



ugr | Universidad
de Granada

TRABAJO FIN DE GRADO
INGENIERÍA INFORMÁTICA

Control y coordinación de drones Crazyflie

Autor

Ángel Hurtado Flores

Tutor

Héctor García de Marina Peinado



E.T.S. DE INGENIERÍAS INFORMÁTICA Y DE TELECOMUNICACIÓN

Granada, Junio de 2024

Control y coordinación de drones Crazyflie

Ángel Hurtado Flores

Palabras clave: Paparazzi UAV, robótica, firmware, drones, GVF

Resumen

Los recientes avances en electrónica, robótica e informática han permitido el desarrollo y creación de un sinfín de nuevas tecnologías. Entre ellas, ha destacado recientemente el crecimiento de los drones, formalmente conocidos como UAV (*Unmanned Aerial Vehicle*), dispositivos que permiten una gran variedad de aplicaciones, desde juguetes y hobbies, hasta aplicaciones profesionales como la agricultura o fotografía.

El abanico de posibilidades es uno de los mayores fuertes de esta tecnología, sin embargo, los resultados obtenidos no siempre son los esperados. La mayoría de soluciones sugeridas recientemente se apoyan en incrementar el número de UAVs trabajando simultáneamente, lo que implica que se necesita control y coordinación entre los vehículos, generalmente con la ayuda de la informática y las telecomunicaciones.

En este trabajo nos centraremos en el control y coordinación de varios UAVs, principalmente en la subcategoría conocida como cuadricópteros, popularmente llamados drones. Se propone como objetivo final conseguir la coordinación con drones *Crazyflies*. La integración de resultados se hará mediante la implementación de un firmware de código abierto basado en el proyecto *Paparazzi UAV* y la inclusión de un algoritmo para movimiento coordinado entre drones.

Control and coordination of Crazyflie drones

Ángel Hurtado Flores

Keywords: Paparazzi UAV, robotics, firmware, drones, GVF

Abstract

Recent advancements in electronics, robotics and computing led to the development of countless new technologies. Among them, the growth of drones, formally known as UAV (Unmanned Aerial Vehicles), has recently skyrocketed.

The range of possibilities is one of the greatest strengths of this technology; however, obtained results are not always what is expected from it. Recently proposed solutions suggest increasing the number of UAVs working simultaneously, which implies the need for control and coordination among vehicles, which is typically assisted by computer science and telecommunications.

In this work, we will focus on the control and coordination of multiple UAVs, primarily in the subcategory known as quadcopters, popularly called drones. The ultimate goal is to achieve coordination among *Crazyflies*. In order to achieve this goal, we will use an open-source firmware based on the *Paparazzi UAV* project and the implementation of an algorithm for coordinated movement between drones.

Yo, **Ángel Hurtado Flores**, alumno de la titulación Ingeniería Informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación**, con DNI 51*****B, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Firmado: Ángel Hurtado Flores

Granada a 21 de Junio de 2024

D. **Héctor García de Marina Peinado**, Profesor del Departamento Ingeniería de Computadores, Automática y Robótica de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado ***Control y coordinación de drones Crazyflie***, ha sido realizado bajo su supervisión por **Ángel Hurtado Flores**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 21 de Junio de 2024

El director:

Héctor García de Marina Peinado

Agradecimientos

A mis amigos, por las “noches de durum”, “miércoles de pollo” y todos los buenos ratos que hemos pasado juntos, ya fuese en el futbolín, jugando a cualquier juego o simplemente charlando en el comedor de la ETSIIT. En especial, a José por tantísimas cosas que hemos hecho juntos: trabajos, jugar, cocinar, el robot...

A mi familia, por el apoyo y paciencia durante estos cuatro años. Por un lado, a mi padre, por enseñarme a “cacharrear”, a mi hermano, por todo lo que me ha enseñado de tecnología e ingeniería y a mi tío Alberto, por hacerme ver la tecnología desde el punto de vista del entretenimiento. Por otro lado, a mi madre, por enseñarme de tantas cosas distintas a la tecnología que siempre son útiles y por tantos otros consejos; y a mi hermana, por contagiarme su simpatía, ánimo y sonrisa que todos los días luce.

A los miembros del despacho DB-4. A Manu, por hacer todos los días en el despacho mucho más divertidos. En especial, a mi tutor Héctor y su estudiante Jesús, por todo el apoyo brindado a lo largo de este trabajo, por ayudarme a salir de todos esos bloqueos, las sugerencias dadas para este trabajo y el extenso conocimiento de matemáticas, física y robótica que me han transmitido. Y por supuesto, por los entretenidos días de pruebas en el aeródromo; seguro que no olvidaremos el día en que un Crazyflie voló a toda mecha hacia el barranco...

ÍNDICE GENERAL

1. Introducción	21
1.1. Motivación	21
1.2. Descripción general del sistema	22
1.2.1. Hardware: Crazyflies y accesorios	22
1.2.2. Software: Paparazzi UAV	23
1.3. Objetivos	24
1.4. Planificación	25
2. Preliminares	27
2.1. Montaje de los drones	27
2.2. Pruebas con software oficial	28
2.2.1. Preparación	29
2.2.2. Control básico con Crazyflie Client	29
2.2.3. Variables de control en Crazyflie Client	30
2.3. Pruebas con Paparazzi UAV	31
2.3.1. Paparazzi Center	31
2.3.2. Simulaciones en Paparazzi Center	32
2.3.3. Firmware de Paparazzi para Crazyflie 2.1	33
2.4. Desarrollo básico en Paparazzi	35
2.4.1. Desarrollo de módulos de control	35
2.4.2. Uso de módulos en una simulación	38
3. Control de un Crazyflie	39
3.1. Control básico	39
3.1.1. PID en Paparazzi	39
3.1.2. Metodología para ajuste del PID	41
3.1.3. Hovering	41
3.2. Algoritmos GVF	42
3.2.1. Fundamentos matemáticos	42
3.2.2. Algoritmos GVF para rotorcrafts	44

3.2.3. Limitaciones de GVF sin posicionamiento	45
3.3. Implementación del algoritmo GVF	46
3.3.1. Control mínimo por GVF de un rotorcraft	46
3.3.2. Ampliaciones al módulo de GVF en Paparazzi	47
4. Coordinación con Crazyflies	51
4.1. Algoritmos de coordinación	51
4.1.1. Formación circular	51
4.1.2. Formación en segmentos paralelos	53
4.1.3. Comparativa entre formaciones	56
4.2. Implementación de los algoritmos de coordinación	56
4.2.1. Adaptación de formaciones circulares	56
4.2.2. Implementación de formaciones en segmentos	58
4.3. Simulación de las formaciones	59
4.3.1. Simulación de formaciones circulares	60
4.3.2. Simulación de formaciones en segmentos	61
4.4. Coordinación y formación de Crazyflies	63
4.4.1. Comparación entre firmware oficial y de Paparazzi . .	63
4.4.2. Control en el firmware de Bitcraze	64
4.4.3. Formación en segmentos en el firmware de Bitcraze .	65
4.4.4. Resultados de la coordinación	67
5. Conclusiones	71
5.1. Análisis de los objetivos y planificación	71
5.2. Aplicaciones y usos	73
5.3. Trabajos y mejoras futuras	73
5.4. Conclusión	75
A. Instalación de Paparazzi en Debian	77
A.1. Instalación de Paparazzi Center	77
A.2. Instalación de Paparazzi GCS	78
A.3. Instalación completa	79
B. Instalación de dependencias	81
B.1. Dependencias a instalar	81
B.2. Configuraciones y ajustes	82
C. Manual de usuario	83
C.1. Uso de Crazyflies en Paparazzi	83
C.2. Coordinación de Crazyflies bajo firmware oficial	85
C.2.1. Demo	85

ÍNDICE GENERAL **15**

C.2.2.	Demo para varios Crazyflies	85
C.2.3.	Segmento para un solo Crazyflie	86
C.2.4.	Coordinación en segmentos paralelos para dos Crazyflies	86
C.2.5.	Coordinación en segmentos paralelos simulada	87

Bibliografia	91
---------------------	-----------

ÍNDICE DE FIGURAS

1.1.	Imagen de un Crazyflie 2.1 montado	22
1.2.	Loco Positioning Deck (izquierda). Loco Positioning Node (derecha)	23
1.3.	Captura de la interfaz de Paparazzi GCS	24
1.4.	Planificación de este trabajo	25
2.1.	Componentes de un Crazyflie 2.1	28
2.2.	Crazyradio 2.0	29
2.3.	Interfaz de Crazyflie Client	30
2.4.	Ejes de rotación de un vehículo aéreo	31
2.5.	Captura de Paparazzi Center bajo Debian 12 con KDE	32
2.6.	Paparazzi GCS ejecutando la simulación por defecto	32
2.7.	Ajustes para cargar el firmware de Crazyflie en Paparazzi	33
2.8.	Conexión del Crazyradio al Crazyflie mediante Paparazzi	34
2.9.	Estructura del firmware de Paparazzi	37
2.10.	Módulo de demostración en Paparazzi	38
3.1.	Comparativa entre la altura conseguida por el PID y el throttle respecto al mismo tiempo. En azul, el throttle, en rojo, la altura. Notación: $h(t)$ altura respecto al tiempo, h_d altura deseada, m_h es el nominal hover throttle y m_d throttle para altura deseada	40
3.2.	Crazyflie haciendo hovering	42
3.3.	Representación gráfica de un vehículo bajo el efecto de un GVF. Imagen extraída de [19] y basado en [18].	43
3.4.	Flow Deck V2 de Bitcraze	45
3.5.	Comparación de dron sin alinear (izquierda) vs alineado (derecha). Si nos fijamos en el triángulo que posee el icono del dron, en el caso de la izquierda siempre mira al norte. En el caso de la derecha, mira hacia el sentido del campo en su posición actual	48

4.1. Ejemplo de segmentos paralelos y normalizados para dos drones	53
4.2. Situación inicial para el ejemplo descrito de formación en segmentos	55
4.3. Ejemplo de simulación de formaciones circulares con drones. Se solapan ambos drones, al estar (como se desea) a 0 grados de error entre ambos	60
4.4. Ángulo entre drones en simulación de formación circular. En rojo el ángulo entre drones deseado	61
4.5. Simulación de formación en segmentos en Paparazzi. Podemos ver los drones que están al lado y en paralelo	62
4.6. Desajuste de la posición sobre el segmento normalizado entre drones en simulación. En rojo el desajuste deseado	62
4.7. Dos drones manteniéndose paralelos. En este caso $p^* = 0$, $v_n =$ 0.2 m/s , $K = 0.12$	67
4.8. Animación del experimento que verifica que el algoritmo fun- ciona	68
4.9. Diferencia de la posición entre los drones. En verde la diferen- cia deseado	68
4.10. Velocidad del dron rojo (simulado). En verde la velocidad no- minal objetivo en ese instante	69
5.1. Resultados reales de la planificación	72

ÍNDICE DE TABLAS

2.1.	Componentes de un Crazyflie	2.1	27
2.2.	Variables de control de un Crazyflie en Crazyflie Client		30
3.1.	Valores PID vertical		41
3.2.	Valores PID horizontal		42

Capítulo 1

INTRODUCCIÓN

En este trabajo nos centraremos en el control y coordinación de varios UAVs, principalmente en la subcategoría conocida como cuadricópteros, popularmente llamados drones.

Se propone como objetivo final conseguir la coordinación entre *Crazyflies*, cuadricópteros ligeros desarrollados por la empresa Bitcraze, mediante la implementación de un firmware de código abierto basado en el proyecto *Paparazzi UAV* [1] y la inclusión de un algoritmo para coordinación.

1.1. Motivación

Los drones Crazyflie son una plataforma ideal para explorar conceptos de control, comunicación y coordinación en robótica. Este TFG permitirá desarrollar habilidades prácticas en programación, electrónica y sistemas embedidos, así como aplicar teorías complejas de control y algoritmos de coordinación en un entorno real y simulado paralelamente.

Los drones tienen numerosas aplicaciones prácticas, desde la vigilancia y el rescate hasta la agricultura de precisión o entrega de mercancías. Desafortunadamente, existen limitaciones tanto hardware como software en el mundo de la robótica, por ejemplo, las limitaciones de batería o el simple control por *waypoints* de un dron.

Muchas de estas limitaciones se pueden sobreponer aplicando enjambres de drones, permitiendo que las aplicaciones prácticas actuales puedan expandirse e incluso permitir nuevas aplicaciones que previamente eran impensables con el control limitado de un sólo dron.

En general, este TFG tiene como objetivo el avance en el control y, en es-

pecial, coordinación entre sistemas de dos o más robots, aplicados al entorno de los drones.

1.2. Descripción general del sistema

Para la coordinación entre dispositivos necesitaremos como mínimo dos o más Crazyflies, un accesorio para localización como, por ejemplo, el de Bitcraze por *anchors* para los Crazyflies y un dispositivo de control central, generalmente un portátil con un software para el control de los UAVs.

Los términos **UAV**, **rotorcraft** y **dron** se utilizarán indistintamente para referirnos, por lo general, a los Crazyflies u otro dispositivo similar.

En las siguientes subsecciones detallaremos cada componente del sistema individualmente, divididos por hardware y software.

1.2.1. Hardware: Crazyflies y accesorios

Se trabajará concretamente con los **Crazyflie 2.1**, drones de tan solo 27 gramos y dimensiones 92x92x29mm [2]. Su bajo peso y tamaño implican que tan solo es deseable usarlos en espacios interiores. Entre sus características más destacables, se encuentra la capacidad de sustituir el firmware oficial, consecuencia de ser un dispositivo *open source* y lo que nos permitirá que este trabajo pueda llevarse a cabo.



Figura 1.1: Imagen de un Crazyflie 2.1 montado

Aunque los Crazyflies se pueden utilizar independientemente con un mando radiocontrol o un portátil con el conjunto del software y firmware oficial, el objetivo es la coordinación de movimientos entre los drones, por tanto necesitamos conocer la posición relativa o absoluta de todos los drones en tiempo real. Por ejemplo, Bitcraze dispone de unos módulos basados en UWB (*Ultra WideBand*, banda ultraancha en español) que permiten la localización de los Crazyflies en el espacio con buena precisión. Este sistema, llamado **Loco Positioning System** se divide en dos componentes [3]:

- **Loco Positioning Node:** conocidos como *anchors* al estar estáticos en una sala como puntos de referencia. Se sitúan al menos 6 para permitir la triangulación en tres dimensiones de la posición de un Crazyflie. Utilizan los módulos de UWB Decawave DWM1000, ampliamente usados y conocidos módulos de UWB.
- **Loco Positioning Deck:** módulo de expansión para un Crazyflie, que le permite comunicarse con las *anchors*. También utiliza los Decawave DWM1000.



Figura 1.2: Loco Positioning Deck (izquierda). Loco Positioning Node (derecha)

Alternativamente, tenemos otro tipo de accesorio simplificado que, si bien tiene menos precisión, es menos dependiente de infraestructura externa. Se detallará en el capítulo 3, al no cobrar sentido su uso hasta ese capítulo.

1.2.2. Software: Paparazzi UAV

Las pruebas iniciales con el sistema serán con el conjunto del firmware oficial para los Crazyflies y el software oficial para ordenador. Una vez se verifique el correcto funcionamiento de cada dron, se pasará al firmware de Paparazzi para los Crazyflies. Junto al software de control, **Paparazzi Center** y **Paparazzi Ground Control Station** [4] se completa la configuración e instalación del sistema para el control y coordinación de los UAV.

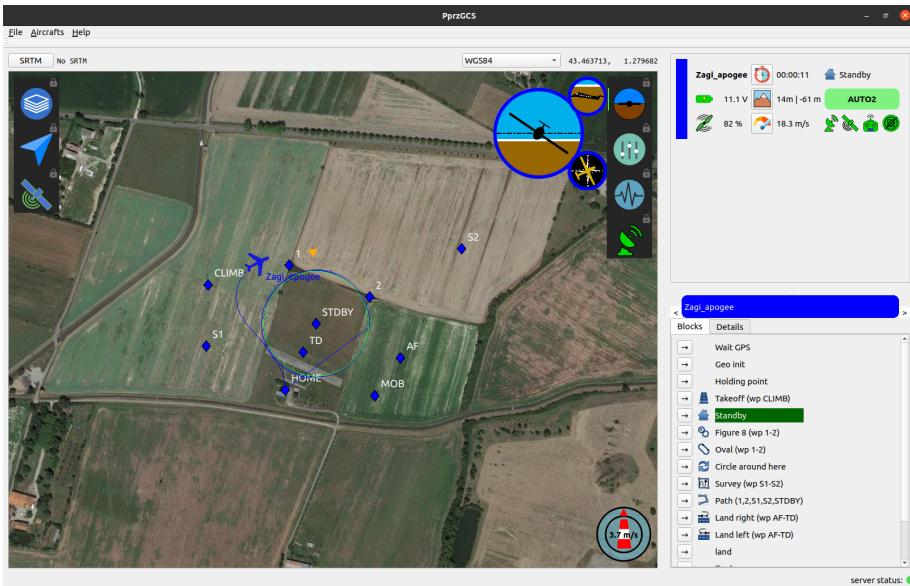


Figura 1.3: Captura de la interfaz de Paparazzi GCS

Para conseguir un funcionamiento completo del sistema, se necesita la implementación del posicionamiento en Paparazzi. Actualmente solo se dispone del firmware para los Crazyflie [5] el cual únicamente permite un funcionamiento básico sin posicionamiento preciso, al depender de exclusivamente los valores del acelerómetro.

1.3. Objetivos

Dada la descripción general del sistema, ya es posible entender los objetivos de este trabajo de fin de grado. Los objetivos, ordenados por capítulos y en orden cronológico de desarrollo, son los siguientes:

- **OBJ-1:** introducción, objetivos, planificación y preparación general. Preparación para la realización de este TFG, incluyendo la realización de este capítulo, familiarización con el ámbito, preparar la plantilla de LaTeX...
- **Capítulo 2 (OBJ-2):** preliminares. Se trata de verificar que el hardware funciona de forma correcta siguiendo los pasos del fabricante, así como una breve familiarización con el control básico de un rotorcraft. Tras verificación con el firmware y software oficial, se realizarán las mismas pruebas básicas con Paparazzi.

- **Capítulo 3 (OBJ-3):** control de un Crazyflie. Se conseguirá un control mínimo y estabilizado de un solo Crazyflie (*hovering*). Se usará un algoritmo de seguimiento de trayectorias que será implementado en el firmware de Paparazzi.
- **Capítulo 4 (OBJ-4):** coordinación entre Crazyflies. Se añade al objetivo 3 la capacidad de coordinar dos o más drones para seguir trayectorias de forma coordinada (formaciones).
- **Capítulo 5 (OBJ-5):** terminamos este TFG con las conclusiones extraídas, resultados obtenidos, análisis de la planificación y objetivos cumplidos.
- **OBJ-6:** Revisión y mejora general previo de este trabajo. Afecta a todos los objetivos. Se añade la realización completa de la presentación en diapositivas y la finalización de esta memoria.

Resumidamente, se trata de conseguir que los Crazyflies puedan seguir trayectorias de forma coordinada utilizando el conjunto de firmware y software *open source* de Paparazzi UAV.

1.4. Planificación

Para cumplir los objetivos provistos se ha realizado la siguiente planificación:

Planificación General																			
Febrero				Marzo				Abril				Mayo				Junio			
W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4
OBJ-1																			
	OBJ-2																		
		OBJ-3										OBJ-4				OBJ-5			
																OBJ-6			

Figura 1.4: Planificación de este trabajo

Por supuesto, esta planificación es aproximada y flexible hasta cierto punto. Debido a la naturaleza de este trabajo, es difícil prever con exactitud el correcto desarrollo de la planificación, al requerir un estudio constante de una tecnología en desarrollo con multitud de opciones, así como la dependencia de un proyecto de ámbito *open source*, en constante evolución con los posibles problemas y contratiempos que ello puede ocasionar, por ejemplo, las actualizaciones de este.

En el último capítulo, se hará una comparación entre esta planificación y el resultado final de esta, con objetivo de entender que ha pasado durante el desarrollo de este trabajo y si el desarrollo ha ido como fue planeado.

Capítulo 2

PRELIMINARES

Tras una breve introducción a los dispositivos que se usarán para el desarrollo de esta obra, procederemos a las primeros pasos necesarios para la configuración y control de los drones.

En adelante, se detallan los procedimientos para lo que es conocido como *Maiden Flight*, el primer vuelo de cualquier tipo de vehículo aeronáutico o aeroespacial. Posteriormente, tras las pruebas con el firmware y software oficiales, se realizarán las mismas pruebas con el firmware del proyecto Paparazzi UAV.

2.1. Montaje de los drones

Cada Crazyflie 2.1 necesita de montaje manual. Para ello, se ha seguido la guía oficial del fabricante [6]. Los componentes de un Crazyflie necesarios para el montaje son los siguientes:

Componente	Cantidad
Crazyflie 2.1 Board	1
Hélices tipo CW	2
Hélices tipo CCW	2
Anclajes para motores	4
Motores DC	4
Batería LiPo	1
Anclaje para batería	1
Pines de expansión	1

Tabla 2.1: Componentes de un Crazyflie 2.1

Además, podemos ver en esta figura los componentes de un Crazyflie previo a su montaje:

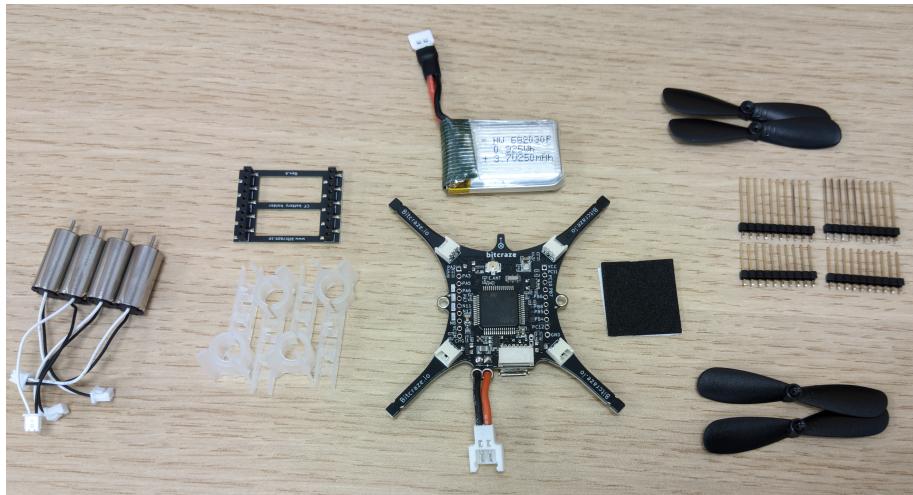


Figura 2.1: Componentes de un Crazyflie 2.1

El procedimiento de montaje, a efectos simplificados, es el siguiente. Se sigue, como bien se ha indicado previamente, la guía oficial [6]:

- Montaje de los motores en cada anclaje para estos.
- Insertar los anclajes en el Crazyflie 2.1 y conectar los motores eléctricamente a este.
- Añadir las hélices a los motores, diferenciando las CW (*ClockWise*) de las CCW (*CounterClockWise*).
- Montar los pines de expansión y la batería.
- Anclar la batería utilizando el anclaje de la batería, que se conecta a los pines de expansión.

2.2. Pruebas con software oficial

Con un Crazyflie montado, podemos proceder a las pruebas iniciales con el conjunto de software y firmware oficial. Desde la guía oficial de montaje se recomienda utilizar la máquina virtual preconfigurada por Bitcraze para conseguir volar el rotorcraft lo antes posible [7]. En esta máquina virtual se incluye los programas para el control, código fuente y otros recursos software de utilidad para el desarrollo con Crazyflies.

2.2.1. Preparación

Previo a la conexión con un Crazyflie, debemos de cargar el firmware al Crazyradio 2.0, el módulo USB que nos permite controlar remotamente un Crazyflie y recibir telemetría de este. Para ello, se sigue la guía oficial del fabricante [8].



Figura 2.2: Crazyradio 2.0

Con ello, ya podemos conectarnos a un Crazyflie desde el software de control Crazyflie Client, no obstante, se recomienda actualizar el firmware del Crazyflie 2.1 previamente utilizando Crazyflie Client. Se sigue la guía oficial de actualización de firmware [9].

2.2.2. Control básico con Crazyflie Client

El último paso previo a cambiar al proyecto Paparazzi consiste en familiarizarse con el control básico de un solo Crazyflie mediante el software oficial. De esta forma se confirma también el correcto funcionamiento de cada dron, asegurando que no es fallo de firmware/software no oficiales (Paparazzi, en este caso). Aquí podemos ver la interfaz de Crazyflie Client:

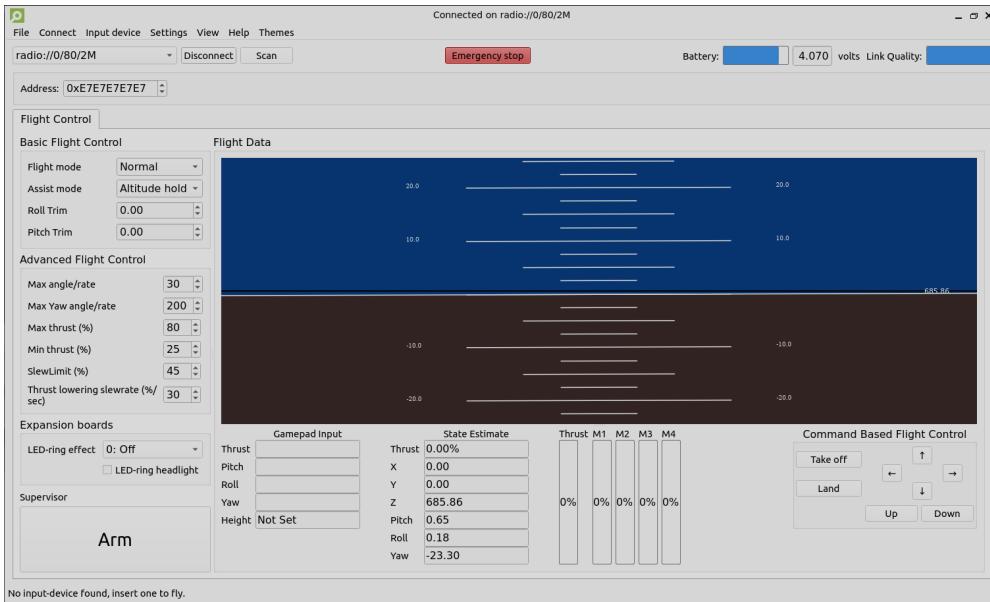


Figura 2.3: Interfaz de Crazyflie Client

2.2.3. Variables de control en Crazyflie Client

Previo a continuar con el desarrollo de este trabajo, es importante entender las variables básicas que definen como se desarrolla el vuelo de un rotorcraft. Aunque existen más, introduciremos aquí todas las variables que Crazyflie Client nos permite saber de un Crazyflie.

En la siguiente tabla se detallan las variables, así como la unidad del Sistema Internacional que se utiliza. Se indican unidades alternativas entre paréntesis, muy utilizadas en otras situaciones o programas.

Variable	Descripción	Unidad
Roll	Rotación sobre el eje X	Grados (Radianes)
Pitch	Rotación sobre el eje Y	Grados (Radianes)
Yaw	Rotación sobre el eje Z	Grados (Radianes)
X	Posición en el eje X	m
Y	Posición en el eje Y	m
Z	Posición en el eje Z	m
Height	Altura respecto al suelo	m
Thrust	Empuje de los motores	%
Battery	Nivel restante de batería	V (%)

Tabla 2.2: Variables de control de un Crazyflie en Crazyflie Client

Cabe destacar las variables **roll**, **pitch** y **yaw** que indican la rotación de un objeto respecto a alguno de sus ejes. Son variables muy usadas en el ámbito de control de vehículos aéreos y aeroespaciales. La siguiente imagen describe de forma gráfica cada una de estas variables.

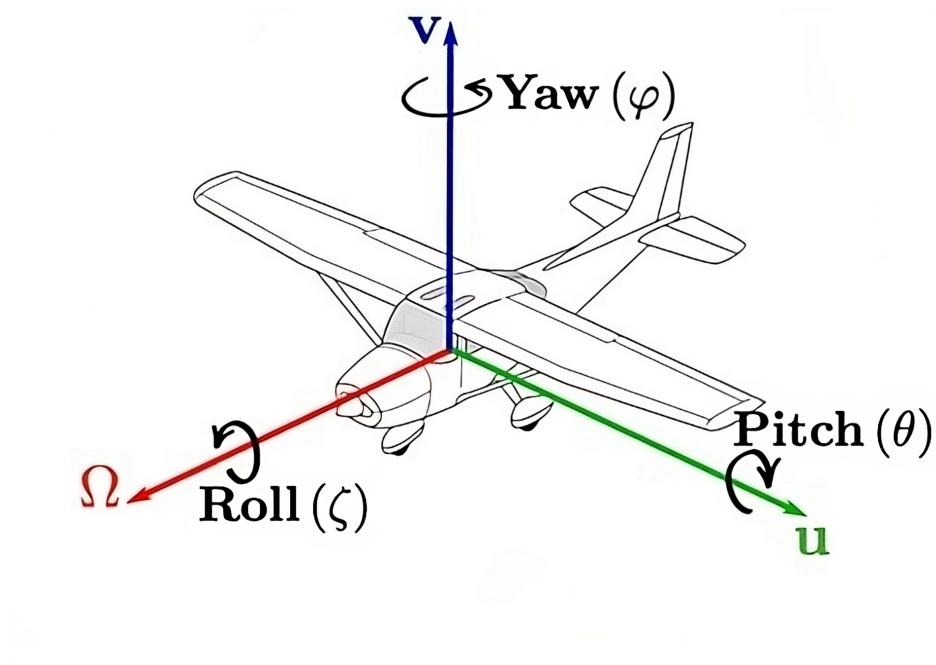


Figura 2.4: Ejes de rotación de un vehículo aéreo

2.3. Pruebas con Paparazzi UAV

Realizadas el montaje y pruebas iniciales, el siguiente paso consiste en la familiarización con el ecosistema Paparazzi. Se instalará Paparazzi Center y acto seguido se realizará una breve simulación de vuelo en Paparazzi GCS. Tras ello, se cargará el firmware de Paparazzi para un Crazyflie y se realizará la primera prueba real.

2.3.1. Paparazzi Center

La instalación de Paparazzi Center se ha realizado siguiendo la guía oficial [10], con leves modificaciones para adaptarlo a Debian 12 (véase Apéndice A). En la siguiente imagen podemos ver la interfaz de Paparazzi Center en Debian 12 con KDE Plasma 5:

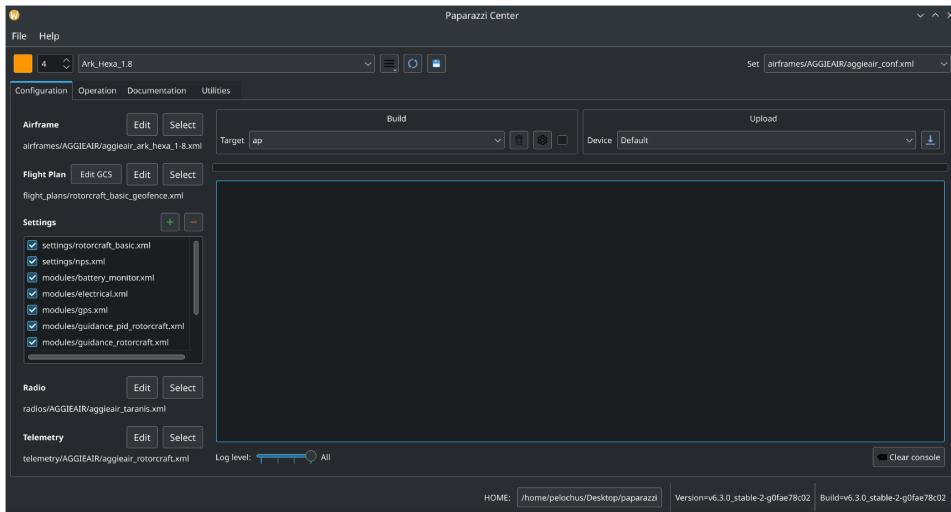


Figura 2.5: Captura de Paparazzi Center bajo Debian 12 con KDE

Desde Paparazzi Center se puede realizar simulaciones, planes de vuelo, observar mensajes de la consola, seleccionar el UAV deseado y un largo etcétera de posibilidades. Se utilizará principalmente los paneles de operación de vuelo, simulación de vuelo, ajustes y consola para depuración.

2.3.2. Simulaciones en Paparazzi Center

Siguiendo la documentación de Paparazzi podemos realizar la primera simulación [12]. Seguir los pasos resulta en la apertura de Paparazzi GCS.



Figura 2.6: Paparazzi GCS ejecutando la simulación por defecto

Desde Paparazzi GCS tenemos un sinfín de posibilidades más que desde Crazyflie Client. Cabe destacar, por ejemplo, la capacidad realizar simulaciones en una zona del planeta cualquiera. Esto se puede aplicar a las misiones de vuelo reales, es decir, mediante GPS indicar en qué zona estamos volando.

Por otro lado, tenemos la capacidad de observar una gran cantidad de variables, incluso personalizar cuáles queremos ver y cuáles no, controles y órdenes de todo tipo, capacidad para elegir *waypoints* etc...

2.3.3. Firmware de Paparazzi para Crazyflie 2.1

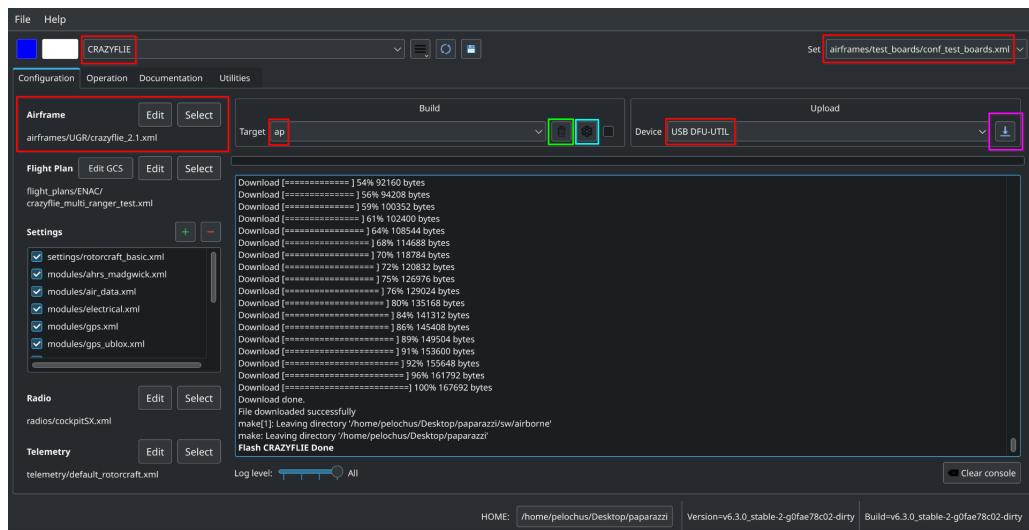


Figura 2.7: Ajustes para cargar el firmware de Crazyflie en Paparazzi

Se seguirá la guía oficial para cargar el firmware [5]. Previamente, es necesario utilizar una versión previa del firmware oficial para conseguir compatibilidad con el firmware de Paparazzi [15]. Este procedimiento consiste en los siguientes pasos:

1. Se instalan las dependencias necesarias. Se encuentran el repositorio de este TFG [17]. Para un lector que no vaya a hacer pruebas, se recomienda leer el Apéndice B, que contiene más detalles.
2. Se mantiene presionado el botón de encendido del Crazyflie mientras se conecta por USB. Esto permite entrar en el modo DFU, que permite cambiar el firmware.
3. Desde Paparazzi se carga el firmware acorde con la Figura 2.7:

- Se ajustan los parámetros que estan en **rojo** de la misma forma.
- Se pulsa el botón resaltado en **verde**, para limpiar archivos, análogo de Paparazzi a hacer un *make clean* al compilar un ejecutable.
- Se pulsa el botón resaltado en **cyan**, para autocompilar el firmware. De nuevo, podemos hacer analogía con *make*.
- Se pulsa el botón en **morado** para cargar finalmente el firmware en el Crazyflie. Esto no debería fallar si se han seguido correctamente los pasos para instalar dependencias.

Por último, se debe probar que Paparazzi funciona en el Crazyflie. Se probará la conexión mediante Crazyradio. Se siguen los siguientes pasos:

- Para empezar, es necesario haber instalado las dependencias. Se incluyen en el previamente mencionado Apéndice B.
- Siguiendo la Figura 2.8 podemos conectar un Crazyflie a Paparazzi Center usando la Crazyradio. Se siguen los siguientes pasos:
 - Se añade uso del Crazyradio 2.0 mediante el menú *Add Tool - Crazyradio/IvyBridge*. Esto añade la configuración marcada en **rojo**.
 - Similarmente, desde el menú *Add Tool - Messages* se añade la configuración marcada en **verde**. Además aparece la ventana señalada en **morado** tras seleccionar el módulo, que muestra la recepción y envío de datos.

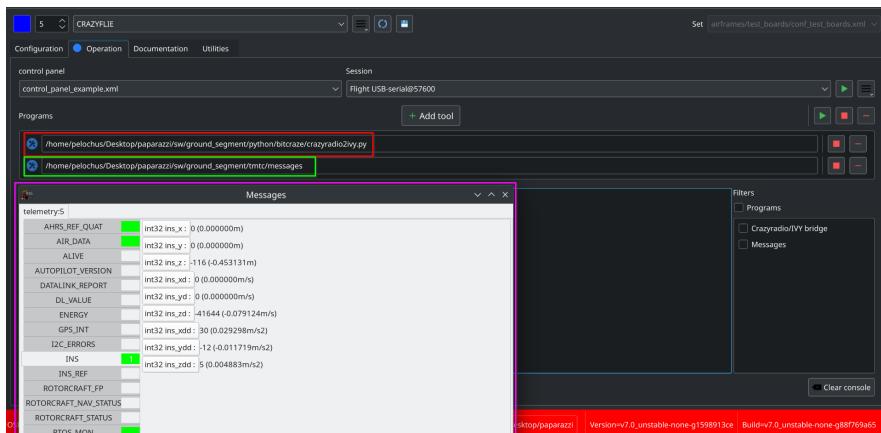


Figura 2.8: Conexión del Crazyradio al Crazyflie mediante Paparazzi

2.4. Desarrollo básico en Paparazzi

Es importante entender como se desarrolla módulos de control en Paparazzi. Los módulos se ocupan de controlar el UAV correspondiente de la forma más genérica posible, permitiendo que varios UAVs distintos puedan utilizar un mismo módulo. Además, permiten adaptar la interfaz del GCS para, por ejemplo, colocar un deslizador que permita cambiar el control de una variable como la velocidad de un UAV.

En esta sección se desarrollará un pequeño módulo de ejemplo según la guía oficial [14] de Paparazzi, así se puede entender mejor como se desarrollará este TFG.

2.4.1. Desarrollo de módulos de control

Paparazzi define los módulos en XML. Este formato permite una estructura clara y modular para implementación de módulos pero no permite el desarrollo de código. Para solucionar esto, el código de los módulos se implementa en el lenguaje C, en el cual se puede implementar algoritmos de control o manejar datos de, por ejemplo, sensores. Estos XML se encuentran en `./conf/modules/`.

La principal utilidad de los módulos aparece al compilar un firmware, donde se indica que módulos se usarán para el control del dispositivo. Por ejemplo, un dron con GPS y acelerómetro podrá incluir el módulo de control GPS de Paparazzi y de uso del acelerómetro. También existen módulos genéricos, que permiten para un tipo de dispositivo, incluir los módulos básicos. Para un rotorcraft, será por lo general módulos de control de 4 motores, IMU (*Inertial Measurement Unit* que incluye acelerómetro y giroscopos), control por radio etc...

Aquí vemos el módulo de demostración de Paparazzi.

```

1 <!DOCTYPE module SYSTEM "module.dtd">
2
3 <module name="demo_module">
4   <doc>
5     <description>Demo module</description>
6   </doc>
7   <header>
8     <file name="demo_module.h"/>
9   </header>
10  <init fun="init_demo()"/>
11  <periodic fun="periodic_1Hz_demo()" freq="1." start="start_demo()" stop=
12    "stop_demo()" autorun="TRUE"/>
13  <periodic fun="periodic_10Hz_demo()" period="0.1" start="start_demo()"
14    stop="stop_demo()" autorun="FALSE"/>
```

```

13   <makefile>
14     <raw>
15 #Example of RAW makefile part
16   </raw>
17     <define name="DEMO_MODULE_LED" value="2"/>
18     <file name="demo_module.c"/>
19 </makefile>
20 <makefile target="demo">
21   <define name="SOME_FLAG"/>
22   <configure name="SOME_DEFINE" value="bla"/>
23 </makefile>
24 </module>

```

Por otro lado, es importante entender que la programación de los módulos esta en C. Estos archivos, por su parte, se encuentran en `./sw/airborne/modules/`. Por ejemplo, veamos el código C para el módulo de demostración:

```

1 #include "demo_module.h"
2 #include "led.h"
3
4 void init_demo(void)
5 {
6   // this part is already done by led_init in fact
7   LED_INIT(DEMO_MODULE_LED);
8   LED_OFF(DEMO_MODULE_LED);
9 }
10
11 void periodic_1Hz_demo(void)
12 {
13   LED_TOGGLE(DEMO_MODULE_LED);
14 }
15
16 void periodic_10Hz_demo(void)
17 {
18   LED_TOGGLE(DEMO_MODULE_LED);
19 }
20
21 void start_demo(void)
22 {
23   LED_ON(DEMO_MODULE_LED);
24 }
25
26 void stop_demo(void)
27 {
28   LED_OFF(DEMO_MODULE_LED);
29 }

```

En conjunto, ambos archivos previos mostrados permiten generar automáticamente el encendido y apagado de un LED de forma periódica en un sistema usando el firmware de Paparazzi. Como bien se ha explicado antes, se puede diferenciar de forma que el archivo C permite generar la lógica mínima de control (en este caso, de los LEDs) y el archivo XML añade documentación, ajustes para compilación, frecuencia o periodicidad de cada función... entre otras funciones.

La razón del uso de archivos de XML, además de modularidad y claridad, es debido al uso extendido de ChibiOS en Paparazzi [16], un sistema operativo de tiempo real que se ocupa de ejecutar las funciones de forma periódica como bien permite definir el XML, utilizando, como es propio de un sistema RT, diversos hilos de ejecución.

Existe la posibilidad de no usar ChibiOS y utilizar un programa en bucle, no obstante, no es el comportamiento más usado, ni generalmente preferible. La plataforma **Crazyflie 2.1 en Paparazzi** utiliza ChibiOS.

Finalmente, hay que añadir al archivo de configuración del firmware del Crazyflie en Paparazzi este pequeño trozo de código para que, en este caso el Crazyflie, pueda ejecutar el módulo de demostración:

```
1 <module name="demo_module" />
```

En general, se puede resumir la estructura de los módulos y como se ubican en Paparazzi con el siguiente diagrama:

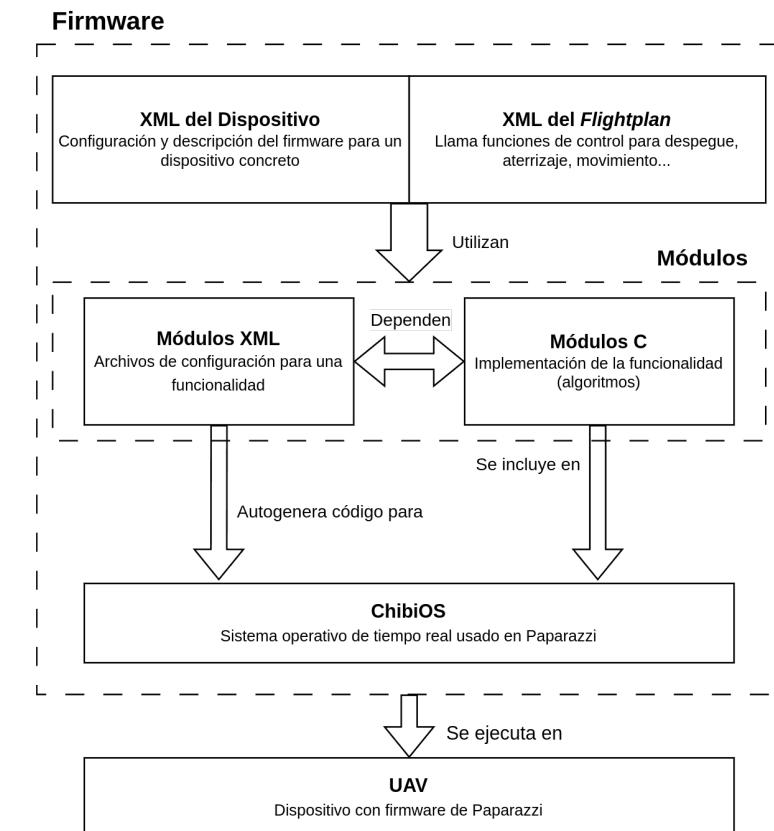


Figura 2.9: Estructura del firmware de Paparazzi

2.4.2. Uso de módulos en una simulación

En esta sección se verá en ejecución el módulo de demostración de la sección anterior. Los módulos permiten añadir comportamientos y funcionalidades extra a los dispositivos con Paparazzi. Normalmente, este comportamiento se puede observar desde Paparazzi GCS. En la siguiente figura podemos ver el módulo de demostración en Paparazzi GCS.

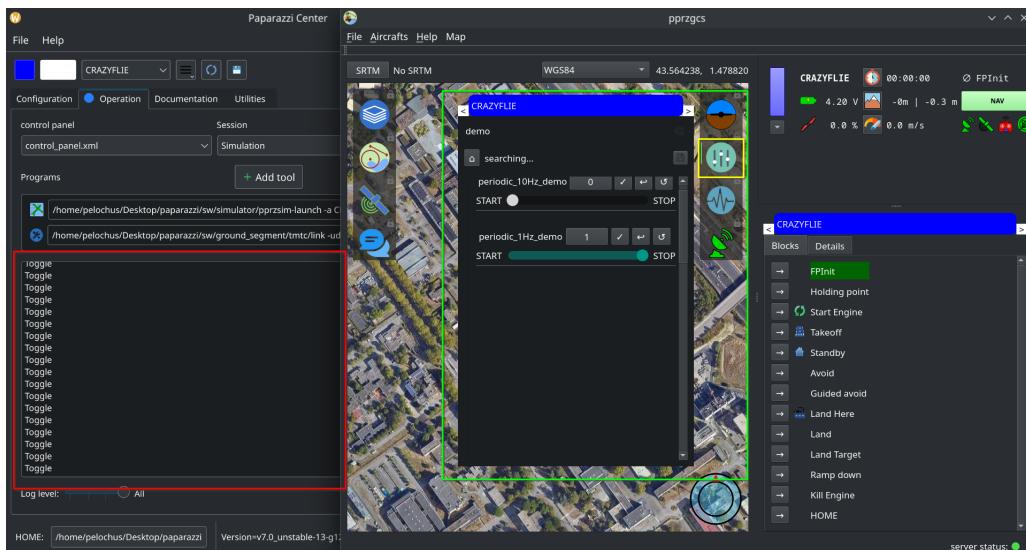


Figura 2.10: Módulo de demostración en Paparazzi

Si nos fijamos en la figura, podemos dividir por las siguientes partes:

1. Se asume que se parte de una simulación, con Paparazzi GCS ya ejecutándose
2. Desde el menú rodeado en color verde, podemos ajustar parámetros del módulo. Para mostrar este menú se presiona el botón rodeado en amarillo.
3. Por último, podemos ver los resultados en la terminal rodeada en rojo. Al ser una simulación, simplemente se imprime por pantalla lo que se supone que se está haciendo, en este caso cambiar el LED de estado cada segundo.

Con los Crazyflies montados, funcionales con Paparazzi y entendido el funcionamiento básico de este, el siguiente capítulo consistirá en empezar el desarrollo para este dron en el marco de Paparazzi.

Capítulo 3

CONTROL DE UN CRAZYFLIE

Dado un conocimiento básico de Paparazzi, este capítulo se centra en la implementación de los algoritmos de control para un sólo Crazyflie. Se realizarán implementaciones parcialmente agnósticas del posicionamiento entre drones, al no ser necesaria la coordinación entre drones.

Se entenderá el funcionamiento de un algoritmo GVF y se verá su implementación en Paparazzi para el Crazyflie, terminando con un buen control de este.

3.1. Control básico

Previo a conseguir control con algoritmos avanzados, se debe conseguir *hovering*, es decir, que el dron flote estático a cierta altura. Debido a que la implementación de Paparazzi actual esta enfocada a control manual con mando, el PID no está debidamente ajustado. Conseguir hovering se reduce a un correcto ajuste del PID vertical y horizontal, al estar el control por PID ya implementado en el firmware del Crazyflie.

3.1.1. PID en Paparazzi

Paparazzi reduce el control de un rotorcraft a dos controladores PID: **uno vertical (eje Z) y otro horizontal (plano y ejes XY)**. El control PID en Paparazzi no difiere mucho de un control PID estándar, no obstante, cabe destacar una leve diferencia en el control PID vertical:

$$m(t) = m_h + K_p e(t) + K_i \int_0^t e(t) dt + K_d \frac{de(t)}{dt} \quad (3.1)$$

Donde:

- $m(t)$ es el throttle (velocidad de los motores de 0 a 100 %) respecto al tiempo
- m_h es el *Nominal Hover Throttle*, es decir, el valor mínimo de throttle deseado. Se fija a un valor muy levemente por debajo del valor mínimo para que un Crazyflie despegue. Se encontró que un Crazyflie despegue con un throttle de **entre 65 % y 70 %**
- $e(t)$ es el error respecto a la altura deseada

Nuestra principal diferencia, como se puede observar, reside en el Nominal Hover Throttle. Con este valor se consigue que el dron parta de un valor inicial de throttle con sentido, es decir, que le permita despegar del suelo. La siguiente figura ilustra perfectamente la idea recién explicada:

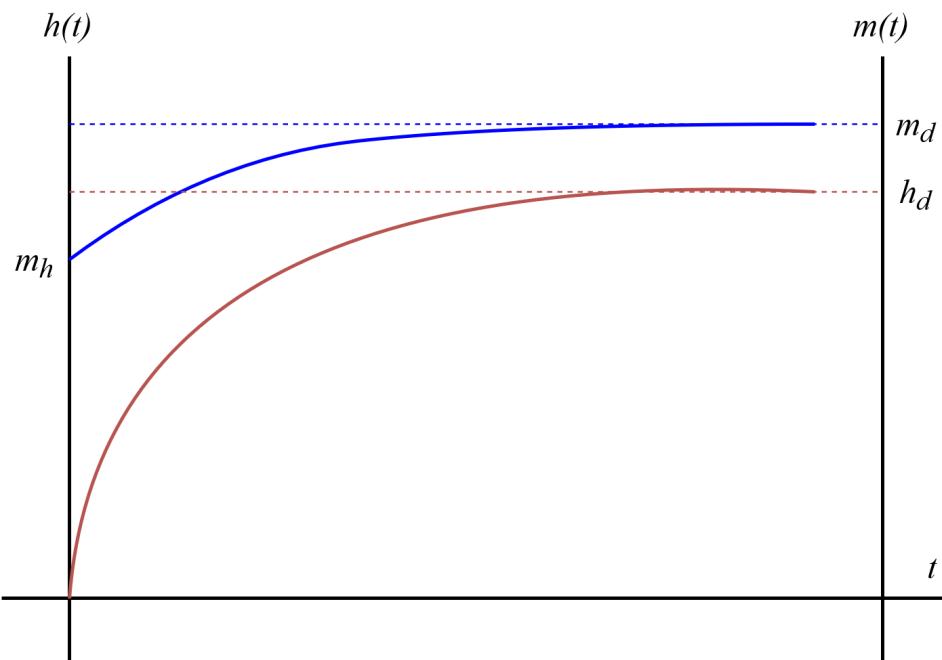


Figura 3.1: Comparativa entre la altura conseguida por el PID y el throttle respecto al mismo tiempo. En azul, el throttle, en rojo, la altura. Notación: $h(t)$ altura respecto al tiempo, h_d altura deseada, m_h es el nominal hover throttle y m_d throttle para altura deseada

3.1.2. Metodología para ajuste del PID

Idealmente, se puede automatizar el ajuste de un PID con un *testbench* para drones, que permite automáticamente ajustar los valores más precisos de manera automática. No obstante, las capacidades de telemetría de Paparazzi permiten diagnosticar manualmente el resultado de un control PID. En general, se siguió la siguiente metodología:

- **Ajuste del control proporcional:** se ajusta el control proporcional hasta que se consiga que el dron despegue hasta una altura no muy lejana del objetivo, con error constante.
- **Ajuste del control integral:** se añade lentamente constante integral hasta que se consiga ver que esta compensa los errores del proporcional y oscila alrededor de la altura deseada.
- **Ajuste del control derivativo:** se ajusta la constante derivativa hasta reducir el overshoot inicial lo suficiente sin que desestabilice el control PI.

Para averiguar el nominal hover throttle se hizo a ensayo y error, debido a que es fácil averiguarlo: probar numeros de 0 a 1 (0 a 100 %) hasta que el dron de pequeños saltos.

3.1.3. Hovering

En la siguiente tabla tenemos los valores del PID para el controlador vertical. Como en cualquier control de tipo PID, tenemos las constantes proporcionales, derivativas e integrales. Se añade en este caso el *nominal hover throttle*, como bien se ha explicado previamente.

Variable	Valor
K_p	175
K_i	30
K_d	70
m_h	0,625

Tabla 3.1: Valores PID vertical

Paralelamente, tenemos en la siguiente tabla los valores para el PID horizontal, que se ocupa de evitar que el Crazyflie se mueva en los ejes X e Y.

Variable	Valor
K_p	50
K_i	20
K_d	125

Tabla 3.2: Valores PID horizontal

Con ambos PID bien ajustados, se consigue que el dron se pueda mantener estático a una altura dada:

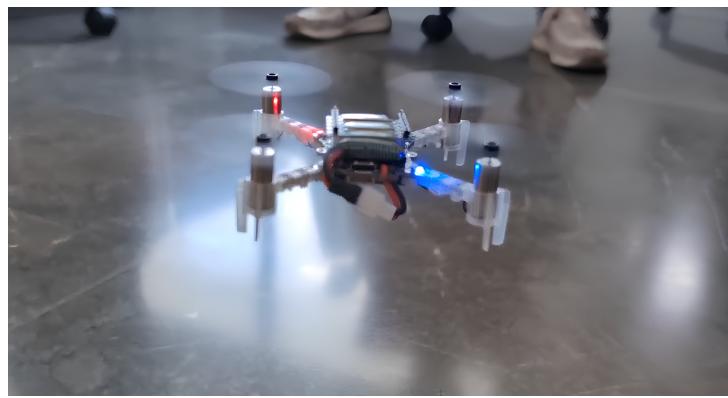


Figura 3.2: Crazyflie haciendo hovering

3.2. Algoritmos GVF

Entre la variedad de algoritmos de control que se puede implementar, se ha elegido un algoritmo GVF (*Guidance Vector Field*). Este tipo de algoritmo permite recorrer trayectorias en 2D mediante la creación de un campo vectorial que indica velocidad y trayectoria del vehículo en un punto dado. La base matemática de este algoritmo permite bajo consumo de recursos (ideal en sistemas empotrados como el Crazyflie) y adaptabilidad a cualquier figura 2D representable matemáticamente.

3.2.1. Fundamentos matemáticos

Como bien se ha explicado previamente, un algoritmo GVF trata de recorrer un camino o trayectoria mediante un campo vectorial. Matemáticamente, podemos definir un camino $\mathcal{P} \subseteq \mathbb{R}^2$ donde $\varphi(p) : \mathbb{R}^2 \rightarrow \mathbb{R}$ mediante la siguiente ecuación implícita:

$$\mathcal{P} := \{p : \varphi(p) = 0\} \quad (3.2)$$

Esta representación, extraída de [18] permite ocupar todo el plano \mathbb{R}^2 mediante las curvas de nivel $\{p : \varphi(p) = c\}$, donde la curva que verifica $c = 0$ es la deseada. Con esta representación, a su vez, podemos representar el error de distancia (no tiene porque coincidir con la distancia euclídea) del vehículo respecto a la curva deseada como:

$$e(p) := \varphi(p) \in \mathbb{R} \quad (3.3)$$

Este error, nos permite variar las componentes de un vector de forma que podamos, en función de como de grande sea $e(p)$, incrementar o decrementar la *agresividad* con la cual el vehículo se aproximará a la curva de nivel deseada. Ello, junto a una constante k_e que permitirá variar la agresividad general, concluye en el siguiente campo vectorial de guiado propuesto por los autores de [18]:

$$\dot{p}_d := \tau(p) - k_e e(p) n(p) \quad (3.4)$$

Donde \dot{p}_d es el vector de velocidad deseada, $\tau(p)$ es la tangente de este vector y $n(p)$ es la componente normal del mismo vector. La siguiente figura explica gráficamente todo lo explicado hasta ahora:

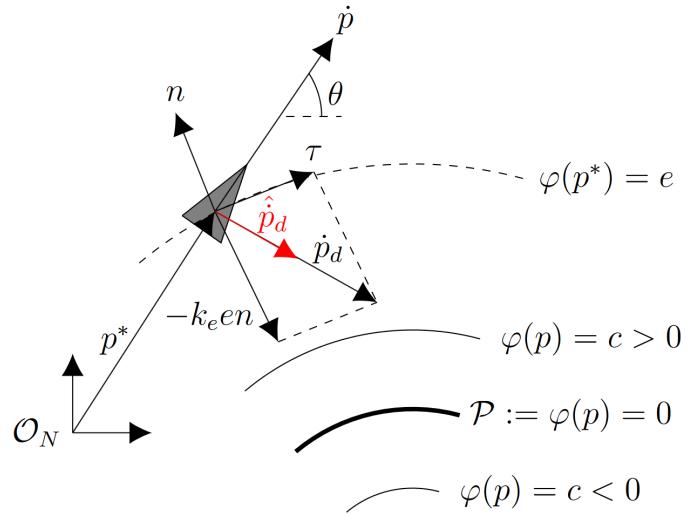


Figura 3.3: Representación gráfica de un vehículo bajo el efecto de un GVF. Imagen extraída de [19] y basado en [18].

En esta figura se debe entender también lo siguiente:

- p^* es el punto en el plano donde se encuentra el vehículo.
- \dot{p} es la velocidad actual del vehículo.
- \mathcal{O}_N es el origen de coordenadas.
- $\hat{\dot{p}}_d$ es el vector con módulo normalizado a la unidad (ocasionalmente interesa solo dirección y sentido del vector) de la velocidad deseada.
- Tanto n como τ son calculadas mediante $\nabla\varphi(p^*)$.

Como podemos observar en la figura, el resultado \dot{p}_d tiende hacia la curva $\varphi(p) = 0$, es decir, el camino deseado \mathcal{P} . Es intuitivo observar que, variando la constante k_e y en función de la curva de nivel e , el resultado de \dot{p}_d será más normal (perpendicular) respecto al camino deseado \mathcal{P} , aumentando la previamente llamada agresividad.

3.2.2. Algoritmos GVF para rotorcrafts

Aunque GVF se puede aplicar a cualquier vehículo, los autores de [18] continúan aportando una solución específica para un *fixedwing UAV* el cual solo puede modificar sus ángulos de *roll* y *pitch* haciendo un movimiento usando la **velocidad angular** (ω).

Por contraparte, un rotorcraft puede modificar su *yaw* y puede realizar movimientos más bruscos debido a su capacidad de moverse lateralmente, permitiendo mucho mejor control en general. Se propone utilizar las **velocidades en los ejes X e Y** para control de los motores del dron, ya que las velocidades son ya calculadas en GVF, se consigue una aplicación directa de una de las variables ya calculadas.

Paparazzi incluye un algoritmo de control llamado **INDI** (*Incremental Nonlinear Dynamic Inversion*) [20] que se ocupa del control de forma transparente en base a una posición, velocidad o aceleración dada. Gracias a este algoritmo, una vez calculada la aceleración deseada en GVF podemos cambiar una variable en el código para que INDI se ocupe de controlar los motores del dron para conseguir el efecto deseado.

Debido a la flexibilidad de INDI, se podría haber usado posición o aceleración en vez de velocidad, pero se usó la velocidad al ser lo más intuitivo y ser un prerequisito ya calculado para GVF.

3.2.3. Limitaciones de GVF sin posicionamiento

Sin añadir nada a un Crazyflie, las únicas variables que este puede calcular son la aceleración en los ejes X, Y y Z, la presión con un barómetro (que permite hacer aproximación de la altura) y los ángulos de roll, pitch y yaw mediante la IMU que posee [21].

Gracias a la diversas capacidades de Paparazzi, este contiene un módulo llamado **INS** (del inglés, *Inertial Navigation System*), que permite un cálculo aproximado de la velocidad y posición integrando la aceleración. Desafortunadamente, la naturaleza de este algoritmo y de las operaciones de punto flotante que utiliza, acumulan error respecto al tiempo que no se puede corregir sin otro sistema de posicionamiento como el GPS.

Como solución, se va a utilizar uno de los módulos de Bitcraze para el Crazyflie llamado **Flow Deck v2** [22]. Este módulo contiene dos sensores esenciales:

- **Altímetro:** que sustituye las medidas aproximadas del barómetro por unas de mucha mayor precisión.
- **Opticflow:** cámara que permite detectar velocidades en X e Y al estar mirando hacia el suelo. Necesita que el suelo tenga textura, pero es mucho mas precisa que integrar la aceleración dada por la IMU.

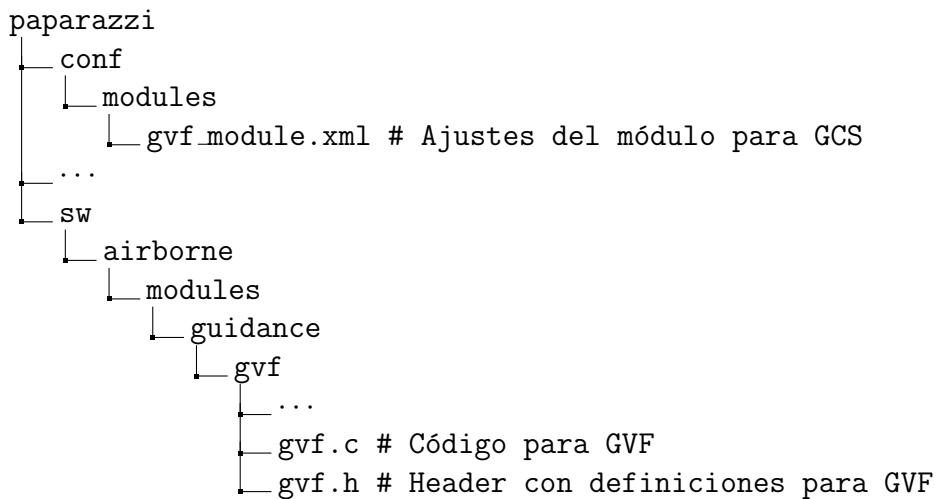


Figura 3.4: Flow Deck V2 de Bitcraze

De esta forma, aunque se sigue manteniendo el error por posición (aún se necesita integrar la velocidad, para conseguir posición), la velocidad y la altura (imprescindibles para realizar GVF) ahora son de mayor precisión.

3.3. Implementación del algoritmo GVF

Paparazzi ya incluye un módulo para GVF, actualmente solo aplicado a fixedwings UAVs y rovers, los cuales son idénticos desde un punto de vista de movimiento en 2 dimensiones (utilizan ambos la velocidad angular en GVF). La estructura de los módulos de GVF en Paparazzi es la siguiente:



Además, viendo este esquema, se puede ver sobre los archivos y carpetas lo explicado sobre la estructura de los módulos de Paparazzi en la Figura 2.9 de forma más visual e intuitiva.

Por ejemplo, tenemos en `conf/modules` la configuración XML del módulo de GVF y en `sw/modules/guidance/gvf` el código (con su correspondiente algoritmia) de GVF.

En este trabajo se trabajará especialmente con los archivos `gvf.c` y `gvf.h`. Las siguientes subsecciones detallarán exactamente de que forma se implementa GVF para rotorcrafts en Paparazzi.

3.3.1. Control mínimo por GVF de un rotorcraft

Como se mencionó previamente, el control de GVF no es agnóstico al dispositivo que lo usa y por tanto, se ha optado por dar un control al Crazyflie comandando velocidades al controlador INDI, que se ocupa de traducir una velocidad dada en throttle de los motores.

La implementación mínima para conseguir GVF en un rotorcraft es tri-

vial, ya que GVF calcula las velocidades en los ejes X e Y y por tanto solo se debe mandar esto a INDI. El resultado se puede ver en el siguiente trozo de código en el archivo `gvf.c`:

```

1 void gvf_control_2D(...) {
2     //
3     // Previous code...
4     //
5
6     #if defined(ROTORCRAFT_FIRMWARE)
7
8         // From sw/airborne/firmware/rotorcraft/navigation.h
9         nav.setpoint_mode = NAV_SETPOINT_MODE_SPEED;
10
11    // md_x and md_y are normalized speeds
12    nav.speed.x = gvf_control.speed * md_x;
13    nav.speed.y = gvf_control.speed * md_y;
14
15    // Optionally align heading with the trajectory
16    if (gvf_control.align)
17        nav.heading = atan2f(md_x, md_y);
18
19    #else // FIXEDWING_FIRMWARE and ROVER_FIRMWARE
20
21    //
22    // Following code...
23    //
24 }
```

En este trozo de código se incluye además parte de la siguiente subsección, pero solo **interesan las líneas 6 a 13**, que son las que se ocupan de mandar la velocidad (calculada previamente en la misma función)

3.3.2. Ampliaciones al módulo de GVF en Paparazzi

Conseguido un funcionamiento mínimo de control con un dron se añaden funciones nuevas para el control de este que no están integradas por defecto en Paparazzi.

GVF con velocidad constante

Utilizando GVF por defecto, la velocidad (\dot{p}_d) va en función de:

- Componente tangencial de la velocidad, τ
- Componente normal de la velocidad, n
- Constante k_e , que modifica la agresividad de la normal

- Curvas de nivel $e(p)$, que aumentan la normal en función de la curva actual

Es evidente que la velocidad no es fácilmente controlable de manera constante sin modificaciones al campo vectorial de guiado. A pesar de ello, existe una solución fácil y simple para conseguir seguir un GVF con velocidad constante sin modificar las variables previamente mencionadas. Esta solución se reduce a comandar lo siguiente:

$$\dot{p}_c := s\hat{p}_d \quad (3.5)$$

Donde \dot{p}_c es la velocidad comandada al dron y s es la velocidad deseada. De esta forma, se consigue seguir el vector deseado sin modificar el campo vectorial, pero consiguiendo una velocidad constante.

Por otro lado, la implementación en el código es trivial. Se añade una función `gvf_set_speed(float speed)` que guarda una variable de velocidad. Posteriormente, en la función general de GVF, se sustituye la velocidad deseada por el vector unitario de esta por la variable de velocidad previamente guardada en la función; en otras palabras la ecuación 3.5.

Alineación con la trayectoria

Los rotorcrafts pueden moverse libremente sobre los ejes X, Y y Z sin afectar a su yaw. Esto permite que un Crazyflie pueda seguir trayectorias en GVF alineándose o no con ella (es decir, que el frontal mire hacia donde se dirige el dron o no).

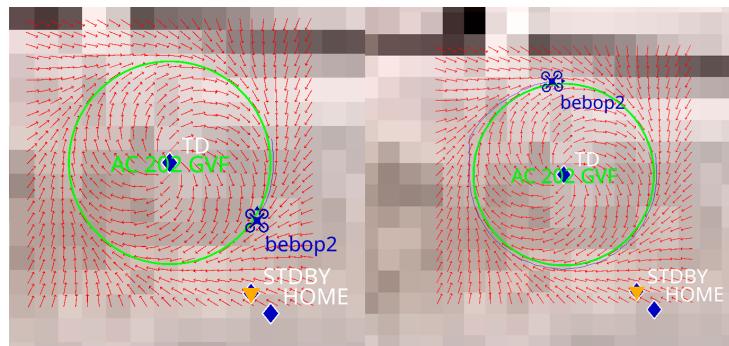


Figura 3.5: Comparación de dron sin alinear (izquierda) vs alineado (derecha). Si nos fijamos en el triángulo que posee el ícono del dron, en el caso de la izquierda siempre mira al norte. En el caso de la derecha, mira hacia el sentido del campo en su posición actual

La imagen previa ilustra perfectamente en simulación las diferencias entre activar la alineación o no. La implementación, por su parte, se realiza de manera similar al ajuste de velocidad:

- Se crea la función `gvf_set_align(bool align)`, que guarda si se desea alinear o no
- Si se desea alinear, en la función principal existe el control de flujo correspondiente para cambiar el comportamiento
- En caso de que si se alinee, se computa el arcotangente de los vectores unitarios de velocidad en X e Y, es decir, $\text{atan2}(\hat{p}_{dx}, \hat{p}_{dy})$. Esto devuelve el ángulo deseado del yaw que será comandado al dron.

Capítulo 4

COORDINACIÓN CON CRAZYFLIES

Conseguido el funcionamiento de GVF para un Crazyflie, se extenderá este funcionamiento de forma que dos o más Crazyflies sean coordinables, es decir, que los recorridos que estos realicen tengan algún tipo de coordinación y dependencia mutua. Dicho de otro modo, formaciones de drones.

4.1. Algoritmos de coordinación

El algoritmo de coordinación elegido es una extensión de GVF para poder realizar formaciones circulares [23]. Además, se propone una variante de este algoritmo para poder realizar GVF coordinado en segmentos paralelos.

4.1.1. Formación circular

Se empezará dando un resumen de la formalización matemática dada por [23]. Para ello, se recomienda entender las nociones matemáticas de GVF previamente descritas en el capítulo 3.

Sea un conjunto de vehículos $n \in \mathbb{N}$ tal que $n \geq 2$ cuyas posiciones están definidas por $p_i \in \mathbb{R}^2$ para $i \in \{1 \dots n\}$ se define el grafo no dirigido $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ donde $\mathcal{V} = \{1 \dots n\}$ es el conjunto de vértices y $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ es el conjunto de aristas. De este grafo, nos interesa la matriz de incidencia $B \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{E}|}$ que es definida de la siguiente forma:

$$b_{ik} \triangleq \begin{cases} +1 & \text{para } i = \mathcal{E}_k^{\text{tail}} \\ -1 & \text{para } i = \mathcal{E}_k^{\text{head}} \\ 0 & \text{en otro caso} \end{cases} \quad (4.1)$$

Donde \mathcal{E}_k^{tail} y \mathcal{E}_k^{head} son los nodos finales e iniciales de una arista $\mathcal{E}_k = (\mathcal{E}_k^{tail}, \mathcal{E}_k^{head})$.

Con esta matriz de incidencia, podemos realizar una topología de enlaces entre vehículos de forma que visualmente se pueda ver en la matriz que UAV toma como referencia un UAV dado. Esto cobrará más sentido conforme se avance en esta subsección.

Recordemos que al estar en un entorno GVF para círculos, nuestras curvas de nivel se rigen por la ecuación $\varphi(p) = p_x^2 + p_y^2 - r^2$ donde $r \in \mathbb{R}^+$ es el radio.

Por otro lado, la coordinación se realizará en base a un ángulo, es decir, cuanto ángulo hay de desfase entre un vehículo y otro. Por ello, el ángulo de un vehículo i en un punto p respecto a $\mathcal{O}_{\mathcal{N}}$ del plano \mathbb{R}^2 viene dado por [24]:

$$\theta_i(p) = \text{atan2}(p_y, p_x) \in (-\pi, \pi] \quad (4.2)$$

Evidentemente, los ángulos que más interesan son los que hay entre un vehículo y otro, no el ángulo respecto a el origen de coordenadas $\mathcal{O}_{\mathcal{N}}$, solo se necesitan para calcularlos. Para ello, los autores de [23] proponen calcular un vector de ángulos entre vehículos de la siguiente forma:

$$z = B^T \theta \quad (4.3)$$

Donde θ es el vector de ángulos calculados usando la Ecuación 4.2 y $z = \{z_1 \dots z_k\}$, $k = |\mathcal{E}|$ es el vector de ángulos entre vehículos. Supóngase ahora un conjunto de ángulos deseados z_k^* , se define e_{θ} como el conjunto de error entre el ángulo actual y el deseado, regido por la ecuación:

$$e_{\theta_k}(t) = z_k(t) - z_k^* \quad (4.4)$$

Se desea que $e_{\theta}(t) \rightarrow 0$ para $t \rightarrow \infty$

Con toda esta base, los autores de [23] terminan proponiendo la siguiente ecuación para control y coordinación.

$${}^i u_r = k_r B_i e \quad (4.5)$$

Donde:

- B_i es la i-ésima fila de la matriz de incidencia
- e es el vector de errores entre ángulos de vehículos
- $k_r \in \mathbb{R}^+$ es una ganancia de control para ajustar el resultado final

- ${}^i u_r$ es la acción de control resultante para el radio r del vehículo i

Finalmente, se aplica sobre el vehículo la acción de control de forma que el radio nuevo sea tal que permita, bajo **velocidad constante**, que los vehículos vayan convergiendo al ángulo deseado. Por tanto, aplicamos lo siguiente para cada vehículo i :

$$r_i = r_i + {}^i u_r \quad (4.6)$$

Resumidamente, el algoritmo consiste en:

- Definir un grafo \mathcal{G} y usar su matriz de incidencia B para definir que drones se coordinarán entre sí
- Calcular el vector de ángulos entre vehículos z en base al vector de ángulos θ respecto a \mathcal{O}_N del plano \mathbb{R}^2
- Calcular el vector de error entre ángulos de vehículos e en base a z y un vector de ángulos deseados z^*
- Calcular las acciones de control u_r en base a e , B y una ganancia de control k_r
- Aplicar la acción de control u_r al radio actual r

Por simplicidad, se han obviado ciertas demostraciones, pasos y definiciones. Si se desea profundizar, se recomienda leer [18] y [23].

4.1.2. Formación en segmentos paralelos

La formación en segmentos paralelos consistirá en dos o más UAVs cada uno siguiendo su propio segmento que serán paralelos entre sí. Es preferible usar segmentos de misma longitud. Por simplicidad en la implementación, se realizará una aplicación de los segmentos de -1 a 1, siendo -1 un extremo y 1 el opuesto como en la figura:

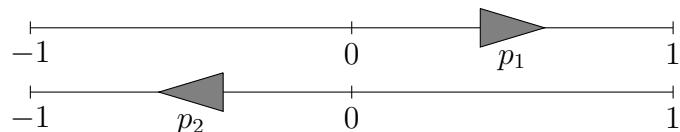


Figura 4.1: Ejemplo de segmentos paralelos y normalizados para dos drones

Para realizar formación coordinada en segmentos paralelos, este trabajo se basará en el modelo Kuramoto [25]. El modelo de Kuramoto se utiliza para describir la sincronización de sistemas oscilatorios bajo la siguiente ecuación:

$$\frac{d\theta_i}{dt} = \omega_i + \frac{K}{N} \sum_{j=1}^N \sin(\theta_j - \theta_i), \quad i = 1 \dots N \quad (4.7)$$

En base a esta ecuación y la Figura 4.1, se propone la siguiente versión adaptada para el control en segmentos paralelos:

$$v_i = v_n + K \sum_{j=1}^N (p_j - p_i - p_{ij}^*), \quad i = 1 \dots N \quad (4.8)$$

Donde:

- $v_i \in \mathbb{R}$ es la velocidad resultante para el dron i
- $v_n \in \mathbb{R}$ es la velocidad nominal (o velocidad base) de los drones
- $K \in \mathbb{R}$ es la ganancia
- $N \in \mathbb{N}$ es el número total de drones
- p_i es la posición del dron sobre el segmento normalizado, por tanto $p_i \in [-1, 1]$, $\forall i$
- p_{ij}^* es el desajuste de la posición deseado entre el dron i y el dron j que valdrá entre $[-2, 2]$. Por ejemplo, para 0, se tiene que los drones han de estar uno al lado del otro y en paralelo.

Intuitivamente, podemos comprender el siguiente algoritmo para el caso $N = 2$, $K = 1$, $p_{12}^* = p_{21}^* = 0$ de la siguiente forma:

- La ecuación resultante para $i = 1$ será la siguiente: $v_1 = v_n + (p_2 - p_1)$
- Para $i = 2$ será: $v_2 = v_n + (p_1 - p_2)$
- Supongamos que $v_n = 1 \text{ m/s}$, que $p_1 = 0.5$ y que $p_2 = 0$
- Supongamos también que el sentido y velocidad de los drones es hacia 1 en sus segmentos, es decir, actualmente están incrementando sus p_i

- Esto da como resultado $v_1 = 0.5 \text{ m/s}$ y $v_2 = 1.5 \text{ m/s}$, en otras palabras, la velocidad del dron 2 es superior y la de 1 inferior para que el dron 2 pueda alcanzar al dron 1 lo antes posible. Sus velocidades irán disminuyendo/aumentando en función de la cercanía entre un dron y otro en sus segmentos.
- Cuando $p_1 = p_2$, se tiene que $v_1 = v_2$, por tanto, ambos se mueven de forma sincronizada y paralela.

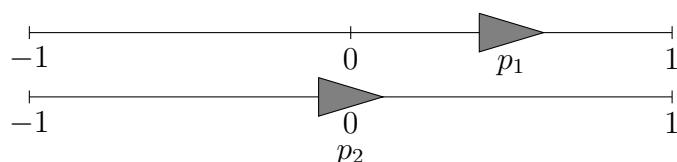


Figura 4.2: Situación inicial para el ejemplo descrito de formación en segmentos

Ajuste de la ganancia

Al ser este algoritmo diseñado por el autor para este TFG, se propone la siguiente forma de ajustar la ganancia:

- Se supone $N = 2$ y $p_{ij}^* = 0$
- Se recomienda que la velocidad **mínima** de un dron sea 0 ($v_i = 0$). En base a esto, el resultado de $K(p_j - p_i)$ deberá ser, **como máximo** igual a $-v_n$
- Se sabe que, como máximo, $p_j - p_i = \pm 2$

En base a todo esto, podemos decir $0 = v_n - 2K$ que da como resultado $K = \frac{v_n}{2}$, por tanto, se recomienda usar una K que sea la mitad de la velocidad nominal deseada. La velocidad máxima será $v_i = v_n + K\max(p_j - p_i) = v_n + 2\frac{v_n}{2} = 2v_n$

Evidentemente, este método es un ejemplo para ajustar la ganancia. En función de los resultados deseados se preferirá un método u otro para ajustarla, pero no se recomienda reducirla ni aumentarla mucho más, al poder haber problemas.

4.1.3. Comparativa entre formaciones

Aprovechando que ya se han explicado y entendido ambos algoritmos, es interesante comparar ambos algoritmos para ver sus similitudes. Recordemos ambas ecuaciones:

$$\begin{aligned} r_i &= r_i + k_r B_i e \\ v_i &= v_n + K \sum_{j=1}^N (p_j - p_i - p_{ij}^*) \end{aligned}$$

Nótese que se ha expandido la Ecuación 4.6 usando la Ecuación 4.5.

Es evidente, que hay grandes similitudes. A grandes rasgos, pueden compararse de la siguiente forma:

- r_i equivale a v_i
- k_r equivale a K . La principal diferencia es que en segmentos solo hay una constante para todos, pero se podría intercambiar por una matriz de ganancias k_{ij}
- e es equivalente a $p_j - p_i - p_{ij}^*$. De hecho, si expandimos, $e = z_k - z_k^*$
- B_i equivale indirectamente a los distintas expresiones resultantes del sumatorio para todos p_{ij}^* (esto último se puede expresar como la matriz de deajustes deseados p^*)

4.2. Implementación de los algoritmos de coordinación

Similarmente a GVF, Paparazzi ya incluye formación circular, pero no esta adaptada a rotorcrafts [26]. Adaptar esto implica diversos cambios referidos al ámbito de los rotorcrafts que se verán en la siguiente subsección.

4.2.1. Adaptación de formaciones circulares

Paparazzi ya incluye formaciones circulares utilizando un script de Python ubicado en `sw/ground_segment/python/gvf` llamado `circularFormation.py`. Este script, similarmente a GVF, no tiene adaptación para rotorcrafts. En resumen, el script consiste en:

- Ejecución del script con un parámetro obligatorio que sera un archivo JSON con parámetros deseados
- Lectura del archivo JSON para recabar ajustes como los IDs internos de los vehículos en Paparazzi
- Sincronización del script con la instancia actual de Paparazzi. Hasta que no se encuentre todos los IDs de los UAVs funcionando bajo GVF tipo círculo el script no se comenzará a ejecutar
- Lectura de la telemetría recibida por los UAVs, para conseguir información como la posición actual
- Cálculo del algoritmo, es decir, cálculo de los ángulos entre vehículos, errores... para poder calcular el radio deseado para cada dron
- Se devuelve la telemetría a través de la librería de Paparazzi para Python que permite comunicarse con los UAVs

La estructura de un archivo JSON suele ser como la siguiente:

```

1  {
2      "ids": [1,2],
3      "topology": [
4          [ 1],
5          [-1]
6      ],
7      "desired_intervehicle_angles_degrees": [0],
8      "gain": 10,
9      "desired_stationary_radius_meters": 80
10 }
```

Donde podemos ver los IDs de los vehículos, la topología deseada (matriz B), ángulo deseado entre vehículos (z^*), ganancia (k_r) y el radio base deseado (r_i).

Para conseguir que esto también funcione para rotorcrafts y por tanto los Crazyflie, se han hecho los siguientes cambios:

- Se añade un nuevo parámetro obligatorio al ejecutar el script. Este parámetro será el tipo de vehículo utilizado, en este caso, hay dos opciones:
 - **Fixedwing:** la ya implementada para aviones de ala fija
 - **Rotorcraft:** la nueva implementación para drones como el Crazyflie

- Internamente en el script, se añade una condición que hará que se suscriba a los mensajes de NAVIGATION o ROTORCRAFT_FP en función de el tipo de UAV deseado. Estos mensajes de telemetría son los encargados de mandar las posiciones relativas (respecto al punto de inicio) de los fixedwing y rotorcraft, respectivamente.

El resto del algoritmo no depende de ningún parámetro extra que pueda ser exclusivo a un tipo de UAV u otro. Consecuentemente, no se necesitan más adaptaciones para las formaciones circulares en rotorcrafts.

4.2.2. Implementación de formaciones en segmentos

En base a la ya implementada formación en circulos, se ha hecho un nuevo programa en Python que implementa la formacion en segmentos, ya que Paparazzi no incluye este programa. Los cambios son:

- Ajustes generales al script del estilo nombres de variables, parámetros de entrada, comentarios...
- Cambio del algoritmo y de las clases para adaptarse a segmentos
- Cambios en la telemetría para sacar otros datos necesarios
- Solo funciona para rotorcrafts, al seguir estos un segmento de mejor forma que un fixedwing

Además, se ha cambiado la estructura del archivo JSON levemente para adaptarse al script. Por ejemplo:

```

1 {
2   "ids": [8,9],
3   "desired_normalized_offset": [0],
4   "nominal_speed": 0.25,
5   "gain": 0.15
6 }
```

Por último, al no ser explicado en la teoría de este algoritmo, se muestra aquí el trozo de código que hace la normalización del segmento:

```

1 # Segment normalization
2 i = 0
3 for ac in self.aircraft:
4     # Calculate x and y distances, from segment and current pos
5     max_dist_x = ac.WP2[0] - ac.WP1[0]
6     max_dist_y = ac.WP2[1] - ac.WP1[1]
```

```

7     dist_x = ac.XY[0] - ac.WP1[0]
8     dist_y = ac.XY[1] - ac.WP1[1]
9
10    # Vector modulus (Pythagoras)
11    max_dist = math.sqrt(max_dist_x**2 + max_dist_y**2)
12    dist = math.sqrt(dist_x**2 + dist_y**2)
13
14    # Having this, we can normalize the distance from 0 to 1
15    norm_dist = dist / max_dist
16
17    # Map segment values from range(0, 1) to range(-1, 1)
18    def map_range(d, in_min, in_max, out_min, out_max):
19        if (in_max == in_min):
20            return 0
21        else:
22            return (d - in_min) * (out_max - out_min) / (in_max - in_min) +
23            out_min
24
25    self.mapped_pos[i] = map_range(norm_dist, 0, 1, -1, 1)
i += 1

```

En resumen:

- Se calculan las distancias en X y en Y
- Se calcula el módulo del vector de las distancias
- Se normaliza a 1
- Por último, se realiza un *mapping* del vector normalizado (0 a 1) a la normalización propia (-1 a 1)

Evidentemente, existen más métodos para conseguir esta normalización (u otros tipos de normalización), pero se ha optado por este al ser simple de realizar, bajo coste computacional y fácil de entender. Funciona además, para un segmento en cualquier ángulo y dirección.

4.3. Simulación de las formaciones

Para acabar este capítulo, veremos los resultados en simulación de los algoritmos implementados. Se verá tanto formaciones circulares como en segmentos. En el caso de las circulares, se obviará los casos reales ya que Parrotazzi no permite realizar círculos de menos de 1 metro de diámetro. Esto último es difícil de realizar en espacios interiores (los Crazyflies no funcionan correctamente en exteriores debido a su bajo peso y, por tanto, el efecto del viento sobre estos)

4.3.1. Simulación de formaciones circulares

Se ha utilizado el siguiente archivo JSON:

```

1 {
2   "ids": [8,9],
3   "topology": [
4     [ 1],
5     [-1]
6   ],
7   "desired_intervehicle_angles_degrees": [0],
8   "gain": 12,
9   "desired_stationary_radius_meters": 75
10 }
```

Y se ha lanzado el script con los siguientes parámetros:

```
1 python circularFormation.py formation/circular_two_bebops.json rotorcraft -v
```

Los resultados tras unos 500 segundos de simulación han sido los siguientes:

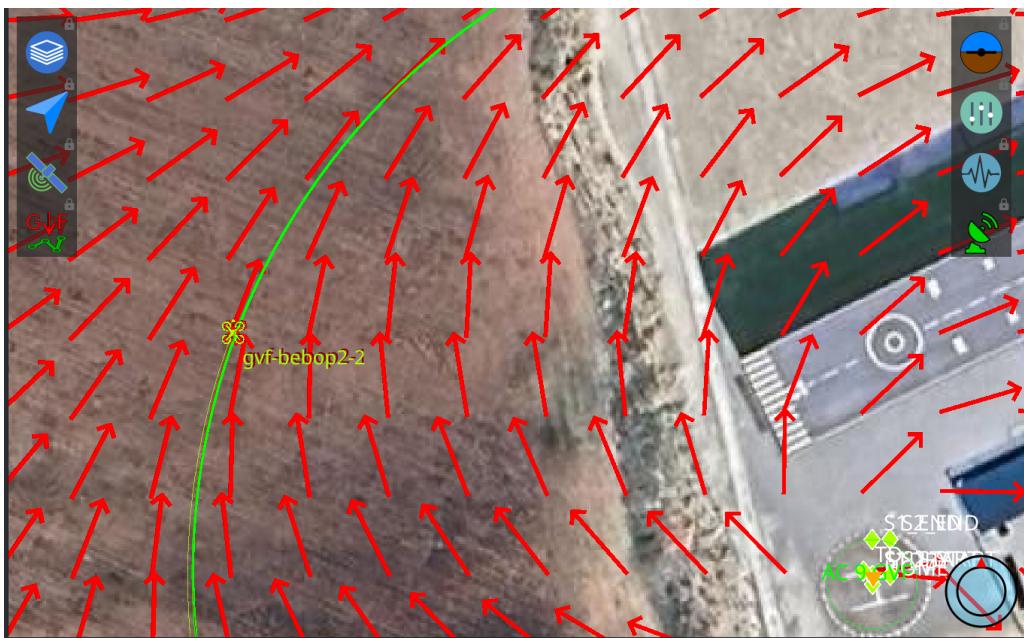


Figura 4.3: Ejemplo de simulación de formaciones circulares con drones. Se solapan ambos drones, al estar (como se desea) a 0 grados de error entre ambos

Además el error del ángulo a lo largo del tiempo ha sido el siguiente. Se parte de ambos haciendo GVF en un radio de unos 80 metros, con un error inicial entre ángulos de unos -6 grados sexagesimales:

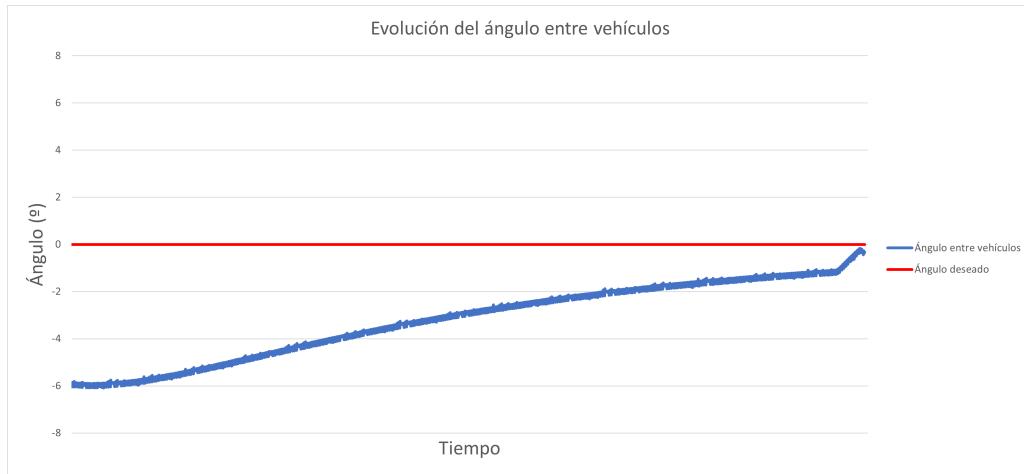


Figura 4.4: Ángulo entre drones en simulación de formación circular. En rojo el ángulo entre drones deseado

Cómo podemos ver, converge lentamente a cerca de 0 grados de error. No se termina de llegar a 0 ya que tomaría mucho tiempo, debido a que k es aparentemente bajo para este escenario.

Tras esto, se mantiene estable en aproximadamente $\pm 1^\circ$ sexagesimales de error entre ambos drones.

4.3.2. Simulación de formaciones en segmentos

Se ha utilizado el siguiente archivo JSON:

```

1 {
2   "ids": [8,9],
3   "desired_normalized_offset": [0],
4   "nominal_speed": 0.25,
5   "gain": 0.18
6 }
```

Y se ha lanzado el script con los siguientes parámetros:

```
1 python segmentFormation.py formation/segment/segment_two_bebops.json -v
```

En esta captura podemos ver dos drones en segmentos paralelos coordinados a la misma distancia en el segmento, es decir, sobre el mismo punto normalizado dentro del segmento (aproximadamente sobre 0.1):

Además el error en la posición del segmento normalizado a lo largo del tiempo ha sido la siguiente. Se parte de ambos haciendo GVF para segmentos



Figura 4.5: Simulación de formación en segmentos en Paparazzi. Podemos ver los drones que están al lado y en paralelo

con un desajuste inicial aproximado de entre -0.5 y 0.5 (según instante de tiempo):



Figura 4.6: Desajuste de la posición sobre el segmento normalizado entre drones en simulación. En rojo el desajuste deseado

Similarmente a formaciones circulares se converge lentamente a 0 de error,

es decir, que uno este al lado del otro. Tras estabilizarse, el error no suele superar los ± 0.05 sobre los segmentos normalizados, exceptuando en los extremos de los segmentos, que se suele ganar algo de error.

4.4. Coordinación y formación de Crazyflies

Por último, se han de probar los experimentos de simulación en los Crazyflies reales. Como se ha explicado previamente, se van a obviar los experimentos de formaciones circulares ya que no se dispone de un buen lugar para estos experimentos.

4.4.1. Comparación entre firmware oficial y de Paparazzi

Las primeras pruebas resultaron ser prometedoras, pero no suficiente, ya que el Flow Deck, aunque mejor que la IMU sola, acumula suficiente error como para que antes de que se coordinen ambos drones se choquen o desvíen demasiado del objetivo, sin llegar a coordinarse. En otras palabras, se puede realizar GVF y/o coordinación bajo un tiempo límite si se utiliza posición relativa, pero la coordinación necesita más de ese tiempo límite. Aquí es un buen ejemplo donde el Loco Positioning System explicado en el capítulo 1 hubiese ayudado, pero se descartó finalmente por falta de un lugar apropiado donde instalarlo.

Consecuentemente, se probó con el firmware de Bitcraze (que se utilizó en el capítulo 2) la estabilidad a lo largo del tiempo y resultó ser considerablemente mayor. En otras palabras, el ajuste de estabilización del PID y lectura de los sensores es **claramente superior en el firmware oficial a en Paparazzi**. En resumen:

- El firmware de Paparazzi es superior en versatilidad, opciones de módulos, telemetría... y es capaz de hacer simulaciones.
- Por contraparte, el firmware de Bitcraze es superior en control del Crazyflie, facilidad de uso (tanto programación como uso general) y documentación.

Por ello, tras demostrar la viabilidad del algoritmo en simulación, se va a realizar la demostración de la coordinación **bajo el firmware oficial de Bitcraze**.

4.4.2. Control en el firmware de Bitcraze

Para entender mejor esta subsección y las siguientes, se recomienda leer paralelamente el Apéndice C, concretamente la sección para Bitcraze. Veamos un script de ejemplo:

```

1 # crazyflie_demo.py
2 # ...
3
4 import cflib.crtp
5 from cflib.crazyflie import Crazyflie
6 from cflib.crazyflie.syncCrazyflie import SyncCrazyflie
7 from cflib.positioning.motion_commander import MotionCommander
8 from cflib.utils import uri_helper
9
10 URI = uri_helper.uri_from_env(default='radio://0/80/2M/E7E7E7E701')
11 DEFAULT_HEIGHT = 0.5
12
13 # ...
14
15 def move_linear_simple(scf):
16     with MotionCommander(scf, default_height=DEFAULT_HEIGHT) as mc:
17         time.sleep(1)
18         mc.forward(0.5)
19         time.sleep(1)
20         mc.turn_left(180)
21         time.sleep(1)
22         mc.forward(0.5)
23         time.sleep(1)
24
25 # ...
26
27 if __name__ == '__main__':
28     cflib.crtp.init_drivers()
29
30     with SyncCrazyflie(URI, cf=Crazyflie(rw_cache='./cache')) as scf:
31         # ...
32         move_linear_simple(scf)
```

... representa fragmentos de código que no se han escrito porque no son necesarios para entender la totalidad del código.

Cómo podemos ver en el fragmento de código anterior, el control de un Crazyflie es trivial en comparación con Paparazzi. El ajuste de PID, los diversos módulos, la necesidad de recompilar el firmware... todo esta abstraído de forma que pueda ser más simple el control básico.

A grandes rasgos, tan sólo hay que inicializar los drivers, crear el objeto `scf` para control del Crazyflie y llamar a una función que indica que se va a comandar. En la función tan sólo hay que indicar si se quiere ir hacia delante, atrás, rotar... Por supuesto existen otros métodos de control, por ejemplo, dirigirse hacia cierta posición, o comandar directamente el roll, pitch, yaw y throttle.

4.4.3. Formación en segmentos en el firmware de Bitcraze

Para implementar el algoritmo de coordinación sobre el firmware oficial, se ha decidido obviar GVF ya que, aunque es preferible debido al buen control que ofrece, no es necesario para hacer una demostración del algoritmo de coordinación. Por ello la implementación de la coordinación en segmentos se ha realizado de la siguiente forma [28]:

```

1 # crazyflie_segment_formation.py
2 # ...
3
4 uris = [
5     'radio://0/80/2M/E7E7E7E701',
6     'radio://0/80/2M/E7E7E7E702',
7 ]
8
9 waiting_uri = 'radio://0/80/2M/E7E7E7E702' # Crazyflie that waits
10 all_positions = {}
11
12 NOMINAL_SPEED = [0.2, 0] # x, y respectively
13 DEFAULT_HEIGHT = 0.5 # In meters
14 SEGMENT_LIMIT = 1 # Meters
15 FREQ = 0.2 # How many seconds until we update the speed
16
17 # ...
18
19 def kuramoto(my_position, other_positions):
20     k = 0.12
21     desired_offset = 0
22
23     delta = 0
24     for position in other_positions:
25         delta += position[0] - my_position[0] - desired_offset
26
27     return k * delta
28
29 def get_xy_from(radio, positions):
30     return (positions[radio].x, positions[radio].y)
31
32 def coordinated_segment(scf):
33     with MotionCommander(scf, default_height=DEFAULT_HEIGHT) as mc:
34         my_nominal_speed = {}
35         my_nominal_speed[scf._link_uri] = NOMINAL_SPEED
36
37     # ...
38
39     while True:
40         # First we calculate positions
41         my_position = get_xy_from(scf._link_uri, all_positions)
42
43         # Calculate every other drone position
44         other_positions = []
45         for uri in uris:
46             if uri != scf._link_uri:
47                 other_positions.append(get_xy_from(uri, all_positions))
48

```

```

49         # Change direction if passed the limit
50     if my_position[0] > SEGMENT_LIMIT:
51         my_nominal_speed[scf._link_uri][0] = -NOMINAL_SPEED[0]
52     elif my_position[0] < -SEGMENT_LIMIT:
53         my_nominal_speed[scf._link_uri][0] = NOMINAL_SPEED[0]
54
55     speed_x = my_nominal_speed[scf._link_uri][0]
56
57     # Add Kuramoto speed
58     speed_x += kuramoto(my_position, other_positions)
59
60     mc.start_linear_motion(speed_x, speed_y, 0)
61     time.sleep(FREQ)
62
63 def recover_positions():
64     global all_positions
65     while True:
66         all_positions = swarm.get_estimated_positions()
67         time.sleep(FREQ / 2)
68
69 if __name__ == '__main__':
70     # ...
71     with Swarm(uris, factory=factory) as swarm:
72         # ...
73         recover_positions_thread.start()
74         swarm.parallel(coordinated_segment)
75         # ...

```

Similarmente al ejemplo anterior, `# ...` representa trozos de código que se han obviado al no ser relevantes para entender la funcionalidad principal.

Podemos dividir el código en las siguientes secciones:

- `recover_positions()`: función que se ocupa de recuperar las posiciones de los drones cada `FREQ / 2` segundos. Se ejecuta en un thread aparte.
- `kuramoto()`: recibe las posiciones de todos los drones y calcula para cada dron cuanto debe aumentar o reducir su velocidad con el algoritmo para segmentos paralelos.
- `coordinated_segment()`: función principal para coordinar los drones. Calcula posiciones, invierte la velocidad lineal de un dron si ha pasado del segmento y comanda la velocidad calculada por el algoritmo cada `FREQ` segundos.

En general, esta implementación simplificada sigue un segmento en X sin importar su posición en Y. Su dirección cambiará cuando llegue a 1 metro y cuando llegue a -1 metro, permitiendo obviar la normalización. Si bien puede ser simple, el objetivo final que es aplicar el algoritmo, se puede conseguir igualmente.

4.4.4. Resultados de la coordinación

Ejecutando el programa mostrado en la subsección anterior, los resultados fueron prometedores: el algoritmo funciona correctamente, pero actualmente hay un error en el código de Bitcraze que impiden que se den la vuelta correctamente en el segmento.

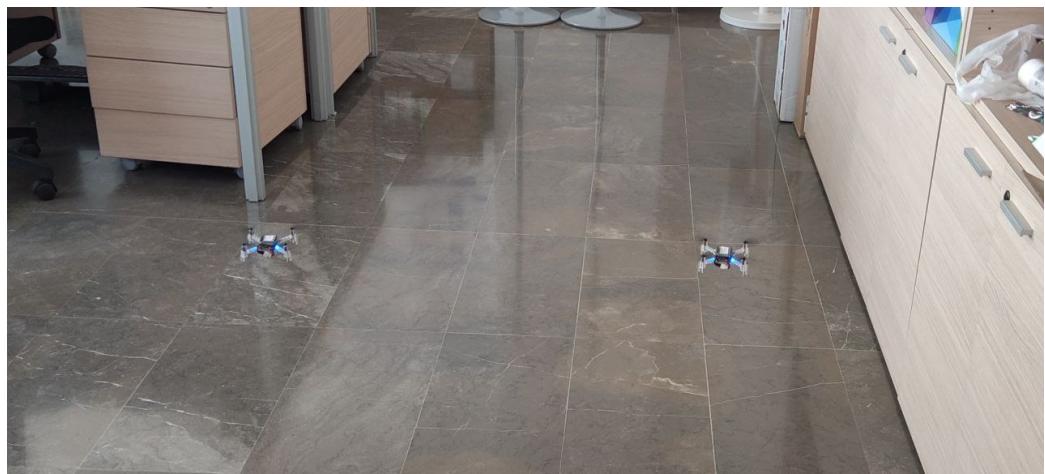


Figura 4.7: Dos drones manteniéndose paralelos. En este caso $p^* = 0$, $v_n = 0.2 \text{ m/s}$, $K = 0.12$

Para verificar que esto no es culpa de un error en la programación o algoritmo, se ha decidido añadir un último experimento que utiliza un log real de un sólo dron siguiendo un segmento y un dron simulado aplicando el algoritmo en base a las posiciones reales del log, es decir, tratando de mantenerse coordinado con las posiciones del dron real guardadas en el log. Mas información de este experimento en el Apéndice C.

Para este experimento:

- $p^* = 0$, $v_n = 0.1 \text{ m/s}$. Para el caso del log sacado del dron real, se mantuvo una velocidad fija de 0.1 m/s .
- Se fijo un segmento con posiciones $(-0.5, 0)$ y $(0.5, 0)$, es decir, para este caso no se normalizó el segmento. Consecuentemente, se ha decidido que $K = 0.3$ (3 veces más alto que la velocidad nominal), para compensar por el hecho de que el segmento es más corto.
- También, se ha elegido un valor de K más alto para que se pueda mantener más cerca el dron simulado del real, ya que al ser una simulación controlada se pueden evitar los posibles errores de elegir un K alto.

- Por último, se ha decidido fijar las posiciones en el eje Y. En el caso del dron real hubo una ligera variación, pero por simplificar, se ha asumido que los valores de p_y fueron constantes.

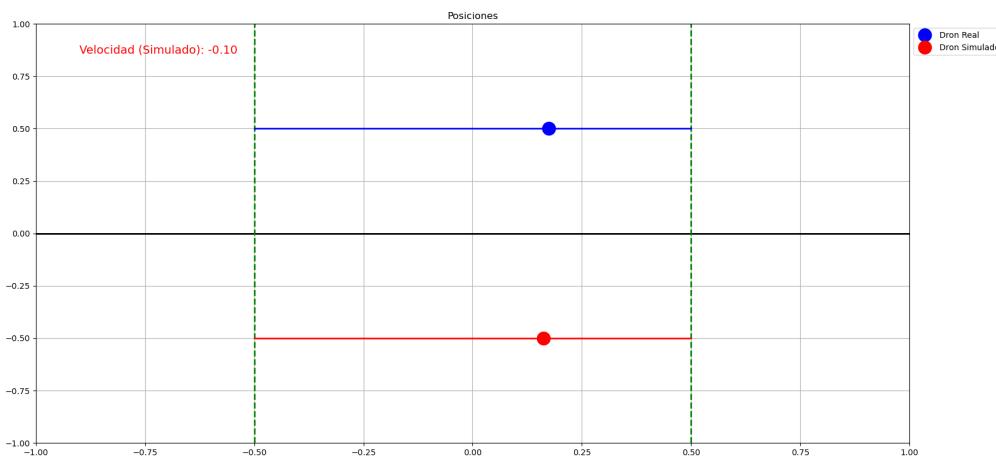


Figura 4.8: Animación del experimento que verifica que el algoritmo funciona

En la siguiente figura podemos ver el error de la posición entre los drones. Se obvian los dos primeros segundos que el dron simulado está quieto:



Figura 4.9: Diferencia de la posición entre los drones. En verde la diferencia deseada

Podemos ver que el error no sube de 0.1 m tras estabilizarse. Si obviamos los máximos locales (se producen al dar la vuelta en el segmento), el error está típicamente por debajo de 0.05 m, es decir por debajo del 10 % de error.

Por último, la velocidad del dron simulado fue la siguiente (para los mismos instantes de tiempo):

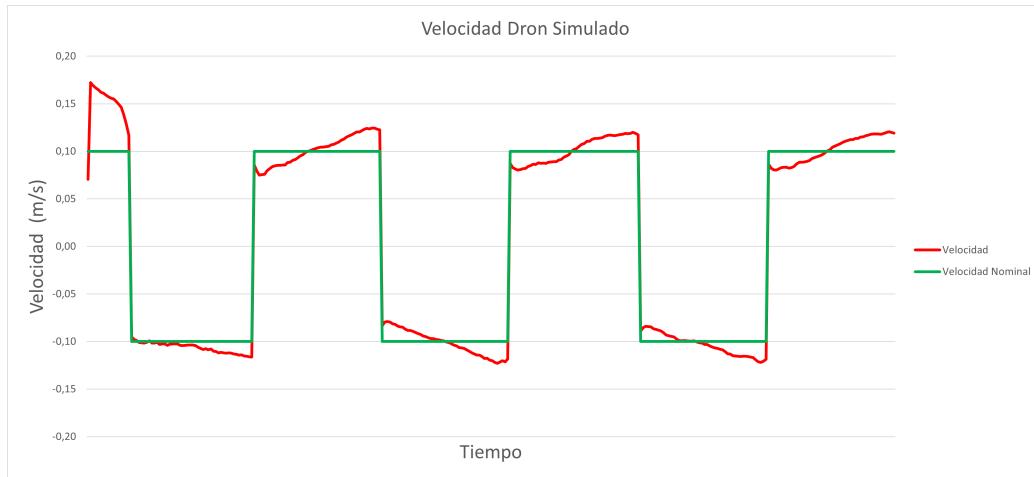


Figura 4.10: Velocidad del dron rojo (simulado). En verde la velocidad nominal objetivo en ese instante

Es interesante ver los bruscos cambios, al ser un dron simulado sin GVF.

Este experimento también sirve para entender las impresionantes capacidades de Paparazzi para simulación. Es evidente que para ello, Paparazzi es superior a otras alternativas software, en especial para los Crazyflies, que no dispone de una extensa variedad.

Finalmente, se recomienda para ambos experimentos ver las animaciones y vídeos correspondientes en la presentación por diapositivas de este TFG para entender mejor los resultados que en las figuras [29].

Capítulo 5

CONCLUSIONES

En este último capítulo, se extraen conclusiones en base a los resultados obtenidos. Se incluye, por supuesto, un análisis de los objetivos y planificación, así como posibles aplicaciones, para poder extraer conclusiones desde un punto de vista de la ingeniería en general, además de los resultados más tangibles y observables a simple vista.

5.1. Análisis de los objetivos y planificación

Es importante observar hasta qué punto se cumplieron las metas establecidas. Por ello, a continuación se muestra un pequeño análisis de si se cumplió cada objetivo, y hasta qué punto:

- **OBJ-1:** introducción, objetivos, planificación y preparación general. Es evidente que se cumplió este objetivo, al ser introductorio y de carácter anterior al resto de objetivos.
- **Capítulo 2 (OBJ-2):** preliminares. Se cumplió perfectamente este objetivo. No obstante, hubo ciertos retrasos que veremos en la siguiente subsección de planificación.
- **Capítulo 3 (OBJ-3):** control de un Crazyflie. Se consiguió implementar el control de un Crazyflie en Paparazzi, tanto en simulación como en la realidad. El único problema fue que el control en movimiento en Paparazzi fue peor a lo esperado.
- **Capítulo 4 (OBJ-4):** coordinación entre Crazyflies. Se consiguió este objetivo plenamente, los resultados finales han sido prometedores

y se ha utilizado tanto Paparazzi como el software y firmware oficial. La única diferencia es que no se pudo conseguir control aceptable bajo Paparazzi, negando la posibilidad de intentar coordinación en este software.

- **Capítulo 5 (OBJ-5):** conclusiones. Similar al OBJ-1, este objetivo se puede considerar evidente, al ser un objetivo que se cumple casi automáticamente al terminar el resto de objetivos.
- **OBJ-6:** Revisión y mejora general. Si bien se ha calificado como objetivo, este último es un poco más ambiguo y subjetivo de poder hacer una evaluación de su cumplimiento. En general, considero personalmente que fue buena elección dedicar y planificar un tiempo a la revisión y mejora general del trabajo en su conjunto. En los resultados de la planificación (siguiente subsección) veremos que se decidió finalmente hacerlo en paralelo al trabajo (intercalando) y no sólo al final.

Acompañado al análisis de objetivos, para comparar si se ha realizado una buena planificación, se ha realizado otro diagrama de Gantt, que muestra como se distribuyeron los objetivos a lo largo del desarrollo:



Figura 5.1: Resultados reales de la planificación

En general, los dos mayores cambios se realizaron en el objetivo 2 y 6 respecto a la planificación inicial en la Figura 1.4. Se encontraron multitud de problemas durante el objetivo 2 que necesitaron mucho más tiempo del planeado, cómo fallos en el firmware de Paparazzi para el Crazyflie. Esto provocó un desplazamiento de los objetivos posteriores y se tuvieron que realizar un poco más deprisa.

Por otro lado, el objetivo 6 se decidió realizarse paralelamente al resto de objetivos, es decir, se realizaban cambios y revisiones casi constantemente, así como se realizaba la memoria en paralelo con lo que se iba desarrollando. Esto último permitió que fuese más cómodo escribir la memoria, al ir plasmando lo recientemente aprendido sobre esta antes de que se olvidasen detalles en las semanas posteriores.

5.2. Aplicaciones y usos

Conseguidos los fundamentos de esta tecnología de coordinación, es importante analizar las posibles aplicaciones finales. Entre ellas, destaca principalmente la experimentación e investigación de la tecnología, ya que este trabajo consigue directamente esta aplicación sin mayores modificaciones y adiciones extra.

Además, la base de esta tecnología permite otros usos (por supuesto, con previas modificaciones si fuesen necesarias) como pueden ser la vigilancia mediante enjambres de drones, aplicaciones en agricultura, defensa o espectáculos de drones independientes del control por *waypoints* fijados por una estación central.

Es evidente que para terminar de ver las posibles aplicaciones y usos, se ha de ver que ha de mejorarse antes, por ello, lo exploraremos en la siguiente subsección.

5.3. Trabajos y mejoras futuras

Si se desease mejorar o continuar sobre la base de este trabajo, las primeras mejoras que se deberían aplicar serían las siguientes:

- **Mejor control en Paparazzi:** al ser el primer objetivo parcialmente no cumplido, es evidente que se necesita mejor control bajo el software de Paparazzi, ya que este es bastante superado por el software oficial para los Crazyflie.
- **Coordinación real en Paparazzi:** consecuencia de no poder conseguir el correcto control en Paparazzi, es evidente que esto también es necesario corregirlo. Es posible que consiguiendo arreglar el control no se necesite más, pero no es seguro.

- **Implementación para N drones:** si bien el algoritmo se ha diseñado pensando en varios drones, las implementaciones se han simplificado para tan sólo dos drones. Por suerte, esta mejora solo requiere cambios a la hora de introducir los parámetros de desajuste deseado y los calculos respecto a estos.
- **Implementación y uso de otros sistemas de posicionamiento:** como bien se vio en el capítulo 1, existen más sistemas de posicionamiento compatibles con los Crazyflies. Estos sistemas podrían dar mayor precisión, evitar acumulación de error e incluso ayudar a la hora de implementar mejor control en Paparazzi.

Por supuesto, esto son las mejoras directas a este TFG, si hablamos de trabajos más grandes a mayor plazo sobre esta tecnología, podemos hablar de lo siguiente:

- **Implementación de GVF paramétrico:** una versión para trayectorias en tres dimensiones de GVF. Más compleja, parcialmente implementada en Paparazzi.
- **Mejoras diversas al software de Paparazzi:** como puede ser mayor facilidad de uso, corregir las simulaciones para el Crazyflie (aunque se pueden usar otros drones), mejorar la documentación, simplificar el software en cuestión de características (son muchas)...
- **Mejora general de los algoritmos:** si bien son buenos, es interesante ver hasta qué punto se pueden conseguir algoritmos con mucha mayor precisión, donde el error es mínimo. También se pueden hacer algoritmos más inteligentes, que no solo sigan trayectorias semi-predefinidas.
- **Coordinación descentralizada y distribuida:** aplicar estos conceptos pero sin haber un nodo central, en este caso, el ordenador que controla todo. En otras palabras, coordinar N drones que se comunican entre ellos y deciden qué hacer solos en base a estos algoritmos, sin un PC central que los coordine a todos.
- **Aplicaciones reales:** como puede ser espectáculos de luces basados en GVF, en vez de ser scripts de a qué posición debe ir cada dron (como es hoy en día, generalmente).

5.4. Conclusión

En conclusión, este trabajo ha conseguido mayoritariamente sus objetivos, planificación y permite diversos usos. Se ha demostrado el buen funcionamiento de los algoritmos de estilo GVF y se han probado dos algoritmos de coordinación, uno diseñado en exclusiva para este trabajo.

Los resultados, si bien prometedores, hacen los Crazyflies ideales exclusivamente para la experimentación, investigación y educación en el ámbito de control y coordinación de enjambres de drones, debido a las mejoras que aún pueden añadirse, como el mejor control en Paparazzi.

Para conseguir más aplicaciones y usos, es recomendable la adición de otros componentes a este trabajo, como pueden ser cámaras (para vigilancia) o LEDs (para espectáculos de luces), así como el desarrollo extra en mejorar tanto el control como la coordinación e incluso avances de hardware que escapan al ámbito de este trabajo.

Capítulo A

INSTALACIÓN DE PAPARAZZI EN DEBIAN

Paparazzi UAV solo se puede instalar oficialmente en Ubuntu [10] sin realizar pasos complejos como compilación desde cero [11].

Desde el equipo de Swarm Systems Lab preferimos el uso de Debian a Ubuntu y, debido a su similitud, el autor de este TFG ha descubierto un método para la instalación de Paparazzi en Debian 12, más detallado en el repositorio [17].

La instalación completa se basa en ejecutar el siguiente comando en Debian. Se recomienda ejecutar el script bajo el directorio donde se desee instalar Paparazzi:

```
1 $ wget https://raw.githubusercontent.com/Pelochus/bt-crazyflies/main/scripts  
2      /debian/paparazzi-debian-install.sh && bash paparazzi-debian-install.sh
```

En las dos siguientes secciones se explica como se instaló en detalle Paparazzi en Debian.

A.1. Instalación de Paparazzi Center

Se ha creado el siguiente script de instalación en Bash para automatizar el proceso:

```
1 #!/bin/bash  
2  
3 # Needed to force a PPA repo in Debian  
4 sudo apt install -y build-essential devscripts  
5  
6 # Important note: [trusted=yes] is not recommended  
7 echo "deb [trusted=yes] https://ppa.launchpadcontent.net/paparazzi-uav/ppa/  
     ubuntu jammy main" | sudo tee -a /etc/apt/sources.list
```

```

8 echo "deb-src [trusted=yes] https://ppa.launchpadcontent.net/paparazzi-uav/
      ppa/ubuntu jammy main" | sudo tee -a /etc/apt/sources.list
9
10 # Rest of the official Paparazzi guide
11 sudo apt update
12 sudo apt install -y paparazzi-dev gcc-arm-none-eabi gdb-multiarch python-is-
      python3 paparazzi-jsbsim dfu-util
13
14 # Clone Paparazzi
15 git clone --origin upstream https://github.com/paparazzi/paparazzi.git
16 cd paparazzi
17 git checkout v6.3 # Currently last stable version
18
19 # Compile latest stable
20 make

```

Para entender este script, hay que hacer hincapié en las siguientes líneas:

- **Líneas 11 y 12:** Añadimos los repositorios de descarga para Paparazzi. Para que funcionen en Debian, se añade `[trusted=yes]` de forma que se fuerce la confianza en el repositorio, ya que no es posible añadirlos de la forma natural según la guía. Se indica además, que se utilice los paquetes para la última versión de Ubuntu que disponen desde el equipo de Paparazzi, en este caso, `jammy`.
- **Línea 16:** Se instalan todos los paquetes según la guía oficial para Ubuntu, exceptuando el paquete para Paparazzi GCS, que no es posible utilizarlo en Debian

El resto de líneas son extraídas de la guía oficial de instalación.

Este script debe funcionar en Debian 12 recién instalado si se posee conexión a la red. Es posible, aunque no ha sido comprobado, que funcione en distribuciones basadas en Debian u otras versiones de Debian como Debian 11. No se recomienda utilizar en distribuciones basadas en Ubuntu, se puede seguir la guía oficial en estos casos.

A.2. Instalación de Paparazzi GCS

Como se ha comentado en la sección previa, el paquete correspondiente a Paparazzi GCS no se puede instalar bajo Debian. Para ello, se puede utilizar el **AppImage** oficial, que funciona similar a un ejecutable en Windows o un contenedor de Docker.

Este AppImage necesita ser integrado con el ejecutable de Paparazzi, ya que por defecto este asume que se ha instalado el GCS desde el gestor de paquetes `apt` y, por tanto, hay acceso universal.

En este caso, se utiliza este script para la configuración e integración de Paparazzi GCS en Debian:

```

1 #!/bin/bash
2
3 cd paparazzi
4
5 # Make Paparazzi GCS AppImage Work
6 echo "# Needed for Paparazzi AppImage to work" >> /home/$USER/.bashrc
7 echo "export PAPARAZZI_HOME=$(pwd)" >> /home/$USER/.bashrc
8 echo "export PAPARAZZI_SRC=$(pwd)" >> /home/$USER/.bashrc
9
10 # Get AppImage and move to /usr/bin/pprzgcs so it can be launched by
11 # Paparazzi Center
12 sudo apt install -y wget # Just in case, you should have it anyway
13 sudo wget https://github.com/paparazzi/PprzGCS/releases/download/v1.0.11/
14     pprzgcs-v1.0.11-x86_64.AppImage -O /usr/bin/pprzgcs
15 sudo chmod 755 /usr/bin/pprzgcs

```

En este caso, es mucho más sencillo entender este script:

- Primeramente se establecen las variables de entorno `PAPARAZZI_HOME` y `PAPARAZZI_SRC`, que son necesarias para el funcionamiento del AppImage.
- Utilizando el ejecutable `wget`, descargamos automáticamente el AppImage y lo movemos a `/usr/bin/pprzgcs`, para que simule el comportamiento de descargar el paquete oficial de la repo.

En general, la instalación de Paparazzi en Debian se condensa en ejecutar el primer script de este apéndice, que se ocupa de llamar a los otros dos scripts, de forma que se pueda modularizar la instalación de ambos componentes

A.3. Instalación completa

Para la instalación completa del código fuente de Paparazzi, se ejecuta el siguiente comando en el directorio raíz del repositorio:

```
1 $ git submodule update --init --recursive .
```

Este paso no es necesario a no ser que se necesite compilar código o ejecutar simulaciones que necesiten de código o librerías externas. No es necesario para el funcionamiento de Paparazzi Center y GCS.

Capítulo B

INSTALACIÓN DE DEPENDENCIAS

En el Apéndice A se incluye como se adaptó e instaló Paparazzi para Debian, no obstante, no se incluye que librerías y configuraciones son necesarias para el uso de Crazyflie en Paparazzi.

En este segundo apartado del apéndice se incluye una breve descripción de las librerías y configuraciones necesarias. Se incluye aquí otro script de Bash que permite realizar todo esto de manera rápida:

```
1 $ wget https://raw.githubusercontent.com/Pelochus/bt-crazyflies/main/scripts  
     /deps/deps-install.sh && bash deps-install.sh
```

B.1. Dependencias a instalar

Se necesitan las siguientes dependencias:

- **dfu-util y dfu-programmer:** Paquetes de Ubuntu/Debian para cargar el firmware en el Crazyflie.
- **cflib:** Librería de Python para poder comunicarse con el Crazyradio 2.0 y, por tanto, un Crazyflie.
- **gedit:** Se usa por defecto para editar archivos en Paparazzi desde Paparazzi Center, al estar pensado para Ubuntu. Es opcional, pero es recomendable tenerlo instalado en Debian como opción *fallback*.

B.2. Configuraciones y ajustes

Por otro lado los ajustes necesarios son los siguientes:

- **Añadir el usuario al grupo dialout:** Esto sirve para poder cargar el firmware en el Crazyflie sin permisos de superusuario.
- **Añadir permisos para el Crazyradio:** Similarmente, se añaden permisos para poder acceder al Crazyradio por USB sin ser *root*. Se sigue la guia oficial [13].
- **Cambiar el editor de texto por defecto de Paparazzi Center:** De nuevo, opcional, pero se puede desde *File - Edit settings - Text editor*

Capítulo C

MANUAL DE USUARIO

Este manual de usuario se divide en dos partes: Paparazzi y coordinación con Bitcraze. Se asume que ya se puede ejecutar Paparazzi para la primera parte. Para la segunda, se asume que se tienen las librerías y aplicaciones de Bitcraze. Se puede usar la máquina virtual provista por ellos.

Si esto no es así aún, se recomienda leer los apéndices A y B o seguir las guías oficiales de instalación y configuración, tanto de Paparazzi como de Bitcraze, según que se vaya a utilizar.

C.1. Uso de Crazyflies en Paparazzi

Al usar el repositorio de GitHub del autor de este TFG [17], si se utiliza el submódulo de Paparazzi, el uso de los Crazyflies se ha simplificado respecto al explicado en el capítulo 2 y se han añadido los pasos pertinentes para el correcto funcionamiento. Los requisitos previos son:

- Se solucionaron algunos errores en el firmware. Esto ya se ha integrado en la repo de Paparazzi oficial.
- Se ha de cargar, previo a cargar el firmware de Paparazzi, el **firmware oficial de septiembre de 2019 (2019.09)**. Si esto no es así, no habrá correcta comunicación con la radio, ya que el firmware de la radio no es modificado por Paparazzi y este entiende a versiones antiguas.
- Se recomienda añadir el Flow Deck V2 mencionado en el capítulo 3, Figura 3.4. Sin este, los resultados reales serán pésimos.
- Por último, se han añadido diversos archivos de configuración XML para facilitar un poco más el uso en Paparazzi

Para el uso de un Crazyflie, se tienen los siguientes nuevos pasos respecto al capítulo 2. Se recomienda leer el capítulo 2 para las correspondientes figuras.

- En el menú desplegable arriba a la derecha, seleccionar `conf_crazyflies.xml`
- En el superior izquierda, seleccionar `crazyflie-2.1` para usar el Crazyflie 2.1. Se tiene soporte experimental para Crazyflie 2.0 añadido por el autor de este TFG, pero no es plenamente funcional.
- Para simulación, los Crazyflie no funcionan (fallo de Paparazzi), así que se recomienda seleccionar `gvf-bebop2-x`, que tiene el mismo funcionamiento que el Crazyflie (mismos flightplan, telemetría...) pero funciona correctamente en simulación.
- En los dos pasos previos, por como se ha diseñado los XML, no se necesita seleccionar otros XMLs. Los flightplan, airframe files, telemetry files y demás configuraciones son las correctas si no se modifican, incluyen calibraciones como las del PID.
- En base a si se desea simulación o real, usar uno de los dos previos y compilar `nps` para simulación o `ap` para cargar posteriormente al Crazyflie, como en el capítulo 2.
- Por último, dirigirse a la pestaña de `Operation` y lanzar una sesión de `Simulation` o `Crazyflie Flight` según cual se deseé usar. Con esto, si se cumplen todos los pasos previos y apéndices pertinentes, debería poderse utilizar el GCS para GVF.

Finalmente, para coordinación, hay que usar los scripts que se encuentran en `sw/ground_segment/python/gvf` según se deseé formaciones circulares o segmentos paralelos. Se recomienda **SÓLO USAR** para simulación, ya que la implementación real no da buenos resultados.

La ejecución ha de realizarse tras mandar los comandos de GVF a los drones. Una vez estos estén siguiendo una trayectoria, ejecutar el script correspondiente. Ambos scripts tienen sus menús de ayuda correspondientes para entender mejor como se ejecutan. También se recomienda ver los ejemplos provistos en la sección 4.3 si los menús de ayuda no son suficientes.

C.2. Coordinación de Crazyflies bajo firmware oficial

Para empezar, se recomienda cargar el último firmware oficial de Bitcraze para utilizar sus herramientas [9]. Además, para el uso como swarm, es necesario asignar una URI distinta a cada Crazyflie mediante la configuración de este en Crazyflie Client. Como último requisito, se requiere el Flow Deck v2 y la Crazyradio, el primero para correcto movimiento y el segundo para la conexión.

Con esta base, y asumiendo la disponibilidad de las dependencias necesarias (estarán todas disponibles si se parte desde Bitcraze VM), se pueden utilizar los programas en la carpeta `python` del repositorio de este TFG [17]. Alternativamente, se pueden ver los ejemplos oficiales de Bitcraze [27]

La ejecución de todos los programas que se mencionarán ahora será de la siguiente forma:

```
1 $ python example.py
```

C.2.1. Demo

El archivo es `crazyflie_demo.py`

Se recomienda utilizar este primer programa para asegurar el correcto funcionamiento de cada Crazyflie por separado. Tan sólo se debe modificar la URI del programa para que coincida con la del Crazyflie que se vaya a usar.

El programa consiste en:

- Despegar el Crazyflie a 0.5 metros
- Moverse 0.5 metros hacia delante
- Volver 0.5 metros hacia atrás, tras girar 180 grados
- Aterrizar

C.2.2. Demo para varios Crazyflies

El archivo es `crazyflie_swarm_demo.py`

Similarmente al caso anterior, esto prueba que el funcionamiento en enjambre funciona. Se debe modificar las URIs deseadas con los Crazyflies que vayan a participar.

Este programa consiste en:

- Parpadear los LEDs de ambos Crazyflie para asegurar que funcionan coordinadamente
- Despegar ambos Crazyflie a 1 metro
- Esperar 3 segundos
- Aterrizar

C.2.3. Segmento para un solo Crazyflie

El archivo es `crazyflie_single_segment.py`

Este programa tiene como objetivo hacer el funcionamiento de la coordinación en segmentos pero para un solo dron, de forma que se pueda verificar que los drones se mueven correctamente a lo largo de un segmento.

Este programa consiste en:

- Despegar el Crazyflie a 0.5 metros sobre el (0, 0)
- Moverse hasta (0.5, 0), volver hacia atrás hasta la posición estimada (-0.5, 0). Las unidades son metros. Esto se hará en bucle infinito.
- El aterrizaje se hará tras comandar `Ctrl+C` al programa

C.2.4. Coordinación en segmentos paralelos para dos Crazyflies

El archivo es `crazyflie_segment_formation.py`

Este programa tiene como objetivo hacer el funcionamiento de la coordinación en segmentos que tiene este TFG como objetivo. Se puede ampliar para funcionar con más de dos drones, pero requiere varias modificaciones.

Este programa consiste en:

- Despegar ambos Crazyflie a 0.5 metros sobre el (0, 0). Cada Crazyflie en este caso, tendrá un (0, 0) distinto, se recomienda poner uno al lado del otro, separados aproximadamente medio metro.

- Moverse hasta (1, 0), volver hacia atras hasta la posición estimada (-1, 0). Las unidades son metros. Esto se hará en bucle infinito.
- Mientras esto se hace, ambos Crazyflie tratarán de mantenerse uno al lado del otro. Por defecto, el script hará a uno de los 2 Crazyflies esperar a que el otro avance, de esta forma, tiene más sentido la coordinación. Se deberá cambiar la variable `waiting_uri` para elegir cual esperará.
- El aterrizaje se hará tras comandar `Ctrl+C` al programa

Si este programa funciona correctamente, deberán obtenerse resultados similares a los de este TFG.

C.2.5. Coordinación en segmentos paralelos simulada

El archivo es `sim/experiment.py`

Este programa tiene como objetivo hacer el funcionamiento de la coordinación en segmentos que tiene este TFG como objetivo, pero simulando uno de los dos drones. Dado que este programa es un breve experimento, no está pensado para ser usado sin previas modificaciones, añadidos o trabajos manuales.

Para usar este programa, se recomienda previamente tener un log similar al provisto en `sim/log.txt`, es decir, una sola columna con números que son las posiciones del dron real. Alternativamente, para probar el funcionamiento se recomienda probar con el archivo provisto de ejemplo. Se adjunta un script para convertir las comas a puntos si fuese necesario.

Para conseguir un log así se puede utilizar el programa para un solo segmento del Crazyflie explicado en este apéndice y extraer la columna de posiciones.

BIBLIOGRAFÍA

- [1] Paparazzi Team, “Paparazzi UAV – Source Code.” <https://github.com/paparazzi/paparazzi>.
- [2] Bitcraze, “Crazyflie 2.1 Product Page.” <https://www.bitcraze.io/products/crazyflie-2-1/>.
- [3] Bitcraze, “Loco Positioning System Overview.” <https://www.bitcraze.io/documentation/system/positioning/loco-positioning-system/>.
- [4] Paparazzi Team, “Paparazzi GCS.” <https://github.com/paparazzi/PprzGCS>.
- [5] Paparazzi Team, “Paparazzi Wiki – Crazyflie 2.0/2.1.” https://wiki.paparazziuav.org/wiki/Crazyflie_2.0.
- [6] Bitcraze, “Getting Started with the Crazyflie 2.X.” <https://www.bitcraze.io/documentation/tutorials/getting-started-with-crazyflie-2-x/>.
- [7] Bitcraze, “Bitcraze Virtual Machine.” <https://github.com/bitcraze/bitcraze-vm>.
- [8] Bitcraze, “Getting started with the Crazyradio 2.0.” <https://www.bitcraze.io/documentation/tutorials/getting-started-with-crazyradio-2-0/>.
- [9] Bitcraze, “Crazyflie Client Guide – Firmware Upgrade.” https://www.bitcraze.io/documentation/repository/crazyflie-clients-python/master/userguides/userguide_client/#firmware-upgrade.

- [10] Paparazzi Team, “Quick Install – Paparazzi UAV ReadTheDocs.” <https://paparazzi-uav.readthedocs.io/en/latest/quickstart/install.html>.
- [11] Paparazzi Team, “Paparazzi Wiki – Installation in Linux from scratch.” <https://wiki.paparazziuav.org/wiki/Installation/Linux>.
- [12] Paparazzi Team, “First Simulation – Paparazzi UAV ReadTheDocs.” https://paparazzi-uav.readthedocs.io/en/latest/quickstart/first_simulation.html.
- [13] Bitcraze, “USB Permissions for Crazyradio 2.0.” https://www.bitcraze.io/documentation/repository/crazyflie-lib-python/master/installation/usb_permissions/.
- [14] Paparazzi Team, “Paparazzi Wiki – Modules.” <https://wiki.paparazziuav.org/wiki/Modules>.
- [15] Paparazzi Team, “Fix for Crazyflie ChibiOS.” <https://github.com/paparazzi/paparazzi/issues/3252#issuecomment-2031553762>.
- [16] Paparazzi Team, “Paparazzi Wiki – RT Paparazzi.” https://wiki.paparazziuav.org/wiki/RT_Paparazzi.
- [17] A. Hurtado Flores, “Control and coordination of Crazyflie drones — Bachelor’s Thesis.” <https://github.com/Pelochus/bt-crazyflies>, 2024.
- [18] H. Garcia de Marina, Y. Kapitanyuk, M. Bronz, G. Hattenberger, and M. Cao, “Guidance algorithm for smooth trajectory tracking of a fixed wing UAV flying in wind flows,” 2017.
- [19] J. Bautista Villar, “Diseño e integración de sistemas de guia-do navegación y control para un enjambre de vehículos autónomos.” https://github.com/jesusBV20/MT-DATCOM/blob/main/pdf/TFM__DATCOM.pdf, 2023.
- [20] Paparazzi Team, “Working with INDI - PaparazziUAV.” https://wiki.paparazziuav.org/wiki/Working_with_INDI.
- [21] Wikipedia, “Inertial Measurement Unit - Wikipedia.” https://en.wikipedia.org/wiki/Inertial_measurement_unit, 2024.
- [22] Bitcraze, “Flow Deck v2 Product Page.” <https://www.bitcraze.io/products/flow-deck-v2/>, 2020.

- [23] H. G. de Marina, Z. Sun, M. Bronz, and G. Hattenberger, “Circular formation control of fixed-wing UAVs with constant speeds,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5298–5303, 2017.
- [24] Wikipedia, “atan2 - Two argument arctangent.” <https://en.wikipedia.org/wiki/Atan2>.
- [25] Wikipedia, “Modelo de Kuramoto.” https://es.wikipedia.org/wiki/Modelo_de_Kuramoto.
- [26] Paparazzi Team, “Guidance vector fields - Circular formations.” [https://wiki.paparazziuav.org/wiki/Module/guidance_vector_field#Circular_formations_\(centralized_from_the_GCS\)](https://wiki.paparazziuav.org/wiki/Module/guidance_vector_field#Circular_formations_(centralized_from_the_GCS)).
- [27] Bitcraze, “crazyflie-lib-python - Examples.” <https://github.com/bitcraze/crazyflie-lib-python/tree/master/examples>.
- [28] Bitcraze, “Swarm Interface: cflib.” https://www.bitcraze.io/documentation/repository/crazyflie-lib-python/master/user-guides/sbs_swarm_interface/.
- [29] Ángel Hurtado Flores, “Presentación de Control y Coordinación de drones Crazyflie.” <https://github.com/Pelochus/bt-crazyflies/tree/main/docs/PowerPoint>.
- [30] Bitcraze, “On-chip debugging for Crazyflie 2.1.” https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/development/openocd_gdb_debugging/.