [6].

6.11. Curvas de Bézier y el dron Crazyflie

Creadas por el ingeniero del mismo apellido, estas son una forma de curvas paramétricas creadas originalmente para el diseño de automóviles y empleadas en un gran número de programas de diseño de gráficos por computadora. Estas permiten la representación de curvas del mundo real que de otra manera no podrían ser representadas matemáticamente, o serían muy complejas. Entre otras aplicaciones, estas curvas pueden representar la velocidad con que se mueve un objeto de un punto A hacia un punto B [25].

El dron Crazyflie emplea este tipo de curvas para el almacenamiento de trayectorias en su memoria interna, para lo cual requiere dar formato a los puntos que se encuentran en sus trayectorias. Sin embargo, se cuenta con la limitante de que la memoria para trayectorias es de 4 kB, y cada punto de la trayectoria ocupa 132 bytes: un dato de tipo flotante para la duración de la trayectoria y otros 8 datos del mismo tipo por cada coordenada y la guiñada. Por tanto, el máximo almacenable son 31 segmentos de trayectoria [26].

CAPÍTULO 7

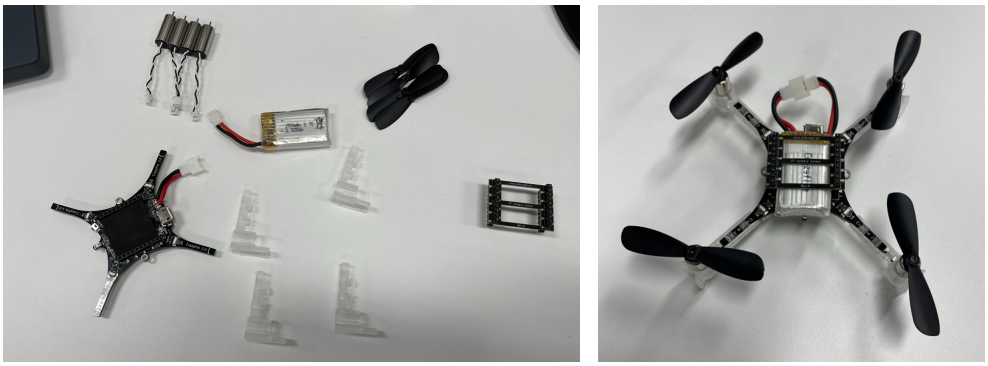
Conexión y pruebas de vuelo usando Crazyflie y Robotat

7.1. Metodología

Se ensambló el dron como fase inicial siguiendo las recomendaciones de trabajos previos y los creadores del dron. También se instaló la máquina virtual, versión de Sanabria, y se instalaron los paquetes y repositorios directamente en una máquina física. Posterior a ello se modificaron y probaron los programas de ejemplo *basiclog.py* y *Ramp.py* incluidos en el repositorio *crazyfie-lib-python*, los cuales se emplearon para para validar la instalación de los repositorios y el funcionamiento de la radio con el dron. También se emplearon para testear la conexión con el dron y realizar una prueba de vuelo simple. Se empleó también la librería *threading* de Python para permitir el funcionamiento en simultaneo del sistema de captura de movimiento, los comandos enviados al dron mediante la antena de radio y la implementación de un botón de emergencia.

7.2. Ensamblaje

La caja del dron contiene un juego de piezas y repuestos necesarios para montar el dron, que se pueden observar en la Figura 13a. Se empezó conectando la placa por USB a la computadora. Las luces de la placa indicaron que el dron está encendido y que todo está en orden, de acuerdo con las guías [2] [9]. Se ensamblaron los motores en sus retenedores plásticos y se trenzaron los cables, para luego montarlos a la placa base del dron y conectarlos a sus puertos correspondientes. Finalmente se insertaron las aspas en los motores, cuidando la colocación de las aspas A en el espacio correcto. El resultado de este ensamblaje se muestra en la Figura 13b



(a) (b)

Figura 13: Partes del dron Crazyflie 2.0 (a) y su montaje final (b) [2].

7.3. Instalación de la máquina virtual

La máquina virtual de Bitcraze posee todas las dependencias y repositorios necesarios para controlar el dron y actualizar el firmware de los dispositivos. La versión oficial, de acuerdo con su repositorio [27] depende de la aplicación Packer para crear una imagen de la máquina virtual, aunque su creación debe ser en un sistema Linux. La versión de Sanabria es la misma máquina virtual más algunas librerías de Python instaladas previamente, por lo que la diferencia es insignificante. Tanto la versión de Sanabria como la oficial dependen del uso un software que permita hacer funcionar una máquina virtual. En este caso se empleó Oracle VirtualBox, que se puede descargar desde su página oficial [28].

Al tener la imagen de la máquina virtual, se importó a VirtualBox y se ejecutó para verificar su funcionamiento y dependencias. Como la máquina tenía la mayoría de elementos ya instalados, solo fue necesario actualizar algunas librerías de Python para hacerla funcionar completamente, como se ve en la Figura 14.

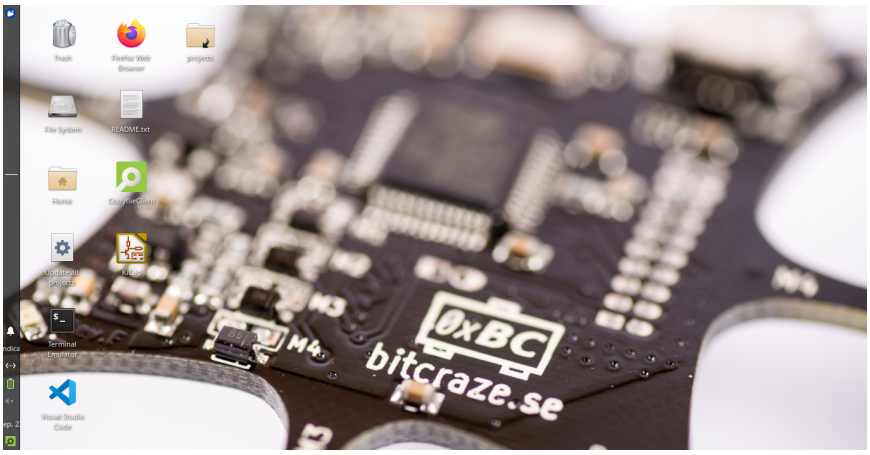


Figura 14: Escritorio de la máquina virtual.

7.4. Instalación en máquina física

Este proyecto se ha trabajado en una computadora personal con Windows 10, en donde también se instaló la máquina virtual. Durante algunas pruebas de uso de la radio en la máquina virtual se detectó que la velocidad de funcionamiento podría ser un problema para las siguientes etapas del trabajo, por lo que se decidió realizar la instalación de los repositorios y dependencias necesarias para operar el dron en la máquina física.

Para esto se descargaron los repositorios crazyflie-lib-python y crazyflie-clients-python de sus respectivos repositorios [12] [13] y se instalaron acorde a sus instrucciones. Además, se instaló el módulo de radio utilizando la herramienta Zadig, de la cual se habla en la documentación del módulo de radio [11]. Una vez instalados, se instaló Visual Studio Code como entorno de desarrollo, dado que es el mismo que se encuentra en la máquina virtual, aunque con cualquier otro editor habría funcionado también. Finalmente, se instalaron algunas librerías de Python necesarias para hacer funcionar ambos repositorios. El cliente funcionando en Windows 10 se muestra en la Figura 15.

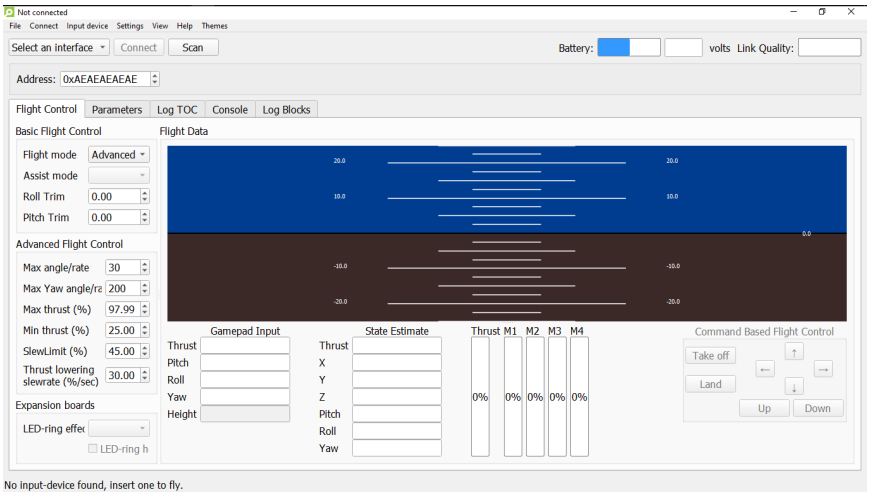


Figura 15: Cliente del Crazyflie.

7.5. Pruebas de funcionamiento de los repositorios

Para probar la instalación de los repositorios en ambos sistemas se probó la conexión con el dron mediante el cliente. Este requiere escanear con la radio los entornos para hallar señales de un dron encendido. Al conectar la radio y asignarle la dirección correcta, el programa fue capaz de encontrar la señal del dron y conectarse a ella. Adicionalmente, el dron puede conectarse directamente mediante la dirección de radio, ancho de bits y canal adecuados. Más información de esto puede encontrarse en la documentación [2] [13].

Como segunda prueba, se ejecutó el programa *basiclog.py*, disponible en los ejemplos del repositorio *crazyflie-lib-python*. Este código activa el registro de datos, también llamado *Log:*, de un estimador de estabilización programado en el dron y entrega la información a la radio en un intervalo recurrente de 100 milisegundos, para luego imprimir la informa-

ción recibida en la consola durante una ventana de tiempo de 10 segundos.

Es importante destacar que ocurría un pequeño retraso de la información recibida en la máquina virtual, aunque bien podría deberse a la cantidad de recursos con los que funciona. Por ello se decidió emplear la máquina física para las siguientes etapas, y usar la máquina virtual solo en caso de actualizar el firmware o hacer pruebas. Los resultados de estos experimentos se muestran bajo el título respectivo.

7.6. Prueba de vuelo

Para verificar que el dron funcione correctamente, se empleó el programa *ramp.py*. Este programa ejecuta una rutina de control del dron para generar empuje vertical suficiente para elevar el dron, y que este vuele con una inclinación de 30 grados hacia adelante y hacia atrás, para terminar en el punto de inicio. Se modificó este programa para generar un movimiento sin inclinación con el suficiente empuje vertical, de modo que el dron se eleve momentáneamente y luego aterrice. Se ejecutó este programa en la máquina física y la máquina virtual.

La velocidad de envío de datos no fue un problema, debido a la simpleza del código. Sin embargo, se detectaron errores en la ejecución de hilos de trabajo en ambas máquinas. Este problema está relacionado al funcionamiento de la librería *Thread*, pero como no ocurría con frecuencia, se decidió simplemente reiniciar el programa cuando fuera necesario. Sin embargo, resultó de mucha utilidad el uso de un hilo de trabajo auxiliar que sirviera como botón de emergencia, en caso de que el dron se comportara de una manera inesperada.

7.7. Uso de la librería *Thread*

Esta librería permite la ejecución de múltiples hilos de trabajo, empleados para arrancar varias funciones en simultaneo. Esta es parte de la librería *threading*, cuya documentación se puede encontrar en línea [29]. La creación de un objeto *Thread* se realiza como muestra el siguiente código:

```
from threading import Thread
def newThread(args):
    ...
...
tr = Thread(target = function, args = (args))
```

Donde `newThread` es una función que se desea añadir al nuevo objeto *Thread*, llamado `tr`, y `args` son los argumentos que requiere la función para operar. Luego, se puede ejecutar el hilo mediante la función:

```
tr.start()
```

Con esto el nuevo hilo se activará y estará ejecutándose hasta que la función termine to-

das sus operaciones. Se puede usar esta librería para darle sus propiedades a una clase. La creación del objeto y la ejecución se deben realizar de forma diferente.

```
from threading import Thread
class threadClass(Thread):
    def __init__(self):
        Thread.__init__(self)
        self.start()

    def run(self):
        # Codigo de la funcion
```

Donde la clase `threadClass` recibe las propiedades del objeto *Thread*. El nuevo hilo creado activará el código en la función `run()` de la clase `threadClass`, cuando se cree un objeto derivado de esta clase.

Esta librería se empleó en esta etapa para la creación de un botón de emergencia, que al presionar la tecla espacio ejecuta una rutina para apagar los motores del dron, cortar la conexión de radio y terminar el programa en ejecución. En las otras etapas del proyecto se empleó la librería para el envío constante de datos al dron y para controlar la conexión al Robotat.

7.8. Uso de la librería *Event*

La librería Event permite la creación de un objeto encargado de gestionar diversos eventos dentro del programa. También se utiliza para controlar pausas dentro de los hilos de trabajo. La documentación de esta librería también es parte de la librería *threading* [29].

Se puede definir un objeto *Event* y controlar un intervalo de tiempo como se muestra a continuación:

```
from threading import Event

event = Event()
event.clear()
event.wait(1)
```

El hilo actual se detiene por un segundo, y este objeto puede ser usado en múltiples hilos, debido a que posee una bandera interna, que se puede usar para controlar si el objeto de eventos sigue activo. Para ver un ejemplo de uso, se recomienda revisar los anexos.

Esta librería se empleó también para la creación del botón de emergencia y como temporizador. Esto debido a que dentro de los hilos no se puede usar cualquier tipo de temporizador, o de otro modo se corre el riesgo de paralizar todos los hilos de trabajo al mismo tiempo, ocasionando que hilos importantes, como el de control del dron, deje de funcionar y el dron se estrelle.

7.9. Resultados

De las pruebas de repositorios, se consiguió la conexión del dron con el cliente, mostrado en la Figura 16.Al conectarse, fue posible recibir todos sus parámetros, datos de registro y se pudo observar visualmente la información del acelerómetro. Esta prueba también fue realizada en la máquina virtual para validar que los repositorios funcionaran correctamente, logrando el mismo resultado.

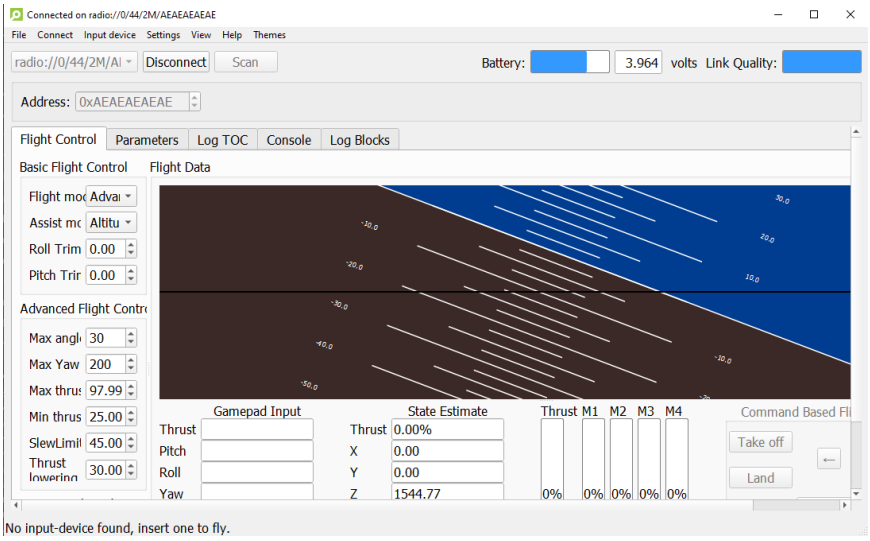


Figura 16: Cliente conectado al dron.

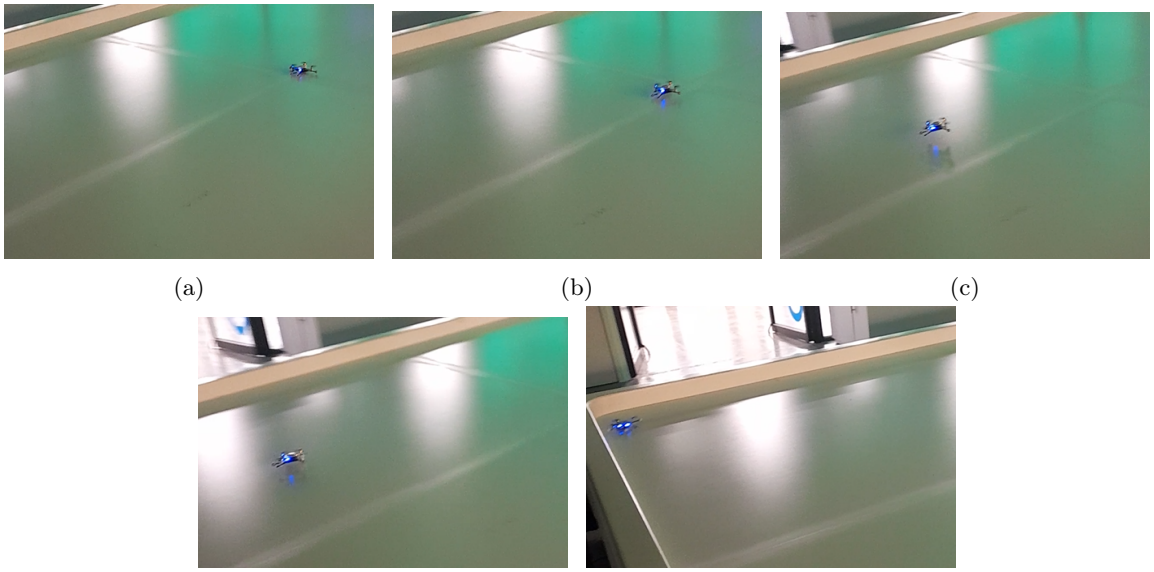
En el caso de la recepción de datos, tanto en la máquina virtual como en la física se recibieron los datos del estabilizador de forma constante. La Figura 17 muestra la recepción de datos en la consola del programa.

```
PS C:\Users\SteveChex> & "C:/Program Files/Python39/python.exe" "c:/Users/SteveChex/VirtualBox VMs/SharedFolders/Bitcraze/MainControl/crazyflie-lib-python/examples/basiclog.py"
Connecting to radio://0/44/2M/AEAEAEAEAE
Connected to radio://0/44/2M/AEAEAEAEAE
[122374][Stabilizer]: stateEstimate.x: 0.000 stateEstimate.y: 0.000 stateEstimate.z: 1502.392 stabilizer.roll: 0.330 stabilizer.pitch: -1.451 stabilizer.yaw: 2.650
[122474][Stabilizer]: stateEstimate.x: 0.000 stateEstimate.y: 0.000 stateEstimate.z: 1502.284 stabilizer.roll: 0.333 stabilizer.pitch: -1.454 stabilizer.yaw: 2.654
[122574][Stabilizer]: stateEstimate.x: 0.000 stateEstimate.y: 0.000 stateEstimate.z: 1502.237 stabilizer.roll: 0.333 stabilizer.pitch: -1.454 stabilizer.yaw: 2.660
[122674][Stabilizer]: stateEstimate.x: 0.000 stateEstimate.y: 0.000 stateEstimate.z: 1502.290 stabilizer.roll: 0.330 stabilizer.pitch: -1.455 stabilizer.yaw: 2.667
[122774][Stabilizer]: stateEstimate.x: 0.000 stateEstimate.y: 0.000 stateEstimate.z: 1502.362 stabilizer.roll: 0.330 stabilizer.pitch: -1.456 stabilizer.yaw: 2.673
[122874][Stabilizer]: stateEstimate.x: 0.000 stateEstimate.y: 0.000 stateEstimate.z: 1502.411 stabilizer.roll: 0.329 stabilizer.pitch: -1.459 stabilizer.yaw: 2.679
[122974][Stabilizer]: stateEstimate.x: 0.000 stateEstimate.y: 0.000 stateEstimate.z: 1502.451 stabilizer.roll: 0.327 stabilizer.pitch: -1.460 stabilizer.yaw: 2.685
[123074][Stabilizer]: stateEstimate.x: 0.000 stateEstimate.y: 0.000 stateEstimate.z: 1502.484 stabilizer.roll: 0.328 stabilizer.pitch: -1.458 stabilizer.yaw: 2.692
[123174][Stabilizer]: stateEstimate.x: 0.000 stateEstimate.y: 0.000 stateEstimate.z: 1502.400 stabilizer.roll: 0.328 stabilizer.pitch: -1.459 stabilizer.yaw: 2.694
[123274][Stabilizer]: stateEstimate.x: 0.000 stateEstimate.y: 0.000 stateEstimate.z: 1502.293 stabilizer.roll: 0.325 stabilizer.pitch: -1.460 stabilizer.yaw: 2.701
[123374][Stabilizer]: stateEstimate.x: 0.000 stateEstimate.y: 0.000 stateEstimate.z: 1502.247 stabilizer.roll: 0.326 stabilizer.pitch: -1.458 stabilizer.yaw: 2.707
[123474][Stabilizer]: stateEstimate.x: 0.000 stateEstimate.y: 0.000 stateEstimate.z: 1502.257 stabilizer.roll: 0.326 stabilizer.pitch: -1.461 stabilizer.yaw: 2.713
```

Figura 17: Salida de la consola del programa *basiclog.py*.

Fue posible, tanto al usar la máquina virtual como la máquina física, lograr que el dron se elevara y volara por un momento, mediante el uso del programa *ramp.py*, pero al no estar controlando la posición del dron en el espacio, este se desplazaba ligeramente en dirección horizontal durante la prueba. La Figura 18 muestra una evolución descriptiva de este comportamiento.

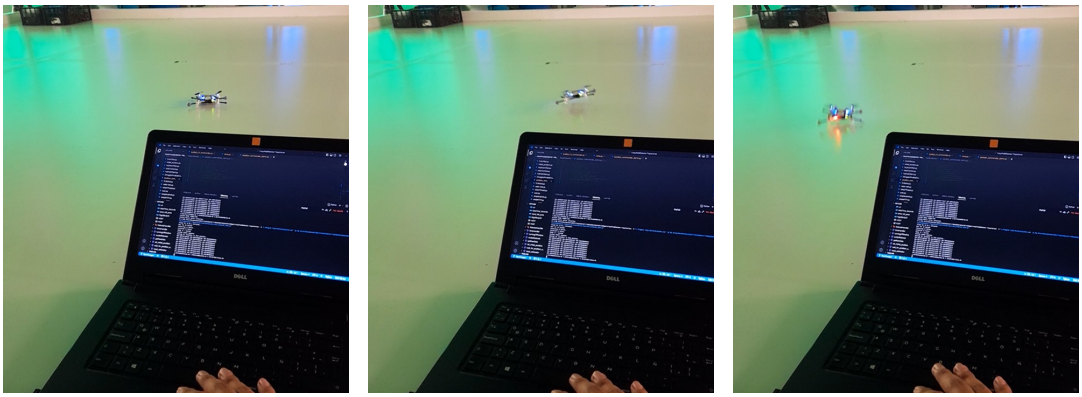
También se logró implementar el botón de emergencia y se probó con un código derivado del ejemplo *ramp.py*. La Figura 19 muestra la evolución de la ejecución de este código, pulsando el botón de emergencia entre el intervalo de tiempo que ocurre en las figuras 19c y 19d, La Figura 19e muestra la salida de la consola al presionar el botón.



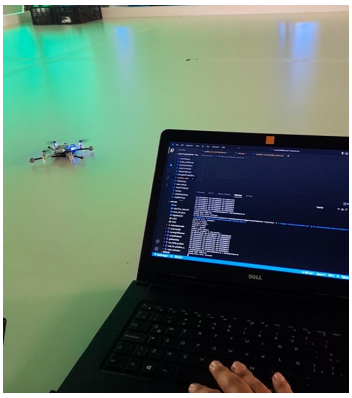
(a) (b) (c)

(d) (e)

Figura 18: Evolución de la ejecución del programa *ramp.py*.



(a) (b) (c)



(d)

```
>
Thread for no.2 Initiated…
Wait for 10 seconds
Connecting to Motive…
Connected to Motive
Ready in 3 seconds
Running Simple Sequence
Waiting for estimator to find position…
999.9997447204369 999.9997446883353 999.999650596515
999.9997447204369 999.9997446883353 999.999650596515
999.999765181783 999.9997651574522 999.9997004003671
999.999765181783 999.9997651574522 999.9997004003671
999.999765181783 999.9997651574522 999.9997004003671
999.999765181783 999.9997651574522 999.9997004003671
999.999765181783 999.9997651574522 999.9997004003671
999.999765181783 999.9997651574522 999.9997004003671
999.999765181783 999.9997651574522 999.9997004003671
5.047184822615236e-05 5.046943260822445e-05 9.765307186171412e-05
CLOSING DRONE CONNECTION
Closing Motive server connection
>
```

(e)

Figura 19: Evolución de la ejecución de la rutina de emergencia.

CAPÍTULO 8

Obstáculos en el Optitrack

8.1. Metodología

Se definieron cuerpos rígidos en Motive usando elementos plásticos que sujetaban marcadores reflectivos. Los datos de estos cuerpos rígidos eran transmitidos a la red local del Robotat constantemente. Se implementó, mediante Python y la librería *Socket*, una conexión de red entre la computadora y la red del Robotat, para obtener la información de posición enviada por Motive, para lo cual se empleó una dirección IP y un puerto específicos en la red. Los datos recibidos en Python fueron procesados posteriormente para obtener valores numéricos en formato de punto flotante, los cuales correspondían a la pose del cuerpo rígido registrado en el sistema. Se empleó el programa modificado *qualisis_hl_commander.py* para crear una clase que funcione con la librería *threading* para enviar la información de la posición del dron mediante un hilo paralelo al programa principal, al tiempo que se recibía la información del estimador interno del dron. También se hizo girar al dron con base en la información de otro cuerpo rígido.

8.2. Identificación y uso de cuerpos rígidos en Motive

La calibración de las cámaras Optitrack fue realizada por uno de los encargados del laboratorio en todo momento, al igual que la creación de cuerpos rígidos, aunque su procedimiento se documenta muy bien en [6]. Se definió un cuerpo rígido en el Robotat empleando un soporte plástico que con al menos 3 marcadores reflectivos. Estos, luego de ser detectados por las cámaras, fueron mostrados en el software dentro de un área que representa la zona capturada por las cámaras. Luego, se seleccionaron los marcadores, y al pulsar clic derecho y seleccionar la opción *create*, se definió un nuevo cuerpo rígido.

El software continuamente presenta la información de posición y orientación del marcador, siempre que el cuerpo rígido se encuentre dentro del área de captura de las cámaras. Estos datos se emplearon como la posición del dron. Para lograr este objetivo, se hizo el montaje mostrado en la Figura 20 entre el dron y marcador de posición empleando velcro.

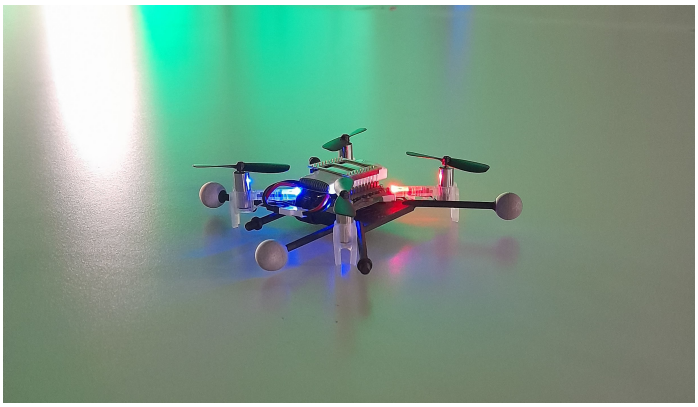
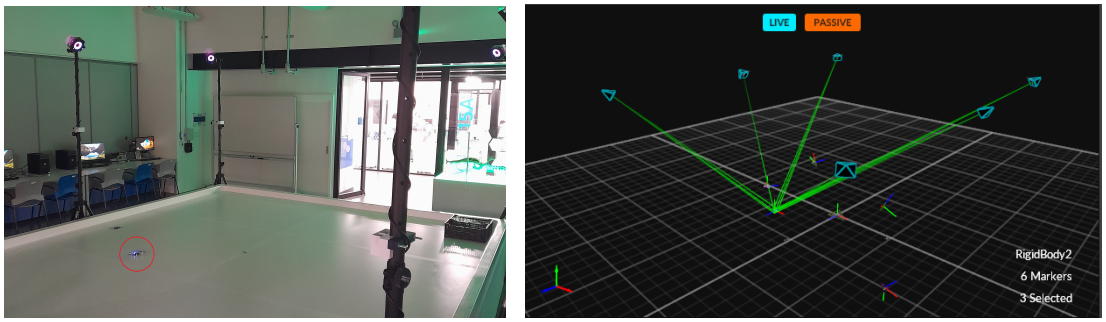


Figura 20: Montaje de dron en el marcador.

A partir de este momento, ya sea empleando el software o la información del código se trató a este marcador dentro del sistema de captura de movimiento como si fuera el dron, tal como se representa en la Figura 21



(a) (b)

Figura 21: Comparativa del dron en la realidad (a) con la imagen virtual en Motive (b).

8.3. Uso de librería *Socket*

Esta librería de Python es empleada en NatNet para la conexión a la red empleando un cable de red. El uso de esta librería de forma individual permite también establecer una conexión de red usando WiFi. La creación de un objeto usando la librería *Socket* permite utilizar una serie de funciones para enviar y recibir información de manera controlada e iterar las lecturas para recibir datos constantemente.

La definición de un objeto se realiza con el comando:

```
import socket
```

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

El objeto creado ahora posee todas las propiedades de la clase *Socket*, y se puede usar para el manejo de datos. La conexión se realiza mediante la siguiente línea:

```
s.connect((HOST, PORT))
```

Donde `HOST` es la dirección IP y `PORT` es el puerto donde se debe conectar la red.

Si el objeto recibe información a través del puerto, la lectura de datos se puede hacer con el comando:

```
data = Robotat.recv(BUFF) # Cantidad de datos a recibir
```

Donde `BUFF` es la cantidad de datos que se espera recibir, en Bytes.

Para enviar información, se emplea la siguiente linea:

```
s.sendall(data.encode())
```

Donde `data` es el dato enviado por la red. La función `encode()` transforma la variable en bytes, y luego la función `sendall` envía los bytes a través del objeto y hacia la red, hasta que encuentre un error, o hasta completar la transmisión.

Esta librería se empleó para realizar la conexión de un socket de red con el software Motive, para recibir los datos de los marcadores de posición. Esto dependió de que en todo momento la computadora estuviera conectada a la red del Robotat

8.4. Conexión y procesamiento de datos recibidos

La conexión al Robotat se realizó empleando las librerías de Python anteriormente mencionadas. Se deja un código de ejemplo como referencia en los anexos. Los datos que recibe el objeto están en formato de bytes y fue necesario decodificarlos para poder usarlos en el programa. Se creó una función llamada `getRawData` que recibe los datos y los transforma en valores de punto flotante.

Es necesario comprender que los datos que envía el Robotat tienen la forma

```
b'[-0.15432748198509216, 0.00577193358913064, 1.3654552698135376,
   -0.9452436566352844, 0.0016223863931372762, -0.32636088132858276,
   -0.0005746895913034678]'
```

La función creada sirvió para traducir los bytes a una cadena de texto, eliminar espacios, separar los valores obtenidos, convertir las cadenas de texto a valores de punto flotante y entregarlo en dos variables separadas.

```
[-0.15432748198509216, 0.00577193358913064, 1.3654552698135376]
[-0.9452436566352844, 0.0016223863931372762, -0.32636088132858276,
```

```
    -0.0005746895913034678]
```

El primer juego de datos es el de posición, y el segundo es el cuaternión del cuerpo rígido, encargado de dar orientación al objeto.

8.5. Datos de posición enviados al dron

Se modificó el código *qualisis_hl_commander.py* para que funcionara con un socket de red y los datos de Motive. Dentro de este código se se empleó la función `send_extpos`, parte de la librería *Crazyflie*, empleada para la creación de un objeto para controlar el dron. Con esta función se actualiza la posición externa del cuadricóptero, de modo que los estimadores internos puedan corregir sus cálculos de posición, que se obtuvieron de la función `getRawData`,

La lectura de datos de posición se hizo creando un objeto derivado de la clase `RobotatHandler`, modificada del código de ejemplo mencionado anteriormente, y empleando la conexión con un socket establecida entre el servidor de Motive y el programa. Luego se almacenó la información decodificada en el objeto usando las variables internas `bodypos` y `bodyquat`. Finalmente, se envió la información de posición al dron usando la función mencionada al inicio. Todo esto se realizó mediante la clase, que a su vez funciona con la librería *Thread*, permitiendo el envío de datos al tiempo que el programa principal sigue en marcha.

Para fines ilustrativos, se muestra un ejemplo de uso de la función `send_extpos`.

```
from cflib.crazyflie import Crazyflie
from cflib.utils import uri_helper
...
uri = uri_helper.uri_from_env(default='//0/0/')

cf=Crazyflie(rw_cache='./cache')
cf.open_link(link_uri)
pose = [0.00 0.00 0.00] # Posicion del dron
quat = [0.00 0.00 0.00] # Cuaternion del dron
cf.extpos.send_extpos(pose[0],pose[2],pose[1])
# O alternativamente
cf.extpos.send_extpos(pose[0],pose[2],pose[1],\
                  quat[0],quat[1],quat[2],quat[3])
...
```

Importante notar el intercambio de las coordenadas y-z reflejado en el intercambio de las variables `pose[2]` y `pose[1]`, debido a que el sistema de coordenadas de Motive usa estos dos ejes de coordenadas de manera distinta. La forma alternativa se deja como referencia, pues se decidió que el acelerómetro interno se encargara de los datos de orientación. También es importante realizar la conexión con el dron usando la librería *uri_helper*, que sirve para facilitar las conexiones mediante la radio. Ejemplos de su uso están en los códigos de ejemplo y en otros trabajos sobre el dron [2].

Se hizo uso usó la función *euler_from_quaternion()*, la cual permite obtener los

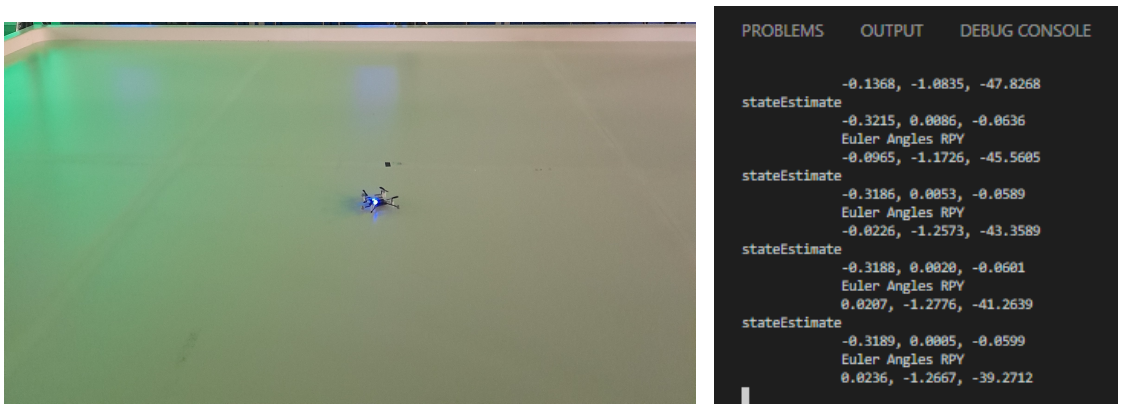
ángulos de Euler de un cuerpo rígido con la información de su cuaternión. Esta información no se envió al dron, sino que se usó en el programa principal como condición para activar el dron y hacerlo girar, de acuerdo con un umbral definido en el código. Para ello simplemente se editó el programa para recibir la información de otro cuerpo rígido definido en el software, y cuando este se moviera manualmente, se enviaría al dron un parámetro de movimiento en el ángulo de guiñada, para hacerlo girar. Esta vez se usó la función *Crazyflie.commander.setpoint* para enviar los parámetros de activación del dron. Para más información del uso de este comando, consultar la documentación o los trabajos previos [13] [2].

8.6. Resultados

Fue posible recibir los datos de posición del dron y enviarlos a este, con el programa modificado. Se muestra en las figuras 22, 23, 24 y 25 la posición y la información del estimador en la consola recibida desde el dron para distintas ubicaciones del dron en el Robotat.

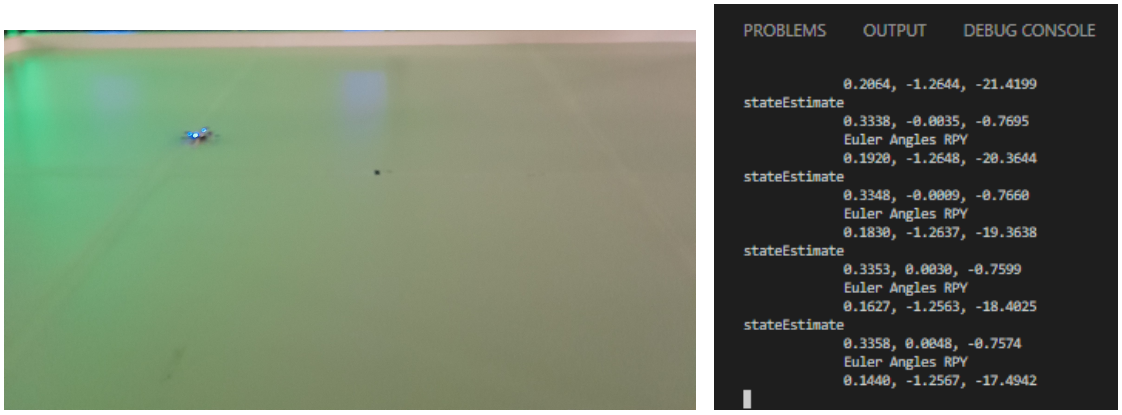
(a) (b)

Figura 22: Dron en posición de prueba 1 (a) y datos del estimador interno (b).



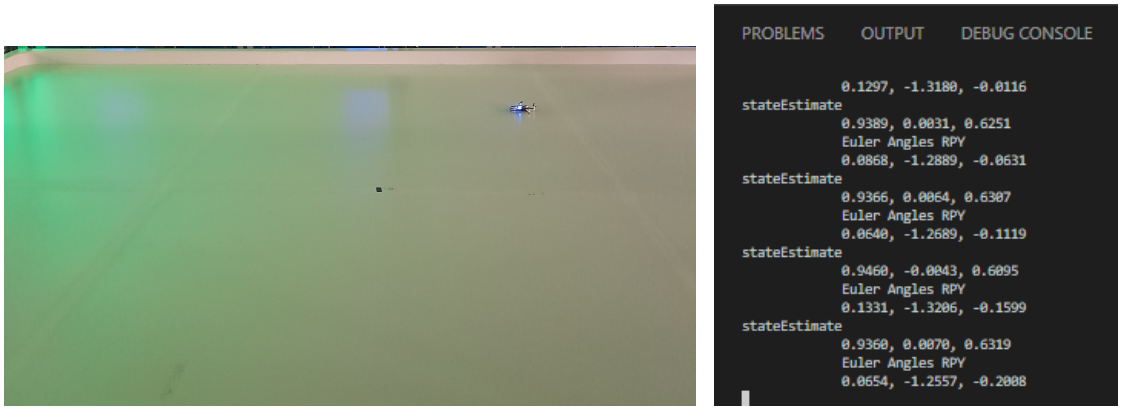
(a) (b)

Figura 23: Dron en posición de prueba 2 (a) y datos del estimador interno (b).



(a) (b)

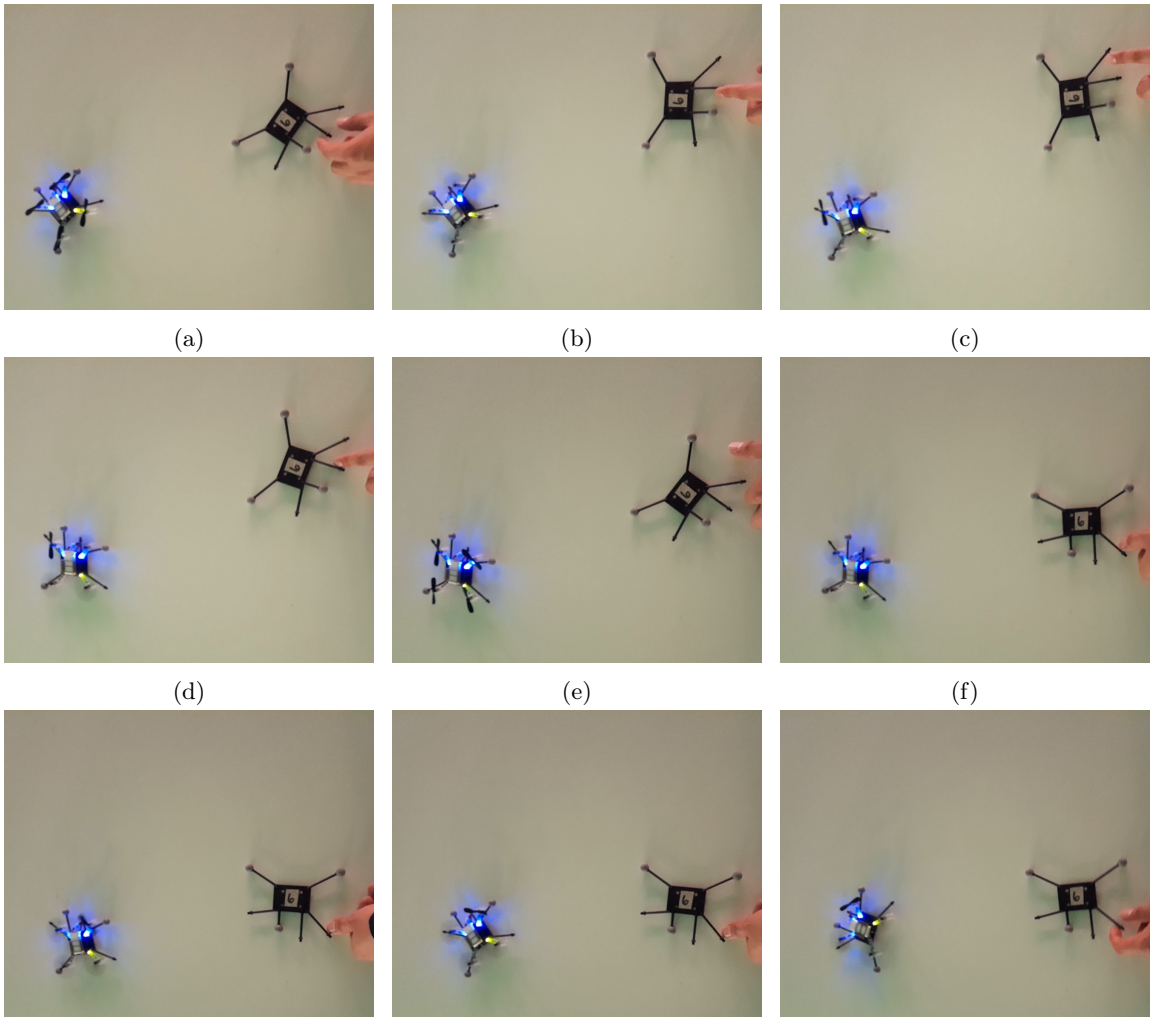
Figura 24: Dron en posición de prueba 3 (a) y datos del estimador interno (b).



(a) (b)

Figura 25: Dron en posición de prueba 4 (a) y datos del estimador interno (b).

También fue posible controlar el dron mediante el uso de un marcador secundario usando los códigos modificados, de modo que cuando el marcador registrara 0.5 o -0.5 radianes rotando en el eje Y, el dron girara hacia la izquierda o hacia la derecha, respectivamente. La Figura 26 muestra la evolución de la ejecución de este código en el tiempo.



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

Figura 26: Control del dron mediante la información de orientación de un marcador secundario.

CAPÍTULO 9

Control del dron en una trayectoria

9.1. Metodología

Se diseñaron dos trayectorias empleando Matlab sin buscar que sean óptimas, una lineal y otra circular, que evita el punto de origen en el sistema de captura de movimiento. Con ayuda del código en *bezier_ trajectory.py*, estas trayectorias han sido transformadas, almacenadas en el dron y ejecutadas dentro del área encerrada por las cámaras. Posteriormente, se utilizó el programa modificado *qualisis_ hl_ commander* para realizar una prueba de suspensión en el aire y una prueba de vuelo punto a punto.

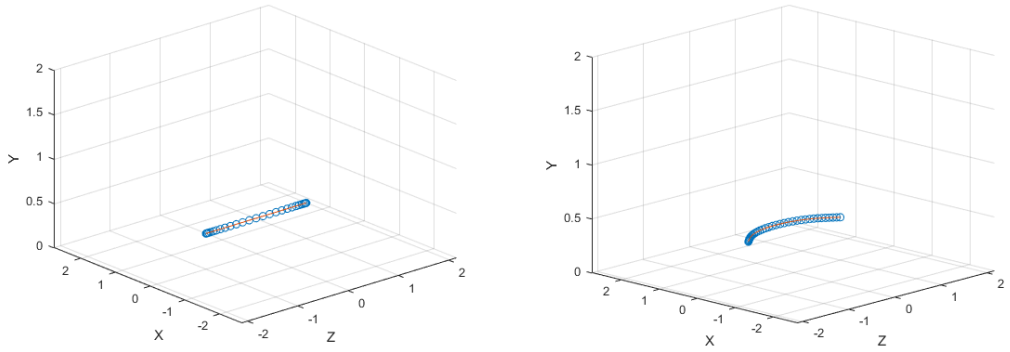
9.2. Creación de trayectorias en Python

Se debe recordar que el dron almacena sus trayectorias empleando los coeficientes de un grupo de polinomios de Bézier. Por lo que las trayectorias a utilizar se deben transformar antes de ser utilizadas. Esto ayuda a reducir espacio en la memoria, para no almacenar una serie de puntos excesivamente grande.

Se usaron las herramientas dadas en *bezier_ trajectory.py* para generar las trayectorias. En esta se usan dos objetos: Segmentos y Nodos. Los nodos son los puntos de referencia que se tienen para la trayectoria, y a su vez contienen otros puntos que sirven para modificar la configuración de la trayectoria. Los segmentos son el polinomio que se forma para generar una conexión entre dos nodos y su configuración. Esta información es la que se envía al dron para que sepa el camino a seguir.

Con las herramientas indicadas y realizando algunas modificaciones al programa de ejemplo se construyeron dos trayectorias. Una recta y una curva casi circular. Además, se

han extraído de estos polinomios una lista de puntos de la trayectoria a recorrer, mostradas en la Figura 27.

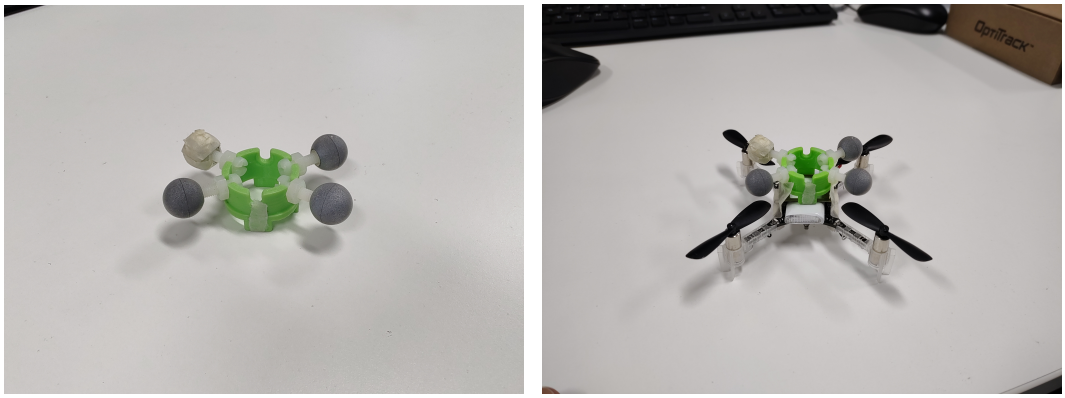


(a) (b)

Figura 27: Trayectorias obtenidas de las curvas de Bézier lineal (a) curva (b).

9.3. Uso de un marcador más preciso.

Al realizar las pruebas de vuelo simple del dron, se observó que este tendía a volar con cierta inclinación, lo cual impedía un vuelo correcto. Esto era provocado por la configuración del marcador en el dron. Pues estos están diseñados para poder brindar una gran cantidad de configuraciones distintas, y no es necesariamente simétrico. Este desbalance es suficiente para alterar el movimiento del dron y hacerlo colisionar. Para resolverlo, se ha diseñado una base de marcadores para sostenerlos por encima del dron y se ha impreso en 3D. Esta base se colocó sobre los extremos de los headers que sujetan la batería y se aseguró su sujeción con masking tape. La pieza y su montaje se muestra en la Figura 28.



(a) (b)

Figura 28: Trayectorias obtenidas de las curvas de Bézier lineal (a) curva (b).

Esta pieza permite generar múltiples configuraciones de marcadores, aunque solo fue necesaria una en este proyecto. Para evitar el desbalance en el vuelo del dron se armó en la base una carga simétrica usando un marcador adicional. Con esta configuración no

se alteró el vuelo del dron en exceso. Sin embargo, ocurrió un problema de detección del marcador, debido a su tamaño. Por lo que el dron tuvo que ser manejado en un área central mostrada en la Figura 29, para asegurar la recepción de datos constante.

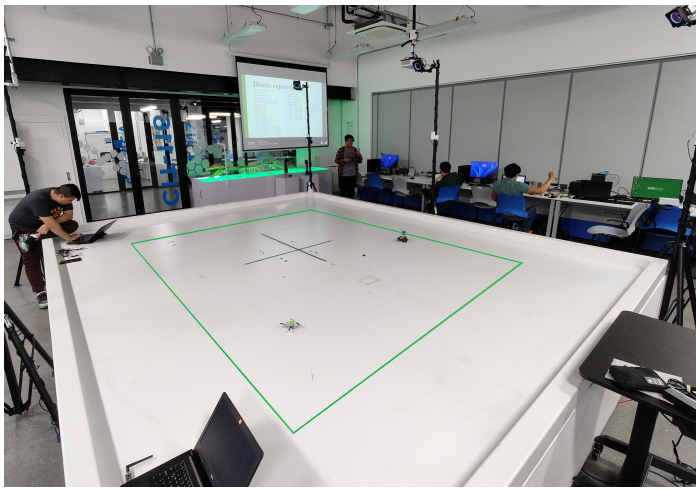


Figura 29: Área confiable del marcador en el Robotat.

9.4. Herramientas para ejecutar trayectorias en el dron

Para que el dron recorra las trayectorias deseadas, se probó emplear la información de las curvas de Bézier junto con el *High Level Commander*, el cual es una versión de alto nivel del *Commander* empleado hasta el momento y permite un control más intuitivo con funciones que realizan un comportamiento específico. Este también administra las trayectorias y las ejecuta, de acuerdo con las instrucciones del código principal.

Esta estructura puede verse en el siguiente extracto del código *autonomous_sequence_high_level.py*.

```
...
def run_sequence(cf, trajectory_id, duration):
    commander = cf.high_level_commander
    commander.takeoff(0.4, 1.0)
    event.wait(2.0)
    relative = False
    commander.start_trajectory(trajectory_id, 1.0, relative)
    event.wait(duration)
    commander.land(0.0, 2.0)
    event.wait(2)
    commander.stop()
...
```

Donde los comandos principales indican, en secuencia y sin considerar los tiempos de espera, elevar el dron a altura 0.4 en 1.0 segundos, iniciar a ejecutar la trayectoria y aterrizar el dron en 2 segundos. Esto refleja lo intuitivo de este módulo. Además de eso, el ejemplo cuenta con una clase que se encarga de cargar las trayectorias del título anterior

en la memoria del dron, para que el *High Level Commander* pueda reconocerla y ejecutarla.

Para cargar las trayectorias al dron, solamente se reemplazó el arreglo de datos de la trayectoria en el ejemplo, y en su lugar se colocó la trayectoria que se usará. Después, se editaron los parámetros en *takeoff()* y *land()* para que coincidan con la trayectoria, siendo el primero de estos la altura y el segundo el tiempo a utilizar para llegar al punto deseado.

Se cargó la trayectoria lineal al dron y se intentó ejecutar su contenido utilizando este ejemplo. Al hacerlo se observaron algunos inconvenientes, como que el dron no se elevaba, aunque intentara ejecutar la trayectoria por cierto intervalo de tiempo. En otras ocasiones se elevaba dando saltos, pero también sin ejecutar la trayectoria adecuadamente. Las razones de esto se desconocen, aunque se cree que está relacionado al funcionamiento del *High level commander*. Y dado que modificar el firmware está fuera del alcance de este trabajo, se optó por realizar trayectorias manualmente con los puntos de las trayectorias generadas por las curvas de Bézier.

9.5. Pruebas de las subfunciones del módulo *commander*

Se decidió probar antes las funciones del *Commander*, a fin de validar que las funciones básicas del dron también se comporten adecuadamente. Para ello se colocaron distintas configuraciones en los subcomandos propios de esta clase. Estas configuraciones, o *setpoints*, son valores que la función codifica antes de enviarlas al dron, y pueden ser valores de posición, ángulo, razón de cambio en un ángulo o un numero dentro de cierto intervalo.

El subcomando *send_setpoint* recibe parámetros de ángulos de Euler de cabeceo y alabeo, taza de cambio de guiñada y empuje vertical codificado en un valor entre 10,000 y 60,000. Para probarlo, se utilizó una configuración con únicamente empuje vertical de 49,000, dejando los ángulos de Euler con valor de 0 todos.

El subcomando *send_zdistance_setpoint* también recibe parámetros de ángulos de Euler de cabeceo y alabeo, taza de cambio de guiñada y una distancia en el eje z del dron. Para probarlo, se asignaron configuraciones con alturas de 0.2 y 0.4 y se dejaron los ángulos de Euler con valor de 0.

EL subcomando *send_hover_setpoint* recibe valores de velocidad en el eje x, velocidad en el eje y, taza de cambio de guiñada y una distancia en el eje z del dron. Para probarlo, se asignaron configuraciones con alturas de 0.2 y 0.4 y el resto de los parámetros con valor 0.

Por último, el subcomando *send_position_setpoint* recibe valores de posición en los ejes x, y, z, y una taza de cambio de guiñada. Para probarlo, se colocó la posición del origen (0, 0, 0) con una taza de cambio de guiñada de 0.

Luego de probar cada subcomando se encontró que no todos trabajan adecuadamente. Por ejemplo, el subcomando de posición logra elevar al dron al punto deseado. Pero luego de hacerlo, o incluso antes de llegar al punto, pierde el control y hace estrellar al dron. Por otro lado, el subcomando de hover no siempre eleva al dron, y cuando lo hace

no logra controlar el dron correctamente, pues el dron si adquiere velocidad en alguno de los ejes. Los que mejor funcionaron fueron los subcomandos *send_ zdistance_ setpoint* y *send_ setpoint*, que se llamarán subcomandos de altura y de control a partir de ahora. Estos dos respondían a lo requerido, como elevar el dron a cierta altura o con cierta fuerza de elevación e inclinar al dron en la dirección deseada.

9.6. Pruebas de control mediante el código principal

Se decidió ejecutar un programa de control que emplee el subcomando de altura y la información del Robotat de la posición del dron para ejecutar un método de control de posición de punto a punto. Primero, se diseñó una función que detecta cuando el dron esté alineado con una de las coordenadas, y si no lo está entonces inclina el dron un ángulo pequeño por un intervalo corto de tiempo.

Se logró que la función detectara estos cambios y respondiera en consecuencia enviando comandos al dron. El código respondió cambiando los parámetros de vuelo del dron cuando este cruzaba un punto de prueba configurable, que se colocó en el origen y en las coordenadas x, $z = (0, 1)$. Al mover manualmente el dron al punto indicado, el código desplegaba texto en el código señalando que se ha llegado al punto deseado.

Sin embargo, al hacer que estas funciones enviaran datos al dron con la información sus parámetros modificados, este presentaba un comportamiento inesperado. El dron volaba hacia el punto deseado, pero luego perdía el control y se estrellaba. Al analizar los datos de la consola se encontró que el código seguía respondiendo, actualizando los parámetros del dron, pero este último no ejecutaba los comandos recibidos.

Se observó un comportamiento extra en esta etapa. El dron recién reiniciado era capaz de encontrar el punto deseado. Sin embargo, al ejecutar nuevamente una secuencia en este dron sin reiniciar, este perdía el control y se estrellaba. De modo que la ejecución de este código dependía fuertemente del estado actual del dron.

Se intentó corregir esto realizando un rastreo de puntos cercanos, para que el dron no pierda el control por tener gran velocidad al acercarse a un punto lejano. El resultado fue un poco mejor en cuanto al movimiento del dron, aunque el problema del reinicio del dron continuó apareciendo.

Por último, para probar el funcionamiento de la subfunción de altura, se decidió ejecutar un código que controle la altura del dron, sin considerar el control en los ejes horizontales. Para esto, se modificó nuevamente el código de control para generar intervalos de tiempo, que luego se usaron para iterar distintas alturas de entre 0.2 y 0.8 metros por encima de la mesa, mientras el dron se encontraba en vuelo. Se obtuvo como resultado que el dron ejecutaba correctamente los cambios de altura, aunque adquiría velocidad lineal que lo desplazaba durante el vuelo.

9.7. Resultados

La ejecución de la trayectoria lineal mediante el código *autonomous_sequence_high_level.py* mostró que el dron no ejecuta adecuadamente las trayectorias cargadas a la memoria. La Figura 30 muestra la línea de tiempo de ejecución de este código, revelando que el *high level commander* no eleva el dron, aunque se pueden observar ligeros cambios en la orientación del dron y que una de las aspas se detiene, mostrando que el dron si intenta ejecutar una función, pero que la respuesta no es adecuada.

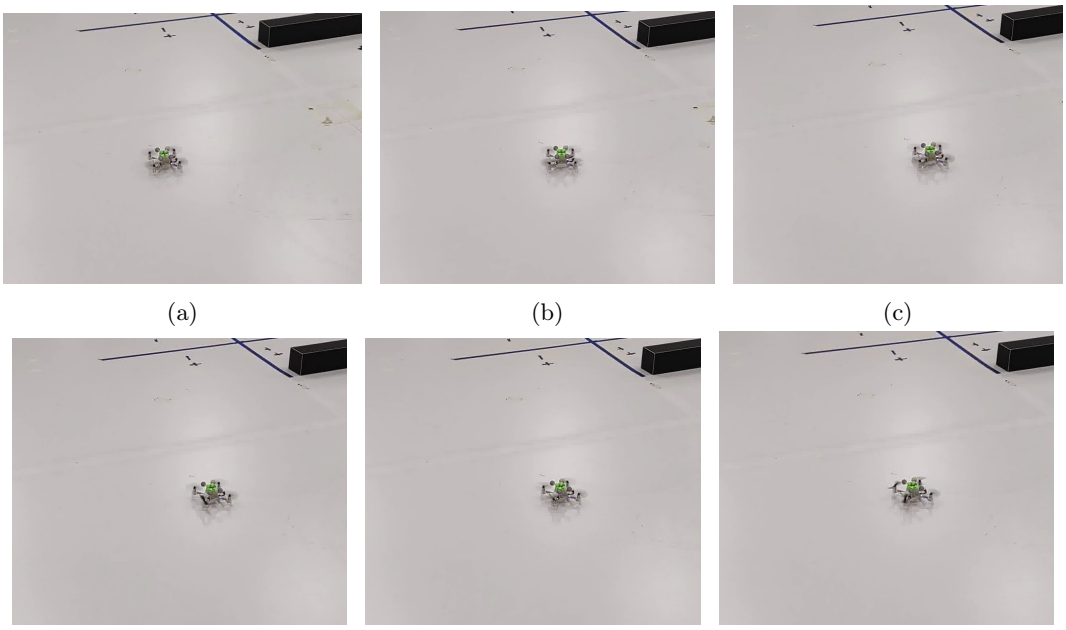
Por otro lado, los comandantes tuvieron comportamientos distintos, aunque podrían resumirse a un caso de vuelo fallido, uno de vuelo inconstante y dos de vuelo irregular. Se incluye en esta sección el caso de vuelo de la subfunción de altura y se dejan los casos restantes como anexos. Se pudo observar que el dron vuela a una altura estable durante la ejecución de esta subfunción, mostrada en la linea de tiempo en la Figura 31. Por esta razón, se eligió esta como base para experimentar con la ejecución de trayectorias usando código.

Al probar el código de control, se hizo primero una versión para llegar a un punto dado sin importar la distancia. Este mostró ser capaz de reconocer la dirección a recorrer. La Figura 32 muestra la línea de tiempo del funcionamiento de la rutina de control intentando alinearse con el origen para luego detenerse, y la Figura 33 muestra una segunda ejecución de la rutina partiendo desde el punto final de la anterior ejecución.

Posteriormente se usó el código de control anterior como bloque de construcción para controlar dos coordenadas a la vez. Se muestra su comportamiento en la Figura 34. Se puede observar que el dron se mueve en dirección al origen, que era el punto esperado. Sin embargo, luego de cruzar este punto, el dron pierde el control y cae a alta velocidad. En este punto se descubrió el comportamiento de fallo del dron, que ocurre en caso de no reiniciarlo antes de ejecutar la rutina de control.

Se decidió cambiar el enfoque del código para que controle la aproximación a un punto cercano, tal como se ve en la Figura 35. Se observa un área cuadrada encerrada con masking tape, que representan el punto objetivo al que el dron debe ir, con un margen de tolerancia. Se observa que el dron se eleva y al estar dentro del área marcada los motores se apagan, respondiendo a la rutina de llegada en el código de control. En este punto se descubrió que, sin un previo reinicio del dron, este siempre pierde el control y se estrella al ejecutar nuevamente el código.

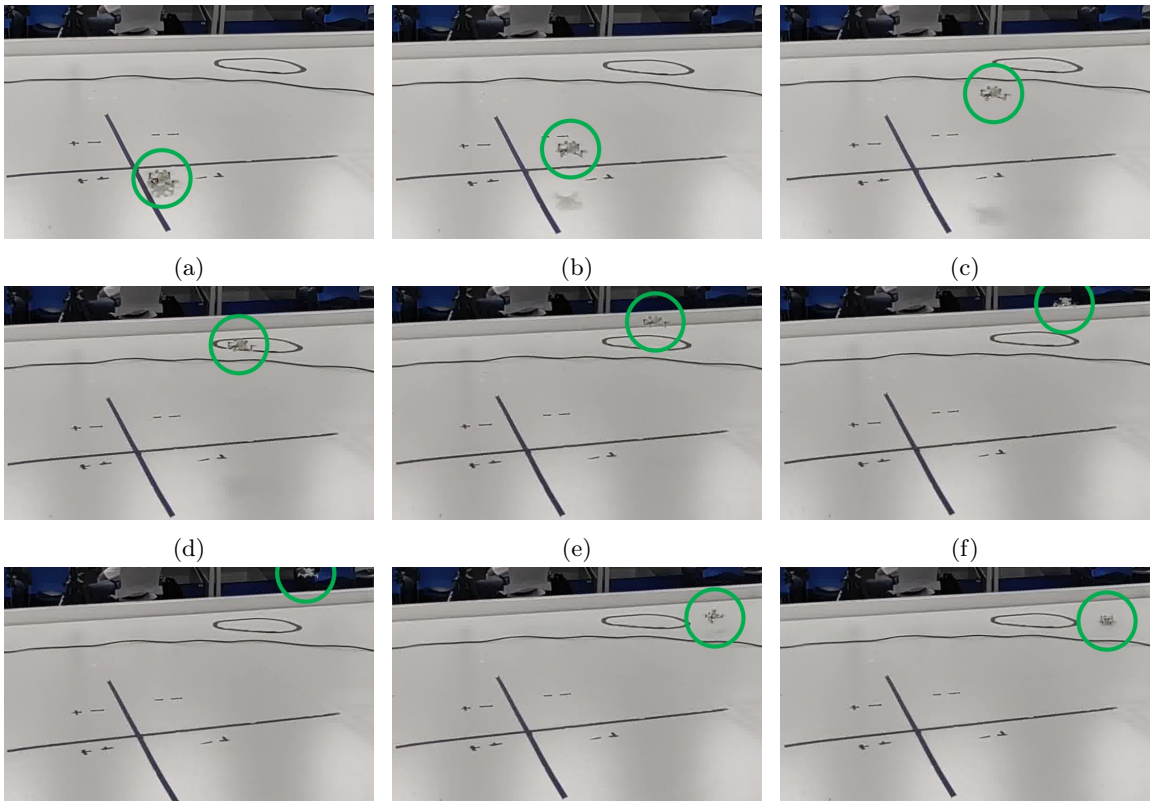
Sin embargo, al controlar la altura, el dron respondió adecuadamente ejecutando una trayectoria de cambios de altura. Aunque debido a la falta de control horizontal, este adquirió velocidad lineal y se trasladó de un extremo a otro de la mesa mientras ejecutaba esta rutina. Este recorrido se muestra en la Figura 36



(a) (b) (c)

(d) (e) (f)

Figura 30: Línea de tiempo de ejecución del código *autonomous_sequence_high_level.py*.

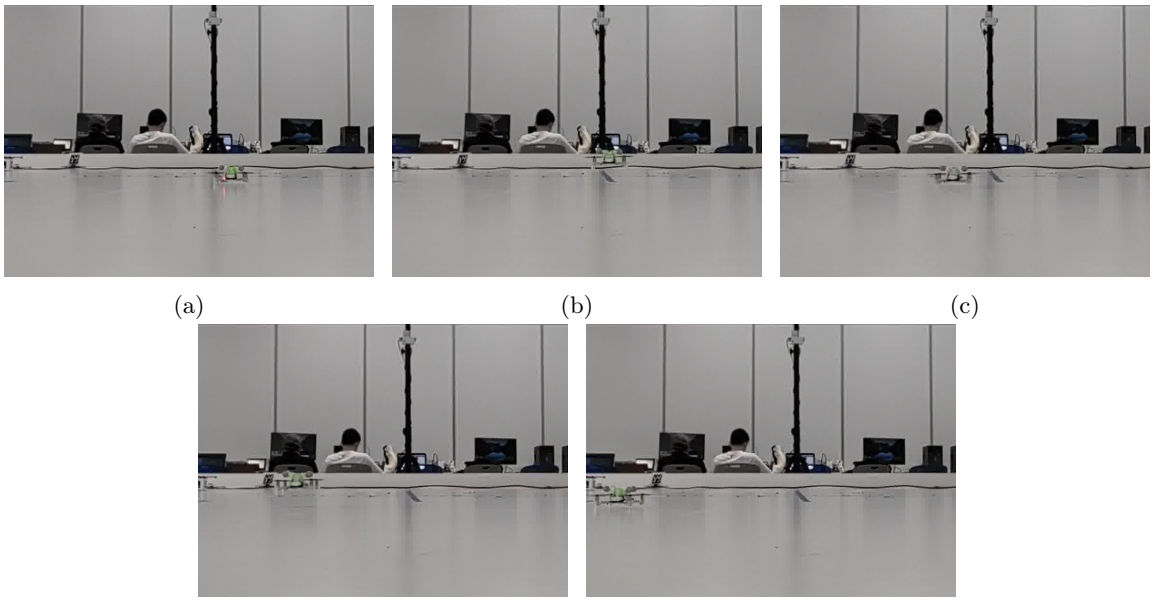


(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

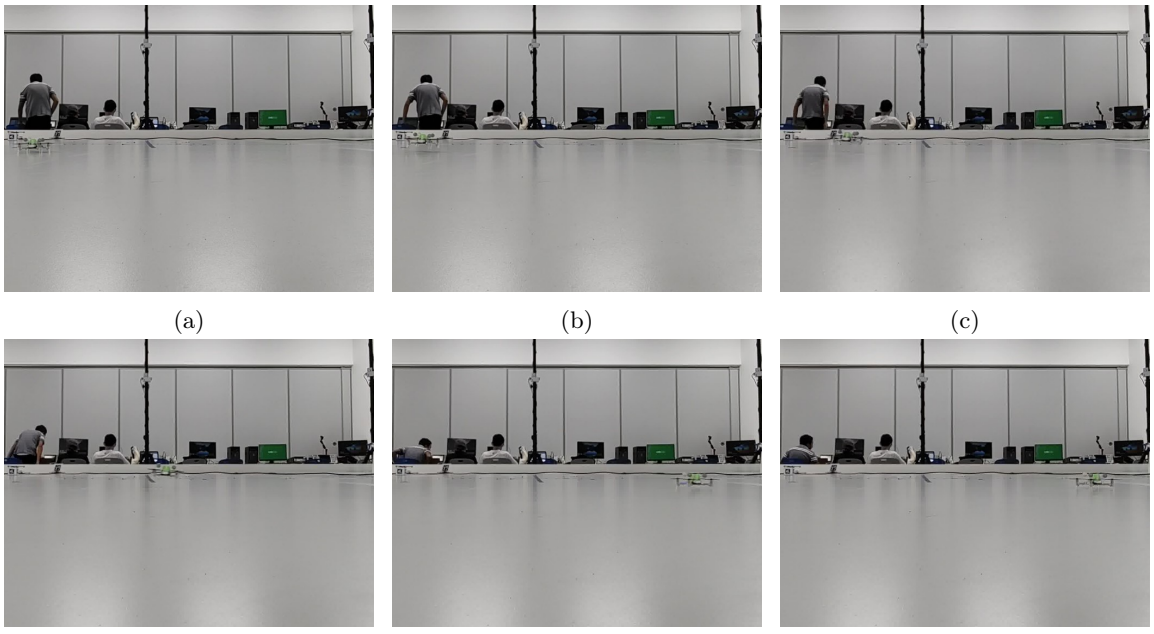
Figura 31: Recorrido del dron al ejecutar el subcomando *zdistance*.



(a) (b) (c)

(d) (e)

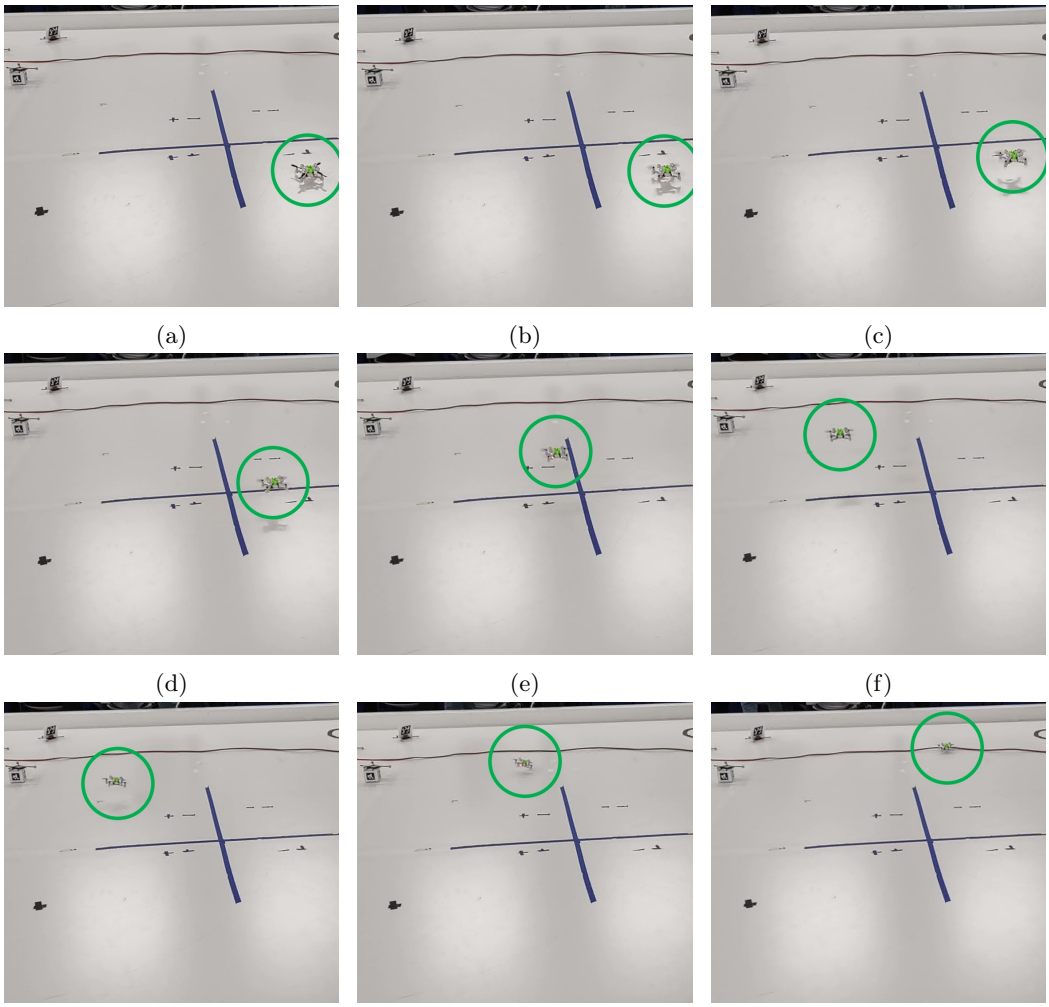
Figura 32: Recorrido del don al ejecutar el código de control en una sola coordenada.



(a) (b) (c)

(d) (e) (f)

Figura 33: Recorrido del don al realizar control en una sola coordenada, partiendo desde el punto final del intento anterior.

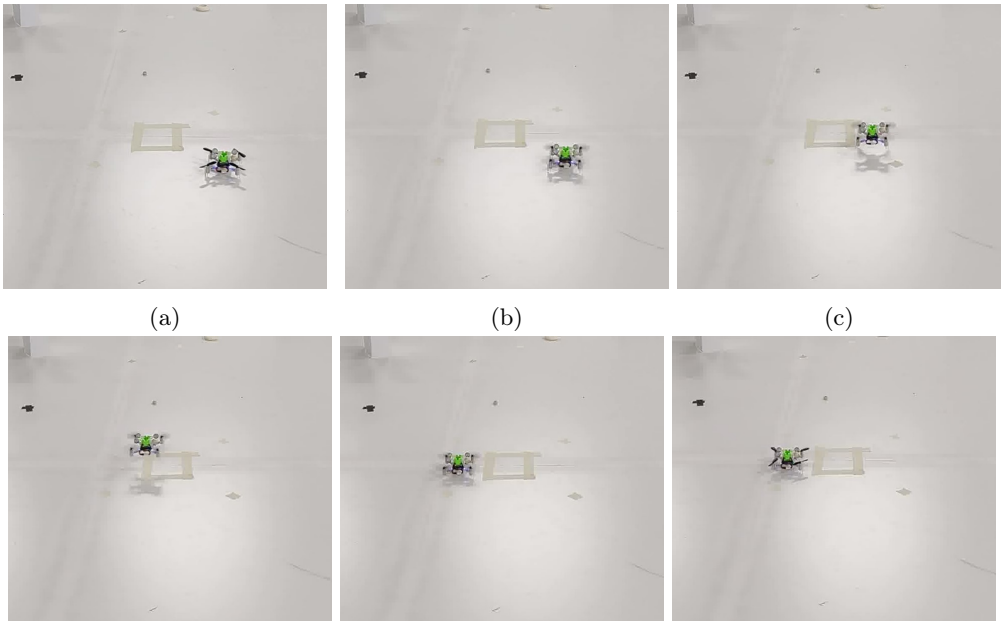


(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

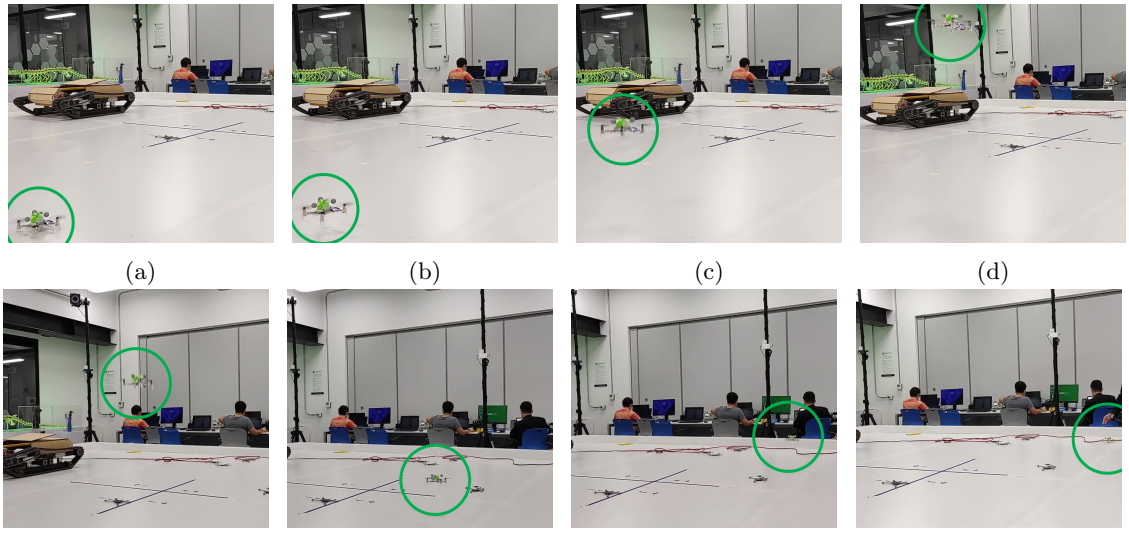
Figura 34: Ejecución del código de control con dos coordenadas.



(a) (b) (c)

(d) (e) (f)

Figura 35: Ejecución del código de control para un punto cercano.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figura 36: Ejecución de control de altura.

CAPÍTULO 10

Conclusiones

- Fue posible establecer una conexión con el dron mediante una máquina física y también con una máquina virtual, logrando obtener una transmisión de los parámetros internos del dron y estableciendo una conexión con el dron usando el cliente.
- Fue posible realizar un vuelo simple con el dron, empleando la librería de Python diseñada por Bitcraze y la radio, logrando activar los motores con una potencia uniforme y suficiente para elevarlo sin considerar el control de posición.
- Fue posible crear cuerpos rígidos en el sistema Optitrack, enviar su información desde el ecosistema Robotat mediante una conexión de red hacia la computadora principal y decodificar los valores numéricos de posición y orientación de un cuerpo rígido específico.
- Fue posible representar al dron mediante un cuerpo rígido, enviar su información al dron y que este actualizara su posición en consecuencia mediante los estimadores internos.
- Se logró controlar el dron mediante el uso de la información de posición de un cuerpo rígido en Motive, logrando activar el dron o actualizar la información de posición de este.
- Se logró codificar una trayectoria en el formato de las curvas de Bézier usado por el dron. Sin embargo la ejecución de esta trayectoria falló debido a un funcionamiento incorrecto en el módulo *High Level Commmander*.
- No se logró controlar al dron mediante rutinas de aproximación a un punto, descubriendo además que el dron tiende a fallar cuando se ejecuta un código de control sin reiniciarlo antes.
- Fue posible ejecutar una trayectoria de cambios de alturas empleando un subcomando del módulo *Commander*.

CAPÍTULO 11

Recomendaciones

- Se recomienda profundizar en el uso de las subfunciones del *Commander* directamente en el firmware, para poder reforzar su comprensión y realizar un mejor control de trayectorias, en caso de realizar el control desde el programa principal.
- Se recomienda emplear una máquina física que tenga instalada la distribución de Linux creada por Bitcraze para realizar futuros trabajos, debido a que muchas de las herramientas disponibles en linea y/o creados por esta empresa, como un generador de trayectorias, fueron pensadas para funcionar en este sistema, y su uso permitirá mucha versatilidad en futuros proyectos.
- Para mejorar las estimaciones del dron, se recomienda trabajar en el firmware para crear más sensores para el dron que permitan brindar mejor información de posición u orientación, de acuerdo con las necesidades de cada proyecto. O usar más sensores preexistentes para desarrollar proyectos con un alcance similar.
- Se recomienda implementar en futuras aplicaciones del proyecto el uso de los valores de velocidad del dron provenientes del Robotat o del dron. Con esta información sería posible implementar el control de posición más fácilmente que usando únicamente la posición.

CAPÍTULO 12

Bibliografía

[1] W. Giernacki, M. Skwierczyński, W. Witwicki, P. Wroński y P. Kozierski, «Crazyflie 2.0 Quadrotor as a Platform for Research and Education in Robotics and Control Engineering,» *22nd International Conference on Methods and Models in Automation and Robotics (MMAR)*, págs. 37-42, 2017.

[2] F. J. Sanabria Salazar, «Diseño e implementación de una plataforma de pruebas para sistemas de control para el dron crazyflie 2.0,» 2021.

[3] M. Gopabhat Madhusudhan, «Control of crazyflie nano quadcopter using Simulink,» 2016.

[4] C. Chadehumbe y J. Sjöberg, «Autonomous flight of the micro drone Crazyflie 2.1 through an obstacle course,» 2020.

[5] D. Warren Mellinger, «Trajectory Generation and Control for Quadrotors,» Tesis doct., 2012.

[6] C. Perafán Montoya, «Robotat: un ecosistema de captura de movimiento y comunicación inalámbrica,» 2021.

[7] C. A. Alonzo Martínez, «Manufactura, modelado y control de un cuadricóptero con comunicación inalámbrica Wi-Fi integrado al ecosistema Robotat,» 2021.

[8] H. A. Burmester Campos, «Diseño de controlador de vuelo para cuadricóptero con capacidad de integrarse a ecosistema Robotat vía inalámbrica Wi-Fi,» 2021.

[9] Bitcraze. «Crazyflie 2.1.» (4 de mayo de 2022), dirección: `https://www.bitcraze.io/products/crazyflie-2-1/`.

[10] B. V. Machine. «Bitcraze.» (7 de mayo de 2022), dirección: `https://wiki.bitcraze.io/projects:virtualmachine:index`.

[11] Bitcraze. «Installing USB Driver on Windows.» (2022), dirección: `https://www.bitcraze.io/documentation/repository/crazyradio-firmware/master/building/usbwindows/`.

[12] ——, «Client installation.» (2022), dirección: `https://github.com/bitcraze/crazyflie-clients-python/blob/master/docs/installation/install.md`.

[13] ——, «Crazyflie-lib-python Installation.» (2022), dirección: https://www.bitcraze.io/documentation/repository/crazyflie-lib-python/master/installation/install/.

[14] B. AB. «The Coordinate System of the Crazyflie 2.X.» (2022), dirección: https://www.bitcraze.io/documentation/system/platform/cf2-coordinate-system/.

[15] Bitcraze. «Home [Crazyradio PA Documentation].» (2022), dirección: https://www.bitcraze.io/documentation/repository/crazyradio-firmware/master/.

[16] J. D. Guerra Quiñones, «Control de actitud de un cuadricóptero,» Tesis doct., 2012.

[17] G. Silano, E. Aucone y L. Iannelli, «CrazyS: a software-in-the-loop platform for the Crazyflie 2.0 nano-quadcopter,» IEEE, inf. téc., 2018.

[18] Coldplay. «Coldplay - Adventure Of A Lifetime (Making Of Video).» (2015), dirección: https://www.youtube.com/watch?v=GnPfDX4mZ40.

[19] ACBP. «Benedict Cumberbatch – Behind-the-Scenes of The Hobbit: Desolation of Smaug.» (2016), dirección: https://www.youtube.com/watch?v=Wu9XPEdBelY.

[20] A. Kushleyev, D. Mellinger y V. Kumar. «A Swarm of Nano Quadrotors.» (2012), dirección: https://www.youtube.com/watch?v=YQIMGV5vtd4&t=1s.

[21] Qualisis. «Motion Capture for Robotics AUV & UAV.» (2022), dirección: https://www.qualisys.com/applications/engineering/cybernetics/.

[22] N. Corporation. «OptiTrack Documentation Wiki.» (5 de mayo de 2022), dirección: https://v22.wiki.optitrack.com/index.php?title=OptiTrack_Documentation_Wiki.

[23] ——, «*Prime*$^{\times}$ *41* - Specs.» (6 de mayo de 2022), dirección: https://optitrack.com/cameras/primex-41/specs.html.

[24] ——, «NatNet SDK.» (7 de mayo de 2022), dirección: https://optitrack.com/software/natnet-sdk/.

[25] M. E. Mortenson, *Mathematics for computer graphics applications*. 1999.

[26] B. AB. «Trajectory formats.» (2022), dirección: https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/trajectory_formats/.

[27] Bitcraze. «Bitcraze Virtual Machine.» (2022), dirección: https://github.com/bitcraze/bitcraze-vm.

[28] Oracle. «Download VirtualBox.» (2022), dirección: https://www.virtualbox.org/wiki/Downloads.

[29] P. S. Foundation. «threading — Thread-based parallelism.» (2022), dirección: https://docs.python.org/3/library/threading.html#event-objects.

[30] V. Sajip. «Logging Cookbook.» (2022), dirección: https://docs.python.org/3/howto/logging-cookbook.html#logging-from-multiple-threads.

CAPÍTULO 13

Anexos

13.1. Programa de Python usando *Event*

El siguiente es un ejemplo modificado [30] que ilustra el uso de un objeto `Event` en múltiples hilos.

```
import threading

def timeHandler(event, no):
    while not event.isSet():
        print(f"Thread {no}")
        event.wait(1)

event = threading.Event()
tr1 = threading.Thread(target=timeHandler, args=(event, 1))
tr2 = threading.Thread(target=timeHandler, args=(event, 2))
tr1.start()
tr2.start()

while not event.isSet():
    try:
        print("Main Thread")
        event.wait(0.75)
    except KeyboardInterrupt:
        event.set()
        break
```

En este ejemplo, el objeto `event` creado se usa como gestor de tiempo en múltiples hilos mediante la propiedad `event.isSet()`. Cuando se realiza una interrupción manual del programa, la bandera interna se activa forzosamente mediante la linea `event.set()`, contribuyendo a cerrar las funciones asignadas a los otros hilos y terminando el programa.

13.2. Conexión a red Robotat

Para definir una conexión de red con el Robotat, fue necesario usar la dirección IP y el puerto al que Motive estaba enviando sus datos. Se creó en Python un programa para realizar la conexión.

```
import socket
HOST = 0.0.0.0 # La ip del Robotat en la red
PORT = 0 # el puerto del Robotat en la red
...
connected = True
try:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((HOST, PORT))
    data = s.recv(1024) # Lectura de datos de bienvenida
except:
    s = []
    connected = False
```

Luego, para recibir la información de un cuerpo rígido, este debe estar definido en Motive y se debe conocer la ID que este posee. En Python, se definió una variable con el número de ID del cuerpo rígido seleccionado.

```
RigidBodyID = 0 # ID del objeto en el Robotat
...
Robotat.sendall(RigidBodyID.encode())
data = Robotat.recv(1024) # Cantidad de datos a recibir
```

Es importante mencionar que se debe conectar la computadora a la red WiFi del Robotat para que la conexión sea exitosa. De otro modo, siempre fallará.

13.3. Enlace a lista de reproducción con los resultados

En el siguiente enlace se muestra una colección de los videos recopilados durante este proyecto. Incluye los resultados documentados en este trabajo junto con otros videos alternos que muestran resultados similares, pero son menos significativos o repiten resultados desde otro punto de inicio u otro ángulo.

Lista de reproducción "Resultados Experimentación":

`https://youtube.com/playlist?list=PLvwB0vJ1kwjbzH5WmL71cNvTCCFcsJdLO`

CAPÍTULO 14

Glosario

Log: Conocido como Registro en Español, es el nombre que recibe una variable definida con las librerías del Crazyflie. Su función es recibir constantemente los valores internos del dron y mantenerlos disponibles en el programa. Se caracteriza por su velocidad y por que solo puede recibir información de las variables programadas en el firmware del dron como variables tipo Log . 21

Matlab: Este es un programa desarrollado para el trabajo matemático usando matrices. Posee múltiples funciones que permiten realizar cálculos matemáticos fácilmente en forma de algoritmos, y también realizar otras operaciones más complejas, como derivación, geometría, simulaciones de algoritmos de robótica, entre otros . 8

Modulación de Ancho de pulso: Llamada *Pulse Width Modulation* en Inglés, esta es una señal alternante de amplitud y frecuencia definidas y generalmente cuadrada. Funciona haciendo variar la duración de un pulso dentro de un periodo de tiempo de la señal, de modo que mientras más tiempo dure el pulso, mayor energía entregará la señal. Suele usarse en el control de diodos emisores de luz y también en el control de motores . 8

Motive Software: Este es un programa desarrollado por la empresa Optitrack, empleado para controlar la información recibida del sistema de captura de movimiento, creado por la misma empresa, y proyectarlo en un entorno amigable con el usuario. También permite el envío de los datos registrados en el software a través de la red . 8

Robotat: Este es un ecosistema empleado para pruebas de robótica en la universidad del Valle de Guatemala. Surgió de los esfuerzos de la universidad y los trabajos de graduación de múltiples estudiantes. El trabajo más reciente realizado en torno al Robotat fue empleado para trabajar este proyecto [6]. 1