

Universidad de la República
Facultad de Ingeniería



ENDRA: Enjambre de Drones Autónomos

Memoria de proyecto presentada a la Facultad de Ingeniería de la Universidad de la República por

Guillermo Cabrera, Agustín Castro, Sofía Salmini

en cumplimiento parcial de los requerimientos para la obtención del título de Ingeniero Electricista.

Tutor

Juan Bazerque ..... Universidad de la República

Tribunal

Juan Bazerque ..... Universidad de la República
Facundo Benavides ..... Universidad de la República
Pablo Monzón ..... Universidad de la República

Montevideo
miércoles 9 junio, 2021

ENDRA: Enjambre de Drones Autónomos, Guillermo Cabrera, Agustín Castro, Sofía Salmini.

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 148 páginas.
Compilada el miércoles 9 junio, 2021.
http://iie.fing.edu.uy/

La innovación es lo que distingue a los líderes de los seguidores.

Steve Jobs

Esta página ha sido intencionalmente dejada en blanco.

Agradecimientos

Queremos agradecer a todas las personas que desde su lugar colaboraron de alguna manera con el desarrollo de este proyecto, especialmente:

A nuestras familias y nuestros amigos, por su apoyo incondicional a lo largo de la carrera. A nuestro tutor Juan Bazerque por su disponibilidad, interés y sus consejos para lograr la finalización de este proyecto de la mejor manera posible.

A Pablo Monzón, por ser el que nos presentó la posibilidad de realizar este proyecto, además de su buena disposición hacia nuestras consultas y sus aportes durante la corrección de este documento. A Leopoldo Agorio, Mariana Del Castillo y Romina Coor por los consejos y aportes, tanto durante el proyecto como durante la elaboración de esta tesis.

Queremos agradecer también al Prof. Alejandro Ribeiro y su grupo de trabajo en la Universidad de Pennsylvania, por permitirnos formar parte de este proyecto y aportar nuestro granito de arena.

En general, al Instituto de Ingeniería Eléctrica y todas las personas que lo conforman, por brindarnos los materiales y conocimientos necesarios para la realización del proyecto en este año tan particular.

Esta página ha sido intencionalmente dejada en blanco.

A nuestras familias y amigos.

Esta página ha sido intencionalmente dejada en blanco.

Resumen

El presente trabajo se centra en la implementación de un enjambre de múltiples drones navegando de forma autónoma, para el cual se desarrollaron acciones de control que involucran planificación de trayectorias, algoritmos de seguimiento y de evasión. El enjambre se compone de tres drones cuadricópteros autónomos, cuya construcción, calibración y configuración fue realizada durante el proyecto.

Específicamente, el sistema desarrollado implementa una dinámica *Follow the leader* con esos tres drones, donde uno de ellos actúa como líder, al cual se le asigna una trayectoria preestablecida, y los otros dos tienen la misión de seguir al líder manteniendo una cierta formación y evitando colisiones. Los drones no cuentan con sensores externos a ellos que permitan tener información acerca de su entorno, por lo que, el desafío está en lograr los resultados esperados utilizando como información principal, las lecturas obtenidas de los módulos GPS, los acelerómetros y las brújulas presentes en cada aeronave. Estos datos deben ser comunicados a los miembros del enjambre para obtener un sistema que cuente con la información necesaria de todos los componentes en un mismo marco de trabajo y de esta forma lograr la dinámica deseada.

Los algoritmos diseñados, se programaron haciendo uso de un entorno robótico llamado ROS, que debió ser previamente estudiado en profundidad e instalado en los miembros del enjambre. Antes de desarrollar la dinámica de seguimiento al líder sobre los cuadricópteros construidos, se realizaron simulaciones por computadora en el software GAZEBO, que permitió probar los algoritmos sin mayores riesgos, utilizando modelos realistas tanto del ambiente como los drones.

Una vez obtenidos resultados satisfactorios en dichas simulaciones, se realizaron pruebas de campo con los drones reales. En este caso particular la misión asignada implicó realizar un círculo a cierta altura y velocidad determinada, manteniendo una formación triangular con ángulos y distancias preestablecidas.

Toda la información de posición, orientación y velocidad de cada integrante del enjambre fue guardada, tanto durante las simulaciones como durante las pruebas de vuelo, para ser analizada y graficada. Luego de este análisis se pudo concluir que todas las pruebas resultaron exitosas, aunque como era de esperar, existen diferencias apreciables entre los resultados que se obtuvieron en simulaciones y los obtenidos durante las pruebas de campo.

Esta página ha sido intencionalmente dejada en blanco.

Tabla de contenidos

Agradecimientos III

Resumen VII

Glosario XIII

1. Introducción **1**
1.1. Descripción del Proyecto 1
1.1.1. Objetivo General 1
1.1.2. Alcance 1
1.1.3. Motivación 1
1.2. Antecedentes 2
1.3. Clasificación de UAVs 4
1.4. Plataforma de Trabajo 5
1.5. Organización del Documento 6

2. Hardware **7**
2.1. Drones 7
2.1.1. Componentes Utilizados 7
2.1.2. Ensamblado 14
2.1.3. Configuración y Calibración de Sensores 15
2.1.4. LIDAR y Computadora a Bordo 16
2.2. Estación Terrestre 18

3. Acciones de Control de Alto Nivel **21**
3.1. Marcos de Referencia 21
3.2. Cambio de Coordenadas 22
3.3. Seguimiento del Líder y Evasión 24
3.3.1. Alternativa I: Estrategia de Tres Estados 26
3.3.2. Alternativa II: Campos Potenciales 29
3.3.3. Estrategia Mixta 37

4. Software **39**
4.1. Herramientas de Software 39
4.1.1. Robot Operating System (ROS) 39
4.1.2. Gazebo 41

4.1.3. SITL . . . 41
4.1.4. MAVROS . . . 42
4.1.5. QGroundControl . . . 43
4.2. Instalación y Tutoriales Realizados . . . 44
4.3. Layout del Proyecto . . . 45
4.3.1. Módulos de Software . . . 45
4.3.2. Organización de Archivos . . . 46
4.4. Implementacion en ROS . . . 47
4.4.1. Archivos Launch . . . 47
4.4.2. Procedimiento para Ejecutar los Launch . . . 50

5. Nodos Desarrollados **51**
5.1. Nodos Líder . . . 51
5.2. Nodos Seguidores . . . 53
5.3. Intercambio de Información . . . 56

6. Comunicaciones **59**
6.1. Comunicación Mavlink . . . 59
6.2. Redes de Comunicación Ad-Hoc . . . 60
6.3. Comunicaciones del Sistema . . . 61
6.3.1. Resumen . . . 63

7. Análisis del Control de Bajo Nivel **65**
7.1. Principios de Funcionamiento de un Cuadricóptero . . . 65
7.1.1. Modelo . . . 65
7.1.2. Interpretación . . . 67
7.2. Modos de Vuelo . . . 68
7.3. Control de Estabilidad y Vuelo . . . 68
7.3.1. Fusión Sensorial y Estimación de Posición . . . 69
7.3.2. Control de Posición y Actitud . . . 69

8. Pruebas Realizadas **73**
8.1. Simulaciones . . . 73
8.1.1. Líder . . . 73
8.1.2. Seguidores . . . 75
8.2. Fake GPS . . . 78
8.3. Pruebas de Campo . . . 78
8.3.1. Líder . . . 78
8.3.2. Seguidores . . . 79

9. Análisis de los Resultados Finales **83**
9.1. Simulaciones . . . 83
9.2. Pruebas de Campo . . . 87

10. Conclusiones y Trabajo Futuro **93**
10.1. Conclusiones 93
10.2. Trabajo Futuro 94

Apéndices **97**

A. Ensamblado y Configuración del Sistema **97**
A.1. Ensamblado - Fase 1 97
A.2. Configuración del Hardware 100
A.3. Ensamblado - Fase 2 102
A.4. Configuración de Comunicaciones 103
A.4.1. Comunicaciones entre drones 103
A.4.2. Comunicación con la Estación Terrestre 104

B. Cambios de Coordenadas **107**

C. Cuaterniones **113**
C.1. Fundamentos Básicos 113
C.2. Utilización de Cuaterniones para Imponer Cambios en la Actitud de un Dron 117

D. Constantes Elegidas **119**

E. Biblioteca ENDRA_lib **121**
E.1. Descripción de las Funciones Definidas en la Biblioteca 121

F. Planificación del Proyecto **125**

Referencias **127**

Esta página ha sido intencionalmente dejada en blanco.

Glosario

AMSL Acrónimo en inglés para altura sobre el nivel medio del mar, *Above Mean Sea Level*. 22

Cartas Terminología matemática en el sentido cartográfico, el objetivo es el de unir una serie de cartas o "mapas" que permitan definir completamente un atlas o "colección de mapas" de la totalidad de un espacio topológico al que se desea estudiar. 23

Collision Avoidance La traducción literal del inglés es evasión de colisiones. 77

ESC Acrónimo en inglés para controladora electrónica de velocidad, *Electronic Speed Controller*. 12

FCU Acrónimo en inglés para unidad controladora de vuelo, *Flight Control Unit*. 10, 12, 15, 41, 43, 49, 59, 68, 69, 78, 99, 101, 103

Follow the leader Dinámica en la que una aeronave líder es seguida por los demás miembros del enjambre, manteniendo una formación y distancia preestablecidas. 1, 13, 46, 51, 75, 81, 93

GCS Acrónimo en inglés para estación de control terrestre, *Ground Control Station*. Es un software que permite comunicarse con una aeronave y conocer su estado actual, así como enviarle comandos a la controladora de vuelo. 59

Motores Físicos Expresión empleada en informática para referirse a un conjunto de programas capaces de realizar simulaciones de ciertos sistemas físicos como la dinámica del cuerpo rígido, el movimiento de un fluido o la elasticidad. Se emplean fundamentalmente para los videojuegos y la simulación robótica. 41

Telemetría Es una tecnología que permite la medición remota de magnitudes físicas y el posterior envío de la información hacia el operador del sistema. Con la telemetría es posible llevar un control más preciso de las métricas relacionadas con un activo. 43

UAV Acrónimo en inglés para vehículo aéreo no tripulado, *Unmanned Aerial Vehicle*. 2–4, 21, 22, 43, 45, 49, 59, 62, 65, 67, 97, 105

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 1

Introducción

1.1. Descripción del Proyecto

1.1.1. Objetivo General

El objetivo de este proyecto fue construir una flota de tres drones y programarlos para que vuelen de forma autónoma realizando una dinámica de seguimiento al líder. Con este fin se debieron desarrollar acciones de control, tanto de seguimiento como de evasión de colisiones, y realizar pruebas de las mismas en un entorno virtual de simulación.

La dinámica *Follow the leader* implementada consiste en que uno de los drones actúe como líder del enjambre, volando en una trayectoria circular predefinida, mientras los otros dos actúan como seguidores, intentando mantener una cierta formación predefinida detrás de él.

La presente propuesta se enmarca en las actividades académicas conjuntas entre el Departamento de Sistemas y Control del IIE y el Departamento de Ingeniería Eléctrica y Sistemas de la Universidad de Pennsylvania.

1.1.2. Alcance

El alcance del proyecto incluyó el ensamblado de los drones a utilizar, su configuración y calibrado, el desarrollo de los algoritmos de control de alto nivel, su programación en C++ (en el sistema operativo robótico ROS) y la simulación de éstos en una flota de tres drones. Inicialmente se había previsto que de ser positivos los resultados de dichas simulaciones se realizaran pruebas de campo, las que finalmente pudieron ser llevadas a cabo.

1.1.3. Motivación

El uso de vehículos aéreos no tripulados se ha popularizado enormemente en los últimos años. Dadas sus características de bajo costo, autonomía y el bajo riesgo humano, ofrecen un gran potencial para uso tanto civil como militar.

En ese sentido, son muchas las aplicaciones que pueden ser realizadas formando enjambres, ya que se puede sustituir un sólo *UAV* (*Unmanned Aerial Vehicle*) con una cierta autonomía dada, por una flota de drones organizados de tal forma de cubrir esa misma tarea en un tiempo menor.

A medida que aumenta el número de robots implicados, mayor importancia adquiere la coordinación y prevención de colisiones. En estos casos los algoritmos reactivos calculan trayectorias en tiempo real que permiten evitar un obstáculo cuando una posible colisión es detectada.

1.2. Antecedentes

Si bien los vehículos aéreos no tripulados de uso civil se encuentran actualmente en pleno auge comercial, sus orígenes se remontan a principios del siglo XX con el surgimiento de algunas aeronaves pequeñas no tripuladas, contemporáneamente con los primeros vuelos exitosos de aviones tripulados.

Como ha sucedido reiteradamente a lo largo de la historia de la ingeniería, los primeros conceptos y desarrollos iniciales de aeronaves que podían operar sin un piloto en el aire tuvieron motivaciones militares. En este sentido, puede considerarse al *RAF Aerial Target* desarrollado entre 1916 y 1917 (durante la Primera Guerra Mundial) bajo las órdenes del ingeniero británico Archibald Low como el primer *UAV* de la historia [22].

El Aerial Target era un pequeño avión monoplano con un motor de 35 HP y un sistema de control remoto por radio frecuencia que permitía pilotarlo [22]. Durante el proyecto, se produjeron seis ejemplares, pero ninguno logró completar un vuelo con aterrizaje sano y salvo, por lo que la Royal Air Force lo transformó en lo que denominaron *"Flying Bomb"* que explotaba al finalizar su vuelo sin necesidad de realizar un aterrizaje.

A partir de esta idea, aplicando los posteriores avances tecnológicos que surgieron con el correr de los años (y las guerras) tales como el desarrollo de las telecomunicaciones, los motores eléctricos o la computación, junto con los grandes avances en la teoría del control, se lograron desarrollar una gran cantidad de vehículos no tripulados para la guerra. Éstos podían realizar tanto misiones radiocontroladas, como misiones completamente autónomas, y se popularizaron como "drones"(traducido del inglés literalmente como "zánganos", así bautizados por la publicación inglesa Popular Science en 1946) [20].

Con la caída del muro de Berlín y el fin de la guerra fría, fue luego de 1990 que se comenzaron a investigar y desarrollar las primeras aplicaciones no bélicas de aeronaves no tripuladas [21]. Un claro ejemplo fue el proyecto ERAST (Environmental Research Aircraft and Sensor Technology) por parte de la NASA, que se sostuvo durante nueve años y produjo grandes aportes a la comunidad científica en dicha área [19].

Durante los siguientes treinta años, y particularmente durante la última década, los drones se han vuelto cada vez más pequeños, más económicos y por tanto más populares. En la actualidad la industria de este tipo de pequeñas aeronaves para uso civil recibe millones de dolares anuales para crear nuevos productos con

diferentes tipos de uso, como la fotografía, cinematografía, agricultura, seguridad y entretenimiento, entre otras [7].

A pesar de todos los avances tecnológicos, existen algunos impedimentos que aún limitan la utilización de UAVs en ciertas tareas. Una de las limitantes más importantes es la autonomía, dada por la duración de las baterías que las aeronaves son capaces de cargar [21]. Si bien se han desarrollado baterías que pueden entregar una gran cantidad de energía (lo que afecta positivamente la autonomía), existe un compromiso entre dicha cantidad, y el peso y tamaño de las mismas (que afectan negativamente la autonomía).

Es aquí donde nace el interés por el estudio de enjambres o grupos colaborativos de drones, que juntos sean capaces de realizar en menor tiempo la misma tarea que un solo *UAV*, o bien que generen un sistema de relevos, permitiendo que otra aeronave continúe la tarea cuando la que se encontraba haciéndola agote su batería.

Hoy en día, en el estado del arte se pueden encontrar investigaciones sobre algoritmos de navegación en flotas de drones [4], comunicación entre varios vehículos aéreos no tripulados [8], o control adaptativo ante situaciones sorpresivas en enjambres [5], entre otros.

Existen diversos antecedentes de enjambres de drones con aplicaciones que ya se encuentran implementadas en distintos ámbitos, algunos ejemplos son [40]:

- Seguridad y patrullaje.
- Agricultura.
- Monitoreo de condiciones climáticas.
- Control de tráfico y peatones en ciudades (Smart-Cities).
- Búsqueda de personas en equipos de rescate.
- Aplicaciones militares.

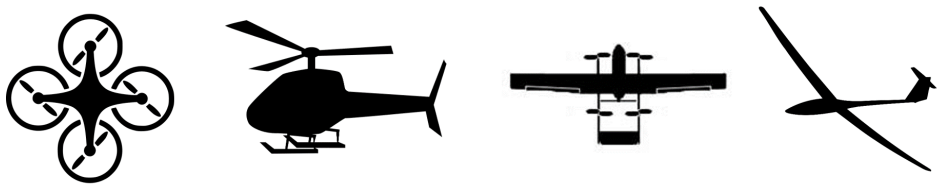


Figura 1.1: Diferentes variedades de UAVs. De izquierda a derecha: Cuadricóptero, Monorrotor, VTOL-Híbrido y Ala Fija.

1.3. Clasificación de UAVs

Existe una gran variedad de vehículos aéreos no tripulados como se aprecia en la Figura 1.1, que pueden ser clasificados según diferentes características, tales como estructura básica, tamaño, capacidad de volar sin ser controlados por un humano o cantidad de motores [40].

Tipo de Control:

Se denominan drones Radio-Controlados a aquellos que necesitan ser guiados permanentemente por un piloto en tierra que los opera mediante un sistema de control remoto. Por otro lado, se conocen como drones Autónomos o Robóticos a los que pueden realizar misiones autónomas que reciben por algún medio, procesan y ejecutan, navegando sin necesidad de contar con un piloto operando en tiempo real. Generalmente, los drones autónomos cuentan con un sistema de radio-control en caso de emergencia.

Estructura Básica:

Se denomina Dron de Ala Fija, a aquellos similares a un aeroplano, que no pueden despegar o aterrizar verticalmente ni permanecer suspendidos. Suelen ser los más veloces y con mayor autonomía. [40]

Se conocen como Multirrotores o Multicópteros a los drones con más de un motor colocado en el plano horizontal y que no cuentan con alas. Son capaces de despegar y aterrizar verticalmente, así como de permanecer flotando en el aire. Al contar con varios motores su autonomía en general es baja.

Un *UAV* con alas fijas montadas sobre un marco horizontal que cuenta con más de un motor (generalmente cuatro), recibe el nombre de VTOL-Híbrido. Es una combinación de las dos categorías anteriores, que permite despegar y aterrizar verticalmente con el sistema multirrotor, y avanzar rápidamente con el sistema de ala fija.

Por último, los drones Monorrotores son todos aquellos que cuentan con un sólo motor, y no cuentan con alas, en su mayoría tienen una estructura similar a

la de un helicóptero, aunque existen ejemplos más innovadores.

Cantidad de Motores:

Dentro de la categoría de los Multicópteros, es posible sub-clasificar las aeronaves según la cantidad de motores que presentan, lo que repercute en su estructura y sus características geométricas.

En este sentido, existen drones Tricópteros, Cuadricópteros, Hexacópteros y Octocópteros; que poseen tres, cuatro, seis y ocho motores respectivamente.

1.4. Plataforma de Trabajo

Siguiendo los criterios anteriores, las tres aeronaves construidas y utilizadas durante este proyecto pueden clasificarse como multirrotores cuadricópteros y autónomos. Serán presentados y analizados con mayor detalle en los capítulos siguientes.

Para poder implementar la dinámica de seguimiento al líder con estas aeronaves, se debieron diseñar algoritmos de control específicos para todos los miembros del enjambre. Por un lado se crearon algoritmos que le permitieran al líder recorrer una cierta trayectoria preestablecida, y por otro lado se desarrollaron dos estrategias distintas de seguimiento y evasión, para lograr que los seguidores mantuvieran la formación deseada y evitaran accidentes. Estas dos estrategias fueron evaluadas y finalmente fusionadas en una tercer estrategia "mixta", según se explica en la sección 3.3.

Dichos algoritmos fueron programados en los drones sobre un sistema operativo robótico (ROS), que se define como un sistema modular, donde cada módulo de programación es llamado "nodo" y los conjuntos de nodos son llamados "paquetes", que intercambian información a través de "tópicos".

En particular, se hizo uso de un paquete especifico para aplicaciones con drones llamado MAVROS y se programaron nodos independientes que implementen cada módulo de control diseñado. El paquete de ROS mencionado, permite traducir las acciones de control a comandos de un protocolo de comunicación estándar para UAVs, conocido como MAVLink, que será fundamental durante la puesta en práctica del proyecto.

Tanto las acciones de control como las características de las comunicaciones y el software utilizado o desarrollado serán estudiados en profundidad a lo largo del documento.

1.5. Organización del Documento

En el Capítulo 2 se describen los componentes y el proceso de ensamblado de las tres aeronaves utilizadas, así como las principales características de la computadora empleada como estación terrestre. A lo largo del Capítulo 3 se presentan las acciones de control diseñadas en el proyecto para llevar a cabo el seguimiento al líder y la evasión de colisiones al mismo tiempo.

Las principales herramientas de software utilizadas se introducen en el Capítulo 4, junto con una exposición detallada de la organización de los módulos programados y los problemas abordados durante el proyecto. Luego, se realiza en el Capítulo 5 una descripción completa de los programas que se ejecutan en cada aeronave, tanto para describir la trayectoria deseada en el líder, como para aplicar las estrategias de seguimiento y evasión en los seguidores.

En el Capítulo 6 se presentan los conceptos relacionados a las comunicaciones del enjambre, y se describen las interconexiones presentes entre los distintos módulos de hardware del sistema. Para completar la descripción de todas las partes del sistema implementado, en el Capítulo 7 se analizan los principios de funcionamiento que le permiten volar a un cuadricóptero, y se presenta brevemente la controladora automática de estabilidad y vuelo presente en las aeronaves del proyecto.

En cuanto a la práctica, en el Capítulo 8 se relata el proceso de pruebas realizadas durante el proyecto, incluyendo la simulación por computadora, las pruebas bajo techo y las pruebas de campo. Los resultados obtenidos en cada una de estas etapas se analizan en el Capítulo 9. Finalmente se expresan las conclusiones del proyecto en el Capítulo 10, junto con algunas ideas de trabajo futuro.

Capítulo 2

Hardware

En este capítulo se relatará el proceso de armado de los drones, junto con los componentes necesarios, además de los pasos importantes de configuración y calibración para que los mismos sean capaces de volar de forma estable y segura. Luego se expondrán las principales características de la estación de monitoreo terrestre y su configuración.

2.1. Drones

La elección de los componentes de hardware fue preestablecida dada la participación de este proyecto entre las actividades conjuntas del Departamento de Sistemas y Control del IIE y el Departamento de Ingeniería Eléctrica y Sistemas de la Universidad de Pennsylvania. En este contexto, el grupo contó desde un inicio con todos los componentes necesarios para construir los tres cuadricópteros autónomos.

2.1.1. Componentes Utilizados

Cada aeronave se puede dividir en tres grandes unidades, el esqueleto del dron, el cual se basa en el modelo comercial DJI F-450, la controladora de vuelo Pixhawk con los sensores y motores conectados a ella, y una computadora a bordo Intel NUC para lograr vuelos autónomos.

Se describen a continuación los componentes necesarios para el armado de los drones, la lista completa de dichos componentes con su descripción detallada puede leerse en la Tabla 2.1 y el conjunto completo se observa en la Figura 2.1.

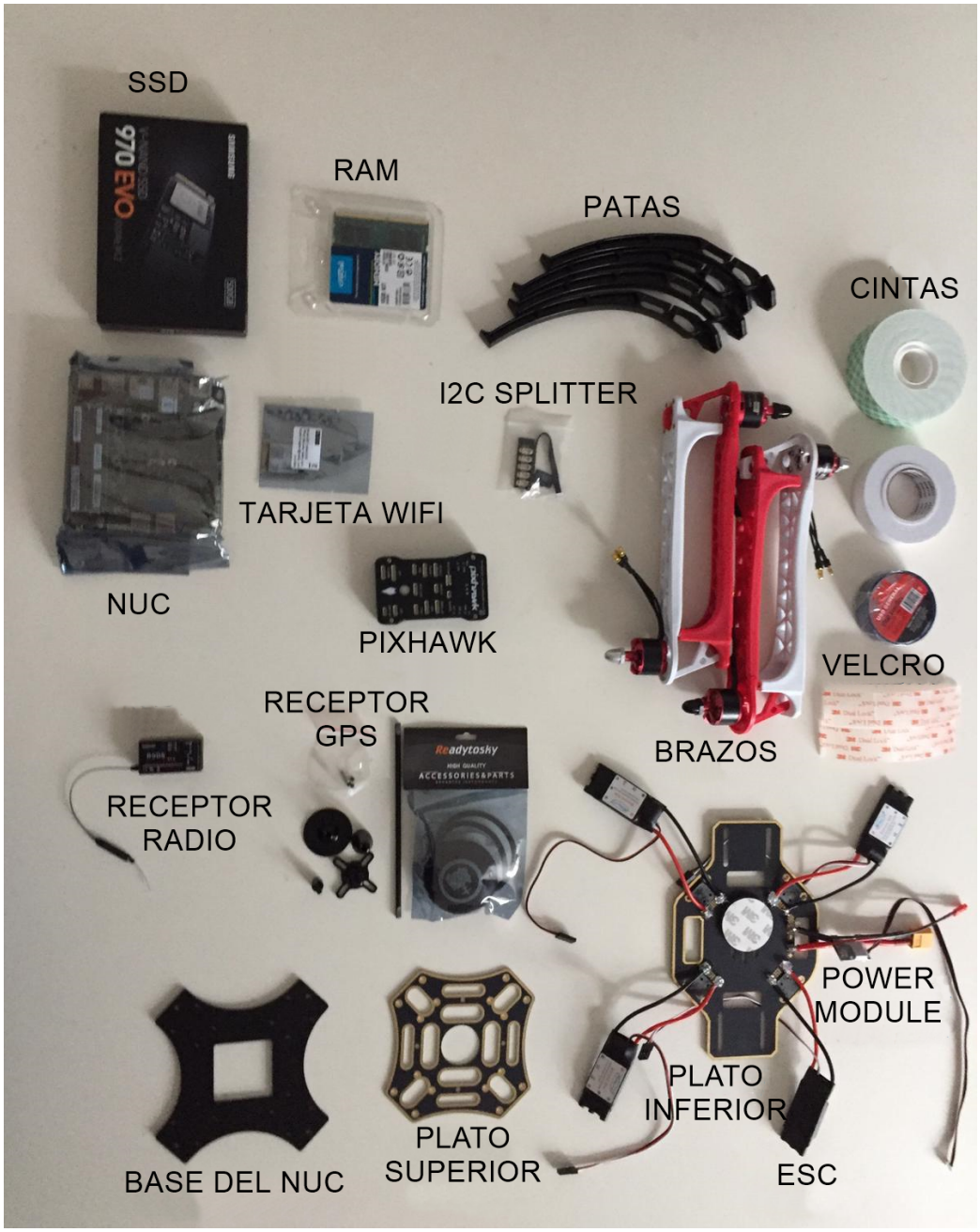


Figura 2.1: Piezas necesarias para construir un cuadricóptero.

Pieza	Cantidad	Marca	Modelo
Plato Inferior	1	DJI	F450
Plato Superior	1	DJI	F450
Brazos	4	DJI	F450
Patas	4	MallofUSA	F450
Hélices	4	GEMFAN	1045
Radio Control	1	RadioLink	AT9S
Receptor Radio	1	RadioLink	R9DS
Motores	4	ReadytoSky	2212 920kv
ESC	4	HobbyPower	Simonk 30A
Módulo de Alimentación	1	CRIUS	9 to 28V
Receptor GPS	1	ReadytoSky	Ublox M8N
LIDAR	1	Garmin	LidarLite V3
FCU	1	3DR	Pixhawk
Computadora	1	Intel	NUC 7 i3DNB
NUC Power Regulator	1	Pololu	D36V50F12
Memoria RAM	2	Crucial	8GB DDR4
Disco SSD	1	Samsung	V-NAND 970EVO
Tarjeta WiFi	1	Intel	AX200

Tabla 2.1: Desglose completo de las piezas que componen cada cuadricóptero.

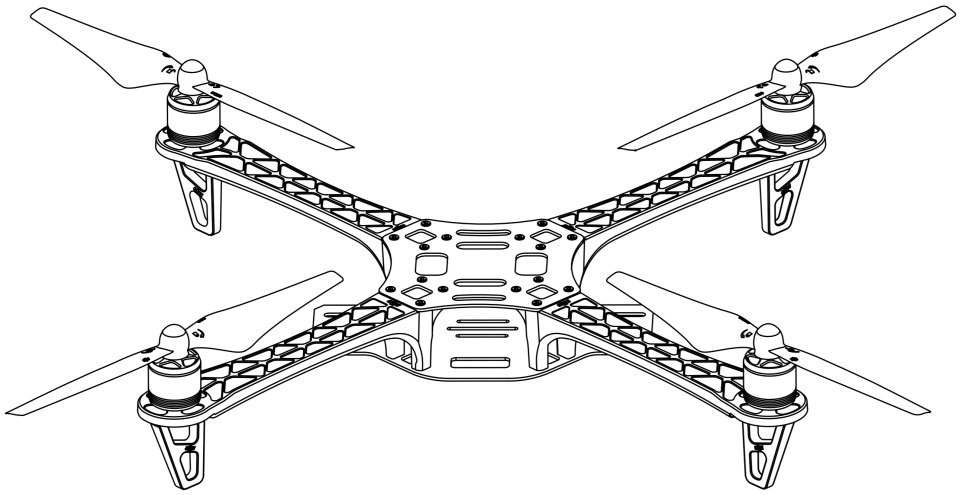


Figura 2.2: DJI F-450 frame.

Estructura

La estructura del cuadricóptero (*frame*) se basa en el modelo comercial DJI F-450 que se ilustra en la Figura 2.2. El mismo se compone de una base o plato central con pistas conductoras para la alimentación, a la que se le ajustan las partes inferiores de los cuatro brazos de plástico resistente a impactos. Por encima de los brazos se coloca un segundo plato metálico, que le aporta rigidez a la estructura y extiende el espacio disponible para colocar los componentes.

Adicionalmente, se le añade debajo de la base un juego de cuatro patas de plástico, que permitan colocar la batería debajo del dron con un velcro con sistema de ajuste.

Pixhawk

La controladora de vuelo utilizada es una Pixhawk Autopilot como la de la Figura 2.3, a la que se le instaló un firmware PX4, versión 1.10.1, el cual se explicará en el Capítulo 7.

También llamada *FCU* (*Flight Controller Unit*), la controladora de vuelo se encarga de obtener y procesar la información de los sensores, y en base a ella enviar órdenes a los actuadores para alcanzar un fin específico, como mantenerse estable en una posición, reaccionar ante las directivas recibidas por radio control, o completar una misión autónoma previamente cargada.

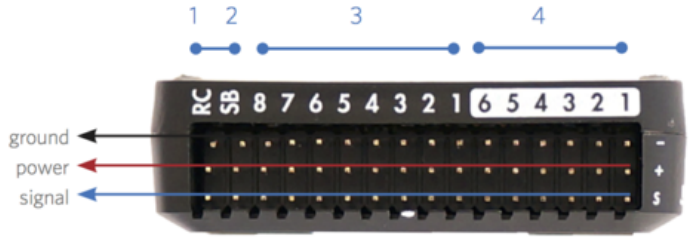
El Pixhawk se alimenta de un voltaje DC de 5 V y tiene un procesador de 32 bits, con un reloj de 168 MHz, 256 KB de RAM y 2 MB de memoria flash, además cuenta con un conjunto de sensores integrados incluyendo dos giróscopos, dos acelerómetros, un magnetómetro (brújula) y un barómetro. El módulo posee diversos conectores para comunicarse con módulos GPS, sensores de altura, cámaras, sensores de viento, motores, servomotores y radio controles [28].



1 Spektrum DSM receiver
2 Telemetry (radio telemetry)
3 Telemetry (on-screen display)
4 USB
5 SPI (serial peripheral interface) bus
6 Power module
7 Safety switch button
8 Buzzer
9 Serial
10 GPS module
11 CAN (controller area network) bus
12 I²C splitter or compass module
13 Analog to digital converter 6.6 V
14 Analog to digital converter 3.3 V
15 LED indicator



1 Input/output reset button
2 SD card
3 Flight management reset button
4 Micro-USB port



1 Radio control receiver input
2 S.Bus output
3 Main outputs
4 Auxiliary outputs

Figura 2.3: Controladora de vuelo Pixhawk con sus puertos de entrada y salida [28].

Motores

Para impulsar al UAV se utilizan cuatro motores eléctricos sin escobillas (*brushless*) Readytosky 2212 con hélices plásticas de 10 pulgadas.

Los motores sin escobillas se utilizan en el ensamblado de drones dadas sus ventajas relativas al costo, peso, tamaño y mantenimiento, pero requieren de un control específico, para el que deben utilizarse controladoras electrónicas de velocidad (ESC, del inglés *Electronic Speed Controller*), comandadas desde la *FCU* Pixhawk.

En este caso se utilizaron ESCs modelo SimonK 30A. Estas controladoras se alimentan directamente desde la batería, y reciben (desde la controladora de vuelo) una señal PWM (Pulse-Width Modulation) proporcional a la velocidad objetivo, que traducen en un voltaje alterno trifásico que se entrega a los motores *brushless*.

Módulo de Alimentación

En el caso más extremo, si la batería LiPo de 4 celdas se encuentra totalmente cargada puede entregar 16,8 V al sistema, y cada *ESC* soporta hasta 30 A, por lo que la potencia entregada por la batería puede ser muy grande y cambiar rápidamente si el dron frena o acelera bruscamente.

Es por esto, que se coloca a la entrada del circuito, luego del conector de la batería, un controlador de potencia que asegura voltaje y corriente estable hacia los motores. El mismo puede recibir hasta 28 V y 90 A, y adaptar también una alimentación de 5 V para la *FCU* Pixhawk, entregando información sobre tensión y corriente total consumida a la misma para poder estimar el consumo de batería.

Radio Control

El sistema de radio control se compone de un transmisor (RadioLink AT9S) y un receptor (RadioLink R9DS), que una vez apareados se comunican a través de una onda de radio en la banda de 2,4 Ghz. Según el fabricante [33], presentan un alcance máximo de 3400 m, y pueden comunicarse con la controladora de vuelo por un puerto SerialBUS o por PWM. En particular en este proyecto solo se utiliza para situaciones de emergencia donde se requiera tener el control directo sobre los movimientos del dron.

Módulo GPS

El módulo GPS se conecta a la controladora para medir la posición global del cuadricóptero. Cuenta con una antena receptora y una magnetómetro (brújula) externo que debe sincronizarse con el magnetómetro interno de la *FCU*. De esta manera, la controladora puede conocer su posición actual en términos de latitud, longitud y altitud sobre el nivel del mar (AMSL), para poder trazar rutas y desplazarse hacia una cierta coordenada durante las misiones o el vuelo autónomo.

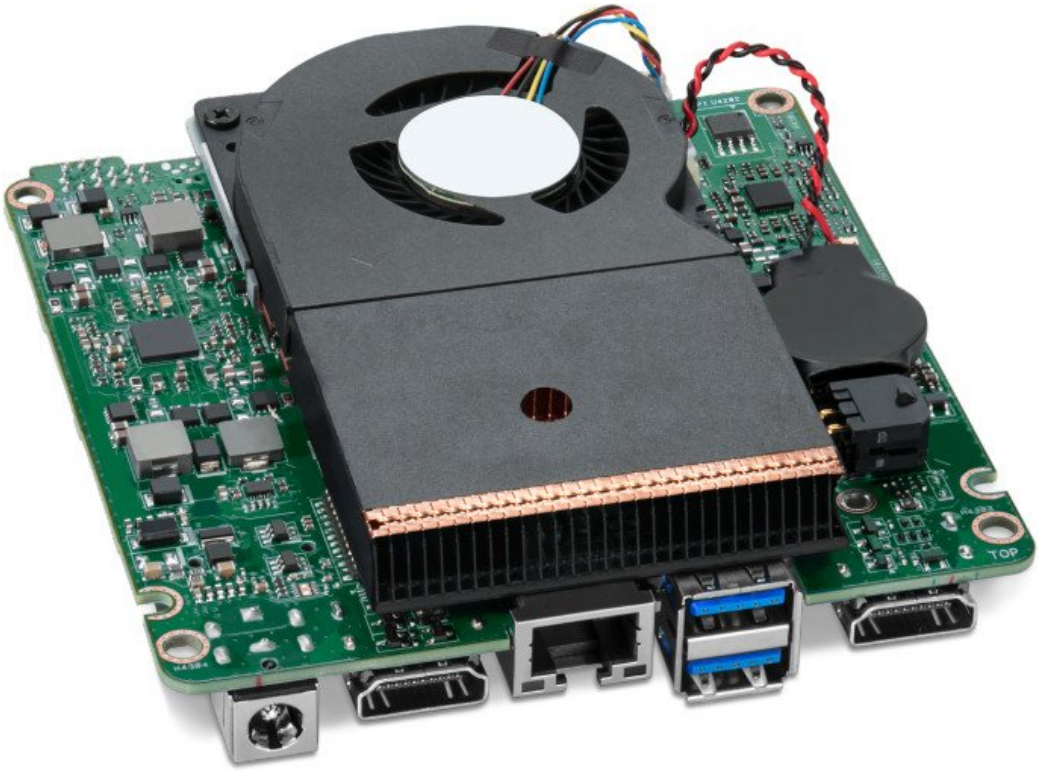


Figura 2.4: Mini-computadora Intel NUC.

LIDAR

El LIDAR (del inglés, *Laser Imaging Detection and Ranging*) es un sensor óptico que emite un rayo láser, mide el tiempo de rebote y con ello estima la distancia a la que se encuentra el objeto sobre el que rebotó. Colocando un LIDAR en el dron, apuntando verticalmente hacia abajo, se obtiene un sensor capaz de informar la altura de la aeronave con respecto al suelo.

Esto resulta particularmente útil para mantener una altura fija sobre el suelo en terrenos desnivelados o vuelos autónomos a grandes distancias, dado que los demás sensores de altura son capaces solamente de medir diferencias con respecto a la altura inicial (barómetro) o respecto al nivel del mar (GPS) y son más imprecisos.

Computadora de a Bordo

Para que cada aeronave pueda considerarse realmente autónoma y finalmente se pudiera llevar a cabo la dinámica *Follow the leader*, cada una de ellas fue equipada con una computadora a bordo, basada en una mini-PC Intel NUC modelo 7 i3DNB que puede observarse en la Figura 2.4. La misma cuenta con un procesador Intel Core i3, varios conectores USB, HDMI y requiere una alimentación de 12 V. A dicha placa se le añadieron dos memorias RAM, un disco SSD y una tarjeta de red WiFi (AX200) con sus respectivas antenas. En la Tabla 2.2 se observan las especificaciones detalladas del sistema.

Componente	Especificación
Procesador	Intel Core i3 - 7100U Dual Core
Memoria RAM	16 GB DDR4
Disco Duro	500 GB SSD
Conectividad	Wi-Fi 6 (802.11ax)
Sistema Operativo	Ubuntu 18.04 LT

Tabla 2.2: Especificaciones técnicas de las computadoras de a bordo utilizadas.

Cada computadora se comunica con las demás a bordo de los otros drones y con la estación terrestre mediante conexión inalámbrica y se conecta a la controladora Pixhawk mediante una conexión serial USB.

Regulador de Tensión para el Procesador NUC

Es un regulador reductor de voltaje, es decir convierte un voltaje de entrada alto en un voltaje inferior. En particular el modelo utilizado en los drones, permite un voltaje de entrada de hasta 50 V y un voltaje de salida de 12 V. Este dispositivo se utilizó para alimentar la computadora a bordo Intel NUC. La batería utilizada puede entregar como máximo 16,8 V y luego de pasar por el regulador de voltaje se conecta al NUC entregando de forma estable los 12 V necesarios.

2.1.2. Ensamblado

Con todos los componentes descritos anteriormente se procedió a ensamblar cada uno de los drones. Se comenzó soldando los cables de alimentación, junto con el módulo de alimentación y las ESCs sobre el plato inferior, al que se le afirmó la controladora Pixhawk, el receptor de radio y el GPS. Luego se agregaron las patas y los brazos a dicho plato inferior, se colocaron los motores y finalmente se añadió el plato superior. Este proceso de ensamblado se describe paso a paso en el Apéndice A.1.

En este punto, se obtuvo un dron como el de la Figura 2.5, en el cual se observa que aun no está presente la computadora de a bordo Intel NUC, por lo que no puede considerarse autónomo.



((a)) Dron- Imágen frontal. ((b)) Dron- Imagen posterior.

Figura 2.5: Resultados de la primera etapa del armado del dron.

2.1.3. Configuración y Calibración de Sensores

Una vez finalizada la primer etapa de ensamblado, se realizó la configuración y calibración de la controladora de vuelo y sus sensores. Si se desea, en el Apéndice A.2 se puede encontrar este proceso de forma detallada.

En primer lugar, fue necesario aparear el control remoto con el receptor presente en el dron, para de esta forma poder controlarlo a través del radio-control.

Posteriormente se configuró y calibró la *FCU* Pixhawk junto con los sensores conectados a ella, para ello se la debió conectar a una computadora a través de un cable USB, y ejecutar la herramienta de software QGroundControl, que permite configurar parámetros de la controladora manualmente, calibrar los sensores (como el acelerómetro o las brújulas) y monitorizar los drones. Se ilustra dicha conexión en la Figura 2.6.

Luego se debió comprobar que el dron ya configurado y con sus sensores calibrados era capaz de volar manualmente. Para esto se colocaron las hélices y se realizó una prueba de vuelo radio-controlado al aire libre.

Antes de comenzar un vuelo, ya sea radio-controlado, en misión o autónomo, siempre es necesario **armar** el dron, esto implica que se han superado todos los chequeos automáticos previos (*pre-arm checks*), por ejemplo que los sensores se encuentren correctamente calibrados, que la batería entregue un nivel aceptable de voltaje, que un *switch* de seguridad en la *FCU* se encuentre desactivado, entre otros. Una vez que se han superado dichos chequeos, se energizan los motores y éstos comienzan a girar lentamente para indicar que la aeronave está en condiciones de comenzar a volar.

En este sentido, es también relevante el concepto de **failsafe**, que corresponde a un sistema de seguridad que ayuda a recuperar el vehículo sin correr riesgos en caso de que existan problemas en medio de un vuelo, como baja batería, pérdida de señal o problemas con la calidad de GPS. Tanto las acciones a tomar, como los eventos que las producen pueden ser configurados en QGroundControl [25].



Figura 2.6: Cuadricóptero radio-controlado, conectado por USB a una PC con QGroundControl para el ajuste de parámetros y calibración.

2.1.4. LIDAR y Computadora a Bordo

Al haber obtenido resultados exitosos durante las pruebas de vuelo con radio control, se procedió a completar el ensamblado del dron autónomo. Para esto fue necesario agregar el sensor LIDAR y la computadora de a bordo Intel NUC (con todos sus componentes), que se montó sobre el plato superior. Este proceso puede encontrarse descrito en el Apéndice A.3.

Luego de tener el dron autónomo completamente armado como se observa en la Figura 2.7, se procedió a instalar el sistema operativo Ubuntu 18,04 LT y las herramientas que se mencionarán en el Capítulo 4 sobre la computadora a bordo, para finalmente realizar alguna prueba sencilla bajo techo, por ejemplo, correr un programa que simplemente intente un despegue y aterrizaje, sin hélices. Este procedimiento se detalla en la Sección 8.2.

Una vez obtenidos resultados positivos en dichas pruebas con cada dron, se procedió a configurar una red Ad-Hoc para que las aeronaves pudieran comunicarse entre sí y con la estación de monitoreo terrestre. El detalle de cómo se estableció la red Ad-Hoc y porqué resulta necesaria se explica en el Capítulo 6.



((a)) Dron terminado - Imagen frontal.



((b)) Dron terminado - Imagen posterior.



((c)) Dron terminado - Imagen superior.

Figura 2.7: Resultado final del armado de los cuadricópteros autónomos.



Figura 2.8: Computadora y Dongle utilizados como estación terrestre durante las pruebas de campo.

2.2. Estación Terrestre

La estación de control terrestre consta de una *notebook* con el Sistema Operativo Ubuntu 18,04 LT, el entorno robótico ROS *Melodic* instalado junto con el paquete MAVROS, y el programa QGroundControl. Cuenta además con un adaptador de red (*Dongle*) conectado a uno de sus puertos USB, como la que se observa en la Figura 2.8. Las especificaciones técnicas de la computadora utilizada durante la simulación y las pruebas de campo pueden observarse en la Tabla 2.3.

Sobre el entorno ROS se ejecutan los programas desarrollados; a través del software QGroundControl se puede revisar el estado de la aeronave en tiempo real, mientras que el estado de la red, y de cada una de las computadoras de a bordo puede ser monitorizado de forma remota mediante conexiones de protocolo *SSH*. A continuación, en el Capítulo 4 se brindan detalles de las herramientas de software mencionadas, y en el Capítulo 6 se detallará el proceso para establecer la conexión *SSH*.

Componente	Especificación
Procesador	2.2 GHz Intel Core i7-8750H Six-Core
Memoria RAM	16 GB, DDR4
Disco Duro	512 GB PCIe SSD
Tarjeta Gráfica	NVIDIA GeForce GTX 1060 (6GB GDDR5)
Conectividad	Wi-Fi 5 (802.11ac)
Sistema Operativo	Ubuntu 18.04 LT
Dongle WiFi	TpLink WN722N 150 Mbps

Tabla 2.3: Especificaciones técnicas de la computadora utilizada en tierra durante el vuelo y las simulaciones.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 3

Acciones de Control de Alto Nivel

Para poder alcanzar los objetivos planteados, se requieren desarrollar acciones de control que gobiernen los movimientos de cada integrante del enjambre. Con este fin, en primer lugar deben identificarse las entradas y salidas al sistema a controlar, y separarlo en capas de control con fines más específicos.

Por un lado, el control de más bajo nivel en el sistema toma como entrada la información de todos los sensores, tales como giroscopio, brújula, módulo GPS, acelerómetro y LIDAR. Luego procesa dicha información, y produce como salida la velocidad angular que debe imprimirse a cada motor para mantener la estabilidad en cada aeronave o moverse en cierta dirección objetivo. Estas acciones de control están estandarizadas, se computan dentro de la controladora de vuelo Pixhawk y serán estudiadas con mayor profundidad en el Capítulo 7.

Por otra parte, debe desarrollarse un control de más alto nivel, que reciba como entradas la posición y velocidad actual de cada componente de la flota, y con ellas sea capaz de determinar cuál debe ser la posición futura de cada UAV en el espacio, de forma de lograr describir la trayectoria deseada y mantener la formación entre líder y seguidores. Dicha posición futura debe calcularse teniendo en cuenta la situación global del enjambre, para evitar colisiones entre los componentes. En el caso del control de alto nivel, las acciones se computan en el entorno ROS, entre las computadoras de a bordo y la estación terrestre, y serán presentadas en este capítulo.

3.1. Marcos de Referencia

Es importante destacar que para conocer la ubicación de una aeronave en el espacio en un instante dado, es preciso conocer su **posición** y **actitud** en dicho instante. Ambas pueden definirse como vectores de tres entradas.

La posición puede expresarse en diferentes marcos de referencia, por ejemplo:

- $\{latitud, longitud, altitud\}$ en coordenadas geográficas (marco global)
- $\{x, y, z\}$ en coordenadas cartesianas (marco local)
- $\{\rho, \phi, z\}$ en coordenadas cilíndricas (marco local)

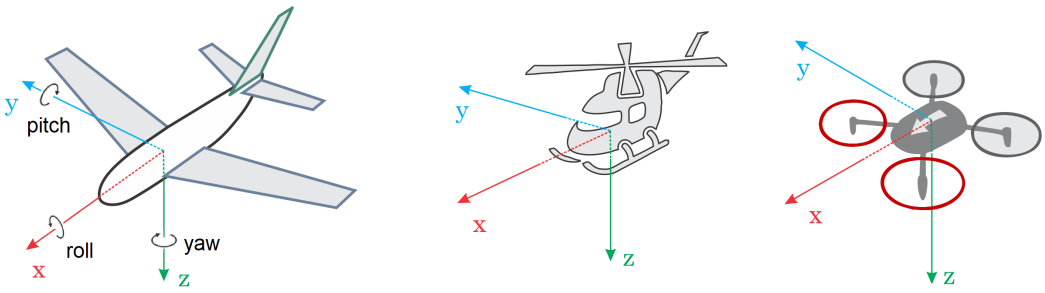


Figura 3.1: Sistema solidario a la aeronave y definiciones de movimientos angulares [26].

Los marcos locales requieren de un origen de coordenadas, que debe ser explicitado cuando se expresa alguna posición en un marco de este tipo. A modo de ejemplo, el firmware PX4 de la controladora Pixhawk utiliza como origen para sus coordenadas locales (cartesianas) el punto donde se enciende la misma.

Además, cuando se utilizan coordenadas geográficas, la altitud puede ser medida con respecto al nivel del mar (*AMSL*), respecto a la altura de despegue (*Home*) o respecto al suelo (*Relative*). En cualquier caso, cuando se habla de posición es necesario especificar el marco de referencia y las unidades utilizadas para la medida.

La actitud indica la orientación angular del dron y siempre se expresa en un marco solidario a la aeronave. Suele mencionarse en la aeronáutica en términos de *pitch* (cabeceo), *roll* (alabeo) y *yaw* (guiño) en correspondencia con el vector de ángulos $\{\theta, \phi, \psi\}$ respectivamente, dichas definiciones se ilustran sobre diferentes tipos de aeronave en la Figura 3.1.

Una forma muy común de representar la orientación son los **cuaterniones** [41]. Éstos son un álgebra sobre $\mathbb{R}$ que permite representar rotaciones en el espacio tridimensional y componerlas de manera similar a como el álgebra de números complejos permite representar rotaciones en el plano. Si el lector desea profundizar en el tema puede recurrir al Apéndice C.

3.2. Cambio de Coordenadas

Para aplicar las estrategias de seguimiento y de evasión, se asume que cada dron seguidor conoce la posición y velocidad de todos los otros drones. Para esto, se debe contar con un sistema de referencia absoluto en el cual cada *UAV* publique sus correspondientes coordenadas. Con este fin, se aprovechó que los drones conocen su posición global en el mundo con buena precisión a partir de reunir la información de todos sus sensores mediante un *filtro de Kalman*, el cual se introduce en la Sección 7.3.1.

Por lo tanto, se hizo uso de la latitud, longitud, y altura sobre el nivel del mar para determinar las posiciones de los drones. Pero este sistema de coordenadas debió convertirse en uno de carácter cartesiano con los ejes en unidades de metros, que permitiera enviar direcciones a cada *UAV* y entender la información de manera simple. La obtención en detalle de este cambio de coordenadas puede encontrarse

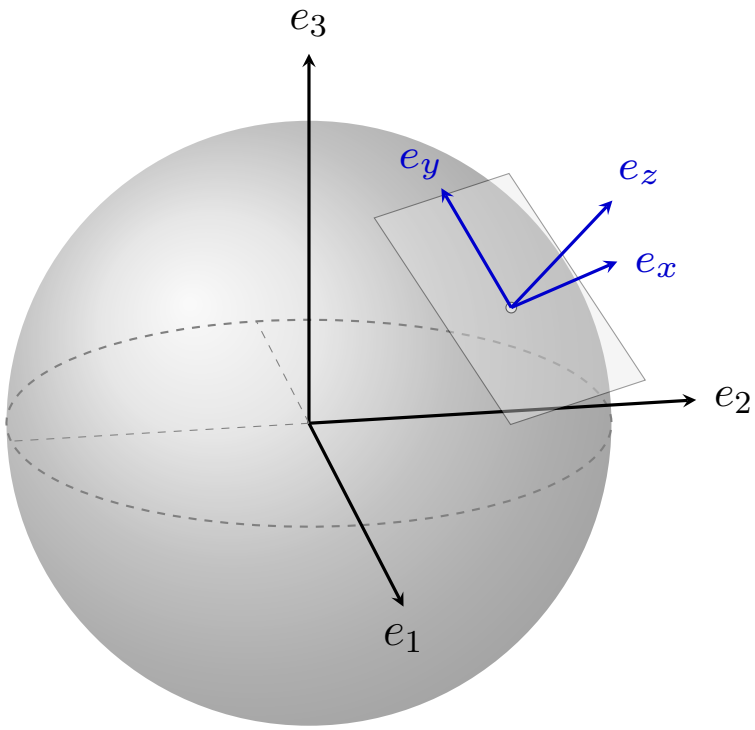


Figura 3.2: Posición sobre la Tierra de los versores $\{e_x, e_y, e_z\}$ del nuevo sistema de coordenadas.

en el Apéndice B.

De forma concisa, para esto se modela al planeta Tierra como un elipsoide, y se hace una aproximación lineal (Taylor de primer orden) a las *Cartas* que definen el sistema de coordenadas $(latitud, longitud, altura)$ en una base que tenga dos vectores tangentes al elipsoide (tangentes al suelo en ese punto) y otro vector normal al elipsoide (indicando la dirección vertical).

La aproximación de primer orden realizada tiene un gran rango de validez sin perder precisión. Esto se debe a que tanto los radios polares como ecuatoriales de la Tierra son extremadamente grandes en comparación con las distancias que se desplazan los drones, haciendo que las variaciones en latitud y longitud sean sumamente pequeñas.

El sistema coordenado cartesiano resultante se expresó en una base tal que las direcciones principales (e_x, e_y, e_z) crecen en dirección hacia el este, hacia el norte y hacia arriba respectivamente, como indica la Figura 3.2.

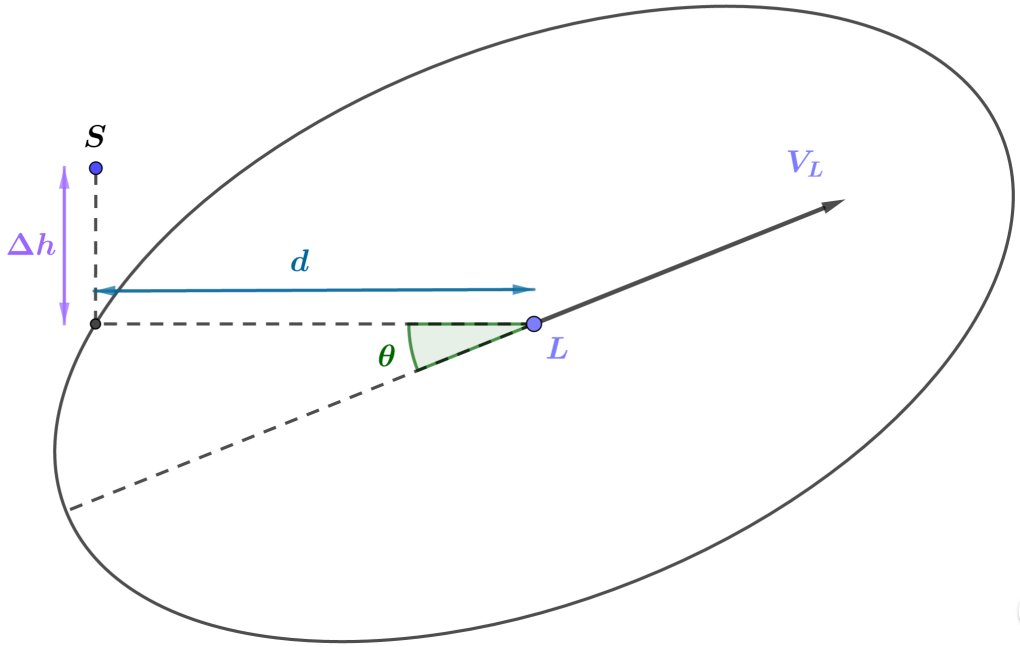


Figura 3.3: Posición donde se desea ubicar a un seguidor (S) respecto del líder (L).

3.3. Seguimiento del Líder y Evasión

En la dinámica de seguimiento del líder, se buscará que los seguidores mantengan cierta formación y eviten colisiones entre drones, es decir que solo los drones seguidores serán encargados de evitar colisiones, mientras que el líder puede desplazarse libremente. Concretamente, mientras el líder (L) está moviéndose con una velocidad con componente horizontal $V_L \in \mathbb{R}^2$, encontrándose en una posición (x_L, y_L, z_L), el seguidor (S) busca mantener la formación que se bosqueja en la Figura 3.3, manteniendo constante el ángulo θ, la distancia horizontal d y la diferencia de altura Δh. Un caso particular de simulación con dos seguidores, siendo $\theta = \pm 30°$, $d = 2$ m y $\Delta h = 0$ m se aprecia en la Figura 3.4.

Es decir, que si se consideran las proyecciones en el plano xy, la posición del seguidor relativa al líder debe mantener un ángulo θ con la recta tangente a la trayectoria del líder (como se observa en la Figura 3.3 proyectando en el plano horizontal, θ es el ángulo que debe formar la velocidad del líder con la posición relativa del líder respecto del seguidor), a una distancia horizontal deseada d. En cuanto a la altura, se buscará mantener una diferencia $\Delta h = z - z_L$ constante con respecto al líder siempre que sea posible (por seguridad, existe una altura mínima de vuelo h_{min} para los seguidores que según la altura del líder podría imposibilitar mantener esta diferencia).

Para lograr esto, se analizaron una serie de trabajos existentes relacionados a estrategias de seguimiento y evasión en robótica. Sin embargo, luego de evaluar las soluciones presentes en el estado del arte, se encontró que ninguna se ajustaba

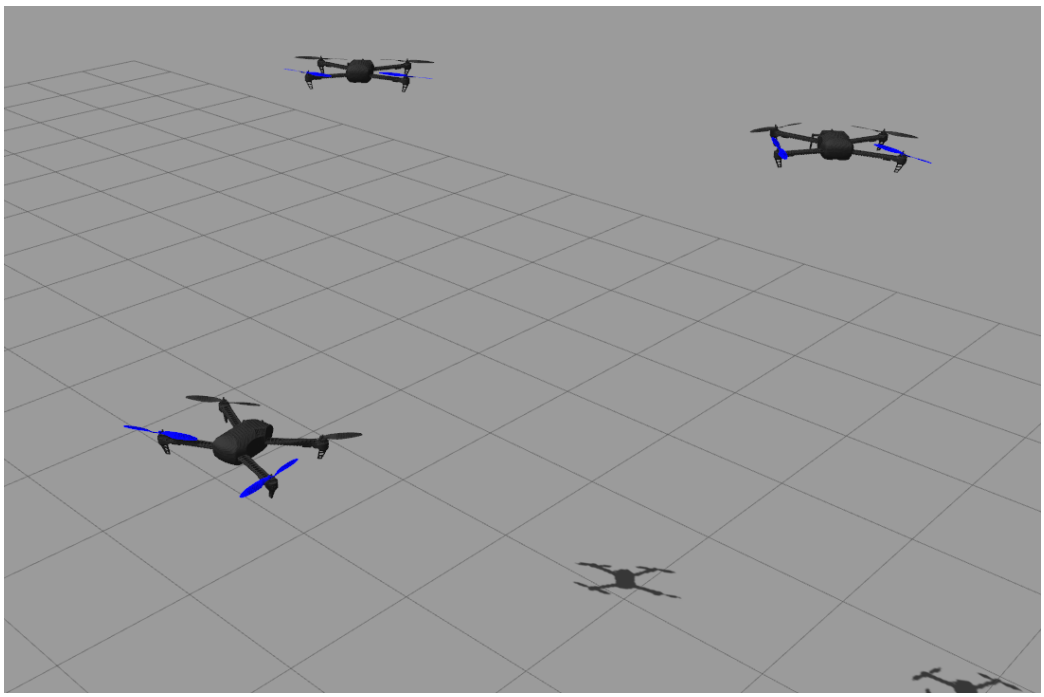


Figura 3.4: Formación de líder (debajo a la izquierda) y dos seguidores realizando una trayectoria circular en el simulador Gazebo. En este caso $\theta = \pm 30°$, $d = 2$ m y $\Delta h = 0$ m

completamente a las restricciones particulares del problema en cuestión. Por un lado, algunos artículos asumen que los obstáculos a evadir se encuentran quietos (como paredes, árboles o barreras) [11] [42], mientras que otros asumen que todos los agentes del sistema (incluyendo al líder) colaboran en la evasión [9] [12].

Finalmente, se plantearon dos estrategias alternativas, descritas en las secciones 3.3.1 y 3.3.2, las cuales fueron combinadas en una nueva estrategia, que se presenta en la Sección 3.3.3, para disponer de las ventajas de las dos anteriores.

Cabe aclarar que si bien las aeronaves se encuentran en el espacio (tres dimensiones), estas alternativas serán planteadas en el plano (dos dimensiones). Esto aumenta la complejidad del problema, no permitiendo que los drones cambien de altura para evitar colisiones, pero brinda mayor seguridad en vuelos que se realicen a poca altura.

Por otro lado, es importante mencionar que los movimientos de los drones en el espacio serán implementados de forma tal que su orientación acompañe la trayectoria, es decir que siempre "miren hacia adelante". Esta decisión podría impactar negativamente en la complejidad y eficiencia energética de la evasión de colisiones, pero se consideró como una oportunidad para aplicar conocimientos de interés para el grupo (como por ejemplo el álgebra de cuaterniones, detallado en el Anexo C).

3.3.1. Alternativa I: Estrategia de Tres Estados

La primer estrategia que se usó para determinar la dinámica del enjambre, fue creada por el grupo de proyecto, y consistió en darle tres estados posibles a cada dron; "esquivar", "detenerse" y "seguir". Esta alternativa se implementó solamente entre un dron seguidor y un dron líder. Dos seguidores pueden implementar esta estrategia al mismo tiempo, pero estos no coordinarán entre sí, por lo que la estrategia no contempla las posibles colisiones entre seguidores. Según se explica a continuación, un dron se encontrará en uno de estos tres estados dependiendo de una serie de factores, como son la distancia entre el dron seguidor y el dron líder, la magnitud de la velocidad del líder, y el ángulo que forma dicha velocidad con la posición relativa entre ambos drones. Se explicará primero en que consiste cada uno de estos tres estados.

Descripción de los Estados:

Seguir: Un seguidor que se encuentra en este estado, intenta posicionarse en el punto X_G a distancia horizontal d del líder, manteniendo el ángulo deseado θ entre su posición relativa al líder y la velocidad del mismo. La actitud de un dron que se encuentra en este estado se elige de modo que se oriente en el sentido de su propia velocidad (los seguidores avanzan "mirando hacia adelante").

Esquivar: Un seguidor que se encuentra en este estado, intenta moverse de manera perpendicular a la velocidad del líder, alejándose de su trayectoria de modo de no interponerse en su camino. El seguidor avanza de manera perpendicular, alejándose de la recta que pasa por el dron líder con la dirección dada por la velocidad del mismo, hasta encontrarse en el punto X_G a una distancia d_{segura} (predefinida) que se considere segura. La actitud de un dron en este estado se elige del mismo modo que en el estado de seguimiento, es decir que el dron se orienta en el sentido de su velocidad.

Detenerse: Es necesario una tercer clase de comportamiento para el caso en que el líder se encuentre quieto (o moviéndose a muy bajas velocidades). Si se usara el comportamiento del estado "Seguir"para este caso, entonces una mínima oscilación del líder (podría ser producida por el viento o incluso deberse a ruido en la medida de la velocidad) haría que el seguidor se moviera bruscamente a distancia d de él.

Un seguidor que se encuentra en este estado, intenta acercarse radialmente hacia el líder hasta alcanzar el punto X_G a la distancia deseada d, sin importar las direcciones de movimiento del líder. Una vez alcanzada esa posición, intenta mantenerse estable allí, manteniendo el ángulo con el líder (sin rotar entorno a él) y la distancia d con el mismo.

Condiciones de Cambio de Estado:

Un dron seguidor se encontrará en el estado **Esquivar** si satisface alguna de las siguientes dos condiciones, que indican riesgo de colisión:

1. La primer condición consiste en que se den simultáneamente tres eventos:
   - El dron seguidor se encuentra dentro del cono cuyo vértice está dado por la posición del líder y su eje está dado por la velocidad del líder, tal que el ángulo del cono al eje es θ_{riesgo} , previamente fijado.
   - La velocidad horizontal del líder supera un cierto umbral v_{riesgo} predefinido.
   - La distancia horizontal del seguidor al líder es menor que una distancia predefinida R_{riesgo}.
2. La segunda condición ocurre si la distancia horizontal del seguidor al líder es menor que una distancia predefinida $R_{int} < d$

Si no se cumplen las anteriores condiciones, el dron cae en el estado **Detenerse**, siempre que se cumple la siguiente condición:

- La velocidad horizontal del líder es menor al umbral v_{riesgo}

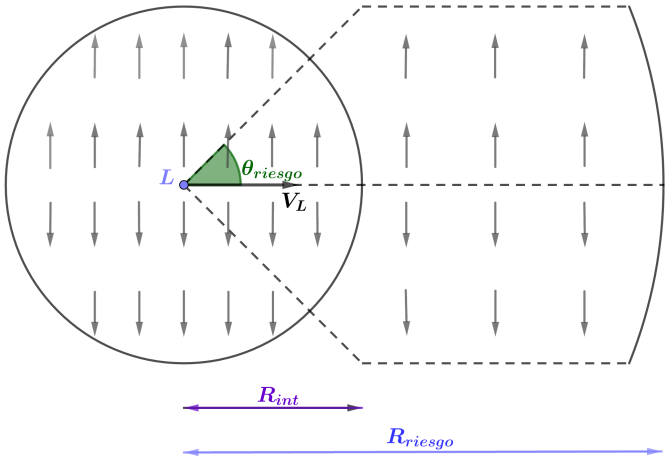
En cualquier otro caso, el dron seguidor se encuentra en el estado **Seguir**, dado que el líder se mueve significativamente y no existe riesgo de colisión.

En la Figura 3.5 se bosquejan las condiciones a cumplir para caer en cada uno de los tres estados posibles, y el comportamiento a tomar por los seguidores en dichos estados.

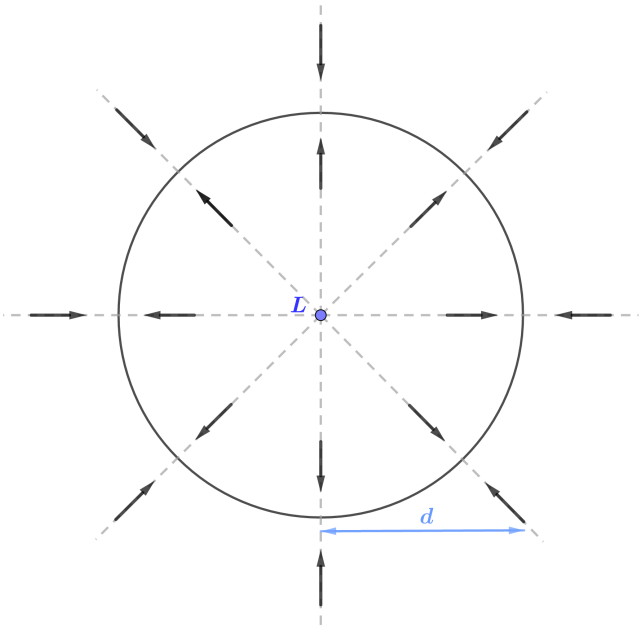
En el caso de la Figura 3.5(a) se ilustra la zona de influencia del estado Esquivar. Como se explicó anteriormente, si el seguidor cumple simultáneamente las condiciones relacionadas a θ_{riesgo}, R_{riesgo} y $V_L = v_{riesgo}$, o si se encuentra dentro de la circunferencia de radio R_{int} centrada en el líder (L), seguirá la trayectoria indicada por las flechas para evitar la posible colisión.

En la Figura 3.5(b) se muestra con flechas la acción que tomará el dron seguidor en el estado Detenerse, para alcanzar la posición objetivo en la circunferencia de radio d centrada en el líder (L). Si se encuentra demasiado cerca, el seguidor buscará alejarse radialmente del líder hasta alcanzar una distancia d, en caso contrario, si se encuentra por fuera del círculo buscará acercarse radialmente hasta alcanzar dicha distancia.

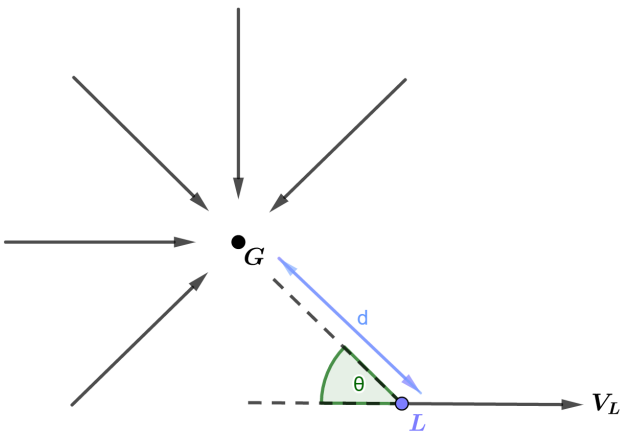
Por último, en la Figura 3.5(c) se muestra la acción que toma el seguidor cuando no está en una situación de riesgo y el líder se esta moviendo a una velocidad (V_L) por encima del umbral. En este caso, el seguidor se aproxima al líder hasta alcanzar una posición objetivo (X_G) como la que se describió en la Figura 3.3.



(a) Zona de influencia del estado Esquivar.



(b) Lugar geométrico de los puntos objetivo (X_G) en Detenerse.



(c) Posición objetivo (X_G) en el estado Seguir.

Figura 3.5: Esquema con las condiciones para pertenecer a un estado y la acción a tomar por un seguidor en cada zona indicada con flechas respecto al líder (L).

Si bien esta estrategia funciona correctamente para controlar la dinámica entre un dron líder y un seguidor, como se mencionó al principio la misma no contempla las posibles colisiones entre los drones seguidores. Por ello se procedió a buscar otros métodos para poder llevar a cabo la dinámica deseada. La búsqueda se orientó en particular a estrategias relacionadas con campos potenciales, llegando a la alternativa que se presenta a continuación.

3.3.2. Alternativa II: Campos Potenciales

Preliminares

La segunda estrategia está basada en la idea presentada en el artículo [9], la cual se apoya sobre potenciales vectoriales. En la misma, se tiene que para cada dron los puntos objetivo (punto que se quiere alcanzar idealmente para mantener una cierta formación) generan campos atractores, y los objetos a esquivar (los otros drones) generan campos repulsores, la trayectoria a seguir se determina siguiendo el flujo dictado por la suma de estos campos. Es decir, que la suma de los campos determina la dirección instantánea neta de la trayectoria. Los campos que repelen tienen mayor magnitud a medida que disminuye la distancia con el obstáculo, de forma que si se está muy cerca de un choque, se le de más prioridad a esquivar al obstáculo que a dirigirse a la posición objetivo.

El artículo plantea, que si se tiene $X_G \in \mathbb{R}^2$ como un punto objetivo que se desea alcanzar, y $X \in \mathbb{R}^2$ es la posición actual del dron seguidor, entonces los **campos atractores** se consideran con la siguiente expresión:

$$AttractionField = K_a.(X_G - X) \tag{3.1}$$

donde K_a es un valor constante que funciona como factor de escala que magnifica o atenúa al campo atractor.

De la misma forma, para los **campos repulsores**, sea $X_O \in \mathbb{R}^2$ la posición de un obstáculo, el artículo considera la siguiente expresión para el campo repulsor:

$$RepField = \begin{cases} K_r \frac{\left(\frac{1}{|X-X_O|} - \frac{1}{d_{rep}}\right)}{|X-X_O|^3}.rot_{\pm\frac{\pi}{2}}(X - X_O) & si \quad |X - X_O| < d_{rep} \\ 0 \qquad si \quad |X - X_O| \geq d_{rep} \end{cases} \tag{3.2}$$

donde la función $rot_{\pm\frac{\pi}{2}} : \mathbb{R}^2 \to \mathbb{R}^2$ es una rotación de ángulo $\frac{\pi}{2}$ centrada en el origen, d_{rep} es un valor a elegir que representa la distancia a la que se debe estar del obstáculo para que empiece a existir un campo repulsor y K_r es un factor de escala constante que magnifica o atenúa dicho campo.

Si se tomaran los campos repulsores puramente radiales (es decir, el campo es saliente desde el obstáculo de manera radial) existirían problemas en caso que el obstáculo se interponga entre el dron y el punto objetivo de manera que estén todos alineados (impidiendo bordear el obstáculo para alcanzar el objetivo). Por este motivo, el artículo sugiere rotar el campo de manera de obtener un campo

puramente tangencial, permitiendo al dron hacer trayectos más cortos para llegar al objetivo a costa de tomar mayores riesgos (pues la nueva trayectoria pasa más cerca de los obstáculos).

En el artículo, el sentido de la rotación del campo se decide teniendo en cuenta la posición y las direcciones de las velocidades de ambos actores (el dron y el obstáculo). Se asume que ambos cooperan para esquivarse, en el sentido que el dron también produce un campo repulsor que repercute en el movimiento del obstáculo, es decir que en el análisis anterior se pueden intercambiar los roles de a quien se llamó "dron" y a quien se llamó "obstáculo".

Modificaciones

Se implementó la técnica descrita en el artículo y se realizaron simulaciones, con la diferencia de que en la implementación realizada el líder no coopera para evadir a los seguidores, de modo que él pueda desplazarse libremente y solo los seguidores se encarguen de evitar las colisiones. Al analizar dichas simulaciones, se observaron algunos problemas para los que se plantearon diferentes soluciones que se enumeran a continuación.

1. En primer lugar los drones seguidores generaban muchas situaciones de riesgo entre ellos al moverse a altas velocidades, debido a que los campos repulsores de cada seguidor no tenían componentes radiales que alejen a un dron del otro. Esto llevaba a que al moverse rápidamente la propia inercia de las aeronaves hiciera que no fuera suficiente poner campos tangenciales para lograr la evasión. Esto pudo solucionarse dando a los campos repulsores una componente radial y otra tangencial, a partir de definir un ángulo $\theta_{rep} \in [0, \pi/2]$ que determine el ángulo que forma el nuevo campo repulsor con respecto a la dirección radial, como se observa en la Figura 3.6.

2. Existieron también problemas al no tener en cuenta la magnitud de la velocidad relativa que cada dron tiene respecto de sus obstáculos. Esto llevaba a que los drones tuvieran un mismo campo repulsor sin importar si el obstáculo se movía rápidamente (en términos de la velocidad relativa) o si estaba quieto, lo que ocasionaba que se tuvieran campos demasiado grandes en situaciones de poco riesgo o muy pequeños en situaciones de más riesgo. Esto fue solucionado cambiando el factor K_r de manera dinámica, de modo que el mismo contenga un factor multiplicativo de la forma

$$\left(\frac{|V - V_O|}{|X - X_O|}\right)^p$$

donde $V - V_O$ y $X - X_O$ son la velocidad y posición relativa de un dron respecto de su obstáculo, y la constante $p > 0$ a elegir determina la velocidad con la que crece K_r con el nuevo factor agregado.

3. En el mismo sentido, se observó que el artículo tampoco contempla la dirección de la velocidad relativa del obstáculo, provocando que los drones

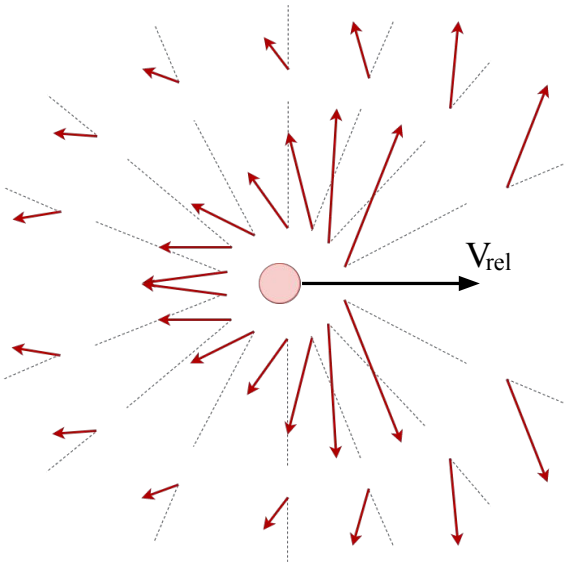


Figura 3.6: Esquema del campo repulsor (con las modificaciones) generado por un obstáculo que se mueve con una velocidad relativa V_{rel} respecto al dron.

seguidores generaran campos repulsores muy grandes en situaciones sin riesgo, en las que no era necesario su efecto. Por ejemplo, con todos los drones en formación moviéndose paralelamente, se podían observar oscilaciones en la posición de los seguidores. Esto fue corregido de modo que los campos repulsores tengan también en cuenta el ángulo que forma la posición relativa con la velocidad relativa que se tiene respecto del obstáculo. Para esto, se agregó un término que toma gran valor cuando la posición relativa del obstáculo va en el sentido contrario a su velocidad relativa y que decrece cuando ambos vectores (posición y velocidad relativa) sean paralelos. La expresión de este término resulta

$$\left(\frac{(X - X_O)^T(V - V_O)}{|X - X_O|.|V_O - V|}\right)^p$$

cuando $(X - X_O)^T(V - V_O) \geq 0$, y 0 en caso contrario.

Llegado este punto, se obtuvo un campo repulsor como el de la Figura 3.6, resultando en un mejor desempeño en la evasión de colisiones entre los seguidores. Sin embargo, debido a que el líder no cooperaba para esquivar a los demás integrantes, aún se observaban colisiones entre los seguidores y el líder. Para solucionar esto, se decidió cambiar la estrategia con la que el líder es esquivado, haciendo uso de la estrategia de "Tres Estados" que se describió anteriormente. Ésta permite seguir y esquivar al líder, y le define a cada dron un punto objetivo que considera las posibles colisiones con el líder. A grandes rasgos, la estrategia de Tres Estados definirá un punto objetivo sobre el cual centrar los campos atractores dados por la estrategia de Campos Potenciales, y esto será presentado con mayor detalle en la Sección 3.3.3.

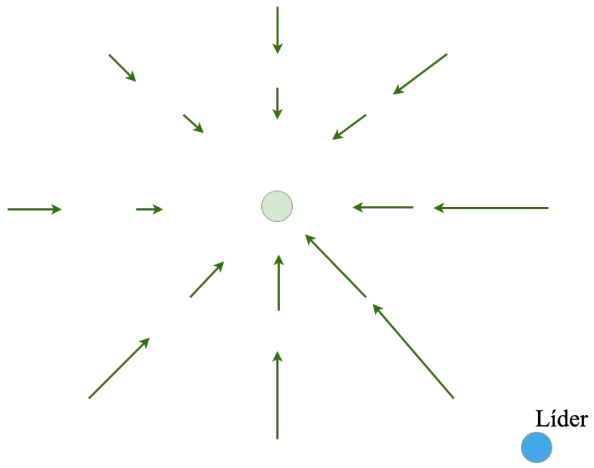


Figura 3.7: Esquema del campo atractor centrado en el punto objetivo con las modificaciones realizadas.

De la misma forma, se enumeran a continuación las modificaciones realizadas sobre los campos atractores.

1. En primer lugar, se observó un atraso entre la posición objetivo de cada seguidor y su posición reportada, impidiendo mantener la formación deseada entre los drones. Este retraso se veía magnificado cuando se aumentaba la velocidad del líder, pues esto genera que el punto objetivo cambie más rápidamente. El problema pudo solucionarse parcialmente, agregando un factor multiplicativo a K_a que aumentara con la velocidad del líder, de la forma
$$Arctg(|V_L|)$$
siendo V_L la velocidad del líder. Este factor también ayuda a esquivar mejor al líder cuando se mueve rápidamente, debido a que esa evasión se manifiesta en el campo atractor luego del cambio explicado en punto anterior.

2. Por otra parte, dado que el campo atractor será quien se encargue de evitar las colisiones con el líder (como se explicó en el punto 4), se hizo que el factor de escala de dicho campo aumentara al disminuir la distancia con el líder como se aprecia en la Figura 3.7. De este modo, se le da más importancia al campo atractor que permite esquivarlo, que al campo repulsor generado por el otro seguidor en caso de que exista riesgo de colisión.

Cabe aclarar, que todas las modificaciones se hicieron de modo que los factores de escala de los campos repulsores y atractores siempre tuvieran un valor mínimo positivo. De esta forma nunca se deja de tener en cuenta a ninguno de los campos para determinar la trayectoria.

Además se intentó que los factores agregados varíen de forma suave, es decir, que no existan cambios abruptos en los campos. Esto se debe a que las discontinuidades en los campos, producen que los drones cambien de dirección muy repentinamente, provocando movimientos muy bruscos que pueden generar situaciones

de riesgo o impedir que se mantenga la formación deseada durante el vuelo. Esto puede apreciarse en las expresiones de los campos que se detallan a continuación.

Campos Implementados

Luego de aplicar las modificaciones mencionadas sobre los campos originales del artículo, se obtuvieron las expresiones que se detallan a continuación.

Campo Atractor:

$$\boxed{AttractionField = K_a.(X_G - X)} \tag{3.3}$$

siendo X_G el punto deseado en la formación para este seguidor (ver figura 3.3), y K_a el factor de escala del campo atractor, elegido a partir de la siguiente expresión:

$$K_a = \begin{cases} (K_{a_{max}} - K_{a_{min}}).\frac{\frac{1}{|X-X_L|} - \frac{1}{d_{K_{a_{min}}}}}{\frac{1}{d} - \frac{1}{d_{K_{a_{min}}}}}.\frac{2Arctg(|V_L|)}{\pi} + K_{a_{min}} & si \quad |X - X_L| < d_{K_{a_{min}}} \\ K_{a_{min}} \qquad si \quad |X - X_L| \geq d_{K_{a_{min}}} \end{cases} \tag{3.4}$$

donde:

- $K_{a_{min}}$ es el menor valor que podrá tomar K_a, cuando el seguidor y el líder se encuentren lejos (a una distancia mayor que $d_{K_{a_{min}}}$).
- $K_{a_{max}}$ es el valor que tomará K_a si el seguidor se encuentra a distancia horizontal d del líder, mientras el líder se mueve a una velocidad V_L en el plano horizontal arbitrariamente grande.
- $d_{K_{a_{min}}} > d$ es una distancia de seguridad a elegir.

A modo de observación, el factor $\frac{2Arctg(|V_L|)}{\pi}$ sirve para aumentar el valor del campo atractor cuando el líder se mueve rápidamente (permitiendo reducir el desfasaje entre la posición del seguidor y su punto objetivo), y se agregó la constante multiplicativa $\frac{2}{\pi}$ para que este factor varíe entre 0 y 1.

Por otro lado, el factor $\frac{\frac{1}{|X-X_L|} - \frac{1}{d_{K_{a_{min}}}}}{\frac{1}{d} - \frac{1}{d_{K_{a_{min}}}}}$ hace que la magnitud del campo atractor aumente cuando el dron seguidor se encuentra muy cerca del líder, de modo de darle más peso a este campo en situaciones de peligro de colisión con el mismo.

El denominador de este último factor ($\frac{1}{d} - \frac{1}{d_{K_{a_{min}}}}$), se eligió de modo que este factor valga 1 cuando la distancia entre el líder y el seguidor corresponde con la distancia deseada d.

Campo Repulsor:

$$RepField = \begin{cases} K_r \dfrac{\left(\frac{1}{|X-X_O|} - \frac{1}{d_{rep}}\right)}{|X-X_O|^3}.rot_{\theta^s_{rep}}(X - X_O) & si \quad |X - X_O| < d_{rep} \\ 0 \qquad si \quad |X - X_O| \geq d_{rep} \end{cases} \tag{3.5}$$

con X_O la posición del obstáculo (habrán dos campos repulsores, uno para el líder y otro para el seguidor restante), $\theta^s_{rep} \in \mathbb{R}$ tal que $|\theta rep^s| = \theta_{rep}$ (recordando que θ_{rep} es un valor predefinido para el ángulo que se desea que exista entre el campo repulsor y la dirección radial saliente del obstáculo) y la obtención de su signo se describirá más adelante, y $rot_{\theta^s_{rep}} = \begin{pmatrix} cos(\theta^s_{rep}) & -sen(\theta^s_{rep}) \\ sen(\theta^s_{rep}) & cos(\theta^s_{rep}) \end{pmatrix}$ es la matriz de giro de ángulo θ^s_{rep}.

El factor de escala K_r se elige como:

$$K_r = \begin{cases} (K_{r_{max}} - K_{r_{min}}).\left(\frac{(X-X_O)^T(V-V_O)}{|X-X_O|^2}\right)^p + K_{r_{min}} & si \quad (X - X_O)^T(V - V_O) \geq 0 \\ K_{r_{min}} \qquad si \quad (X - X_O)^T(V - V_O) \leq 0 \end{cases} \tag{3.6}$$

siendo:

- $K_{r_{min}}$ es el mínimo valor que puede tomar el factor de escala del campo repulsor.
- p es un número positivo a elegir.
- $K_{r_{max}} > K_{r_{min}}$ es otra constante a elegir.

Una observación de esta elección de K_r, es que cuando $(X-X_O)^T(V-V_O) \geq 0$, entonces la expresión anterior puede escribirse como:

$$K_r = (K_{r_{max}} - K_{r_{min}}).\left(\frac{(X - X_O)^T(V_O - V)}{|X - X_O|.|V - V_O|}\right)^p.\left(\frac{|V - V_O|}{|X - X_O|}\right)^p + K_{r_{min}}$$

Aquí, el término $\frac{(X-X_O)^T(V-V_O)}{|X-X_O|.|V_O-V|}$ en K_r corresponde con el coseno del ángulo que forman la posición relativa del seguidor respecto del obstáculo con la velocidad relativa del obstáculo respecto del seguidor. El factor $\left(\frac{|V-V_O|}{|X-X_O|}\right)^p$ sirve para magnificar el campo repulsor cuando el obstáculo se encuentra más cerca y moviéndose rápido (en términos de la velocidad relativa).

En particular, $\frac{|V-V_O|}{|X-X_O|}$ es el inverso del tiempo que le llevaría al obstáculo recorrer la distancia $|X - X_O|$ en el marco relativo al seguidor, si se mantuvieran las velocidades de ambos constantes. La potencia p a elegir existe con el objetivo

de "suavizar" las acciones de control, es decir que no cambie bruscamente durante el vuelo.

En caso de que $\theta_{rep} = 0$, el campo repulsor siempre resultará radial saliendo desde el obstáculo. En los demás casos, el signo del ángulo de repulsión representa el sentido de giro en torno al obstáculo, y se obtiene a partir de la expresión:

$$\theta^s_{rep} = \theta_{rep} \cdot \text{signo}[(X - X_O)^T rot_{\frac{\pi}{2}}(V_O - V)] \tag{3.7}$$

La ecuación anterior permite que el sentido de giro de los drones reduzca el riesgo de colisión. Es decir, el sentido de giro se elige de modo que el seguidor se aleje de la trayectoria dada por la recta que pasa por el obstáculo con dirección igual a la velocidad relativa del obstáculo respecto del seguidor.

En las Figuras 3.8 y 3.9 se bosquejan casos donde, a misma distancia, corresponde girar el campo radial en sentido antihorario o en sentido horario respectivamente.

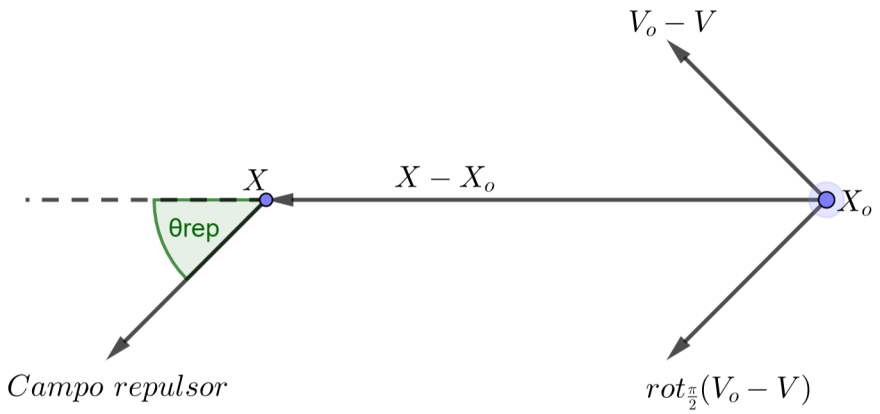


Figura 3.8: Giro antihorario de un campo repulsor. Observar que la proyección del vector $rot_{\frac{\pi}{2}}(V_O - V)$ sobre la recta definida por las posiciones de ambos drones, va en la misma dirección que $X - X_O$, por lo que el giro es antihorario

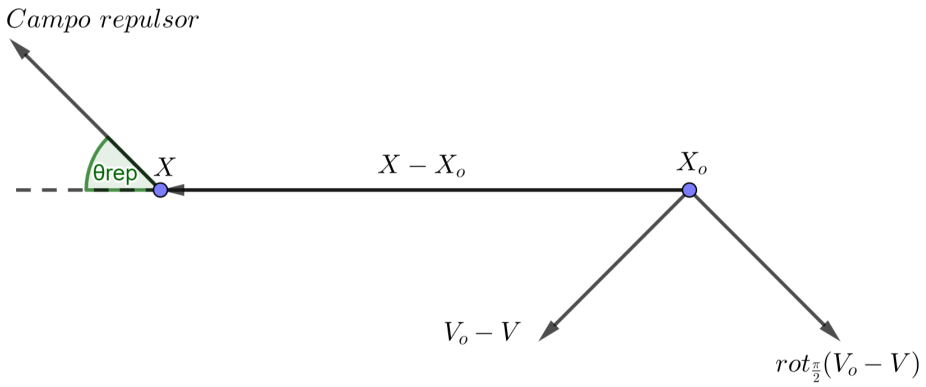


Figura 3.9: Giro horario de un campo repulsor. Observar que en este caso la proyección del vector $rot_{\frac{\pi}{2}}(V_O - V)$ sobre la recta definida por las posiciones de ambos drones, va en dirección contraria a la de $X - X_O$, por lo que el giro es horario.

3.3.3. Estrategia Mixta

Hasta aquí, se habían observado buenos resultados en simulaciones con la estrategia de Tres Estados para que un seguidor evite colisiones con el líder e intente seguirlo. Por lo tanto, se decidió unirla con la estrategia de Campos Potenciales, que arrojó buenos resultados en la evasión de colisiones entre seguidores, intentando mantener la formación deseada.

Para sacar el mejor provecho de cada estrategia, se utilizó la posición objetivo X_G determinada por la estrategia de Tres Estados como centro para generar un campo potencial atractor hacia ella como el presentado en la ecuación (3.3). De esta manera, la posición atractora contempla a su vez las posibles colisiones con el dron líder sin que el mismo deba generar un campo que lo repela.

A modo de ejemplo, si el líder se encuentra estático en un punto, la estrategia de Tres Estados indica el estado **Detenerse**, entregando una posición objetivo sobre la cual se coloca un campo atractor que intente mantener la formación preestablecida. Sin embargo en caso de existir riesgo de colisión, Tres Estados se encuentra en el estado **Esquivar**, resultando en una posición objetivo que intenta evitar el choque, por lo que la estrategia de Campos Potenciales coloca un campo que atraiga al seguidor hacia dicha posición objetivo.

Mientras tanto, se utiliza un campo potencial repelente como el de la ecuación (3.5), centrado en la posición actual de cada seguidor. Esto permite que los drones seguidores cooperen entre ellos para evitar las posibles colisiones que afecten a ambos, dado que las dos aeronaves se evadirán mutuamente, debiendo desviarse lo menos posible de la trayectoria de seguimiento.

Sumando entonces el campo repelente centrado en el otro seguidor, y el campo atractor centrado en el *setpoint* que se generó a partir de la estrategia de Tres Estados, se consigue un campo neto que determina la dirección de la trayectoria que se debe seguir para intentar mantener la formación preestablecida y evitar accidentes con cualquiera de las demás aeronaves del enjambre al mismo tiempo.

Los resultados obtenidos durante las primeras simulaciones con la estrategia Mixta resultaron alentadores, por lo que se optó por seguir adelante con su utilización. La programación de estas acciones de control en un software que pudiera ser interpretado por las aeronaves, así como los detalles de las simulaciones realizadas serán estudiados a lo largo de los capítulos siguientes, siendo necesario introducir antes algunos conceptos importantes para su comprensión.

Las constantes que fueron utilizadas finalmente, tanto para las simulaciones como para las pruebas de campo, pueden encontrarse en el Apéndice D.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 4

Software

En este capítulo se introducirán los conceptos relacionados al software del proyecto, tanto las herramientas utilizadas como las desarrolladas a lo largo del mismo. Se incluye también una breve introducción de cómo se debe trabajar al momento de utilizar el entorno robótico ROS, y se detallará la organización de los archivos creados y su ejecución.

4.1. Herramientas de Software

4.1.1. Robot Operating System (ROS)

El entorno robótico ROS fue utilizado para implementar los comportamientos de cada integrante del enjambre, así como la comunicación y el flujo de información entre los mismos.

ROS es un sistema específicamente creado para el desarrollo de software en robots. Es un software libre, desarrollado originalmente en el Laboratorio de Inteligencia Artificial de Stanford University, y posteriormente mantenido por la empresa Willow Garage y la organización Open Robotics [34].

Su funcionamiento puede describirse como una estructura de grafos, donde los nodos son rutinas o programas e interactúan entre sí mediante tópicos públicos o llamadas de servicios bilaterales. Todo el registro y comunicación entre nodos es monitorizado por un ROS *master*. Este tipo de arquitectura modular de software se acompasa con la arquitectura de hardware de los robots, que generalmente consiste en una red de sensores y controladores que se comunican entre sí.

ROS permite comunicarse con los sensores y actuadores del robot, posibilitando un fácil acceso a la información ya procesada de estos dispositivos, generalmente en forma de tópicos. Existe una gran variedad de paquetes (*ROSPackages*) creados por usuarios con fines específicos para diferentes aplicaciones y dispositivos estándar del mundo de la robótica, como drivers de motores, servomotores, sensores, cámaras y más.

Además cuenta con una variedad de herramientas de desarrollo como compiladores, simuladores y visualizadores que permiten automatizar tareas y analizar los

datos entregados por el robot en tiempo real. A continuación se definen algunos de los conceptos más importantes del entorno ROS [35]:

- **Node**: Un nodo es una rutina o proceso ejecutable que realiza cierta tarea, utilizando como entrada la información de los tópicos que se encuentran disponibles en la red en tiempo real, y obteniendo como salida cierta información que puede o no poner a disposición del resto del grafo. Puede ser programado en *Python* o *C++*, para lo que existen *Client Libraries* específicas llamadas *rospy* y *roscpp*.

- **Topic**: Un tópico es un nombre único que se le otorga a un mensaje de un tipo en particular, al que los nodos pueden suscribirse o publicar. El tipo de mensaje debe tener un formato previamente definido, conteniendo una cierta cantidad de variables estructuradas. Pueden existir varios mensajes del mismo tipo con fines diferentes, por lo que se utiliza el tópico con su nombre asociado para diferenciarlos. Un mismo nodo puede publicar y/o suscribirse a varios tópicos, así como un mismo tópico puede ser publicado o tener suscritos a varios nodos.

- **Service**: Dado que ROS es un sistema distribuido, con muchos nodos conviviendo al mismo tiempo, pueden existir interacciones de tipo cliente-servidor entre dos nodos particulares, es aquí donde se utilizan los servicios. Éstos se definen como un par de mensajes estructurados, uno para la solicitud y otro para la respuesta, la dinámica consiste en que un primer nodo ofrezca un servicio al que le otorga un nombre, enviando la solicitud y aguardando por la respuesta de parte de un segundo nodo.

- **Master**: El ROS *master* es el primer proceso a ejecutar mediante el comando *roscore*. Es el encargado de ordenar y sincronizar el flujo de información en el grafo computacional, así como de registrar los nombres asociados a los nodos, tópicos y servicios. Es el núcleo del entorno ROS.

- **Launch**: Un archivo *launch* permite lanzar (ejecutar) de manera sincronizada un conjunto de nodos. El archivo debe contener la línea de ejecución de cada nodo, con los parámetros necesarios para cada uno, y en el orden deseado. Permite generar sub-grupos de nodos con un nombre en particular que se asigna también a los tópicos y servicios que interactúan dentro de cada sub-grupo.

- **Package**: Los paquetes son la unidad principal de software en ROS. Un paquete es una carpeta que contiene nodos, bibliotecas, definiciones de mensajes y servicios, *launchs*, archivos de configuración y archivos de compilación. Todo se encuentra organizado de tal forma que basta con descargar y compilar un paquete para poder hacer uso de sus herramientas.

- **Bag**: Es un formato que permite guardar información durante la ejecución de un ROS *master*. Puede guardar uno, varios o todos los tópicos disponibles en el grafo, con una estampa temporal asociada. De esta forma, se puede

graficar, analizar e incluso reproducir el flujo de información durante un lapso de tiempo de interés.

En el marco de este proyecto, ROS permitió implementar las acciones de control de alto nivel de cada dron, así como la red de comunicación entre ellos y con la computadora en tierra.

4.1.2. Gazebo

En cualquier proyecto relacionado a la robótica, la simulación es imprescindible para detectar problemas previo al pasaje a los robots reales, y facilitar el proceso de desarrollo sin los riesgos y el tiempo que involucra realizar pruebas de campo.

Gazebo es un simulador de robótica en 3D, que puede utilizar varios *Motores Físicos* de alto rendimiento para simular ambientes interiores o exteriores que incluyen iluminación, condiciones atmosféricas y climáticas, texturas y colisiones entre objetos. Se le pueden importar modelos de robots, sensores y actuadores comerciales, y también pueden construirse sistemas robóticos propios.

Es software *open-source*, y fue desarrollado originalmente por Willow Garage. Al día de hoy cuenta con más de diez versiones y es mantenido por la organización Open Robotics al igual que el entorno ROS, lo que los hace altamente compatibles.

4.1.3. SITL

Un simulador SITL (software in the loop) permite emular el firmware de una *FCU* en una computadora sin necesidad de ningún hardware adicional.

Es una compilación del código de la controladora de vuelo que utiliza un compilador C++ ordinario, lo que brinda un ejecutable que permite probar el comportamiento del código desarrollado, interactuando con el firmware emulado y con el modelo simulado en Gazebo, sin el hardware de la controladora real [30].

Como se mencionó en la sección 2.1.1, los drones utilizados en el proyecto cuentan con una controladora de vuelo Pixhawk, que tiene instalado un firmware PX4. De modo que para realizar las simulaciones, se acudió al SITL de dicho firmware.

El diagrama de la Figura 4.1 muestra un entorno de simulación SITL PX4 típico para cualquiera de los softwares de simulación compatibles. Las diferentes partes del sistema se conectan a través de puertos UDP y se pueden ejecutar en la misma computadora o en otra computadora en la misma red.

Si se aplica el sistema de compilación *SITL make*, tanto SITL como el simulador se iniciarán en la misma computadora y los puertos se configurarán automáticamente como lo indica la Figura 4.1. El sistema de compilación con *make* permite iniciar el simulador del firmware PX4, lanzar un simulador gráfico, agregarle un modelo de vehículo particular y conectarlos. En el caso particular de este proyecto el simulador utilizado es Gazebo, y el modelo del vehículo para la simulación es *Iris*, que emula un cuadricóptero comercial de dimensiones similares a los construidos. En definitiva, para simular en la computadora una *FCU* idéntica

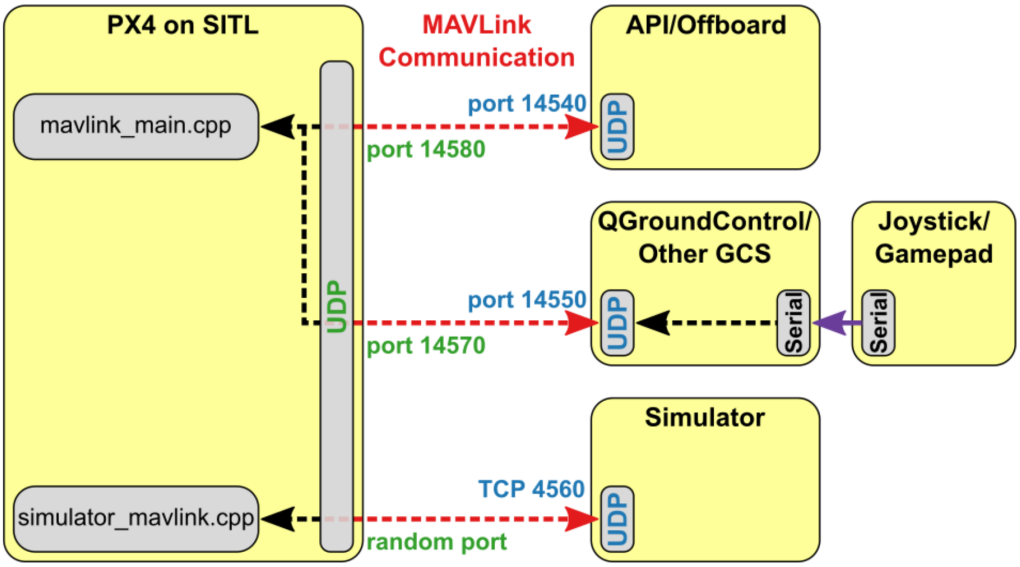


Figura 4.1: Diagrama del entorno de simulación SITL de PX4 [30], donde se observan los diferentes módulos que interactúan durante la simulación de una controladora de vuelo, funcionando abordo de un dron en un mundo virtual.

a la utilizada en la práctica (con los parámetros del dron *Iris*) se debe ejecutar el siguiente comando:

```
$ make px4_sitl_default gazebo_iris
```

4.1.4. MAVROS

MAVROS es un paquete de ROS que permite la comunicación MAVLink entre computadoras que ejecutan ROS, controladoras de vuelo y estaciones de control terrestre que sean compatibles con el protocolo MAVLink. El paquete contiene las bibliotecas que definen los mensajes, así como un conjunto de *launchs* para establecer la conexión con diferentes controladoras y ejecutar comandos particulares, y un set de nodos ejemplificando la simulación en Gazebo [36].

Luego de que se ejecuta un *roscore*, se debe realizar la conexión con la controladora de vuelo (ya sea una controladora real o una simulada en Gazebo) utilizando el archivo *launch* adecuado. Al establecer la conexión, se ejecuta el nodo de nombre *mavros*, y se ponen a disposición una larga lista de tópicos, algunos sobre los cuales el sistema comienza a publicar el estado actual reportado por la controladora, y otros a los que se suscribe, esperando que un agente externo (otro nodo) publique en ellos algún comando. Las definiciones de los tópicos disponibles, así como los tipos de mensajes y servicios, se encuentran en la web oficial del paquete [36].

Por ejemplo, al iniciar MAVROS se habilita el tópico:

`mavros/global_position/global`

donde el nodo *mavros* publica información acerca de la posición global adquirida por el GPS, incluyendo la calidad de la señal y el vector $\{latitud, longitud, altitud\}$. Otro nodo dentro del mismo *master* puede suscribirse a ese tópico y hacer uso de los datos publicados. De la misma manera, se pone a disposición un tópico llamado:

`mavros/setpoint_position/global`

que se encuentra vacío inicialmente, donde otro nodo del ROS *master* puede publicar una posición deseada en el marco global LLA. Si eso sucede, el nodo *mavros* utiliza esa información para traducirla en comandos MAVLink y enviarlos a la *FCU*, que actuará en consecuencia. Una descripción detallada del protocolo MAVLink se encuentra en el Capítulo 6.

En resumen, para que un nodo interactúe con la aeronave, basta con importar las bibliotecas de mensajes MAVROS que se deseen escribir o leer, y utilizar las funciones de publicación y suscripción a tópicos que provee el paquete. La sintaxis que se utiliza varía según el lenguaje elegido (Los nodos de ROS pueden ser programados en *python* o C++) pero el funcionamiento es el mismo ya que el paquete MAVROS soporta ambos.

4.1.5. QGroundControl

QGroundControl (QGC) es el software que se utilizó como estación de monitoreo terrestre. Su principal utilidad es la de informar y visualizar en todo momento el estado de un dron conectado a ella, así como leer y modificar los parámetros guardados en el firmware. También permite crear y cargar misiones o enviar comandos en tiempo real a la aeronave (como despegue, vuelta a casa y aterrizaje), aunque estas últimas funciones no son utilizadas en el proyecto [32].

El programa se comunica con el *UAV* mediante el protocolo MAVLink, y fue desarrollado por el mismo equipo que creó dicho protocolo. La conexión puede establecerse mediante un puerto USB, *Telemetría* por radiofrecuencia, WiFi o un puerto UDP o TCP en la computadora que ejecuta QGC.

La herramienta cuenta con una interfaz gráfica, con una pantalla de vuelo como la que se aprecia en la Figura 4.2, que permite ubicar las aeronaves en un mapa en tiempo real, conocer su orientación, el estado de sus sensores, batería restante y leer las alertas de funcionamiento. También posee una pantalla de configuración, donde se fijan los valores de los parámetros de la controladora y pueden calibrarse los sensores, una pantalla de misiones en la que se crean, editan y cargan misiones para que la aeronave realice de forma autónoma, y una pantalla de análisis donde pueden obtenerse los datos recabados luego de una sesión de vuelo.

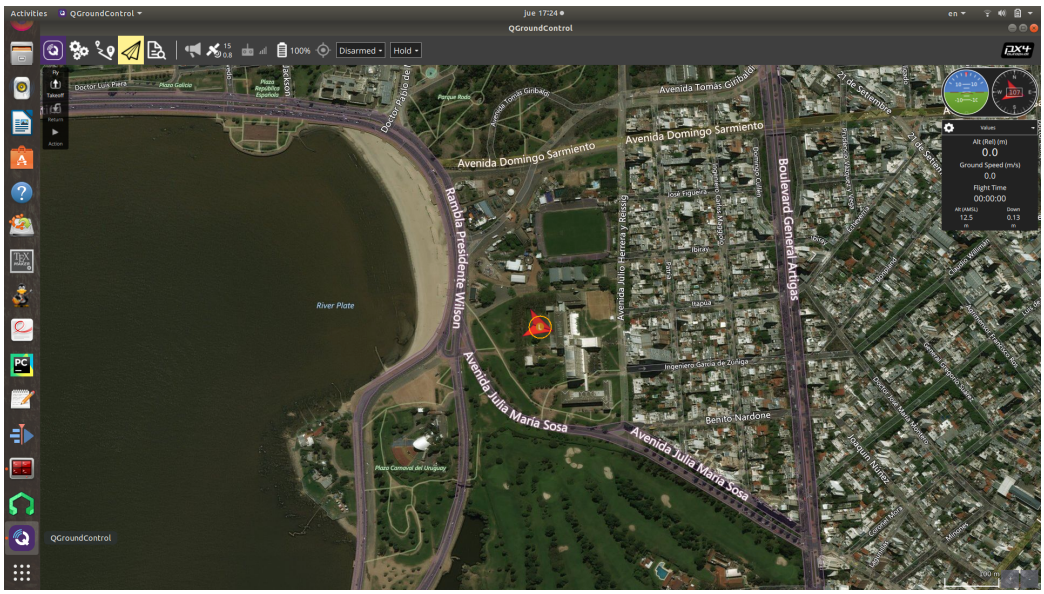


Figura 4.2: Pantalla de vuelo del software *QGroundControl*, desde la que se monitoriza el estado de un dron en tiempo real, volando al costado de la Facultad de Ingeniería.

4.2. Instalación y Tutoriales Realizados

En primer lugar fue necesario instalar el sistema operativo sobre el cual basar el software del proyecto. Si bien actualmente existe una nueva versión de ROS, a la fecha de comienzo de este proyecto la última versión existente (llamada *ROS Melodic*) recomendaba su instalación sobre el sistema operativo *Ubuntu 18.04*, por lo que se procedió a instalar dicha distribución de Linux.

Una vez instalado el sistema operativo, para instalar y aprender lo necesario sobre el funcionamiento de ROS, se comenzó leyendo la documentación [34] y realizando el tutorial de instalación oficial, de donde se obtiene un resumen general de la estructura conceptual del entorno, y se informa acerca de cómo utilizar algunas de las funcionalidades anteriormente detalladas [39]. Al finalizar dicho tutorial e implementar los pasos allí descritos, se cuenta con el sistema ROS instalado, el árbol de directorios creado con los permisos necesarios y un conjunto de paquetes y herramientas básicas que se instalan automáticamente junto con el sistema.

Por último, se procedió a descargar e instalar el paquete MAVROS, y a descargar los archivos necesarios para la simulación SITL de PX4, siguiendo las instrucciones presentes en la documentación del firmware de vuelo utilizado durante el proyecto [27]. Finalmente, se recrearon los ejemplos de simulación provistos por dicha documentación, hasta lograr entablar una conexión MAVLink y ejecutar un despegue y aterrizaje autónomos en el simulador Gazebo con un modelo de cuadricóptero de características similares a los construidos en el proyecto y utilizados posteriormente en las pruebas de campo.

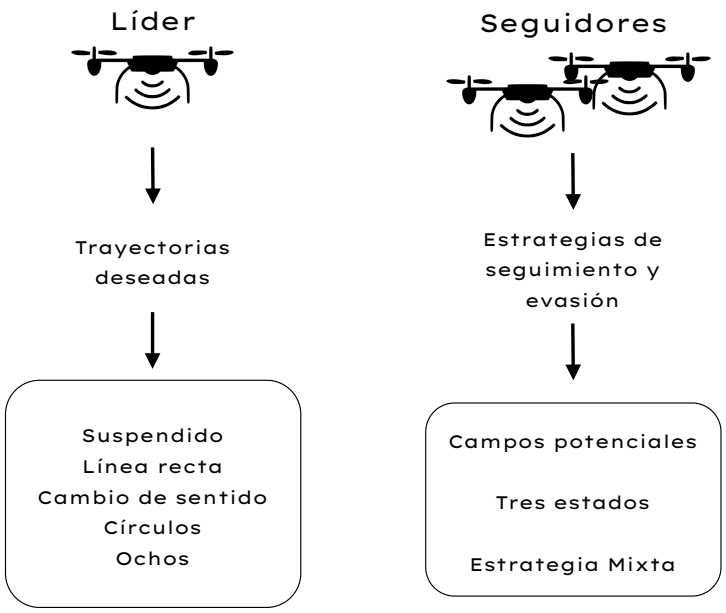


Figura 4.3: Módulos de software por separado, con las variantes programadas para cada uno.

4.3. Layout del Proyecto

4.3.1. Módulos de Software

Las tareas desarrolladas en el proyecto se pueden dividir en dos módulos independientes que intercambian información. Por un lado las tareas relacionadas con la navegación del *UAV* líder, que debe seguir una cierta trayectoria deseada, y por otro lado la implementación de las estrategias de seguimiento y evasión diseñadas para los *UAV* seguidores. En este sentido, se decidió programar nodos de ROS separados para cada módulo, uno para la trayectoria del líder y otro para el seguimiento y evasión, lo que permite una fácil adaptación del sistema a los cambios en la cantidad de aeronaves desplegadas, las variantes realizadas en cada módulo o los eventuales errores que pudieran ocurrir en alguno de ellos. Esta modularización se aprecia en el esquema de la Figura 4.3.

Para el módulo de trayectorias deseadas, se programaron una serie de nodos que luego del despegue, determinan un conjunto de posiciones objetivo (*waypoints*) al dron líder, con el fin de que éste describa una cierta ruta, como por ejemplo una línea recta, círculos, idas y vueltas sobre la misma recta, ochos, o simplemente que permanezca suspendido. Además este nodo publica permanentemente en un tópico la posición del dron líder, junto con su actitud y su velocidad, poniéndolas a disposición de cualquier otro nodo dentro del grafo que quiera escucharlas.

La información publicada por el módulo del trayectorias deseadas, es tomada como entrada al módulo de seguimiento y evasión, en el que se implementan las estrategias de control detalladas en el Capítulo 3. Para esto también se necesita estar al tanto de la posición, actitud y velocidad de los demás seguidores presentes en la dinámica, por lo que cada seguidor ejecuta un nodo de seguimiento y evasión por

separado, que se suscribe a dicha información de los demás componentes y publica la suya en la red. Una descripción en profundidad de los nodos implementados y los tópicos intercambiados por ellos se presenta en el Capítulo 5.

Con todos los nodos disponibles, se pueden ejecutar diferentes dinámicas de seguimiento al líder, realizando combinaciones entre alguna de las trayectorias deseadas para el líder y una estrategia de control para los seguidores.

4.3.2. Organización de Archivos

La forma recomendada de organizar un conjunto de nodos y tópicos con un fin común dentro del entorno robótico es la creación de un *ROS Package*, que permite la portabilidad a la hora de querer utilizar el software en diferentes computadoras. En este proyecto se creó un paquete de ROS llamado **ftl** (acrónimo de *Follow the leader*), que contiene todos los nodos programados en *C++*, así como los archivos *launch*, una biblioteca creada con funciones útiles y los archivos de compilación.

La descripción de la biblioteca creada, así como la definición de todas las funciones que contiene se encuentra en el Apéndice E. Los archivos de compilación declaran todos los nodos ejecutables, y su dependencia con la biblioteca creada, además declaran la dependencia del paquete completo con otros *packages*, como por ejemplo el paquete *mavros*, necesario para traducir las acciones entregadas por los nodos de **ftl** al protocolo MAVLink.

Todos los *ROS Packages* se presentan como una carpeta con el nombre del paquete, y se organizan en sub-carpetas de la siguiente manera [35]:

- Carpeta **src**: Contiene todos los nodos programados en C++.
- Carpeta **scripts**: Contiene los nodos programados en Python.
- Carpeta **include**: Se encuentran los archivos de cabecera de las bibliotecas C++ creadas.
- Carpeta **msg**: En caso de existir, aquí se crean los archivos de declaración de nuevos mensajes.
- Carpeta **launch**: En esta carpeta se encuentran todos los archivos *launch* del paquete.
- Archivo **CMakeLists.txt**: Archivo de compilación que declara ejecutables, dependencias, bibliotecas, mensajes e interdependencia con otros paquetes.
- Archivo **package.xml**: Manifiesto del paquete creado, define nombre, autores, licencias, e interdependencias.

El paquete puede descargarse como la carpeta con el mismo nombre, pegarse en el directorio `catkin_ws/src` creado durante la instalación de ROS, y compilarse mediante el comando:

```
$ catkin build ftl
```

En caso de no contar con alguno de los otros paquetes necesarios para el funcionamiento de **ftl**, el archivo de compilación requerirá su descarga e instalación y por último se compilarán todos los *ROS Packages* nuevos.

4.4. Implementacion en ROS

Cuando se quiere realizar una aplicación con múltiples nodos y parámetros haciendo uso de ROS, para comenzar se debe iniciar el ROS *master* en una terminal con el comando *roscore*. Luego, en nuevas terminales puede ejecutarse cada nodo deseado mediante el comando *rosrun*, seguido del nombre y los parámetros necesarios de cada programa. Esto ejecutará el código implementado para cada nodo dentro del mismo *master*.

En muchas ocasiones el orden en el cual se inicia cada uno de los programas es muy importante. Cada nuevo nodo se debería iniciar en una terminal diferente y en el orden correcto para obtener el funcionamiento deseado de manera adecuada. Finalmente, se tendrían demasiadas terminales abiertas lo cual puede resultar en una situación difícil de manejar.

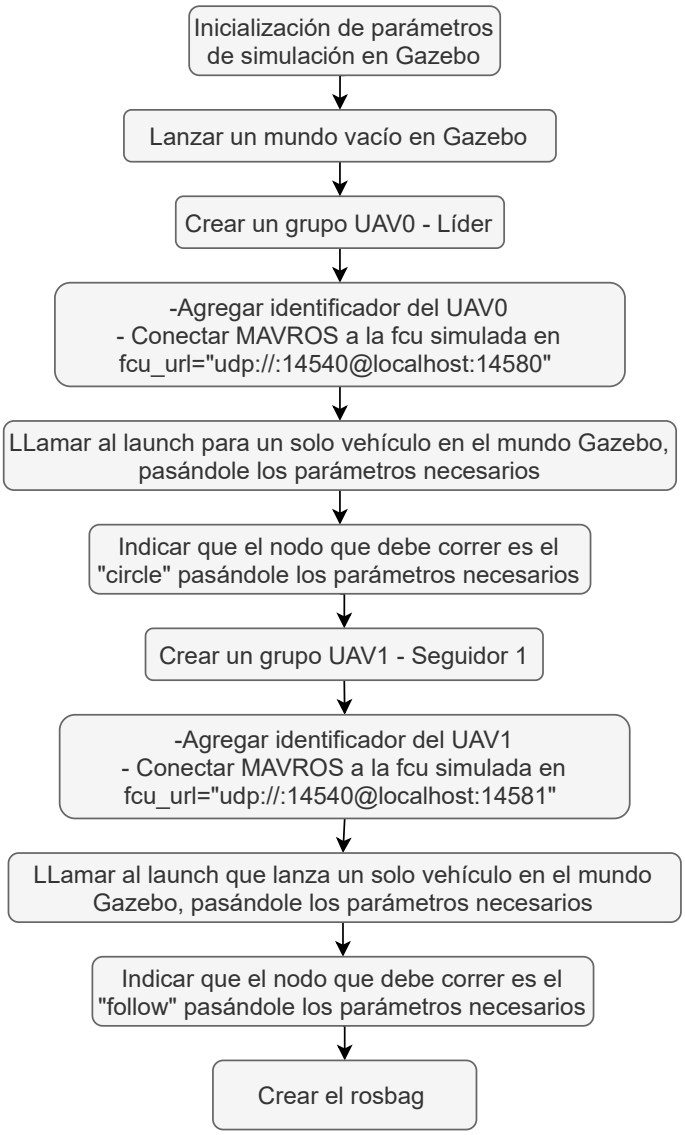
Por este motivo, se utilizarán los ya mencionados ROS *launch*, que permiten iniciar todo lo necesario desde un solo archivo. Éstos ayudan a iniciar cuanto nodo sea necesario y fijar todos los parámetros requeridos de manera ordenada y simple, además de ejecutar el *roscore* si no existiese un *master* en ejecución.

4.4.1. Archivos Launch

Recordando el objetivo particular de obtener tres drones realizando una trayectoria circular, en la cual los drones se van incorporando uno a la vez, primero el líder y luego los seguidores, se implementaron archivos *launch* adecuados.

En el caso de las simulaciones, la dinámica se detalla a continuación. Primero se inicia el dron líder en el origen de coordenadas, el nodo asociado al líder se denominará *circle* haciendo referencia a la trayectoria circular, y se ejecutará a una altura de 2 m. En segundo lugar se arma el primer seguidor, con un desplazamiento en la posición x de 5 m respecto del líder. Este dron seguidor implementará el nodo denominado *follow* a una altura de 3 m. Por último, se iniciará el segundo seguidor con un desplazamiento negativo en la posición x de 5 m, de igual manera que el primer seguidor, este seguidor utilizará un nuevo nodo *follow* a una altura de 3 m. Esta diferencia de altura entre el líder y los seguidores se dejó como margen de seguridad, pensando en el momento de realizar las pruebas de campo.

Además de las alturas y la posición inicial, en el *launch* se deben pasar otros parámetros necesarios para los distintos nodos. En particular, el nodo *circle* requiere saber además, el valor del radio del circulo a desarrollar, que en este caso particular se fijó con un valor de 5 m. Por otro lado, el nodo *follow* requiere saber, además de la altura, el ángulo deseado entre el vector velocidad del líder y el vector formado entre las posiciones del líder y seguidor, que en este caso se fijó en 30°. Otro parámetro importante de este nodo es la diferencia de altura mínima entre líder y seguidor, la cual se indicó con un valor de medio metro por seguridad.


Inicialización de parámetros de simulación en Gazebo
Lanzar un mundo vacío en Gazebo
Crear un grupo UAV0 - Líder
-Agregar identificador del UAV0
- Conectar MAVROS a la fcu simulada en fcu_url="udp://:14540@localhost:14580"
LLamar al launch para un solo vehículo en el mundo Gazebo, pasándole los parámetros necesarios
Indicar que el nodo que debe correr es el "circle" pasándole los parámetros necesarios
Crear un grupo UAV1 - Seguidor 1
-Agregar identificador del UAV1
- Conectar MAVROS a la fcu simulada en fcu_url="udp://:14540@localhost:14581"
LLamar al launch que lanza un solo vehículo en el mundo Gazebo, pasándole los parámetros necesarios
Indicar que el nodo que debe correr es el "follow" pasándole los parámetros necesarios
Crear el rosbag

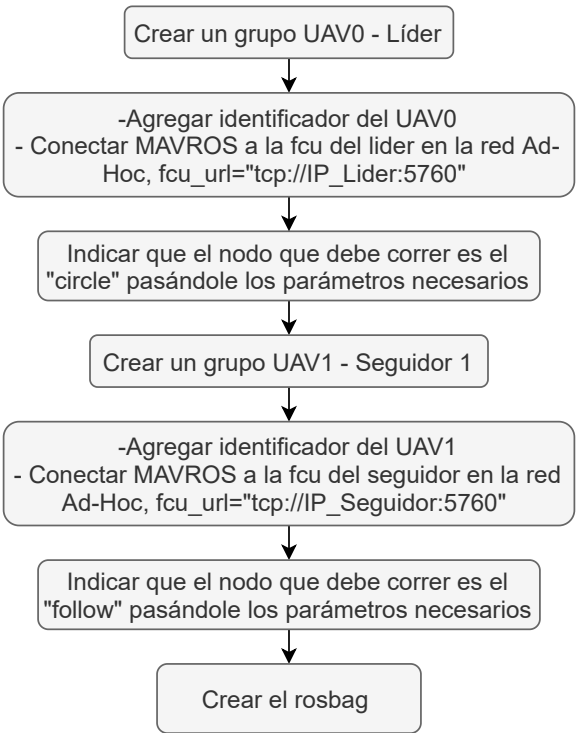


Figura 4.5: Diagrama de flujo del *launch* implementado para realizar las pruebas de campo con un líder y un seguidor.

Cuando se trata de los *launch* creados para las pruebas de campo, los mismos son una versión reducida de los de simulación, como se observa en el diagrama de flujo de la Figura 4.5. En este caso, una vez encendidas las aeronaves y conectadas a la red Ad-Hoc, se debe ejecutar un lanzador que conecta un nodo *mavros* a cada *FCU* mediante la conexión TCP-IP con cada computadora abordo y ejecuta los nodos líder o seguidor en cada caso.

Como puede verse comparando las Figuras 4.4 y 4.5, la ejecución de *mavros* recibe como parámetro el puerto de conexión, que en el caso de la simulación es un puerto UDP local que debe ser diferente para cada aeronave simulada, y en el caso de las pruebas de campo es un puerto TCP en la IP del dron al que se quiere conectar.

Para diferenciar los distintos *mavros* que se encuentran presentes en el mismo ROS *master*, se le asigna un nombre a cada *UAV*, y se agrupa la ejecución de los nodos correspondientes bajo dicho nombre. De esta manera, los tópicos y servicios referidos a cada aeronave comienzan con una etiqueta que los diferencia.

Cabe aclarar, que en ambos diagramas de flujo se presenta el ejemplo para un líder y un solo seguidor, dado que el grupo del segundo seguidor (UAV2) es análogo al del primero, cambiando únicamente el puerto de simulación o la dirección IP.

4.4.2. Procedimiento para Ejecutar los Launch

Una vez programados los nodos y los archivos *launch* a ejecutar, se deben compilar los nodos creados, ejecutando en la carpeta `catkin_ws/` el siguiente comando:

```
$ catkin build
```

Luego de esto, se agregan variables del entorno de ROS necesarias para simular, corriendo (también dentro de la carpeta `catkin_ws/`):

```
$ source devel/setup.bash
```

Si se desean hacer pruebas con drones reales, ya se está en condiciones de ejecutar el archivo *launch* deseado. Para esto, si se desea ejecutar el archivo *launcher_ejemplo.launch*, se debe ejecutar dentro de la carpeta `catkin_ws/` el comando:

```
$ roslaunch ftl launcher_ejemplo.launch
```

En caso de querer realizar simulaciones en Gazebo, antes del comando anterior que ejecuta el *launch* `launcher_ejemplo.launch`, se deben ejecutar los siguientes comandos dentro de la carpeta `Firmware` (instalada en la Sección 4.2):

```
$ DONT_RUN=1 make px4_sitl_default gazebo
$ source Tools/setup_gazebo.bash $(pwd) $(pwd)/build/px4
  _sitl_default
$ export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:$(pwd):$(pwd
  )/Tools/sitl_gazebo
```

Capítulo 5

Nodos Desarrollados

Se describen en este capítulo todos los nodos programados durante el proyecto, tanto para el dron líder como para los seguidores. Se hará especial énfasis en los nodos que fueron utilizados finalmente en la dinámica *Follow the leader*, en la que la trayectoria preestablecida fue circular y la estrategia de seguimiento y evasión fue la "mixta", definida en el Capítulo 3.

5.1. Nodos Líder

El diagrama de flujo de la Figura 5.1 describe de manera concisa el programa implementado para que un dron realice una trayectoria circular, llamado `circle.cpp`.

Los nodos de los líderes reciben como parámetros desde el *launch* únicamente valores que determinan la trayectoria a realizar, como son por ejemplo el tiempo total de vuelo, y parámetros geométricos como la altura de vuelo y similares (por ejemplo, en caso de que la trayectoria corresponda con una circunferencia, el parámetro correspondiente al radio de la misma se brinda desde los archivos *launch*).

Dentro del nodo, el líder es el encargado de publicar el centro del desarrollo de Taylor para el sistema de coordenadas cartesiano absoluto compartido por los tres drones (el cual determina la ubicación espacial del punto origen de coordenadas de los tres drones). Para esto, el líder publica desde un inicio un cierto valor de tipo *NavSatFix* acordado por los tres drones en el tópico `/coord`, el cual indicará que el líder aún no ha publicado el centro del Taylor. Luego de que el líder tenga una estimación relativamente precisa de su latitud y longitud, publicará estos valores en el mismo tópico.

El líder también se encarga de publicar en un tópico de tipo *booleano*, el cual indica cuando está volviendo a su punto de inicio para aterrizar. Es decir que este tópico se mantendrá con valor *False*, hasta el momento en que esté retornando a la base unos segundos antes de aterrizar, publicando en ese momento un valor de *True*.

Una vez armado, el dron guarda su posición de inicio en el marco de coorde-

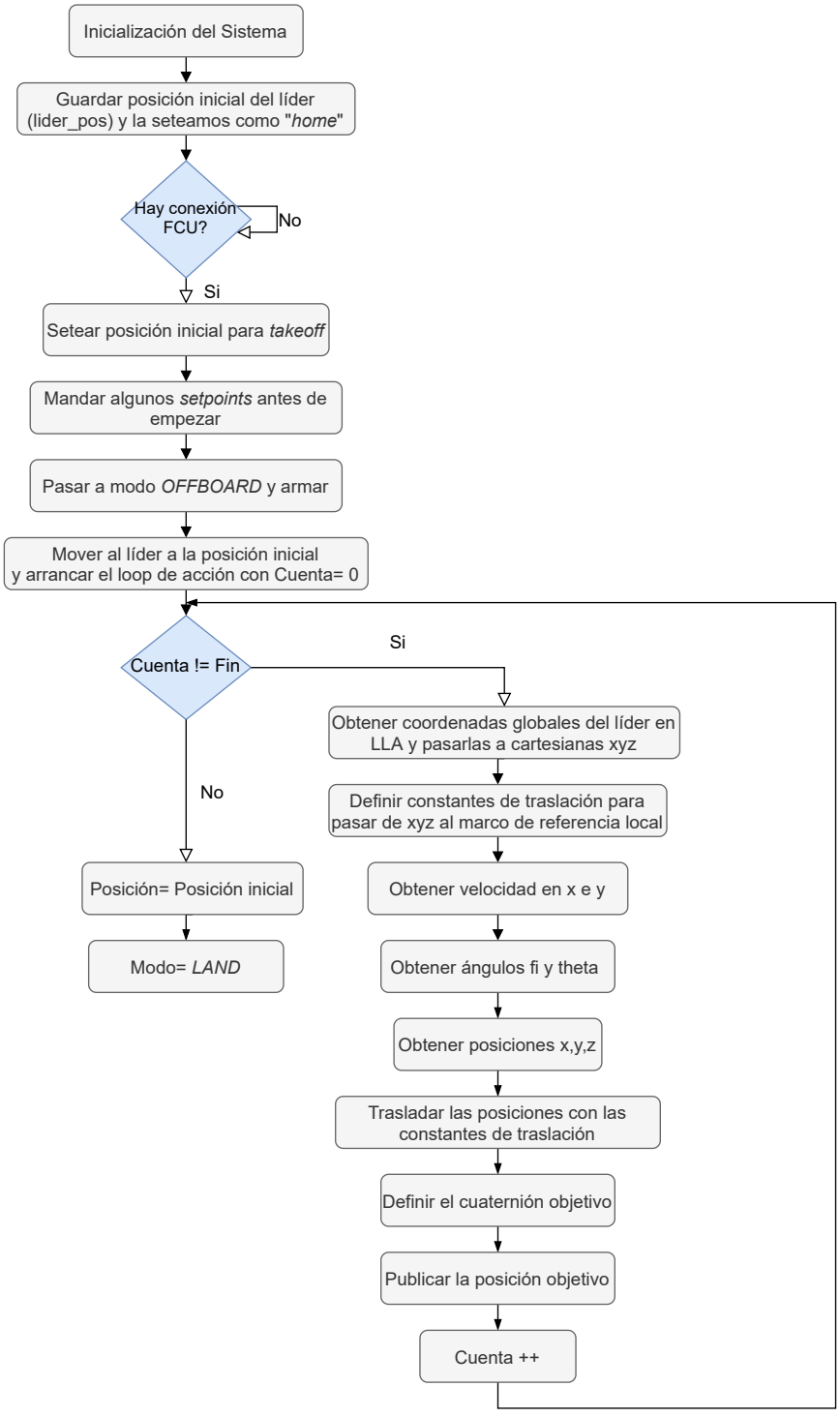


Figura 5.1: Diagrama de flujo del nodo que implementa que el líder siga una trayectoria circular.

nadas absoluto cartesiano, para saber la posición de retorno al final del nodo. El dron líder deberá siempre a partir de la latitud, longitud, y altura sensada en cada instante, encargarse de transformar dichos valores al sistema cartesiano global a partir del Taylor orden 1 y publicarlo en un tópico de tipo *PoseStamped*.

Si se tiene una trayectoria $(x(t), y(t), z(t))$ en el marco de coordenadas absoluto, la cual se desea que el líder siga, entonces se debe estimar en cada instante cómo transformar las coordenadas del sistema local (`/uav0/mavros/local_position/pose`) referido a su posición de origen (donde fue encendido el dron), al sistema cartesiano global generado con el Taylor de orden 1.

Como ambos sistemas de coordenadas están igualmente orientados (los ejes crecen hacia el Este, Norte, y arriba respectivamente), este cambio de coordenadas se estima simplemente tomando las diferencias componente a componente entre ambos sistemas. Una vez estimada la diferencia en el instante t, se convierten las coordenadas $(x(t), y(t), z(t))$ del marco absoluto al marco local, para publicar dicha posición *setpoint* en el tópico `/uav0/mavros/setpoint_position/local`.

Cuando ha finalizado el tiempo de vuelo, se le otorgan al dron 20 segundos para retornar a su posición de inicio y aterrizar (en caso de no alcanzar esta posición en este tiempo, aterriza en el lugar).

A lo largo de todo el nodo, las alturas se manejan siempre tomando la variación de altura con respecto a su valor inicial, y se asume que todos los drones comienzan a igual altura. Esto se hace de esta forma, pues si se intenta usar una referencia absoluta como por ejemplo el nivel del mar, entonces se puede verificar que los drones presentan diferencias apreciables (del orden de varios metros) cuando miden este valor, pues el sistema GPS no es particularmente preciso para medir alturas. Al obtener la altura de esta manera, se está haciendo uso del *sensor fusion* resultante del barómetro y el LIDAR, además del módulo GPS, como se verá en el Capítulo 7.

5.2. Nodos Seguidores

Los nodos de los drones seguidores reciben desde el *launch* parámetros que indican la altura de vuelo a la que se posicionarán luego de despegar, y antes de aterrizar, la diferencia de altura que deben mantener con el dron líder, el ángulo con el que deben mantener su posición relativa al líder con el vector velocidad del líder, un número identificador del seguidor (1 ó 2 en este caso), y condiciones de armado y despegue (que pueden ser condiciones de tiempo, o depender de la posición del líder).

Como lo muestra la Figura 5.2, al inicio del nodo llamado `follow.cpp`, el dron seguidor espera a que se cumpla su condición de inicio antes de armar. Luego de armado, antes de despegar, el dron espera (si es necesario) a que el líder publique la posición del centro del Taylor para tener un marco de referencia absoluto compartido entre los drones.

Una vez que recibe esta posición, la dinámica es similar a la del dron líder, en la que a partir de su longitud, latitud y altura, publica su posición en un tópico de tipo *PoseStamped*, y para crear un *setpoint* en el marco local (referido al punto donde el dron se encendió) se debe primero convertir el *setpoint* del marco absoluto

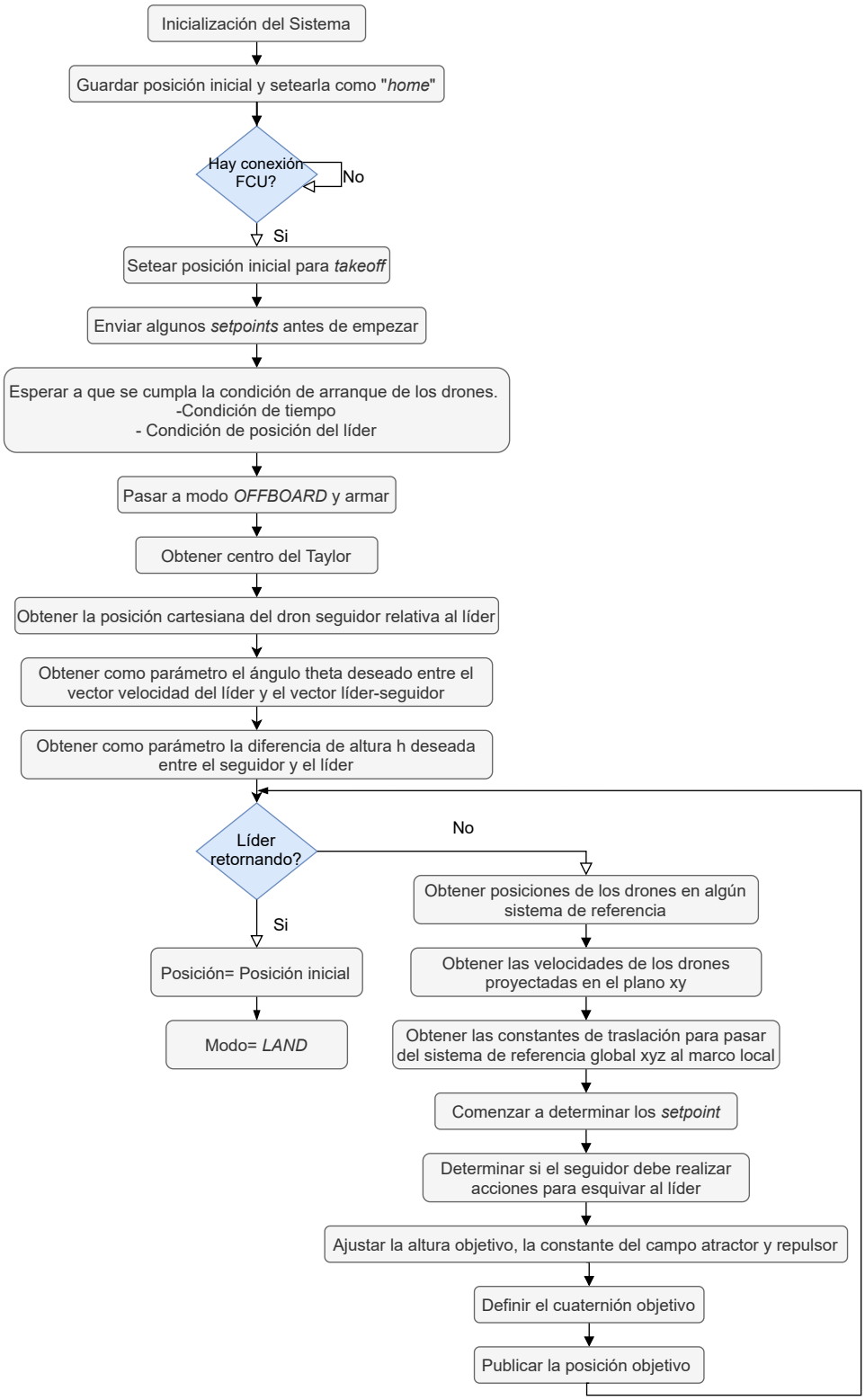


Figura 5.2: Diagrama de flujo del nodo que implementan los seguidores para poder seguir al líder o esquivarlo según sea el caso.

al marco global a partir de las diferencias de traslación entre un marco y otro. El seguidor además guarda su posición de inicio en el marco absoluto cartesiano, para poder retornar al final.

El dron seguidor recibe las velocidades de los otros drones (líder y otros seguidores en caso de existir) y sus posiciones en el marco absoluto generado por el Taylor de orden 1, y usando esta información genera campos potenciales atractores (a partir de la estrategia de **Tres Estados**, que le permiten seguir y esquivar al líder), y campos repulsores (que le permiten esquivar a los otros seguidores). A partir de lo anterior, se obtiene el *setpoint* en el marco absoluto, el cual es convertido al marco local para luego ser publicado.

Cada dron seguidor se encarga de convertir su latitud, longitud y altura al sistema cartesiano absoluto y publicarla para que los demás drones dispongan de dicha información. Además, los drones seguidores están siempre atentos al tópico booleano publicado por el líder para saber en que momento el mismo está retornando a la base para aterrizar.

Una vez que el líder está retornando al punto de despegue, los seguidores generan un campo atractor en su base (la cual había sido guardada en el marco de referencia cartesiano absoluto en un principio) y campos repulsores en los otros drones (incluyendo en este punto un campo repulsor generado por el líder). Disponen también de 20 segundos para retornar a su base, y en caso de no alcanzarla entonces aterrizan en el lugar.

Al igual que sucede en el nodo del líder, las alturas se manejan siempre tomando la variación de altura con respecto a su altura inicial, asumiendo que todos los drones comienzan a igual altura.

5.3. Intercambio de Información

En la Figura 5.3 se muestra un grafo que representa las comunicaciones entre los distintos nodos al realizar una simulación con un dron líder ($uav0$) y dos drones seguidores ($uav1$ y $uav2$). En la misma, se puede observar como el dron líder publica la información de su posición en el marco compartido entre los tres drones (descrito en la Sección 3.2) en el tópico `/uav0/info_pos`, mientras que los demás drones están suscritos a este tópico para obtener la información de posición del líder.

Del mismo modo, los drones seguidores publican sus posiciones en los tópicos `/uav1/info_pos` y `/uav2/info_pos`, mientras se suscriben al tópico de posición del otro (de modo que cada dron seguidor, conoce la posición del otro seguidor y del líder). Junto con esto, los seguidores también están suscritos a los tópicos de velocidad del resto de la flota (siendo `/uav0/mavros/local_position/velocity_local` el del líder, `/uav1/mavros/local_position/velocity_local` el del primer seguidor y `/uav2/mavros/local_position/velocity_local` el del restante), es decir que ambos seguidores tienen acceso a las velocidades de los otros drones.

Las posiciones publicadas en los tópicos `/info_pos` de cada dron, se almacenan en `/rosbag_record_ftl` para tener un registro de las trayectorias de cada dron en el tiempo. El líder también guarda en este tópico su posición en el marco local (`/uav0/mavros/local_position/pose`), para poder comparar la trayectoria medida en este marco con la trayectoria obtenida a partir de la aproximación del Taylor de primer orden.

Además, el líder publica en dos tópicos auxiliares, llamados `/coord` y `/come_back`, a los que ambos seguidores están suscritos. En el tópico `/coord` se almacena la latitud, longitud, y altura (sobre el nivel del mar), medida por el líder antes de despegar. Estas coordenadas son usadas por el resto del enjambre como centro del Taylor (es decir que corresponderán con el punto $(0, 0, 0)$ en el marco cartesiano), de manera de obtener un sistema de referencia común. Por otro lado, el tópico `/come_back` es usado por el líder para comunicar el momento en que está retornando a su base para aterrizar, de modo que $uav1$ y $uav2$ sepan cuando dejar de seguirlo y volver a sus respectivas bases.

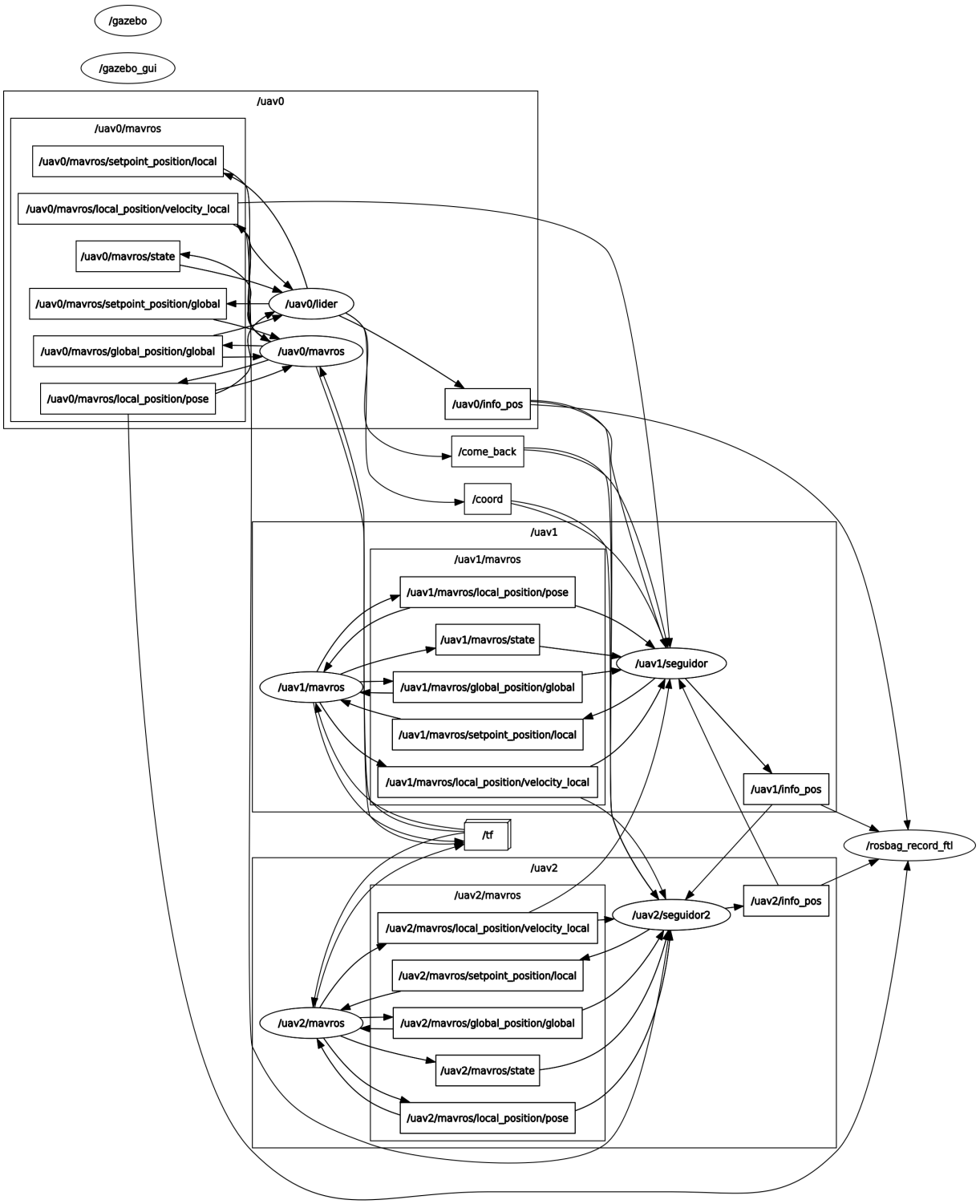


Figura 5.3: Grafo de comunicaciones entre tópicos durante una simulación con un dron líder y dos seguidores. Este grafo se obtuvo automáticamente con el paquete `rqt_graph` [37].

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 6

Comunicaciones

En este capítulo se mencionarán conceptos relacionados con el establecimiento de la comunicación del sistema. En primer lugar se verá el protocolo MAVLink usado en la comunicación entre el Pixhawk y el NUC, y en la comunicación entre la FCU Pixhawk y QGroundControl.

Luego se verá la red de tipo Ad-Hoc que se utilizó para la comunicación entre los drones, y también con la estación terrestre. Finalmente, se detallará lo necesario para establecer todas las comunicaciones del sistema.

6.1. Comunicación Mavlink

MAVLink (*Micro Air Vehicle Link*) es un protocolo de comunicación fuertemente utilizado para transmitir información entre una controladora de vuelo que viaja a bordo de un *UAV* y una estación de control terrestre u otro agente externo. Fue desarrollado en el año 2009 por un equipo de la Escuela Politécnica Federal de Zúrich y posteriormente mejorado con el aporte de la comunidad [16].

La inmensa mayoría de las controladoras de vuelo que se comercializan en la actualidad utilizan MAVLink (en sus versiones 1 o 2 dependiendo de la versión de *firmware*), tanto para comunicarse con las *GCS* (*Ground Control Station*), como para intercambiar información con los sensores y actuadores presentes en el dron [18].

En el marco de este proyecto se utiliza el protocolo MAVLink en la comunicación establecida entre la *FCU* y la computadora a bordo, dado que la controladora Pixhawk así lo requiere. De la misma forma, la comunicación entre la controladora de vuelo y el software QGround Control (que se describe en detalle en la Sección 4.1.5) es implementada haciendo uso de dicho protocolo.

Dependiendo del emisor, receptor e intención del mensaje enviado, MAVLink cuenta con sub-protocolos particulares a los que llama "microservicios", entre los que se encuentran los protocolos de conexión, comandos, estado actual, parámetros, imágenes, video en tiempo real o misiones.

El protocolo en general combina dos modelos de mensajería distintos. Por un lado, para el flujo de datos en tiempo real, los mensajes se publican como *topics*

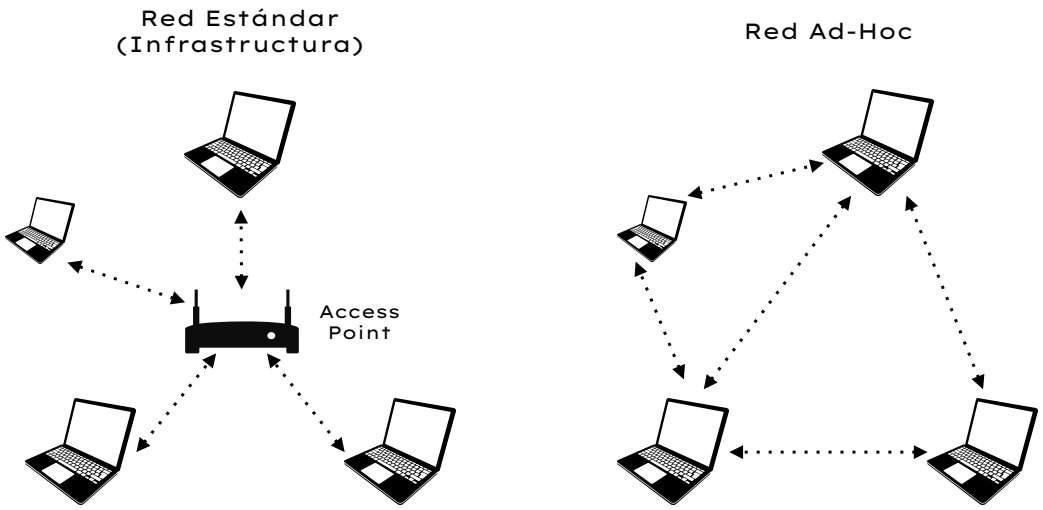


Figura 6.1: Modos inalámbricos.

y el receptor se suscribe a ellos, como ser los tópicos de posición de los drones. En cambio se utiliza una comunicación punto a punto para establecer los subprotocolos de configuración como por ejemplo el de misión o de parámetros.

El conjunto de mensajes admitidos por un sistema MAVlink, también denominado dialecto, es definido en un archivo XML.

El conjunto de herramientas MAVLink también provee un generador de bibliotecas, que utiliza las definiciones de los mensajes en un XML para generar bibliotecas MAVLink en cada uno de los lenguajes de programación admitidos, entre los que se encuentran *C, C++, python, java* o *javascript* [16]. Una vez descargada e importada la biblioteca, pueden escribirse programas que soliciten información a la controladora, o envíen comandos desde una computadora en tierra o una computadora a bordo.

6.2. Redes de Comunicación Ad-Hoc

Al momento de realizar las pruebas de campo, para entablar la comunicación entre los drones, se utilizó el protocolo de redes de comunicación Ad-Hoc. A continuación, realizaremos una breve introducción teórica de este tipo de redes.

El propósito por el cual fueron creadas era permitir que distintas unidades en un área geográfica extensa se comunicaran entre sí mediante enlaces radios, con libertad de movimiento y de forma cooperativa, de modo que cada nodo podía actuar como un dispositivo terminal o como un conmutador de paquetes [6].

Son de particular interés las redes Ad-Hoc en las que sus nodos son móviles, por lo que la topología de la red es variable, de modo que un nodo que tiene un enlace con un nodo vecino, puede alejarse lo suficiente como para salir del área de cobertura de ese nodo vecino y formar un nuevo enlace con un tercer nodo que esté incluido en su área de cobertura. En consecuencia, otra característica importante es que existen cambios en las rutas entre origen y destino de un paquete.

Una red Ad-Hoc se crea para cierto propósito de forma temporal. Al tratarse de una red inalámbrica descentralizada, no depende de una infraestructura preexistente, como por ejemplo, los *routers* en redes cableadas. En la Figura 6.1 se contraponen ambos tipos de conexión para que se comprenda de mejor manera la diferencia entre ambos.

En una red Ad-Hoc, cada nodo toma parte en la tarea de encaminamiento de los paquetes mediante el reenvío de datos hacia otros nodos, y esta tarea se realiza dinámicamente sobre la base de conectividad de la red.

El principal inconveniente de este tipo de redes radica en el número de saltos que debe recorrer la información antes de llegar a su destino. Cada nodo que retransmite la información implica un salto, cuantos más saltos mayor es el tiempo que tarda en llegar la información a su destino y aumenta la probabilidad de que la información se corrompa con cada salto. Como contrapartida, si dichos saltos son más cortos podría mejorar la relación señal a ruido (SNR) [6].

Debido a que una de las principales aplicaciones de los enjambres de drones es la de incrementar el área de cobertura respecto al uso de un único dron, las aeronaves están constantemente cambiando de posición y estos cambios pueden suceder a velocidades altas. Por lo que, el objetivo es tener una comunicación independiente de una estación terrestre y que pueda variar su conexión según la conveniencia, para lograrlo se utilizará una red Ad-Hoc.

6.3. Comunicaciones del Sistema

Por requerimientos preexistentes al proyecto, el flujo de información entre los componentes de la flota y la estación de control terrestre debe implementarse mediante una red WiFi de tipo Ad-Hoc. Para esto es necesario conectar a la PC un adaptador WiFi que soporte redes de este tipo, e instalar los drivers adecuados. En este proyecto, se optó por el adaptador TP-Link WN722N. Una vez configurada la red, se le debió asignar una dirección IP fija a cada integrante de la misma, de forma de poder identificarlos siempre de la misma manera.

ROS brinda la posibilidad de implementar un sistema con un único ROS *master* o utilizar arquitecturas multimaster [38]. La arquitectura *multimaster* permite seleccionar qué información de cada tópico se compartirá entre los distintos *master* y de esta forma permite disminuir el flujo de datos que se intercambian en el sistema. Como contrapartida este tipo de arquitectura conlleva una complejidad mayor a la utilización de un único *master*. Dada la baja cantidad de componentes en la red, se decidió utilizar un único ROS *master* que pudiera ejecutarse en cualquiera de los componentes de la red, asignando su dirección IP de manera que todos los tópicos de los demás componentes sean publicados y leídos en esa dirección.

Si bien, como se explicó en la Sección 4.1.1, el ROS *master* puede ejecutarse con el comando *roscore* en cualquiera de las computadoras que forman la red, se optó por hacerlo en la PC que se encuentra en tierra, dadas las ventajas que otorga para acceder al flujo de datos y controlar los procesos en caso de que ocurra un inconveniente o se pierda la conexión con algún otro componente.

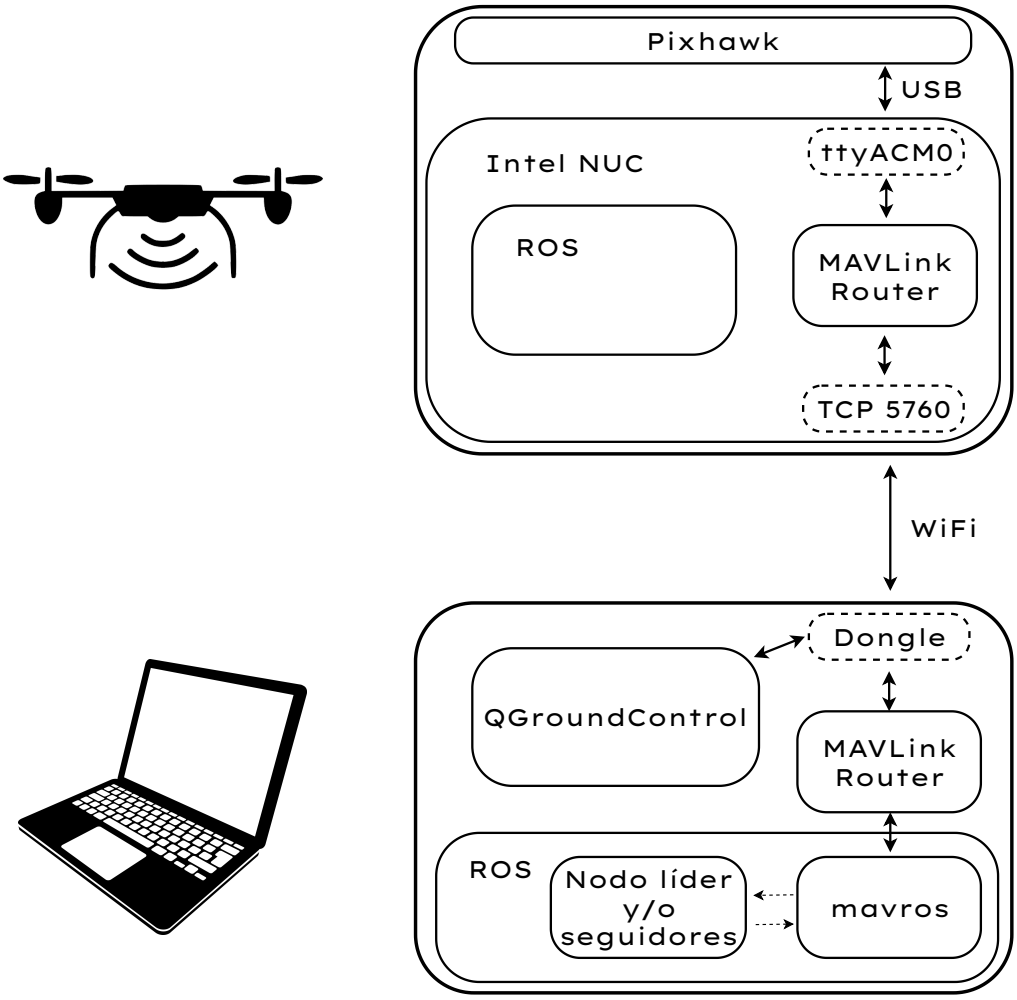


Figura 6.2: Esquema de interconexiones entre los módulos del sistema, los tres drones son análogos y se diferencian por su dirección IP.

Por otra parte, cada Intel NUC se comunica con la controladora de vuelo mediante una conexión serial por cable USB, en el que se establece una comunicación de protocolo MAVLink. Además, la estación en tierra con QGroundControl, se conecta a cada dron mediante una conexión TCP con un puerto de la computadora a bordo, que se encuentra redirigido (por la herramienta MAVLink Router) hacia el mismo puerto serial USB para lograr acceder a la controladora. Un esquema de todas las conexiones mencionadas se ilustra en la Figura 6.2.

Por último, los *UAV* están equipados con un sistema de comunicación por radiofrecuencia, de forma de ser controlados directamente por un radio control, salteando la comunicación WiFi en caso de que la misma falle o que las aeronaves, funcionando en forma autónoma, presenten un comportamiento inesperado o peligroso. Es importante destacar que cada dron cuenta con un radio control diferente, en caso que sea necesario pilotar manualmente varios integrantes de la flota de manera simultánea.

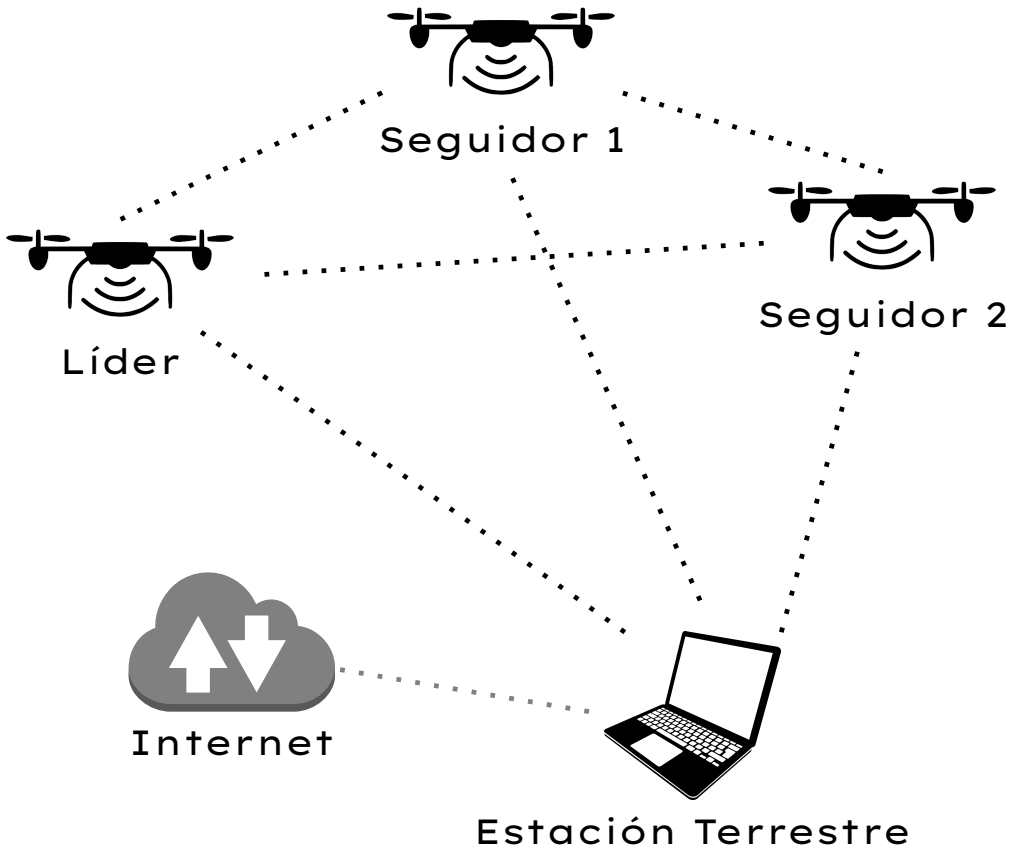


Figura 6.3: Esquema del sistema implementado en este proyecto, donde la estación terrestre ejecuta el ROS *master*, que es compartido entre todos los miembros de la red Ad-Hoc.

6.3.1. Resumen

Como se observa en el esquema de la Figura 6.3, la dinámica de seguimiento del líder se implementó sobre una flota de tres UAVs (un líder y dos seguidores) interconectados entre sí y con una computadora en tierra (estación terrestre), que cuenta además con acceso a internet.

Los tres cuadricópteros presentan idénticas características de hardware, con una computadora a bordo de cada uno, que le da acciones de control mediante comandos MAVLink enviados por un puerto USB a la controladora de vuelo. Dichas acciones de control se computan en un único ROS *master* compartido (que se encuentra alojado en la PC en tierra) y se le comunican a cada computadora a bordo mediante una red WiFi de tipo Ad-Hoc, como lo muestra la Figura 6.2. La traducción de las acciones de control a mensajes del protocolo MAVLink es realizada por el paquete MAVROS.

Para llevar a cabo la dinámica, los nodos de ROS programados aplican diferentes algoritmos de control, que permiten trazar la trayectoria preestablecida para el líder y desarrollar acciones de seguimiento y evasión de colisiones por parte de los seguidores.

Cabe aclarar, que podría excluirse a la estación terrestre de la red y ejecutarse el *roscore* en una de las computadoras a bordo, independizando al enjambre como un conjunto de robots completamente autónomo. En este caso se perdería la posibilidad de monitorear el vuelo en tiempo real o tomar acciones de ningún tipo en caso que algo salga mal (salvo por el radio control si se encontrara dentro del alcance).

Capítulo 7

Análisis del Control de Bajo Nivel

A lo largo de este capítulo, se introducirán brevemente los conceptos principales acerca del funcionamiento de un cuadricóptero y una controladora de vuelo montada sobre él. De esta forma, se intenta acercar al lector a los eventos que se desencadenan una vez que los algoritmos de control de alto nivel (presentados en el Capítulo 3) le envían una posición objetivo a la aeronave.

Estos conceptos no son indispensables para comprender el funcionamiento del enjambre, ni de las pruebas realizadas en el proyecto que serán presentadas a partir del Capítulo 8. Sin embargo pueden resultar interesantes para lograr un entendimiento íntegro del sistema implementado en todos los niveles.

7.1. Principios de Funcionamiento de un Cuadricóptero

7.1.1. Modelo

Para comprender los principios que rigen el funcionamiento de un *UAV* cuadricóptero, como el representado en la Figura 7.1 (y también como los construidos en este proyecto) se debe conocer primero su modelo físico-matemático.

Con ese fin, a lo largo de esta sección se denominan:

- (x, y, z) la posición absoluta del centro de masa del dron.
- (ϕ, θ, ψ) los ángulos de rotación del dron según los ejes x, y y z respectivamente (ángulos de Euler). Se supone que $-\frac{\pi}{2} < \phi < \frac{\pi}{2}$ y que $-\frac{\pi}{2} < \theta < \frac{\pi}{2}$, mientras que ψ puede tomar cualquier valor.
- m es la masa total del dron.
- I_x, I_y, I_z son los principales momentos de inercia del dron, desde el centro de masa en las direcciones x, y y z respectivamente.
- l es la distancia entre el eje de las hélices y el centro de masa del dron.
- w_1, w_2, w_3 y w_4 son las velocidades angulares de las hélices del dron.

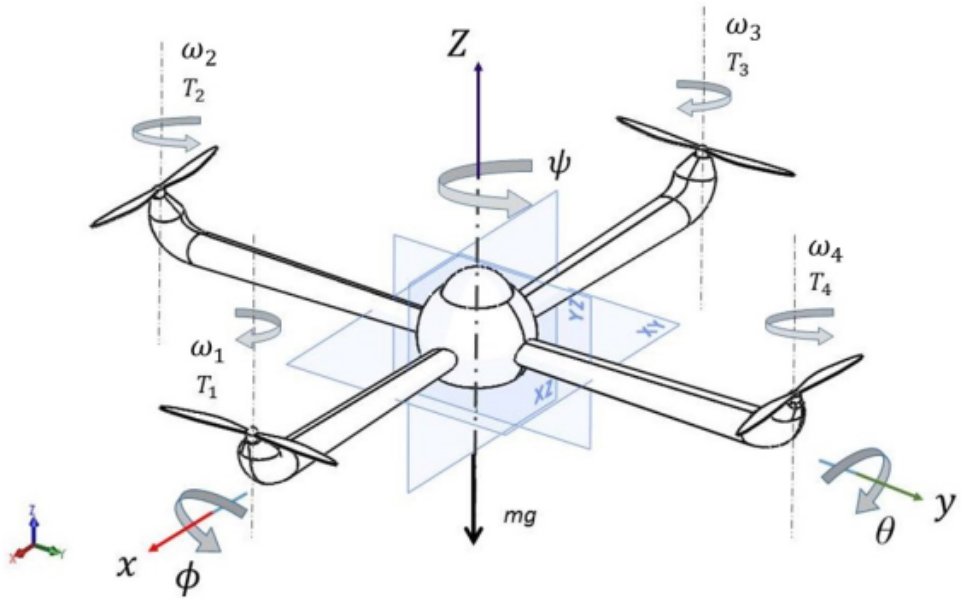


Figura 7.1: Esquema del dron y su sistema de coordenadas [10].

- T_1, T_2, T_3 y T_4 son las fuerzas de empuje de las hélices del motor.
- J_r es la inercia de las hélices.
- $\Omega = w_2 + w_4 - w_3 - w_1$ es la velocidad residual del rotor.

El modelo en variables de estados que domina al dron tiene la siguiente forma [10]:

$$\begin{pmatrix} x' \\ x'' \\ y' \\ y'' \\ z' \\ z'' \\ \phi' \\ \phi'' \\ \theta' \\ \theta'' \\ \psi' \\ \psi'' \end{pmatrix} = \begin{pmatrix} x' \\ [cos(\psi)sen(\theta)cos(\phi) + sen(\psi)sen(\phi)]\frac{U_1}{m} \\ y' \\ [sen(\psi)sen(\theta)cos(\phi) - cos(\psi)sen(\phi)]\frac{U_1}{m} \\ z' \\ cos(\theta)cos(\phi)\frac{U_1}{m} - g \\ \phi' \\ \frac{1}{I_x}[(I_y - I_z)\psi'\theta' - J_r\theta'\Omega + U_2 l] \\ \theta' \\ \frac{1}{I_y}[(I_z - I_x)\psi'\phi' + J_r\phi'\Omega + U_3 l] \\ \psi' \\ \frac{1}{I_z}[(I_x - I_y)\theta'\phi' + U_4] \end{pmatrix} \tag{7.1}$$

Siendo

$$\begin{cases} U_1 := T_1 + T_2 + T_3 + T_4 = K_T(w_1^2 + w_2^2 + w_3^2 + w_4^2) \\ U_2 := T_4 - T_2 = K_T(w_4^2 - w_2^2) \\ U_3 := T_3 - T_1 = K_T(w_3^2 - w_1^2) \\ U_4 := \frac{d}{K_T}(T_1 + T_3 - T_2 - T_4) = d(w_1^2 + w_3^2 - w_2^2 - w_4^2) \end{cases} \tag{7.2}$$

con $d > 0$ una constante que depende de la densidad del aire, del radio y de la forma de las hélices, y K_T es el coeficiente de empuje de las hélices.

7.1.2. Interpretación

Cuando un motor se enciende y hace girar la hélice acoplada a su eje, las palas empujan el aire hacia abajo, generando una fuerza de reacción vertical y proporcional al cuadrado de la velocidad angular.

Para lograr desplazarse de un cierto punto del espacio a otro, el cuadricóptero es capaz de combinar traslaciones en los tres ejes (x, y, z) y rotaciones en los ángulos (ϕ, θ, ψ). Con este fin, se deben coordinar las velocidades de los cuatro motores en cada instante (dos girando en sentido horario y dos en sentido antihorario, como se aprecia en la Figura 7.1), lo que repercute directamente en el empuje que produce cada motor y el momento angular total en ese instante.

Intuitivamente, si los cuatro motores permanecen a la misma velocidad, el dron se encuentra horizontal sin rotar en ninguno de los ángulos, pudiendo subir, bajar o permanecer flotando según la relación que exista entre la suma de los empujes de todos los motores ($U_1 = T_1 + T_2 + T_3 + T_4$) y la fuerza peso ($mg$).

Cuando el motor 3 gira más rápido que el motor 1 ($U_3 > 0$), mientras 2 y 4 tienen misma velocidad, la aeronave se inclina según el ángulo θ y por lo tanto avanza en la dirección del eje x. De forma análoga, si la diferencia de velocidades se produce entre las hélices 2 y 4 ($U_2 > 0$) el UAV avanza en el eje y.

Por último, si la suma de los empujes que producen los dos motores que giran de forma horaria no es idéntica a la suma de los empujes de los motores antihorarios ($U_4 = T_1 + T_3 - T_2 - T_4 \neq 0$), por conservación del momento angular se produce una rotación del ángulo ψ, que cambia "hacia donde apunta" la aeronave.

7.2. Modos de Vuelo

Antes de seguir, resulta necesario introducir el concepto de **modos de vuelo**, que definen cómo la controladora responde ante las entradas y cómo aplica las acciones de control en el movimiento. Los nombres y características puntuales de cada modo de vuelo dependen del firmware instalado en la controladora, pero en general se pueden agrupar entre manuales, asistidos y autónomos.

Existen modos acrobáticos o de carrera, que habilitan al dron a realizar directamente los movimientos comandados desde el radio control, casi sin asistencia de estabilidad. También hay modos que permiten el radio control, pero aportan asistencia en el control de estabilidad en altura o control de estabilidad total en posición, es decir que a pesar de las condiciones climáticas, si no hay comandos desde tierra la aeronave permanece flotando en el lugar. También existen modos de misión autónoma, que ejecutan una ruta previamente planificada y guardada en la controladora, sin interacción con un piloto durante el vuelo, y finalmente modos completamente autónomos, que reaccionan en tiempo real ante los comandos recibidos desde una computadora a bordo o desde otra fuente externa.

En el marco de este proyecto, todos los vuelos fueron realizados en modo autónomo (llamado *Offboard*), donde la controladora de vuelo Pixhawk recibe la posición y actitud objetivo desde la computadora a bordo. El radio control estuvo presente para tomar el control directo en caso de emergencia, pasando a un modo manual con asistencia de estabilidad total (llamado *Position Hold*).

7.3. Control de Estabilidad y Vuelo

Un cuadricóptero es un sistema muy complejo de modelar, y por tanto complejo de controlar. Dicha complejidad está dada por la cantidad de sensores involucrados (que entregan información a una frecuencia de muestreo variada y con distintos marcos de referencia), la cantidad de ángulos y direcciones que componen el movimiento y la cantidad de aspectos físicos que deben tenerse en cuenta en la práctica.

La unidad controladora de vuelo Pixhawk con su firmware PX4, se encarga de determinar y ejecutar las acciones de control que permiten que cada dron se mantenga estable y pueda moverse de forma adecuada. Es por esto, que no fue necesario desarrollar las acciones de control de bajo nivel en particular, pero sí se optó por investigar acerca de ellas, obteniendo lo que se detalla a continuación.

Asumiendo la complejidad anteriormente mencionada, la *FCU* divide el problema en diferentes módulos más pequeños [23], entre los que se destacan: drivers de entradas y salidas, módulo de conectividad, almacenamiento de información, módulo de fusión sensorial, estimador de posición, control de posición y actitud, control de modos de vuelo y módulo de vuelo autónomo. Todo el flujo de información entre módulos se maneja desde un bus de mensajería central.

7.3.1. Fusión Sensorial y Estimación de Posición

Una vez que la información de los sensores es recibida por los drivers, es procesada para obtener una estimación de la actitud y posición actual de la aeronave. Dada la variedad de sensores útiles para obtener la posición, y la redundancia de los sensores que informan la orientación en cada instante, debe realizarse una combinación que resulte en la mejor estimación posible.

Por ejemplo, para obtener la posición de cada dron, se hace uso del sensor de GPS. Sin embargo éste tiene mucha incertidumbre y se perdería mucha información si no se hiciera uso de la lectura en los otros sensores como el acelerómetro, el barómetro y el lidar.

La combinación de información proveniente de distintas fuentes es conocida como **fusión sensorial**, y consiste de un balance ponderado de los datos que ingresan, que en el caso del firmware PX4 se calcula a través de **Filtros de Kalman Extendidos** (EKF) [13] [14].

El filtro de Kalman asume un modelo en variables de estado, donde se consideran variables de estado como la posición y la actitud del dron, así como las velocidades, y otros factores que afecten a la evolución de la trayectoria de los drones que puedan ser de interés. La lista completa de variables de estado se puede encontrar en [31].

Dado el modelo de transición de estados, la fusión sensorial se hace recursivamente en dos pasos. En primera instancia, se estima el estado actual en función de la estimación del estado anterior, para luego corregir la estimación actual usando las nuevas medidas de los sensores.

La versión clásica del filtro de Kalman asume un modelo lineal con las variables de estado (lineal tanto para la transición de estados, como para las magnitudes medidas por los sensores). En este caso, el modelo no es lineal (como puede verse en la ecuación (7.1), donde se ve una relación no lineal que vincula derivadas de la posición con los ángulos de Euler que definen la actitud del dron). Por este motivo el firmware PX4 hace uso de la variante EKF de Kalman, en la cual el filtro realiza una linealización del modelo utilizando la aproximación del Taylor de primer orden.

7.3.2. Control de Posición y Actitud

La información obtenida en el módulo de estimación de posición ($\mathbf{X}$), junto con el cuaternión de orientación ($\mathbf{q}$) que se obtiene a partir de las brújulas, ingresan a los controladores de estabilidad y vuelo, que deben producir como salida un vector de empujes deseados en los cuatro motores ($\mathbf{T}$).

Con ese fin, la *FCU* presenta una arquitectura de control de tipo cascada, en la que la acción de control resultante de cada bloque ingresa como entrada (*setpoint*) al siguiente bloque de control, hasta obtener la salida final. A su vez, cada bloque de control se implementa mediante arquitecturas tradicionales P (Proporcional) o PID (Proporcional, Integral, Derivativo), como muestra la Figura 7.2, que se pasará a explicar a continuación [24].

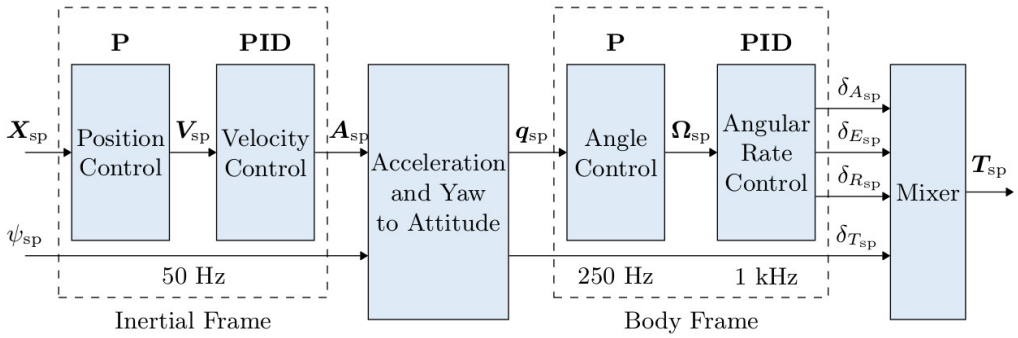


Figura 7.2: Arquitectura de control del Firmware PX4 Multicopter [24].

En primer lugar, la posición deseada $\mathbf{X}_{sp}$ ingresa al controlador proporcional de posición, donde se compara con la estimación de la posición actual $\mathbf{X}$ y se obtiene cuál debe ser la velocidad $\mathbf{V}_{sp}$ para minimizar dicha diferencia. Esa velocidad *setpoint* obtenida, ingresa al PID de control de velocidad, en el que se calcula el error entre la velocidad actual y la deseada, y se computa la aceleración necesaria $\mathbf{A}_{sp}$ para anular dicho error.

Estos dos controladores trabajan con magnitudes que se miden en un marco de referencia inercial y se actualizan a una tasa de muestreo de 50 Hz. Para continuar la cascada de controladores hacia el control de actitud, es necesario cambiar el marco de referencia por uno solidario a la aeronave, como fue explicado en la Sección 3.1.

Con este fin, la aceleración deseada que se obtuvo del bloque anterior ingresa junto con el *yaw setpoint* ψ_{sp} a un bloque que logra expresar la actitud deseada mediante un cuaternión q_{sp}. Dicho cuaternión es controlado por un bloque proporcional, que obtiene como acción de control el vector de velocidades angulares deseadas $\mathbf{\Omega}_{sp}$.

Por último, la velocidad angular ingresa a otro bloque PID, generando un vector de acciones de control que se procesan en un *Mixer*, del que se obtiene finalmente el vector de empujes deseados para cada motor $\mathbf{T}_{sp}$. El bloque *Mixer* calcula las fuerzas de empuje haciendo uso de un modelo matemático muy similar al de la Sección 7.1.1, pero en tiempo discreto [15].

Como se aprecia en el esquema de la Figura 7.2, la frecuencia de funcionamiento de los controladores de actitud es bastante mayor a la de los controladores de posición. Esto se debe principalmente a dos factores; en primer lugar debe tenerse en cuenta que la estimación de posición mediante filtros EKF conlleva un cierto costo computacional, mientras que la información de las brújulas simplemente se promedia (si existe redundancia), ponderando cada magnetómetro según su calibración, lo que involucra muchas menos demoras. En segundo lugar, el control de actitud es crítico para mantener la estabilidad de la nave y evitar vibraciones, por lo que amerita una tasa de refresco superior.

Cabe aclarar que algunos de los bloques de control pueden no ser utilizados en algunos modos de vuelo, por lo que existe una variable *Enable* que los habilita o

deshabilita según resuelva el módulo de modos de vuelo [23]. Por ejemplo el bloque controlador de posición no es utilizado en los modos de vuelo manuales.

En ese mismo sentido, todas las variables *setpoints*, pasan por una función saturación antes de ingresar al siguiente bloque, de forma de mantener los valores obtenidos dentro de un rango permitido. La fijación de dichos valores de saturación también es controlada por el módulo de modos de vuelo [24].

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 8

Pruebas Realizadas

En el transcurso del proyecto, se realizaron varias pruebas de diferentes tipos en cada instancia. Durante el desarrollo de software, fue necesario probar el avance de cada nodo en simulaciones de Gazebo, intentando encontrar errores, seleccionando los parámetros de control y depurando el código hasta obtener una performance aceptable.

Una vez terminada la construcción de los cuadricópteros, se realizaron pruebas de conexión y comunicación entre los integrantes del sistema, las que se efectuaron sin hélices y bajo techo. Por último, habiendo obtenido resultados satisfactorios en las simulaciones y las pruebas *indoor*, se hicieron pruebas de campo para comprobar que era posible implementar la dinámica de seguimiento sobre los cuadricópteros construidos.

En este capítulo se describirán las pruebas realizadas en todos los niveles, y se compartirán algunos de los resultados intermedios obtenidos. Los resultados finales serán analizados con mayor detalle en el Capítulo 9.

Es importante aclarar que las gráficas de trayectorias que se presenten de aquí en adelante (tanto en el plano como en el espacio) cuentan con una representación temporal dada por la intensidad del color en cada punto graficado; siendo el color más oscuro correspondiente al inicio de la trayectoria y el más claro asociado al final de la misma.

8.1. Simulaciones

8.1.1. Líder

Ya realizados los tutoriales que se mencionaron en la Sección 4.2, se contaba con un nodo que ejecutaba la simulación en Gazebo de un dron cuadricóptero con firmware PX4 capaz de armarse, realizar un despegue y un aterrizaje de manera autónoma. A partir de ese ejemplo, se crearon los nodos que describen las trayectorias del líder, tales como círculos, ochos o rectas (nombrados en el Capítulo 4), que fueron simulados en Gazebo durante el proceso de desarrollo, hasta arribar a los resultados finales.

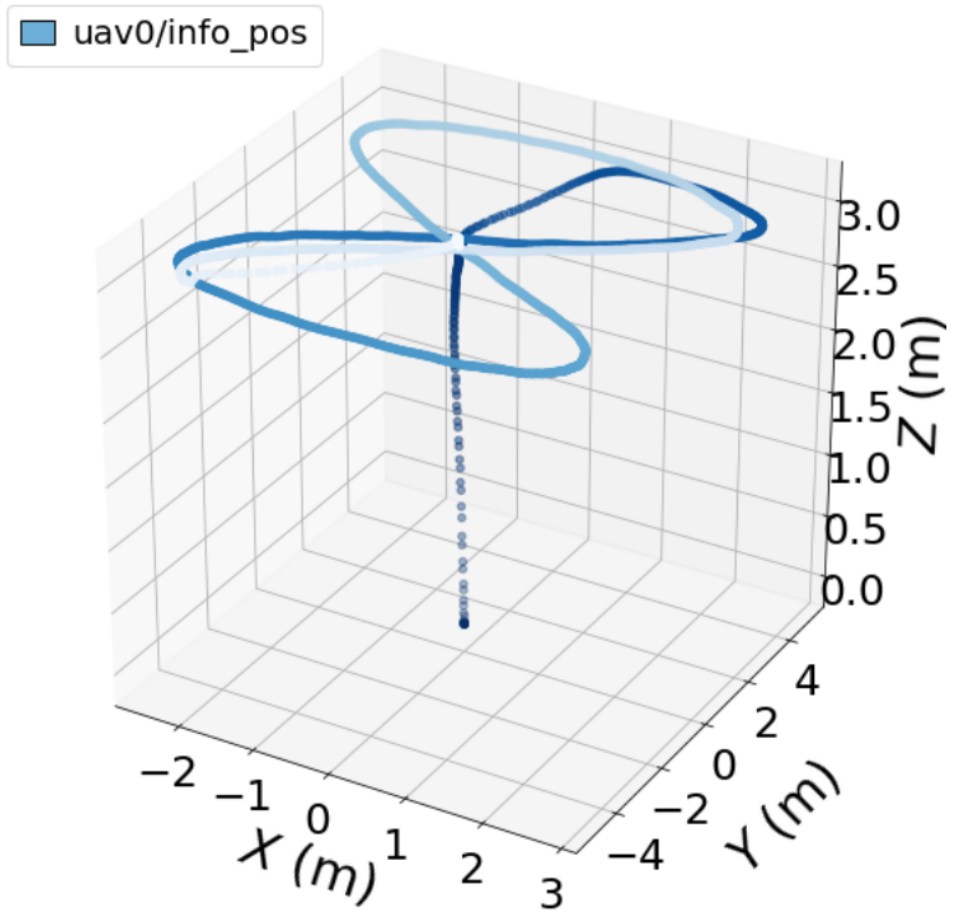


Figura 8.1: Gráfica de posición de un dron, realizada con los datos obtenidos a partir de una simulación Gazebo del nodo *ocho.cpp*.

Fue durante estas simulaciones iniciales, que se comenzaron a recabar los primeros *rosbag* de datos, sumamente útiles para analizar el funcionamiento del software desarrollado, obteniendo resultados como los que se observan en la Figura 8.1 para un líder que despega, describe una trayectoria similar a un número ocho a 3 m de altura y finalmente aterriza sobre la posición inicial.

Las simulaciones de trayectorias del líder también fueron útiles para probar y conocer con mayor profundidad el software de control terrestre QGroundControl, dado que el mismo puede conectarse al puerto local donde *mavros* publica la información de telemetría. De esa forma el vuelo en Gazebo puede ser monitoreado como si fuera una aeronave real, lo que se observa en la Figura 8.2, obtenida durante la misma simulación de vuelo que la Figura 8.1.

Hasta aquí todos los nodos programados hicieron uso de la posición local reportada por *mavros* para calcular los *setpoints*. Como ya fue explicado, se debieron crear funciones que logren realizar estas acciones en un marco de referencia único para todo el enjambre.

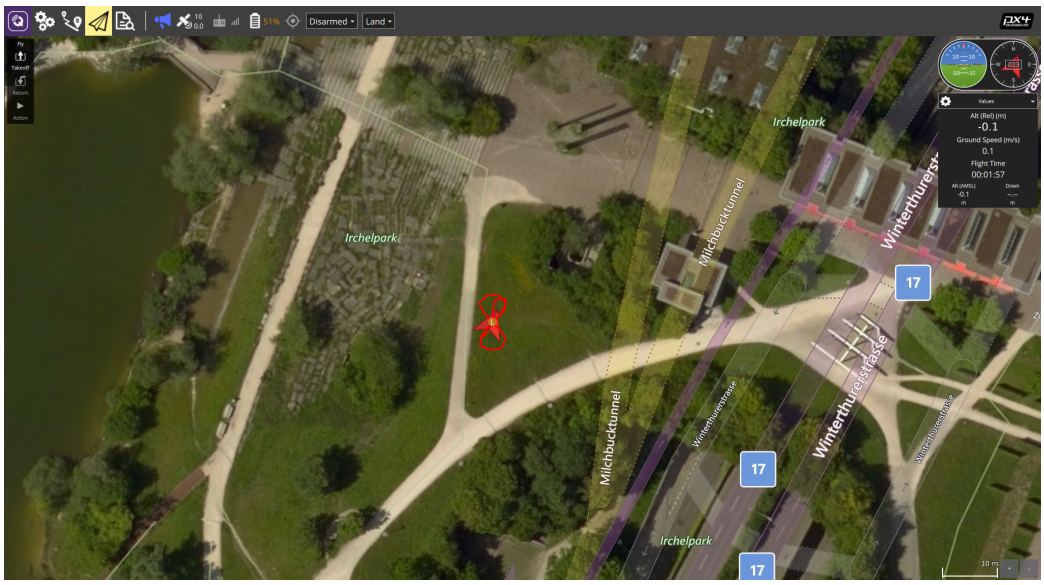


Figura 8.2: Mapa de vuelo en QGroundControl, conectado a MAVROS durante la simulación en Gazebo del nodo *ocho.cpp*.

Fue sencillo comprobar el correcto comportamiento de las funciones de cambio de coordenadas desarrolladas, dado que si el único dron presente es el líder, la posición reportada por el cambio de coordenadas realizado debería coincidir con la posición en el marco local informada por MAVROS, pues en ambos casos el origen del sistema es el punto inicial de despegue. La Figura 8.3 es un gráfico obtenido de una misma simulación, con el fin de comparar las posiciones reportadas en ambos marcos, y comprobar que el error producido por la aproximación de Taylor puede considerarse despreciable. Esta afirmación será estudiada con mayor detalle en el Cápitulo 9, tanto para simulaciones como para las pruebas de campo con aeronaves reales.

8.1.2. Seguidores

Un tiempo después, habiendo definido las dos estrategias de control a aplicar explicadas en el Capítulo 3, se comenzaron a simular los nodos de seguimiento con uno y luego dos drones seguidores. En ambas estrategias de control existen un grupo de constantes que definen el comportamiento de los integrantes del enjambre, algunas afectan la agresividad en la evasión de colisiones, otras las distancias de seguridad o los margenes de error aceptados y más. Por lo tanto, durante esta parte del proyecto, la simulación por computadora fue crucial para lograr determinar las combinaciones de parámetros que lleven a los mejores resultados.

Esta "calibración" de constantes se llevó a cabo simulando repetidas veces la dinámica *Follow the leader*, en principio sobre trayectorias circulares, siendo éste el principal objetivo del proyecto. De esta manera, se buscaron los parámetros que permitieran un mejor desempeño de seguimiento y evasión para cada estrategia, pudiéndose ver a simple vista que los mejores resultados eran obtenidos con la

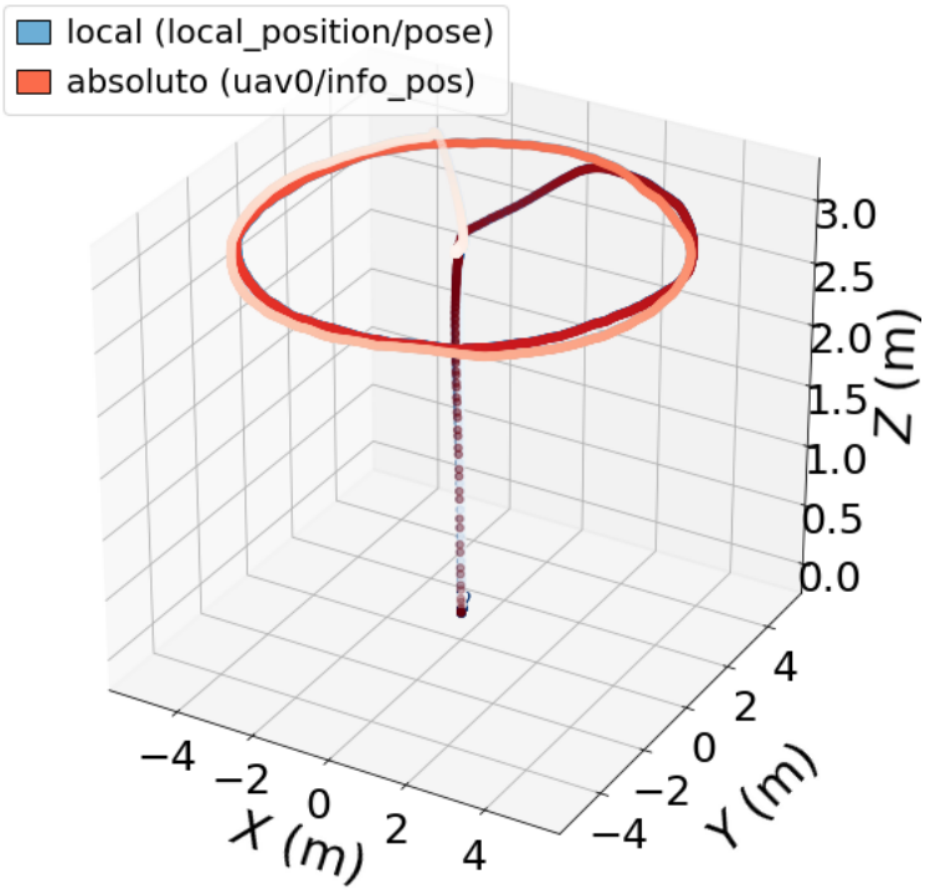


Figura 8.3: Posición reportada por un dron en una simulación, en el marco de referencia local de MAVROS, y en el marco de referencia global del enjambre durante un mismo vuelo. En esta simulación se eligió una altura de vuelo de 3 m, y un radio de 5 m.

estrategia mixta.

Dentro de esa estrategia, se buscó perfeccionar las constantes elegidas, probando con condiciones más complejas, como mayor velocidad de navegación o menor distancia y ángulo entre los componentes. En las Figuras 8.4 se nota un claro ejemplo de lo mencionado, dado que con un ángulo de formación $\theta = 60°$, a una velocidad de 1 m/s se obtienen mejores resultados que con $\theta = 30°$ a la mitad de la velocidad (manteniendo todas las mismas constantes de control).

En una segunda instancia, habiendo obtenido resultados aceptables durante dichas simulaciones, se probaron trayectorias más exigentes, como el cambio de sentido, donde el líder gira repentinamente 180° sobre su propio eje y avanza a gran velocidad sobre la misma recta, requiriendo una rápida evasión por parte de los seguidores como puede verse en la Figura 8.5.

Como se aprecia en los extremos de la trayectoria, cuando el líder gira y avanza en el sentido contrario, los seguidores se ven obligados a evadirlo, girar, y cruzarse con el otro seguidor evitando una colisión, todo en muy poco tiempo hasta re-

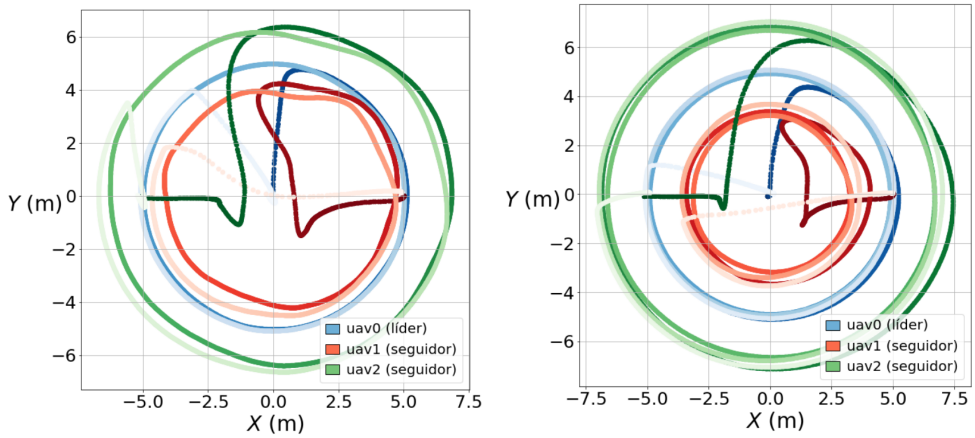


Figura 8.4: Posiciones reportadas por los drones, en el marco de referencia global del enjambre durante un mismo vuelo, con el líder realizando una trayectoria circular de $5\ m$ de radio. En (a), el líder se mueve a $0{,}5\ m/s$ y se eligió $\theta = 30°$, y en (b), el líder se mueve a $1\ m/s$ y se eligió $\theta = 60°$.

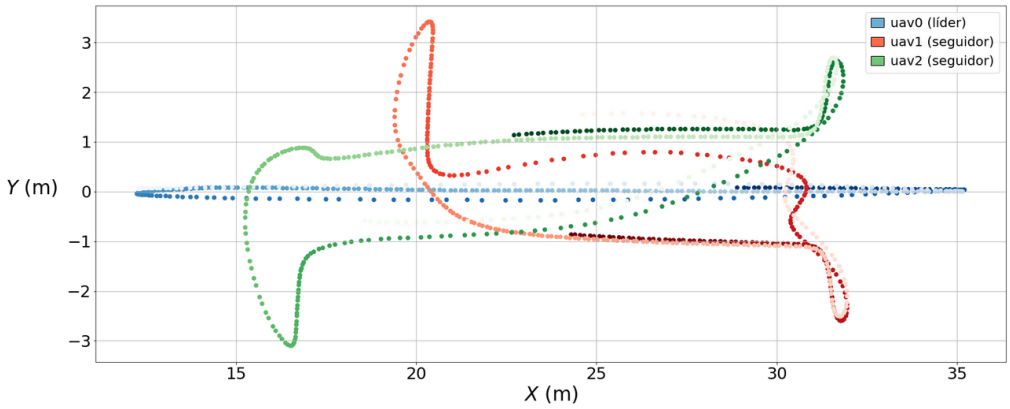


Figura 8.5: Gráfica de posición del líder con dos seguidores, realizada con los datos obtenidos a partir de una simulación Gazebo de los nodos *cambio_sentido.cpp* y *follow.cpp*, con la estrategia de control mixta. El dron líder se desplaza a una velocidad máxima de $3\ m/s$, lo que resulta en un exigente testeo de *Collision Avoidance*.

cuperar la formación preestablecida del lado opuesto al que se encontraban. La exigencia de esta dinámica, aumenta cuanto mayor es la velocidad del líder y menores son la distancia y el ángulo dados para la formación de seguimiento.

Una vez terminado el ajuste fino de los parámetros, y siendo satisfactorios los resultados obtenidos en las simulaciones de la dinámica de seguimiento sobre todas las trayectorias con la estrategia mixta de control, se procedió a implementar el sistema sobre UAVs reales.

8.2. Fake GPS

Las pruebas en Gazebo simulan el vuelo de aeronaves equipadas con una *FCU* con firmware PX4, que reciben una señal simulada de GPS provista por la misma controladora. Esta señal simulada es conocida como *fake GPS* (del inglés, GPS falso) y puede también ser aplicada sobre una controladora de vuelo real, para realizar pruebas en lugares donde no es posible la localización satelital.

Como paso previo a las pruebas de campo, se realizaron pruebas en espacios cerrados, sin hélices y utilizando GPS falso. La habilitación del *fake GPS* puede realizarse mediante el software QGroundControl, en la ventana "Analyze", seleccionando "MAVLink Console" y ejecutando:

```
$ gps stop
$ gps start -f
```

Las pruebas *indoor* fueron particularmente útiles para testear y solucionar problemas relacionados a la conexión de la red, la ejecución del ROS Máster y la comunicación MAVLink entre la controladora Pixhawk y el NUC. En este sentido se efectuaron pruebas variadas hasta lograr establecer la red Ad-Hoc, levantar el *roscore* compartido, ejecutar diferentes nodos en cada componente de la flota y monitorear el correcto intercambio de información entre ellos.

A pesar de haber logrado lo anterior con éxito, se encontraron algunas dificultades de conexión y estabilidad en la red, que fueron asociados en un principio a una mala sujeción de las antenas a las tarjetas de red presentes en cada aeronave, lo que fue en gran medida solucionado.

8.3. Pruebas de Campo

Después de haber superado con éxito las pruebas de funcionamiento del sistema en un entorno simulado, haber comprobado que los cuadricópteros construidos eran capaces de volar manualmente, y que era posible establecer la comunicación entre los drones y la computadora como había sido planificado, se procedió a implementar el sistema en pruebas de campo con los drones reales. Para esto se volaron las aeronaves en una zona amplia, abierta y segura, monitorizando el estado actual desde la estación terrestre y con los radiocontroles listos para tomar el control en caso de necesidad, como se muestran en las Figuras 8.6 y 8.7.

8.3.1. Líder

El orden de los experimentos desarrollados durante las pruebas de campo fue similar al de las simulaciones. En primer lugar se comprobó el funcionamiento de un armado, despegue y aterrizaje autónomo sobre cada una de las aeronaves y luego esa misma dinámica fue progresivamente realizada con dos y tres componentes en simultáneo.

En una segunda instancia, se probaron trayectorias circulares en un sólo dron, basadas en coordenadas locales, para corroborar que la estabilidad y precisión de



Figura 8.6: Drones utilizados durante las pruebas de campo.

la navegación autónoma de los UAVs era similar a la observada en las simulaciones. Antes de poder comenzar las pruebas de campo con varios integrantes del enjambre a la vez, fue necesario comprobar el funcionamiento del cambio de marco de referencia explicado en la Sección 3.2, de forma de poder establecer la posición de todos los drones con respecto al mismo sistema de coordenadas.

Con este fin, se repitió la prueba de trayectoria circular, pero esta vez dirigiendo al dron hacia posiciones calculadas en el marco de referencia general de la flota. A partir de este punto, se contó con todas las herramientas necesarias comprobadas para implementar una dinámica de seguimiento al líder.

8.3.2. Seguidores

La implementación de los seguidores en las pruebas de campo se vio afectada en varias ocasiones debido a fallas en la conexión de la red Ad-Hoc establecida. Esta falla ocurría principalmente en el momento que los drones despegaban y se alejaban de la estación terrestre, provocando que la conexión con la misma se debilitara ocasionando pérdida de paquetes de datos. Los drones deben reportar constantemente su ubicación y velocidad, por lo que la pérdida de esta información (aún por breves instantes de tiempo) puede causar un cálculo erróneo del próximo *setpoint*, lo que en algunas circunstancias podría generar situaciones de riesgo.

Para solucionar esta problemática, se optó por sustituir la antena del *dongle* utilizado, por una de mayor potencia y por ende mayor alcance. Si bien este cambio



Figura 8.7: Prueba de campo típica, con los tres cuadricópteros, la estación terrestre y los radiocontroles.

mejoró la estabilidad de la red, en algunas pruebas, ya más esporádicas, se observó alguna pérdida de conexión. Estudiando el problema con mayor profundidad, se observó que la señal que se emite desde las antenas tiene forma de toroide, por lo que se debía posicionar la antena de tal forma que este toroide alcanzara todos los drones. Se decidió colocar la computadora en el centro de la trayectoria circular realizada por el enjambre, y de esta forma se lograron disminuir las pérdidas de conexión.

Otra problemática que surgió durante las pruebas de campo, fue la presencia de errores en la medición de alturas de vuelo de los drones. En un principio se intentó medir la altura tomando al nivel del mar como referencia común, dada por el GPS. Esto ocasionó problemas debido a que la estimación del nivel del mar tiene una gran incertidumbre, por lo que los distintos drones no se ponían de acuerdo en la altura de referencia (es decir, aunque las aeronaves se encuentren físicamente a una misma altura, los valores que cada uno mide con respecto al nivel del mar pueden diferir en el orden de varios metros).

Este percance implicó que para tomar una referencia común de altura para los distintos drones, se debió hacer el supuesto de que los mismos comenzaban a igual altura (lo cual fue cierto en las condiciones trabajadas en la práctica). Esto permitió definir un marco común a partir de la variación de altura de cada dron con respecto a su altura inicial (la altura inicial se toma como referencia 0 en la

vertical). Esto logró una mayor precisión que cuando se usó la altura obtenida a partir de la referencia común definida por el nivel del mar, pues los drones son capaces de medir variaciones de altura con gran exactitud, haciendo uso de *sensor fusion*.

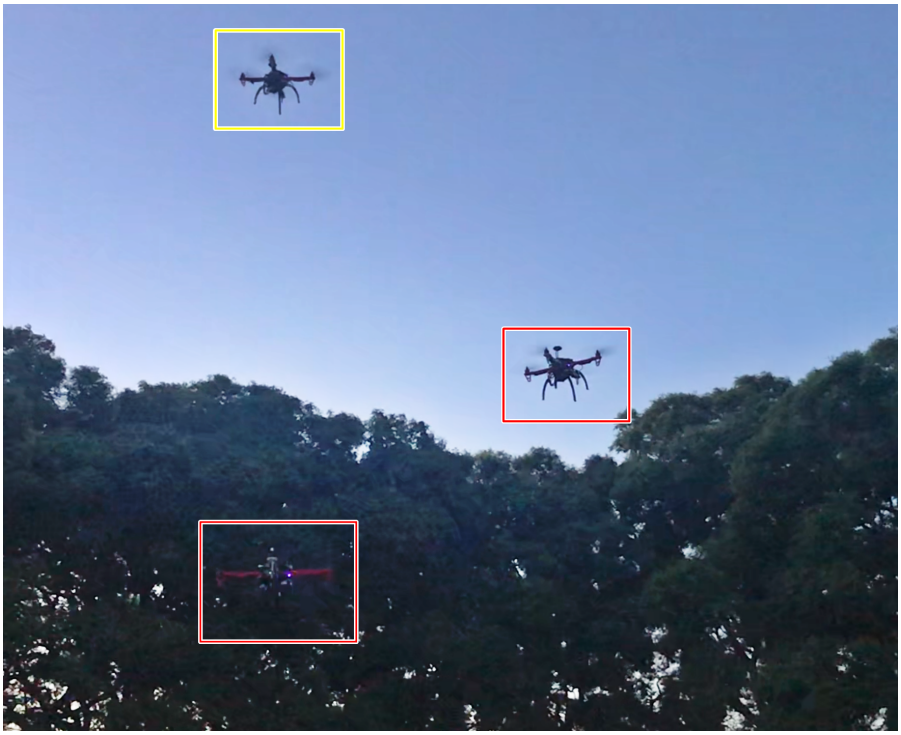
De todas maneras la altura sobre el nivel del mar fue útil para el sistema de coordenadas común entre los drones, pues dicho valor afecta (aunque en pequeña escala) a las distancias en el plano horizontal, como se ve en la ecuación (B.2). Es decir, para medir las distancias en el plano xy, todos los drones hicieron uso de la altura AMSL medida al inicio por el líder como valor de h_0, mientras que en el eje z cada dron midió su variación de altura respecto de su propia altura inicial.

Luego de superadas estas dificultades, se logró realizar con éxito la dinámica de seguimiento al líder, en primer lugar con dos drones (un líder y un seguidor) y posteriormente con los tres integrantes del enjambre, observando resultados satisfactorios, que serán analizados durante el Capítulo 9. Con esto se dio por finalizada la etapa de pruebas del proyecto.

Cabe aclarar, que todas las pruebas de vuelo se realizaron manteniendo una diferencia de altura de seguridad entre los componentes de la flota, de forma de poder comprobar el correcto desempeño en el plano xy de las estrategias de seguimiento y evasión, sin correr riesgos de colisión. En la Figura 8.8 se observan los tres drones volando en formación durante una dinámica *Follow the leader*, manteniendo una distancia vertical de 2 m entre sí.



(a)



(b)

Figura 8.8: Fotografías tomadas durante el vuelo de la flota, implementando una formación de seguimiento al líder, con una diferencia de altura de 2 m por seguridad. El líder está resaltado en amarillo y los dos seguidores recuadrados en rojo.

Capítulo 9

Análisis de los Resultados Finales

Hasta ahora se han descrito las pruebas realizadas tanto en simulaciones como en las pruebas de campo. En estas instancias, se puede ver que existe una diferencia importante entre realizar simulaciones por computadora y probar las mismas estrategias de control sobre los drones reales, dado que durante las pruebas de campo se debieron resolver problemas prácticos que no se habían presentado durante las simulaciones. Con las mejoras realizadas en el transcurso de las pruebas de campo, que se explicaron en el capítulo anterior, se obtuvieron los resultados que se detallarán en este capítulo.

9.1. Simulaciones

En primer lugar se realizó una comparativa entre el marco de referencia local de MAVROS y el marco de referencia global del enjambre, calculado utilizando aproximaciones de Taylor. Para ello, se calculó y graficó la diferencia entre las posiciones reportadas por el dron líder en cada uno de ellos, obteniendo los resultados que se ilustran en la Figura 9.1.

En la Figura 9.1 se aprecia que la distancia promedio reportada entre ambos marcos en la simulación es de aproximadamente 2 cm. Esta diferencia es muy pequeña en relación a las distancias manejadas por las aeronaves, por lo que se justifican las aproximaciones realizadas en el Apéndice B.

A modo de ejemplo, en una trayectoria circular de 3 m de radio, la diferencia entre los marcos produciría un error menor al 1 % en el plano horizontal, mientras que el error producido por la medición del módulo GPS (presente en ambos marcos de coordenadas) puede llegar al orden de un metro.

Luego, se procedió a analizar el desempeño de la estrategia de seguimiento, para lo que se midieron y graficaron las distancias en el plano horizontal entre los integrantes del enjambre en diferentes condiciones. En la Figura 9.2 (a) se puede observar la distancia entre cada seguidor y el líder, cuando este último se mueve a 0,5 m/s implementando una trayectoria circular y con $\theta = \pm 30°$. Al comienzo de la dinámica, los seguidores se mueven hasta posicionarse a cierta distancia del líder, se mantienen estables a esa distancia por un intervalo de tiempo, y en última

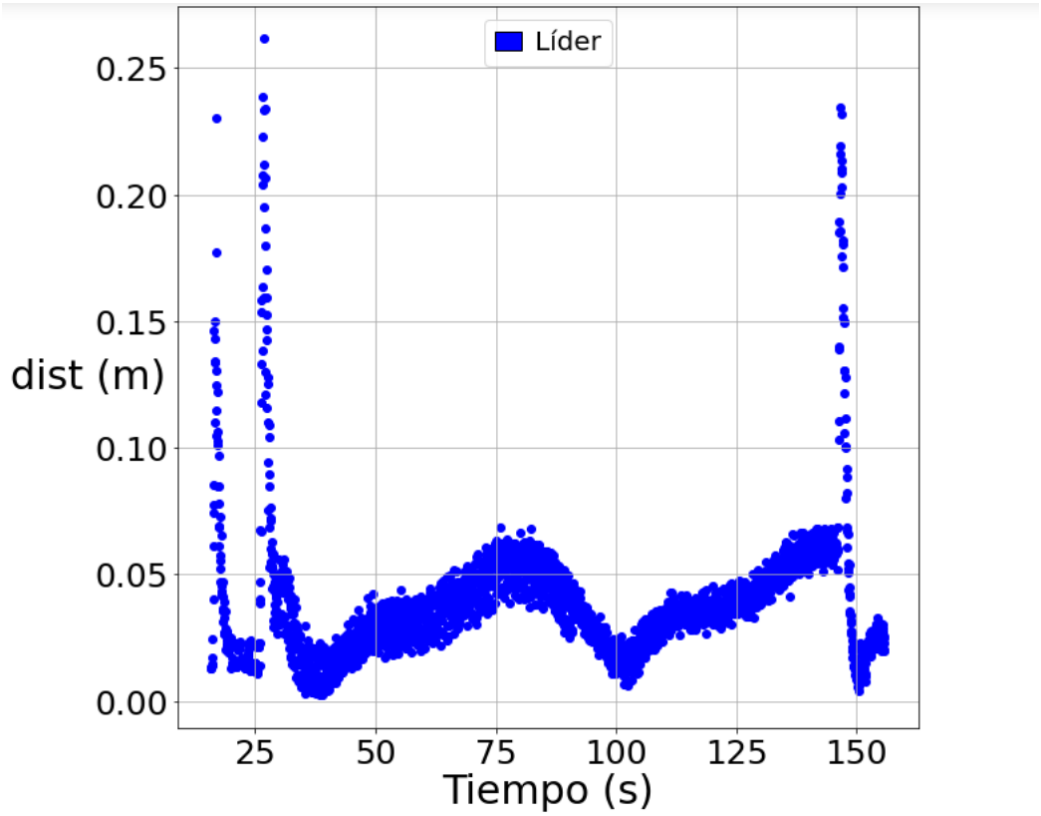


Figura 9.1: Distancia entre las posiciones reportadas por el dron líder en los distintos marcos de referencia (local de MAVROS, y en el marco de referencia global del enjambre) durante las pruebas en simulaciones.

instancia se alejan del líder retornando a sus respectivos orígenes.

La distancia horizontal deseada es de 2 m, pero en la figura observamos que la distancia promedio es de 2,5 m, por lo que se presenta un desfasaje de aproximadamente medio metro. Cuanto menor sea la velocidad del líder, será más fácil para los seguidores acercarse a la distancia de 2 m, este comportamiento se puede comprobar en la Figura 9.2 (b), donde se observa una distancia promedio de 2,3 m. Luego, en la Figura 9.2 (c) el líder despega y luego se queda quieto en una posición estacionaria, y en este caso en cuanto al desempeño de los seguidores se observa que, efectivamente la distancia de los seguidores al líder es de 2 m. Este gráfico es útil para verificar que si se dan condiciones sencillas de seguimiento, los drones son capaces de seguir al líder a la distancia programada sin error.

En la Figura 9.2 (d), se observa qué sucede cuando se deja el parámetro k_a fijo (en vez de usar el k_a dinámico de la expresión (3.4)). En este caso se utilizó $k_a = 1$ y se observa que la distancia promedio se aproxima a 2,8 m. Comparando esta figura con 9.2 (a), se comprueba que utilizar k_a variable mejora claramente el desempeño de los seguidores, tal como se esperaba.

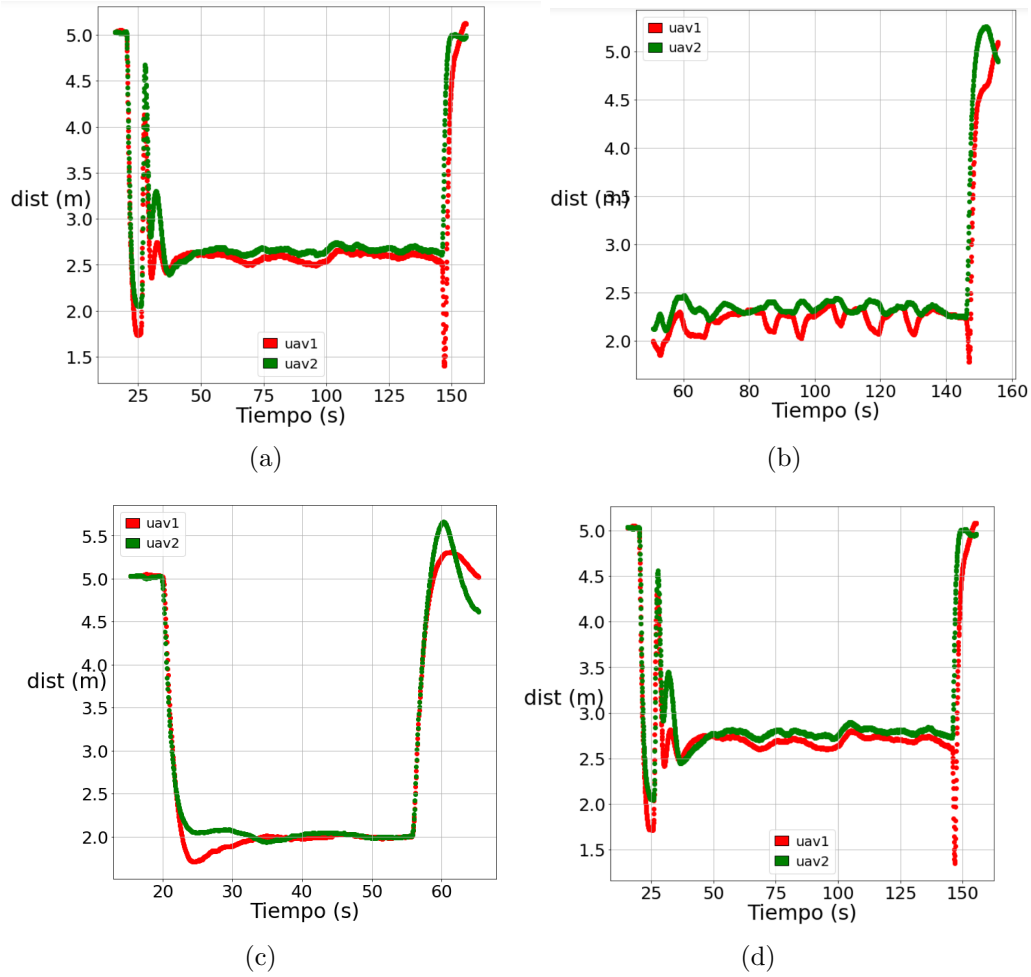


Figura 9.2: Distancias horizontales obtenidas desde los drones seguidores al líder en simulaciones. En (a) se usó el K_a dado por la expresión (3.4), con el líder desplazándose en trayectoria circular a $0{,}5\ m/s$. En (b) se usó el K_a dado por la expresión (3.4), con el líder desplazándose en trayectoria circular a $0{,}2\ m/s$. En (c) se usó el K_a dado por la expresión (3.4) con el líder quieto. En (d) se usó $K_a = 1$ constante, con el líder desplazándose en trayectoria circular a $0{,}5\ m/s$.

Luego de analizar cómo afectan algunos parámetros, como la velocidad del líder o la variable k_a al desempeño de los seguidores en su tarea de seguir al líder a cierta distancia, se verá un enfoque más global del comportamiento del enjambre. La Figura 9.3 se corresponde con la representación tridimensional de la Figura 8.4 (a), donde se observa la trayectoria realizada por el líder y dos seguidores. En este caso, se programó al líder para que ejecutara el nodo circle.cpp, donde el radio del círculo se fijó en 5 m, la velocidad del líder en $0{,}5\ m/s$ y el ángulo $\theta = \pm 30°$. Como ya se ha mencionado previamente, la trayectoria cambia de color de oscuro a claro a medida que transcurre el tiempo. Se puede observar que en la simulación, el comportamiento del enjambre es correcto y estable. Los drones seguidores se

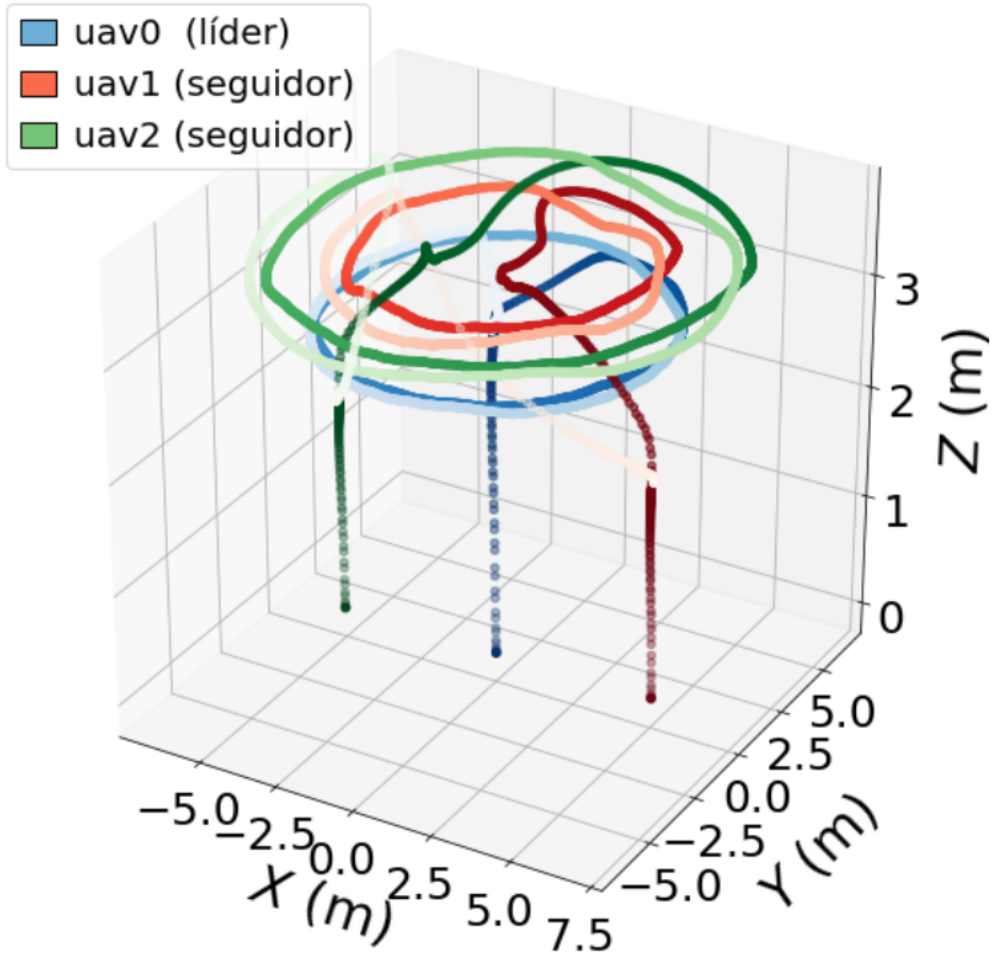


Figura 9.3: Posiciones reportadas por los drones en una simulación, en el marco de referencia global del enjambre durante un mismo vuelo, con el líder moviéndose a $0{,}5\ /s$. En esta simulación se usaron dos seguidores, y el parámetro $\theta = \pm 30°$. El radio elegido para el líder fue de $5\ m$, y las alturas de vuelo fueron de $3\ m$ para el líder, mientras que los seguidores debían mantenerse $0{,}5\ m$ por encima del líder.

mantienen a cierta distancia estable del líder, y la trayectoria ejecutada por este último es efectivamente circular. Se observa además, que los drones despegan y aterrizan exactamente en la misma posición. Con este gráfico se corrobora que los algoritmos implementados funcionan correctamente en las simulaciones.

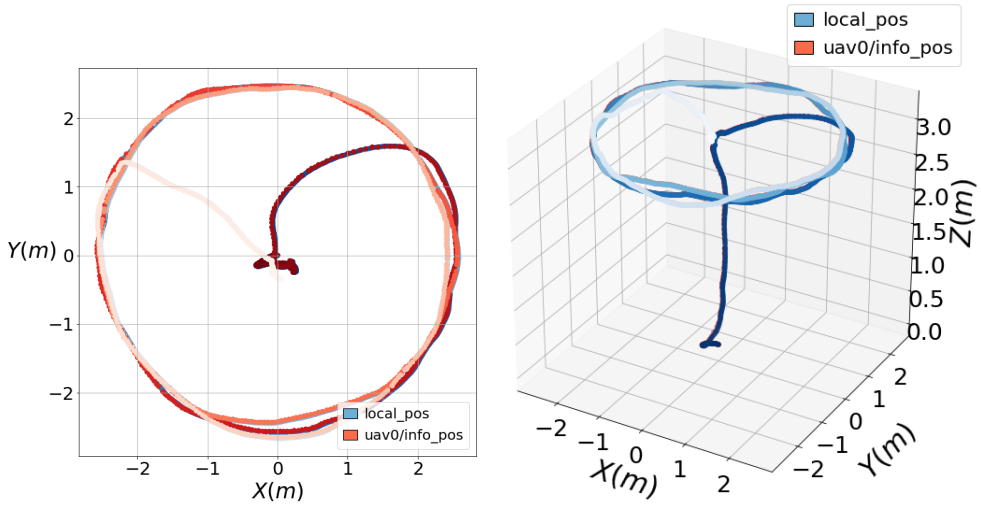


Figura 9.4: Posición reportada por el dron líder en el marco local de MAVROS y en el marco global obtenido a partir del Taylor de primer orden durante una prueba de campo.

9.2. Pruebas de Campo

Del mismo modo que se hizo en las simulaciones, se pudo verificar que el sistema de coordenadas obtenido a partir de las aproximaciones al Taylor de primer orden de la parametrización del elipsoide que modela la Tierra (calculado en el apéndice B), logró una gran precisión. Se hizo una prueba de campo en la que únicamente voló el dron líder, y se compararon las posiciones reportadas por el mismo en el marco global proveniente del Taylor, con las posiciones reportadas en tiempos cercanos en el marco local de MAVROS. En la Figura 9.4 puede verse que las coordenadas reportadas en ambos marcos están prácticamente solapadas.

Observando más detalladamente, la distancia entre un marco y otro se mantuvo con valores pequeños, del orden de unos pocos centímetros, como se ve en la Figura 9.5. Esta diferencia puede ser causada tanto por errores de aproximación debidos a que se recurrió a un Taylor de primer orden, o también a las aproximaciones usadas en el cálculo de los coeficientes de este desarrollo de Taylor. También puede deberse a que las posiciones que se están comparando fueron reportadas en tiempos ligeramente distintos, y existen incertidumbres en los instrumentos con los que el dron determina su posición. A pesar de todas estas posibles causas de error, la distancia promedio entre las posiciones reportadas en estos marcos fue de 4,7 cm, la cual es bastante pequeña y entonces corrobora la validez de las aproximaciones realizadas en el Apéndice B.

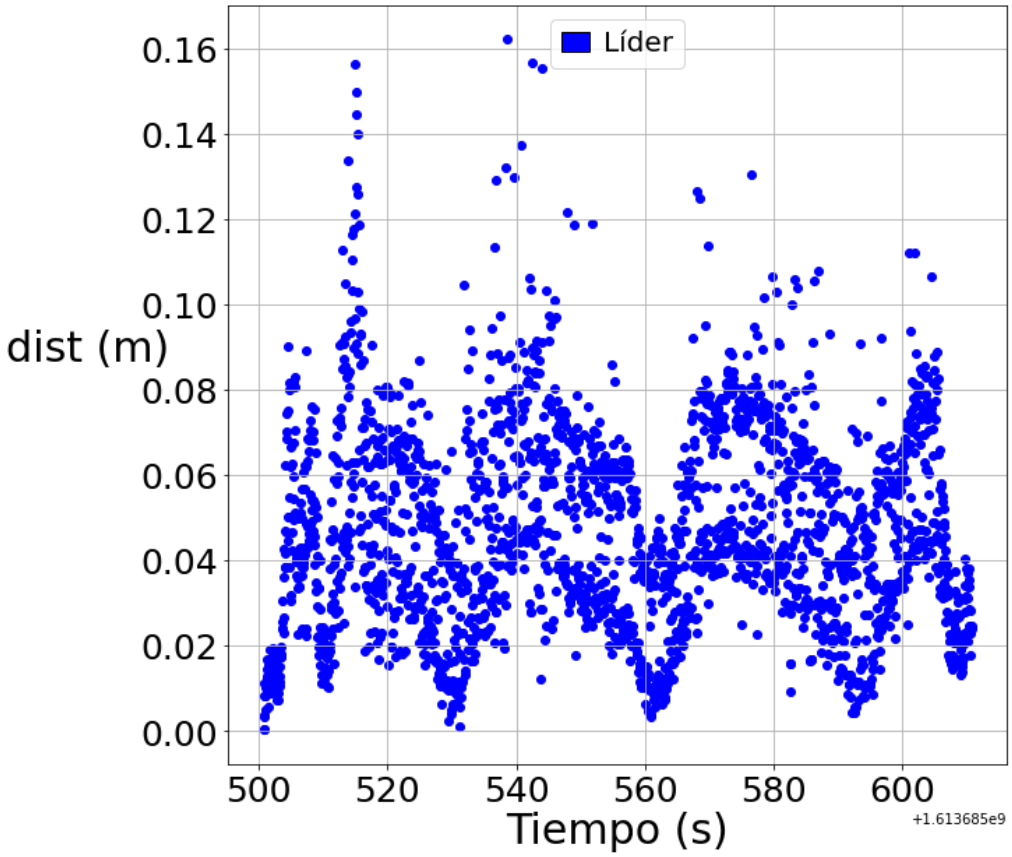
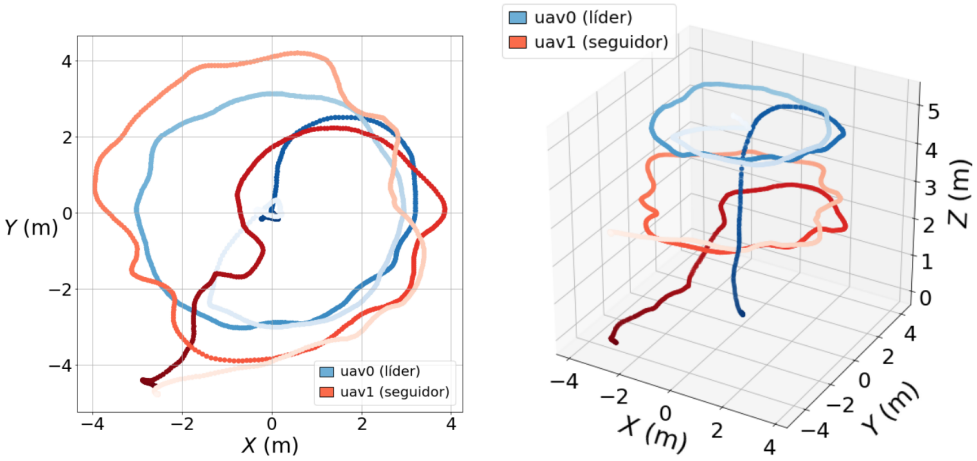


Figura 9.5: Distancia entre las coordenadas reportadas por el líder dadas en el marco local de MAVROS y en el sistema global obtenido a partir del Taylor de primer orden durante una prueba de campo.

En cuanto a las pruebas de campo con varios UAVs, primero se observarán los resultados obtenidos durante el vuelo de dos drones (un líder ejecutando un círculo de 3 m de radio, a 0,5 m/s de velocidad y una altura de 5 m, y un seguidor con parámetro $\theta = 0°$ y una altura de vuelo de 3 m). Los resultados de esta experiencia se presentan en la Figura 9.6. En la Sub-figura 9.6 (a) se grafican los resultados en el plano xy, mientras que en la Sub-figura 9.6 (b) se realiza un gráfico $3D$, donde se puede verificar que las alturas de vuelo son las deseadas. En ambos gráficos, se puede corroborar que los drones despegaron y aterrizaron en el mismo lugar, cumpliendo con el comportamiento esperado al comienzo y al final de la trayectoria.

Durante la misma, el líder describe efectivamente un círculo de radio igual a 3 m, mientras que el seguidor lo sigue de buena manera, aunque manteniendo en el plano horizontal una distancia de entre 2,5 metros y 3 metros con el líder, como se ve en la Figura 9.7.



(a) (b)

Figura 9.6: Posiciones reportadas por los drones en una prueba de campo durante un mismo vuelo, con el líder moviéndose a $0{,}5\ m/s$. En esta prueba se usó solo un seguidor, y el parámetro $\theta = 0°$. El radio elegido para el líder fue de $3\ m$, y las alturas de vuelo fueron de $5\ m$ para el líder, mientras que el seguidor debía mantenerse $2\ m$ por debajo del líder.

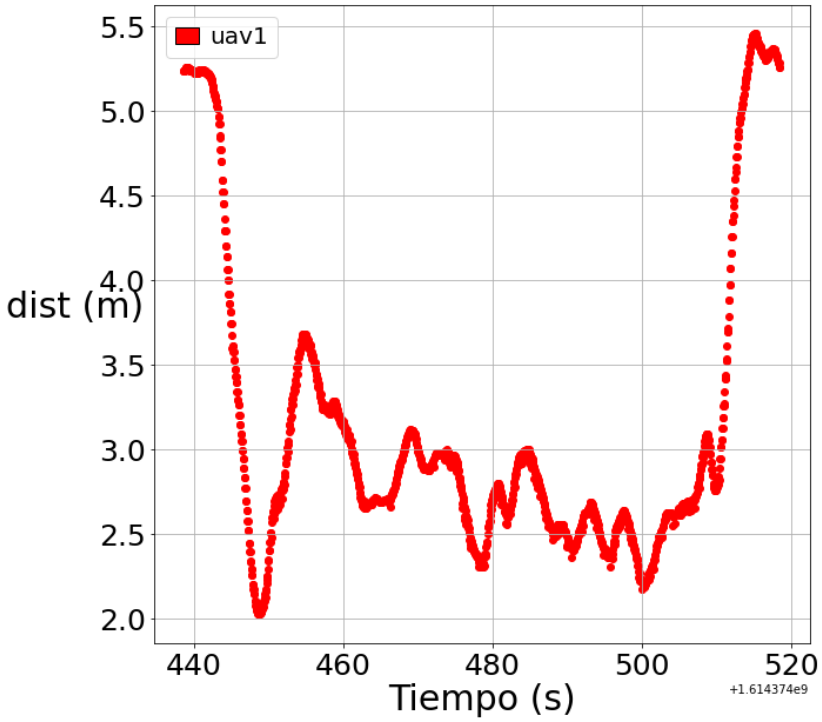
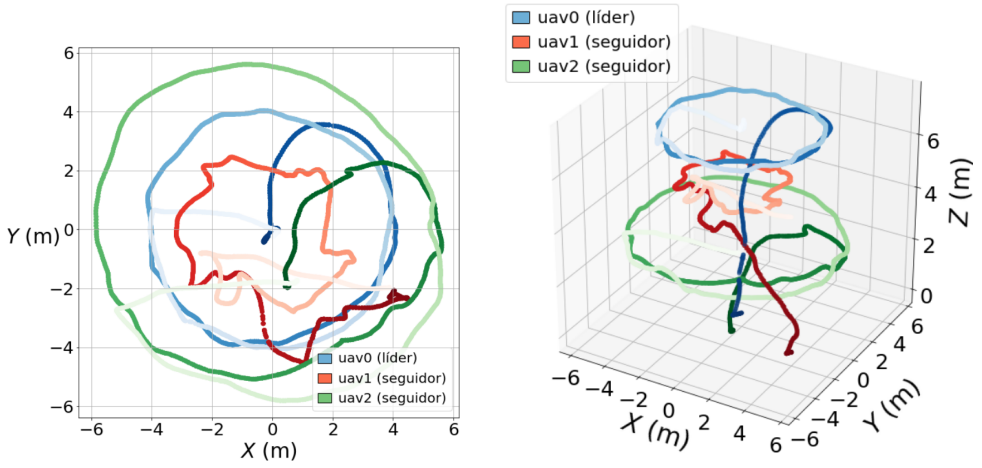


Figura 9.7: Distancia horizontal obtenida desde el dron seguidor al líder en una prueba de campo con un único seguidor, con el líder moviéndose a $0{,}5\ m/s$ y haciendo un círculo de 3 metros de radio. Se usó en esta prueba $\theta = 0$.



(a) (b)

Figura 9.8: Posiciones reportadas por los drones en una prueba de campo durante un mismo vuelo, con el líder moviéndose a 0,5 m/s. En esta prueba se usaron dos seguidores, y el parámetro $\theta = \pm 60°$. El radio elegido para el líder fue de 4 m, y las alturas de vuelo fueron de 7 m para el líder, mientras que el seguidor $uav1$ debía mantenerse 2 m por debajo del líder y $uav2$ debía mantenerse 4 m pode debajo del líder.

Como se puede ver, el comportamiento de los drones en las pruebas de campo presenta leves oscilaciones, producidas fundamentalmente por el error inherente a la medida del módulo GPS para estimar su posición global, así como los errores que se asocian al tiempo de transmisión de la información en la red. Estos efectos no son tenidos en cuenta durante las simulaciones en Gazebo, y son los que justifican la realización de pruebas de campo para poder valorar el funcionamiento de las acciones de control diseñadas. Sin embargo a pesar de esto, las distancias reportadas en las pruebas sobre drones reales no difieren significativamente de las reportadas durante las simulaciones.

De igual modo, las pruebas de campo con tres UAVs arrojaron resultados satisfactorios. La Figura 9.8 muestra que el líder tuvo buen control de su posición, logrando una trayectoria circular de 4 metros de radio. El dron seguidor $uav2$ se mantuvo del lado externo del círculo siguiendo al líder con una trayectoria bastante regular, mientras que el seguidor $uav1$ que debía mantenerse del lado interno del círculo siguió al líder aunque desplazándose de forma poco regular.

A pesar de esto, el $uav1$ se desempeñó mejor en cuanto a mantener la distancia horizontal deseada de 2 metros respecto del líder, como se ve en la Figura 9.9. Esto puede deberse a que a dicho dron se le exigían velocidades menores que al dron $uav2$, dado que el $uav2$ al encontrarse del lado exterior del círculo debía recorrer mayores distancias que el dron $uav1$ en un mismo intervalo de tiempo. Se recuerda que como al líder se le exigió una velocidad de 0,5 m/s, un radio de 4 metros, y la velocidad angular para todos los drones debería ser la misma (asumiendo además que $uav2$ sigue un radio de entre 5 metros y 6 metros), se tendrá que al dron $uav2$ se le exige una velocidad de entre 0,6 y 0,75 m/s. Este valor es cercano a la

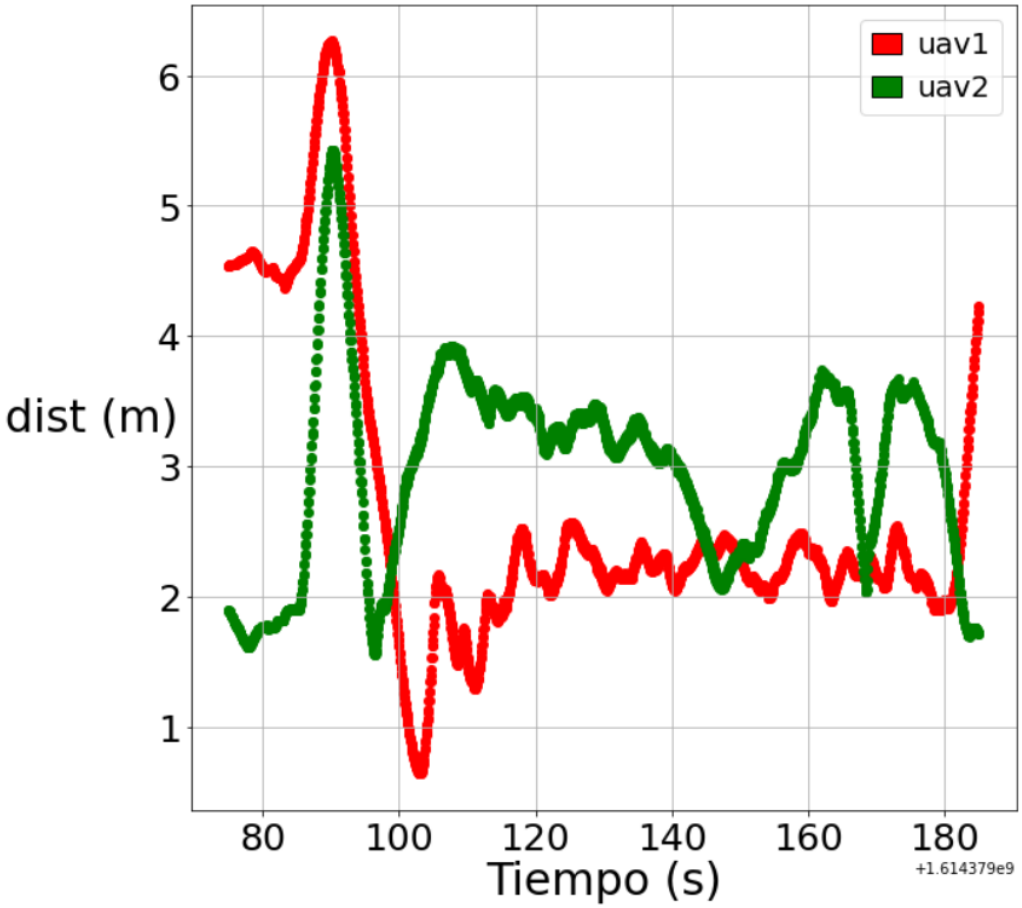


Figura 9.9: Distancias horizontales obtenidas desde los drones seguidores al líder en una prueba de campo con 2 seguidores durante un mismo vuelo, con el líder haciendo un círculo de 4 metros de radio a una velocidad de 0,5 m/s. En esta prueba se usó $\theta_{uav1} = 60°$ y $\theta_{uav2} = -60°$. La dinámica de seguimiento comienza alrededor de $t = 100$ s.

velocidad máxima seteada en *QGroundControl* (de 0,7 m/s), que la controladora de vuelo no permite superar.

Por otra parte, puede observarse que los drones lograron un muy buen control en cuanto a su altura, como se ve en la Figura 9.10, donde el líder mantuvo su altura deseada de 7 metros, y los seguidores mantuvieron sus distancias de 2 y 4 metros por debajo del líder.

En lineas generales, el análisis de los resultados obtenidos nos permite afirmar que el comportamiento del enjambre durante las pruebas de campo sigue la misma línea de lo observado durante las simulaciones en Gazebo, con una disminución en la precisión de los movimientos, originada por aquellos factores que los motores de simulación no pueden contemplar.

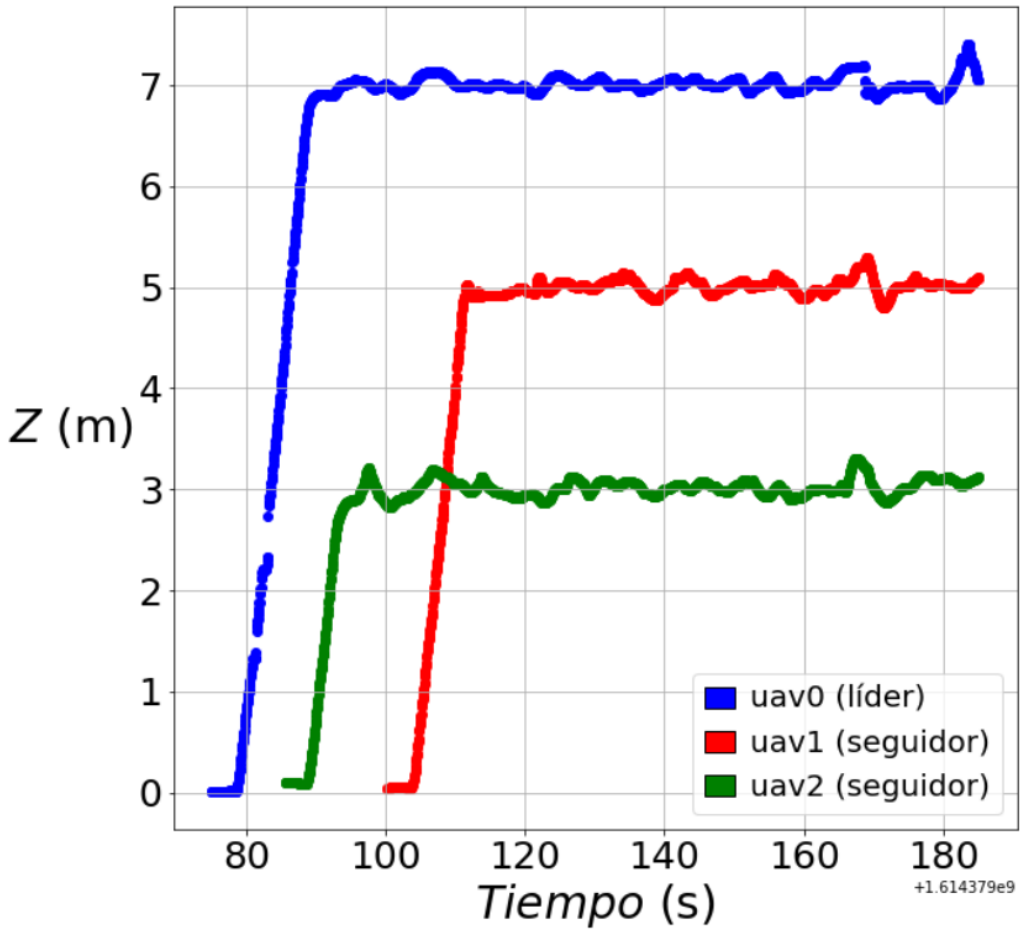


Figura 9.10: Alturas reportadas por los drones en una prueba de campo durante un mismo vuelo, con el líder moviéndose a $0{,}5\ m/s$. En esta prueba se usaron dos seguidores, y el parámetro $\theta = \pm 60°$. El radio elegido para el líder fue de $4\ m$, y las alturas de vuelo fueron de $7\ m$ para el líder, mientras que el seguidor $uav1$ debía mantenerse $2\ m$ por debajo del líder y $uav2$ debía mantenerse $4\ m$ pode debajo del líder.

Capítulo 10

Conclusiones y Trabajo Futuro

10.1. Conclusiones

En general, durante todo el proyecto, la curva de aprendizaje presentó etapas de rápido crecimiento y otras de mesetas. Éstas últimas, durante lapsos breves de tiempo, se dieron luego de haber dominado cierto tema y antes de afrontar el siguiente desafío.

En cuanto al hardware, se logró el correcto armado de los drones, que fueron capaces de cumplir con su cometido de buena manera. El proceso de construcción requirió un tiempo mayor al planificado, debido fundamentalmente a las restricciones generadas por la pandemia en curso, dado que las reuniones fueron limitadas por la emergencia sanitaria, enlenteciendo el proceso.

En lo que refiere al software, el mismo fue desarrollado en su totalidad por el grupo, tomando como referencias otros antecedentes de programación utilizando MAVROS y C++. Se diseñaron e implementaron distintos algoritmos de navegación autónoma con diferentes propósitos. Se comenzó con programas sencillos, que simplemente ejecutaran un despegue y aterrizaje, luego se desarrollaron nodos que además del despegue, realizaran una cierta trayectoria en el espacio (como círculos, ochos y rectas, entre otros).

Posteriormente se diseñaron acciones de control para seguimiento y evasión, logrando diferentes alternativas, algunas de las cuales fueron pensadas y desarrolladas completamente por el equipo, mientras otras se basaron fuertemente en bibliografía existente, con modificaciones particulares para adaptarlas al problema en estudio.

En una segunda etapa de software, se programaron nodos que implementaran las estrategias de control diseñadas para seguir al líder y evadir las posibles colisiones con los demás integrantes de la flota. Finalmente, se unieron todas las piezas para llevar a cabo la dinámica *Follow the leader* con tres UAVs (un líder y dos seguidores). Aquí, los seguidores contaron con la capacidad de esquivar al líder y evadirse entre ellos en caso de tener una proximidad que se considerara peligrosa, y luego de esto buscar la forma más óptima y segura de retomar su misión de seguimiento.

Si bien los algoritmos en todos los casos funcionaron correctamente en las simulaciones por computadora, el paso de dichas simulaciones a las pruebas de campo no resultó inmediato. Se debieron realizar modificaciones en la implementación del código para lograr que funcionara en las aeronaves construidas por el equipo. Los principales inconvenientes identificados se relacionaron con la falta de exactitud de las medidas obtenidas del módulo GPS, que ocasionaron problemas tanto para obtener el origen de coordenadas de nuestro sistema, así como una referencia común de altura sobre el nivel del mar.

Por otro lado, se logró implementar exitosamente una red Ad-Hoc, posibilitando la comunicación entre los UAVs y la *laptop* (que tenía el papel de estación terrestre) a través de Wi-Fi. De esta forma se lograron probar y validar los algoritmos de control diseñados sobre el sistema real.

El alcance inicial del proyecto, contemplaba la construcción de tres drones capaces de volar, el desarrollo de acciones de control, y la implementación de dichas acciones en un entorno de simulación, demostrando de esta manera que era factible implementar el sistema en la flota real. Finalmente, dado el éxito de las etapas anteriores, se logró cumplir con el objetivo adicional de llevar a cabo un sistema funcional real con los tres drones construidos, con un líder realizando una trayectoria circular y dos drones actuando como seguidores, evitando posibles colisiones.

Se destaca la gran variedad de temáticas abordadas en este proyecto. Tal como se describió en los capítulos anteriores, se puede dividir el trabajo realizado en distintos módulos; un módulo de hardware donde se armaron y calibraron los drones desde cero, un módulo de diseño de algoritmos de control, uno de desarrollo de software, uno de comunicación inalámbrica, y un módulo de procesamiento y análisis de resultados, buscando posibles soluciones a los problemas presentados. Gracias a esto, se logró obtener una buena experiencia de desarrollo ingenieril, adquiriendo una amplia gama de conocimientos nuevos y aplicando varios de los conocimientos aprendidos en el transcurso de la carrera. Se logró tener una experiencia de trabajo enriquecedora, trabajando en equipo, manejando los tiempos de forma correcta y buscando soluciones a los imprevistos presentados hasta lograr los objetivos.

Se debe destacar finalmente, no sólo el trabajo ya mencionado, sino las posibilidades que el proyecto brinda para el desarrollo de trabajos futuros. Es un proyecto escalable y adaptable a distintos objetivos, la variedad de posibles aplicaciones es muy amplia, tiene muchos puntos a mejorar, pero con un gran camino ya recorrido.

10.2. Trabajo Futuro

Como se ha mencionado a lo largo del informe, si bien en este trabajo se han logrado muy buenos resultados, el proyecto puede ser adaptable a diferentes situaciones y aplicaciones, de acuerdo al interés existente.

Se pueden realizar varias mejoras, algunas de ellas fueron discutidas por el grupo a lo largo del año, pero finalmente no fueron implementadas por falta de tiempo o por estar muy por fuera de los alcances planteados. La mayoría de estas

mejoras se relacionan con la capacidad de adaptación del enjambre en caso de fallas en algún integrante.

En primer lugar, existe una problemática que yace en una posible falla en la conexión del líder. Dado que la dinámica depende fundamentalmente de él, debería existir cierta configuración de seguridad que implique, que ante la pérdida de conexión del líder, batería baja u otra problemática que lo ausente de la red, se reasigne la misión a alguno de los seguidores, o en su defecto que los seguidores ante la falta de información proveniente del líder, retornen a su base de la forma más segura posible.

Podría existir también una mala conexión con alguno de los seguidores, provocando que por falta de información (debido a la pérdida de paquetes), los algoritmos de evasión implementados no funcionen correctamente. En tal caso, se debería nuevamente buscar la forma de que ese dron seguidor abandone la misión y retorne de manera segura a su base, adaptando la misión para que siga obteniendo buenos resultados.

Actualmente los drones reportan todos los datos de interés a la estación terrestre, que realiza el cómputo de las posiciones deseadas. Otra opción posible sería lograr independizarse de la computadora para que se puedan realizar misiones de larga distancia con buena conexión en la red Ad-Hoc entre los drones. Esto puede lograrse fácilmente si se ejecuta el ROS Máster en alguna de las computadoras a bordo de las aeronaves, o de manera más compleja, haciendo uso de varios Máster que interactúen entre sí (arquitectura multimáster [38]).

Hasta el momento, las principales mejoras que se han discutido están relacionadas a la conectividad de los drones, pero también se pueden hacer cambios que mejoren la dinámica de los mismos. Se observó a lo largo del Capítulo 9 que los drones seguidores no lograban mantener la distancia deseada con el líder. Esto podría corregirse a partir de herramientas clásicas de control, como puede ser la implementación de un controlador P.I.D donde se realimente la diferencia de posición con respecto a la posición deseada, y se obtenga a partir de este control un valor de la variable K_a del campo atractor que ayude a disminuir esta distancia. Otra alternativa podría ser intentar predecir la posición objetivo en un instante futuro, considerando una ventana de tiempo y/o la velocidad instantánea del líder de forma tal que el seguidor centre un campo atractor en el punto que predice.

Por otro lado, recordando que los parámetros obtenidos para las estrategias de seguimiento y evasión se obtuvieron empíricamente mediante simulaciones, una alternativa posible sería aplicar técnicas de optimización para hallar los valores que mejor se adapten al problema.

En cuanto a la escalabilidad de la solución, las estrategias de control aplicadas no propones restricciones sobre la cantidad de seguidores presentes en el enjambre. Dada la modularización del software utilizado se podrían agregar nuevos drones con nodos seguidores separados sin mayor dificultad. En dicho caso la nueva formación deseada podría ser elegida con los parámetros d, θ y Δh de cada uno de los seguidores, teniendo en cuenta la incidencia de los campos repulsores entre los mismos.

Por último se destaca que los drones construidos cuentan con gran capacidad

de cómputo gracias al Intel NUC que tienen equipado, por lo que podrían agregarse sensores de diferentes tipos a las aeronaves, que permitan desarrollar aplicaciones que hagan uso de la flota para llevar a cabo una cierta tarea, como por ejemplo cámaras de video, sensores termográficos o multiespectrales. Se encuentra aquí un gran campo fértil para avanzar en aplicaciones de procesamiento de señales, que no tengan como objetivo enfocarse en el control de navegación y vuelo del enjambre.

Apéndice A

Ensamblado y Configuración del Sistema

A.1. Ensamblado - Fase 1

Los pasos que se siguieron para el correcto armado de cada *UAV* se enumeran a continuación, procurando que sea una tarea sencilla de seguir para el lector.

1. Lo primero que se hizo fue soldar a la base todos los cables de las ESCs, como se muestra en la Figura A.1 respetando las polaridades en cada punto de soldadura.

2. Tal como se ilustra en la Figura A.2, se soldó el cable de la batería y el cable JST que alimenta al NUC. Se agregó silicona en las conexiones para lograr un correcto aislamiento de las mismas.

3. Se pegó el controlador Pixhawk a la base utilizando gel autoadhesivo, el gel es importante para absorber la vibración de la aeronave, que podría producir ruido en la medida de los sensores presentes en la placa del controlador.

   Luego se pegó el receptor de radio R9DS a la base con cinta doble faz, y por último se aseguró la base de la antena del GPS a la base del dron utilizando cinta doble faz y los tornillos con las tuercas que vienen en el empaque del módulo.

4. En las patas, con un taladro, se realizaron agujeros de 2 mm para fijar las piezas cuadradas impresas en 3D, y se colocaron dichas piezas impresas en los espacios rectangulares de la base como se aprecia en la Figura A.3. Estas piezas son un componente extra, que no incluye el modelo del *frame* comercial utilizado. Se añadieron con el propósito de tener una mejor superficie de agarre para las patas del dron.

5. Atendiendo a los sentidos de giro señalados en la Figura A.4, se atornillaron los motores en los brazos.

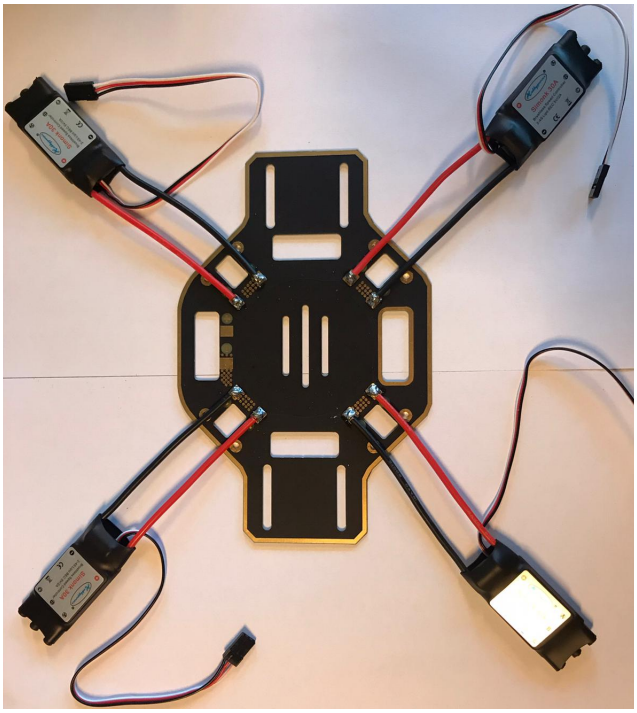


Figura A.1: Base del dron con los ESC.

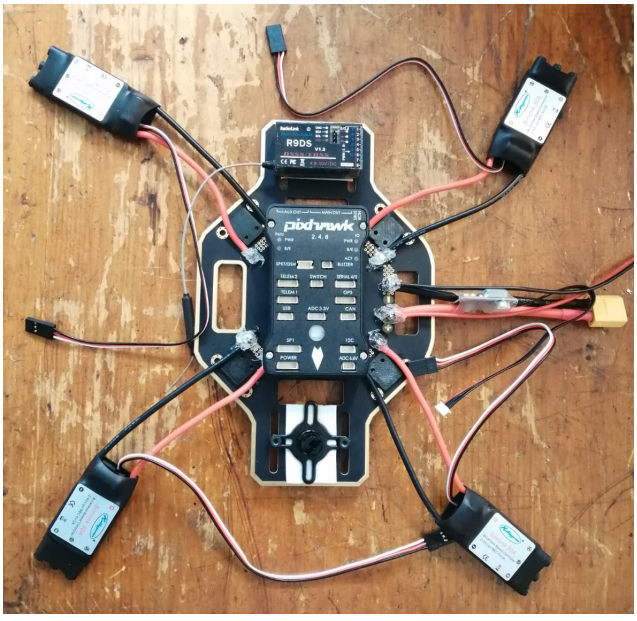


Figura A.2: Base inferior del dron con las ESCs, el cable de la batería, el JST que alimenta al Intel NUC y el Pixhawk pegado sobre el gel anti-vibración.

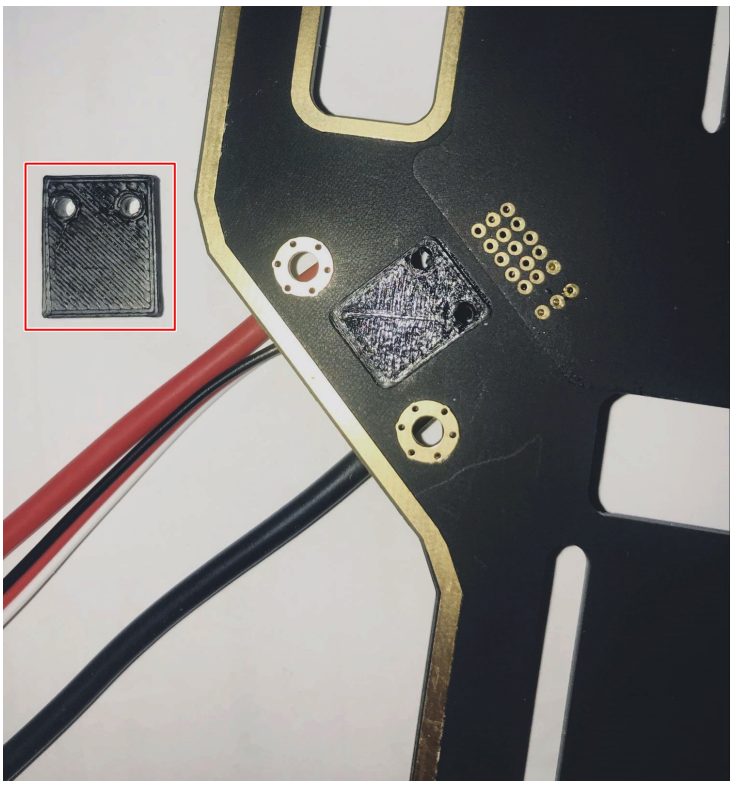


Figura A.3: Arandelas 3D, utilizadas para generar otro punto de agarre para las patas.

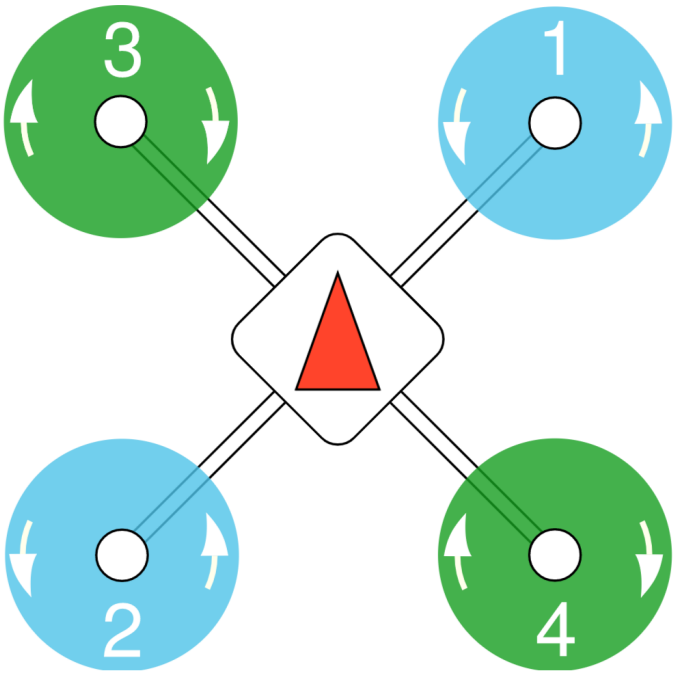


Figura A.4: Sentidos de giro de los motores de un cuadricóptero, con el frente correspondiendo a donde apunta la flecha impresa en la *FCU* Pixhawk [29].

6. Se colocaron tanto los brazos como las patas, atornilladas a la base inferior.

7. Se conectaron las ESCs a los motores y se las aseguró en los brazos utilizando cinta. En primera instancia la conexión de las tres fases es totalmente arbitraria, luego de tener la aeronave ensamblada completamente, se debe verificar que el sentido de giro de los motores sea el adecuado (siguiendo los sentidos dados por la Figura A.4), y en caso contrario se deben intercambiar dos de las distintas fases del motor para lograr el sentido deseado.

8. Se conectaron los cables de datos de las cuatro ESCs a las salidas *MAIN OUT* 1 a 4 del auto-piloto Pixhawk, siguiendo la misma numeración que los motores correspondientes.

9. Se agregó el módulo GPS afirmándolo sobre la antena ya colocada y se lo conectó a la entrada *GPS* del Pixhawk.

10. Se añadió el *switch* de seguridad, asegurándolo con silicona en un brazo, y conectándolo al Pixhawk en la entrada *SWITCH*.

11. Se conectó el receptor de radio RD9S al Pixhawk a través de la entrada *RCIN* y se aseguró la antena del módulo a un brazo del dron con cinta.

A.2. Configuración del Hardware

Se verá la configuración de los parámetros de hardware utilizados específicamente en este proyecto.

En primer lugar, para aparear el control remoto con el receptor presente en el dron se siguieron los siguientes pasos:

1. En el control remoto se cambió el tipo de vehículo a *quadcopter*.

2. Se cambió *system OUT* de PPM a SBUS.

3. Se oprimió el *switch* en el receptor del dron hasta que la luz esté azul.

4. Se apagó y prendió el transmisor para lograr el apareo.

Posteriormente se configuró y calibró el dron haciendo uso de la herramienta de software QGroundControl.

En la pantalla *Configuration* de dicho software, para completar la configuración inicial, se realizaron los siguientes pasos:

1. Se actualizó el firmware, esto puede realizarse fácilmente a través de la pestaña *Firmware* de QGC.

2. En la pestaña *Airframe*, se eligió el esqueleto del dron como *Quad X*.

3. Se calibraron los sensores, siguiendo las instrucciones indicadas en QGC en la solapa *Sensors*.

4. Se calibró la radio, para ello nos dirigimos a la pestaña *Radio*.

5. Se configuraron distintos parámetros de funcionamiento en la sección *Parameters*, notamos que los valores seleccionados son bastantes conservadores en cuanto a las velocidades máximas permitidas, con el fin de lograr un vuelo seguro y minimizar riesgos:

   a) CBRK_USB_CHECK = max value

   b) PWM_MIN = 1000 us

   c) MPC_XY_VEL_MAX_UP = 0,7 m/s

   d) MPC_Z_VEL_MAX_DN = 0,7 m/s

   e) MPC_Z_VEL_MAX_DN = 0,7 m/s

   f) MPC_LAND_SPEED = 1,0 m/s

   g) RTL_RETURN_ALT = 4,0 m

   h) RTL_DESCEND_ALT = 4,0 m

6. En la pestaña *Flight Modes* se realizaron las siguientes configuraciones:

   a) Se configuró el Canal 10 (Switch A del RC) como *kill switch*.

   b) Se configuró el Canal 5 (Switch C del RC) como *mode switch*

   c) Se seleccionaron los modos de vuelo a utilizar (*altitude*, *position*, *land*).

7. Se realizó la calibración de la batería y de las ESCs en la pestaña *Power*, de la siguiente manera:

   a) Se especificó que son cuatro celdas.

   b) Se conectó la batería al dron, se introdujo el voltaje medido manualmente y se oprimió *calculate*.

   c) Se desconectó la batería y el USB. Se reconectó el USB y se procedió a la calibración de las ESCs.

8. En la pestaña *Safety* se debieron configurar los parámetros de seguridad.

   a) Se activó el *Geofence Failsafe Trigger*

   b) Se setearon los valores de altura y radio en 10 m y 20 m respectivamente.

9. Por último se reinició la *FCU* y se intentó armar el dron sin hélices colocadas.

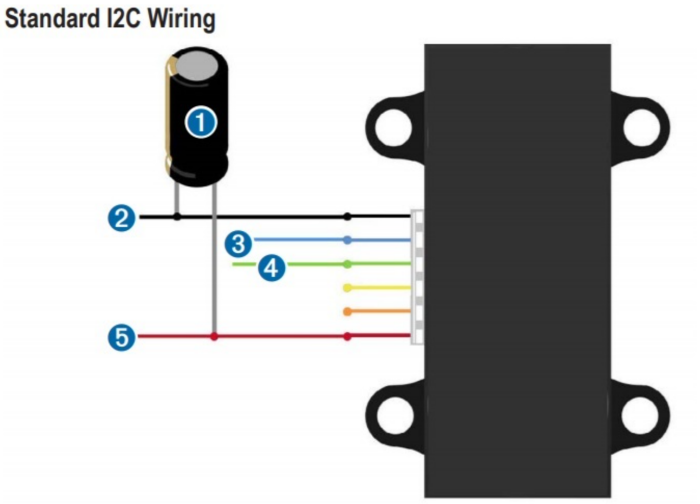


Figura A.5: Conexionado necesario para conectar el LIDAR mediante el puerto I2C.

A.3. Ensamblado - Fase 2

Para completar el armado del dron autónomo se debieron realizar los pasos que se detallan a continuación:

1. Se agregó el *LIDAR*. Para ello se recomienda seguir el tutorial oficial de PX4 [2]. Se realizó el conexionado de la Figura A.5, donde el capacitor utilizado debe ser de 680μF.

2. Se procedió a ensamblar y montar la computadora de a bordo. Para esto fue necesario conectar y atornillar la tarjeta de red con sus dos antenas al Intel NUC, posteriormente se insertaron las memorias RAM y se conectó el disco SSD.

3. Se montó una base plástica, atornillada sobre el plato superior del cuadricóptero, y finalmente se aseguró la mini-PC sobre dicha base con cuatro separadores.

4. Se conectaron teclado, ratón y pantalla a la computadora de a bordo, y se le instaló el sistema operativo Ubuntu 18.04, el marco de trabajo ROS y el paquete *mavros*, como lo detalla la Sección 4.2.

5. Se instaló y configuró la conexión por SSH, de forma de poder acceder a la mini-PC desde la *notebook* para continuar con su configuración.

A.4. Configuración de Comunicaciones

A.4.1. Comunicaciones entre drones

Luego de tener los drones completamente armados y probados, se debió configurar la red AD-HOC para que los drones puedan comunicarse entre sí y configurar ROS en cada dispositivo para que funcione dentro de dicha red. Para eso se debió realizar lo siguiente:

1. Se modificaron las variables ambientales de ROS, asignando un nombre a cada aeronave que participe del ROS *master* compartido, indicando las IP propias y la IP de la computadora en tierra que ejecuta el *roscore*. Para esto se debe modificar el archivo `rosrc` en la carpeta `root`, las principales variables a setear son `ROS_IP, AGENT, ROS_MASTER_URI`.

2. Se descargó y compiló la herramienta MAVLink Router, siguiendo las instrucciones presentes en su repositorio de *github* [17]. Esta herramienta permite enrutar los paquetes MAVLink que se reciben por WiFi en la mini-PC hacia la *FCU* conectada a ella por USB.

3. Se creó el archivo de configuración del MAVLink Router, para esto debe ejecutarse:

   ```
   $ sudo nano /etc/mavlink-router/main.conf
   ```

   Agregando las siguientes lineas para dirigir los paquetes recibidos hacia los puertos de la controladora de vuelo y QGroundControl:

   ```
   [General]
   TcpServerPort=5760

   [UartEndpoint bravo]
   Device = /dev/ttyACM0
   Baud = 921600

   [UdpEndpoint a]
   Mode = Normal
   Address = 127.0.0.1
   Port = 14550
   ```

4. Se habilitó el re-direccionamiento de paquetes mediante los comandos:

   ```
   $ sudo systemctl enable mavlink-router.service
   $ sudo systemctl start mavlink-router.service
   ```

5. Se editó el archivo `interfaces` en la ruta `/etc/network/interfaces` con el comando:

   ```
   $ sudo nano /etc/network/interfaces
   ```

Se cambió la IP por la correspondiente al número de dron y se indicó que el modo sería Ad-Hoc.

6. Se reinició la red con el comando

```
$ systemctl restart networking
```

Y se verificó utilizando *iwconfig*, que la interfaz haya cambiado a modo Ad-Hoc y que todos los drones estén conectados a la misma celda.

7. Se deshabilitó el *Network Manager*, editando el archivo de configuración:

```
$ sudo nano etc/NetworkManager/NetworkManager/conf
```

Modificando:

managed=false

8. Se reinició el servicio y se chequeó:

```
$ systemctl restart networking
$ sysctl net.ipv4.ip_forward
```

En este punto fue posible acceder a internet desde el NUC a través de la computadora de la estación terrestre, y al mismo tiempo contar con la red Ad-Hoc funcionando.

A.4.2. Comunicación con la Estación Terrestre

Como se mencionó en el Capítulo 2, la estación terrestre consta de una *notebook* con ROS y QGC, que es utilizada para monitorizar los drones y ejecutar el ROS *master* compartido con la red Ad-Hoc. Para ello la *notebook* debe ser configurada siguiendo los pasos que se detallan a continuación.

1. En primer lugar, se siguieron una serie de pasos necesarios para poder conectarse a los drones a través de ssh desde la computadora, los que se enumeran a continuación:

   a) Dirigirse al archivo `ssh config` en la carpeta `home` en la laptop.

   b) Editar el archivo anterior indicando para cada uno de los drones el parámetro `User` y `HostName`.

   c) Conectarse por ssh al dron deseado.

2. Por otro lado, para lograr que los drones tengan acceso a internet a través de la computadora se debió habilitar en ésta el reenvío de paquetes o *packets forwarding*. Además, se seteó la tabla de enrutamiento y DNS en los drones como se indicó previamente.

   Para habilitar el reenvío de paquetes en la computadora se siguió un instructivo recomendado por el tutor [1]. Luego en la consola se deben ejecutar los siguientes comandos:

```
$ sudo /sbin/iptables -t nat -A POSTROUTING -o eth0 -
MASQUERADE
$ sudo /sbin/iptables -A FORWARD -i eth0 -o eth1 -m
state --state RELATED,ESTABLISHED -j ACCEPT
$ sudo /sbin/iptables -A FORWARD -i eth1 -o eth0 -j
ACCEPT
```

Para lograr que la configuración quede guardada luego de un reinicio, se instaló un paquete llamado *iptables-persistent*.

Luego se instaló MAVLink Router al igual que en los *UAV*, necesario en este caso para enrutar los paquetes MAVLink de cada aeronave a cada puerto asignado por *mavros*. Para ello nos dirigimos a la referencia [3].

Seguidamente, habilitamos el MAVLink Router como un servicio mediante las siguientes líneas:

```
$ sudo systemctl enable mavlink-router.service
$ sudo systemctl start mavlink-router.service
```

Por último, al igual que en las aeronaves, se seteó la variable ambiental de ROS que determina donde se aloja el ROS *master* compartido. Para esto se modificó el archivo `rosrc` en la carpeta `root` indicando el valor de las variables `ROS_MASTER_URI` y `AGENT`.

Esta página ha sido intencionalmente dejada en blanco.

Apéndice B

Cambios de Coordenadas

Modelando a la superficie Tierra al nivel del mar como un elipsoide de radio ecuatorial R_e y radio polar R_p, obtenemos que la superficie de la Tierra se describe con la siguiente ecuación: $\frac{x^2}{R_e^2} + \frac{y^2}{R_e^2} + \frac{z^2}{R_p^2} = 1$

Parametrizando esta superficie con los parámetros $\theta \in (0, 2\pi)$ y $\varphi \in (0, \pi)$ obtenemos las siguientes cartas:

$$\phi(\theta, \varphi) = \begin{pmatrix} R_e cos(\theta) sen(\varphi) \\ R_e sen(\theta) sen(\varphi) \\ R_p cos(\varphi) \end{pmatrix}$$

Estos ángulos pueden interpretarse a partir de las figuras B.1 y B.2.

Claramente el parámetro θ corresponde directamente con la longitud (ángulo de giro entorno al eje polar de dirección e_3), mientras que φ se puede vincular con la latitud (cuyo vínculo se describirá más adelante). Se describe la posición de un punto que se encuentra a una altura h sobre el nivel del mar, a partir de tomar un punto al nivel del mar y desplazarlo una distancia h según la normal saliente del elipsoide. Para obtener dicha normal, se computan las derivadas parciales de las cartas anteriores.

$$\frac{\partial \phi}{\partial \theta} = R_e(-sen(\theta)sen(\varphi), cos(\theta)sen(\varphi), 0)$$

$$\frac{\partial \phi}{\partial \varphi} = (R_e cos(\theta)cos(\varphi), R_e sen(\theta)cos(\varphi), -R_p sen(\varphi))$$

$$\frac{\partial \phi}{\partial \varphi} \wedge \frac{\partial \phi}{\partial \theta} = \begin{pmatrix} R_e R_p cos(\theta) sen^2(\varphi) \\ R_e R_p sen(\theta) sen^2(\varphi) \\ R_e^2 cos(\varphi) sen(\varphi) \end{pmatrix} \propto \underbrace{\begin{pmatrix} R_p cos(\theta) sen(\varphi) \\ R_p sen(\theta) sen(\varphi) \\ R_e cos(\varphi) \end{pmatrix}}_{\text{elipsoide con radios intercambiados}} =: N$$

Se puede observar a partir de la expresión de la normal que el vínculo entre φ y la latitud geodésica (ángulo que forma el eje polar con el vector normal) no es trivial, pues el coseno de la latitud geodésica φ_L se calcula con la expresión

$$cos(\varphi_L) = \frac{<N,e_3>}{\|N\|} = \frac{R_e cos(\varphi)}{\sqrt{R_p^2 sen^2(\varphi) + R_e^2 cos^2(\varphi)}} \neq cos(\varphi) \qquad si \quad R_e \neq R_p$$

Del mismo modo, φ tampoco coincide con la expresión de la latitud geocéntrica (ángulo que forma el eje polar con el vector posición desde el centro de la Tierra), pues:

$$\frac{\left\langle \begin{pmatrix} x \\ y \\ z \end{pmatrix}, e_3 \right\rangle}{\left\| \begin{pmatrix} x \\ y \\ z \end{pmatrix} \right\|} = \frac{R_p cos(\varphi)}{\sqrt{R_p^2 sen^2(\varphi) + R_e^2 cos^2(\varphi)}} \neq cos(\varphi) \qquad si \quad R_e \neq R_p$$

Puede notarse que φ coincide con la latitud paramétrica, pues si se proyecta el elipsoide sobre una esfera a partir de reescalar el eje polar (multiplicándolo por el cociente de los radios $\frac{R_e}{R_p}$), se tendrá que φ corresponderá con el ángulo que forma el eje polar con la proyección sobre la esfera del punto $\phi(\varphi, \theta)$. Se puede corroborar esto calculando el coseno de la latitud paramétrica como:

$$\frac{\left\langle \begin{pmatrix} x \\ y \\ \frac{R_e}{R_p} z \end{pmatrix}, e_3 \right\rangle}{\left\| \begin{pmatrix} x \\ y \\ \frac{R_e}{R_p} z \end{pmatrix} \right\|} = \frac{\frac{R_e}{R_p} R_p cos(\varphi)}{R_e} = cos(\varphi)$$

¿Cómo se obtiene φ en términos de la latitud geodésica φ_L?

Usando la expresión que se obtuvo para $cos(\varphi_L)$ en función de φ, se deduce:

$$cos(\varphi_L) = \frac{R_e cos(\varphi)}{\sqrt{R_p^2 sen^2(\varphi) + R_e^2 cos^2(\varphi)}} = \frac{R_e cos(\varphi)}{\sqrt{R_p^2 + cos^2(\varphi)(R_e^2 - R_p^2)}}$$

$$\Rightarrow R_e^2 cos^2(\varphi) = \left[R_P^2 + cos^2(\varphi)(R_e^2 - R_p^2)\right] cos^2(\varphi_L)$$

$$\Rightarrow cos^2(\varphi) \left[R_e^2 + cos^2(\varphi_L)(R_p^2 - R_e^2)\right] = R_P^2 cos^2(\varphi_L)$$

$$\Rightarrow cos^2(\varphi) = \frac{R_p^2 cos^2(\varphi_L)}{R_e^2 sen^2(\varphi_L) + R_p^2 cos^2(\varphi_L)}$$

Como el signo de $cos(\varphi)$ es igual al signo de $cos(\varphi_L)$ (pues el cuadrante del ángulo correspondiente a φ siempre coincide con el de φ_L, como se puede ver en la figura B,1), y definiendo cos^{-1} como la inversa de coseno para los ángulos en $[0, \pi]$, se puede escribir entonces:

$$\varphi = cos^{-1}\left(\frac{R_p cos(\varphi_L)}{\sqrt{R_e^2 sen^2(\varphi_L) + R_p^2 cos^2(\varphi_L)}}\right)$$

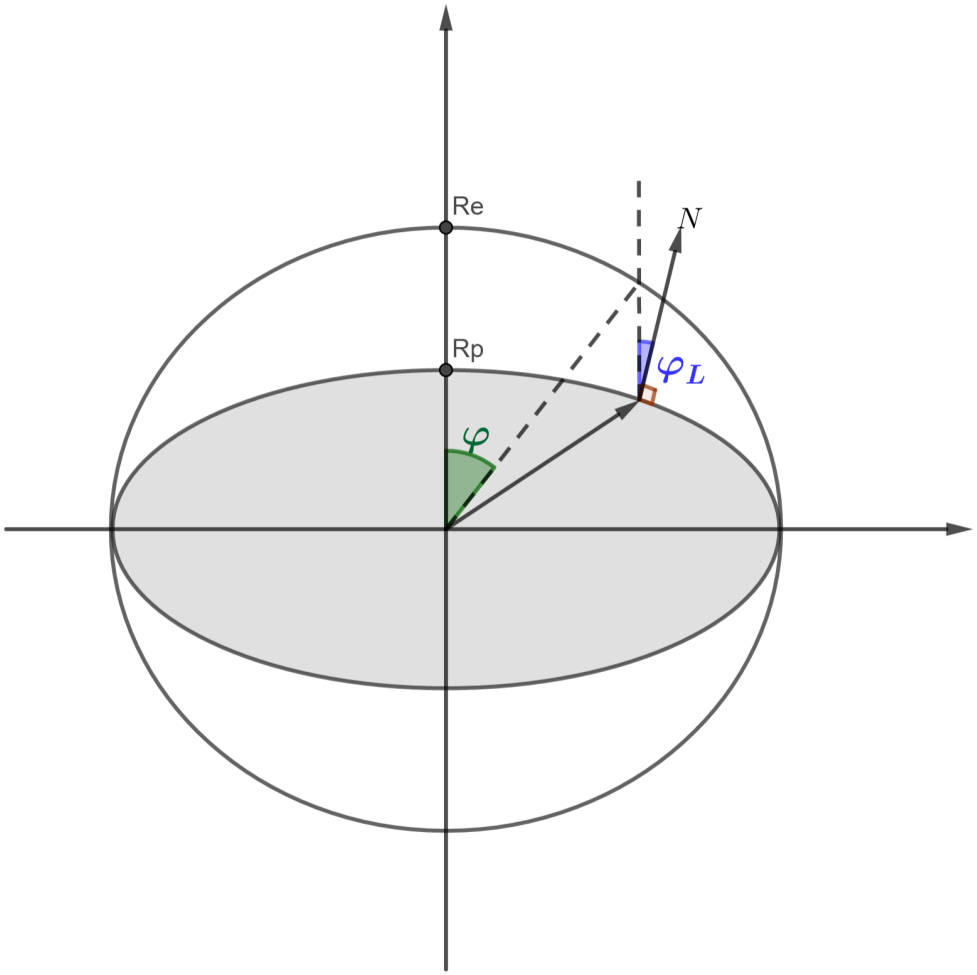


Figura B.1: Vínculo entre la latitud paramétrica φ y la latitud geodésica φ_L.

En resumen, se ha conseguido entonces la siguiente familia de ecuaciones:

$$\begin{cases} \phi(\theta,\varphi) = \begin{pmatrix} R_e cos(\theta) sen(\varphi) \\ R_e sen(\theta) cos(\varphi) \\ R_p cos(\varphi) \end{pmatrix} \\ N = \begin{pmatrix} R_p cos(\theta) sen(\varphi) \\ R_p sen(\theta) cos(\varphi) \\ R_e cos(\varphi) \end{pmatrix} \\ cos(\varphi_L) = \frac{R_e cos(\varphi)}{\sqrt{R_p^2 sen^2(\varphi) + R_e^2 cos^2(\varphi)}} \\ sen(\varphi_L) = \sqrt{1 - cos^2(\varphi_L)} = \frac{R_p sen(\varphi)}{\sqrt{R_p^2 sen^2(\varphi) + R_e^2 cos^2(\varphi)}} \\ cos(\varphi) = \frac{R_p cos(\varphi_L)}{\sqrt{R_e^2 sen^2(\varphi_L) + R_p^2 cos^2(\varphi_L)}} \\ sen(\varphi) = \sqrt{1 - cos^2(\varphi)} = \frac{R_e sen(\varphi_L)}{\sqrt{R_e^2 sen^2(\varphi_L) + R_p^2 cos^2(\varphi_L)}} \end{cases}$$

Haciendo un poco de abuso de notación, se puede escribir un punto posición del dron como:

$$\phi(\theta,\varphi,h) = \underbrace{\phi(\theta,\varphi,0)}_{\phi(\theta,\varphi)} + h\frac{N(\theta,\varphi)}{\|N(\theta,\varphi)\|}$$

$$\Rightarrow \phi(\theta,\varphi,h) = \begin{pmatrix} R_e cos(\theta)sen(\varphi) \\ R_e sen(\theta)cos(\varphi) \\ R_p cos(\varphi) \end{pmatrix} + \frac{h}{\sqrt{R_p^2 sen^2(\varphi) + R_e^2 cos^2(\varphi)}} \begin{pmatrix} R_p cos(\theta)sen(\varphi) \\ R_p sen(\theta)cos(\varphi) \\ R_e cos(\varphi) \end{pmatrix} \tag{B.1}$$

A continuación se tomará un punto de operación (θ_0, ϕ_0, h_0) en torno al cual se hará un desarrollo de Taylor de ϕ, el cual se expresará de manera aproximada en un sistema coordenado ortonormal. Se aprovecha que las derivadas parciales de ϕ son aproximadamente ortogonales (en particular, son ortogonales en $h = 0$), por lo que para expresar el Taylor en una base ortonormal bastará con calcular la norma de estas derivadas.

Se comienza con la derivada parcial respecto θ:

$$\frac{\partial\phi}{\partial\theta} = \begin{pmatrix} -R_e sen(\theta)sen(\varphi) \\ R_e cos(\theta)sen(\varphi) \\ 0 \end{pmatrix} + \frac{hR_p}{\sqrt{R_p^2 sen^2(\varphi) + R_e^2 cos^2(\varphi)}} \begin{pmatrix} -sen(\theta)sen(\varphi) \\ cos(\theta)sen(\varphi) \\ 0 \end{pmatrix}$$

$$\Rightarrow \frac{\partial\phi}{\partial\theta} = \underbrace{sen(\varphi)\left[R_e + \frac{hR_p}{\sqrt{R_p^2 sen^2(\varphi) + R_e^2 cos(\varphi)}}\right]}_{\|\frac{\partial\phi}{\partial\theta}\|} \cdot \begin{pmatrix} -sen(\theta) \\ cos(\theta) \\ 0 \end{pmatrix}$$

Es decir

$$\left\|\frac{\partial\phi}{\partial\theta}\right\| = sen(\varphi)\left[R_e + \frac{hR_p}{\sqrt{R_p^2 sen^2(\varphi) + R_e^2 cos(\varphi)}}\right] \approx sen(\varphi)\left[R_e + h\right]$$

Del mismo modo, se procede con la derivada parcial respecto φ:

$$\frac{\partial\phi}{\partial\varphi} = \begin{pmatrix} R_e cos(\theta)cos(\varphi) \\ R_e sen(\theta)cos(\varphi) \\ -R_p sen(\varphi) \end{pmatrix} + \frac{h}{\sqrt{R_p^2 sen^2(\varphi) + R_e^2 cos^2(\varphi)}} \begin{pmatrix} R_p cos(\theta)cos(\varphi) \\ R_p sen(\theta)cos(\varphi) \\ -R_e sen(\varphi) \end{pmatrix} - \frac{h(2R_p^2 sen(\varphi)cos(\varphi) - 2R_e^2 cos(\varphi)sen(\varphi))}{2(R_p^2 sen^2(\varphi) + R_e^2 cos^2(\varphi))^{\frac{3}{2}}} \begin{pmatrix} R_p cos(\theta)sen(\varphi) \\ R_p sen(\theta)sen(\varphi) \\ R_e cos(\varphi) \end{pmatrix}$$

$$\Rightarrow \frac{\partial\phi}{\partial\varphi} = \begin{pmatrix} R_e cos(\theta)cos(\varphi) \\ R_e sen(\theta)cos(\varphi) \\ -R_p sen(\varphi) \end{pmatrix} + \frac{h}{\sqrt{R_p^2 sen^2(\varphi) + R_e^2 cos^2(\varphi)}} \begin{pmatrix} R_p cos(\theta)cos(\varphi) \\ R_p sen(\theta)cos(\varphi) \\ -R_e sen(\varphi) \end{pmatrix} - \frac{sen(\varphi)cos(\varphi)h(R_e - R_p)(R_e + R_p)}{(R_p^2 sen^2(\varphi) + R_e^2 cos^2(\varphi))^{\frac{3}{2}}} \begin{pmatrix} R_p cos(\theta)sen(\varphi) \\ R_p sen(\theta)sen(\varphi) \\ R_e cos(\varphi) \end{pmatrix}$$

Para aproximar el valor de $\left\|\frac{\partial\phi}{\partial\varphi}\right\|$, se puede observar que la norma del último término de la expresión anterior es despreciable, pues vale:

$$\underbrace{hsen(\varphi)cos(\varphi)}_{\frac{1}{2}sen(2\varphi)}\underbrace{\frac{R_e-R_p}{\sqrt{R_e^2cos^2(\varphi)+R_p^2sen^2(\varphi)}}}_{\approx 0}\underbrace{\frac{R_e+R_p}{\sqrt{R_e^2cos^2(\varphi)+R_p^2sen^2(\varphi)}}}_{\approx 2}\underbrace{\sqrt{\frac{R_p^2cos^2(\varphi)+R_e^2sen^2(\varphi)}{R_e^2cos^2(\varphi)+R_p^2sen^2(\varphi)}}}_{\approx 1}$$

Mientras que la norma del primer y segundo término es:

$$\begin{cases} \left\| \begin{pmatrix} R_e cos(\theta)cos(\varphi) \\ R_e sen(\theta)cos(\varphi) \\ -R_p sen(\varphi) \end{pmatrix} \right\| = \sqrt{R_e^2cos^2(\varphi)+R_p^2sen^2(\varphi)} \\ \left\| \frac{h}{\sqrt{R_p^2sen^2(\varphi)+R_e^2cos^2(\varphi)}} \begin{pmatrix} R_p cos(\theta)cos(\varphi) \\ R_p sen(\theta)cos(\varphi) \\ -R_e sen(\varphi) \end{pmatrix} \right\| = h\sqrt{\frac{R_p^2cos^2(\varphi)+R_e^2sen^2(\varphi)}{R_e^2cos^2(\varphi)+R_p^2sen^2(\varphi)}} \approx h \end{cases}$$

Es decir que el tercer término que aparece en la expresión de $\frac{\partial\phi}{\partial\varphi}$ es despreciable en comparación con los primeros 2. Por lo tanto, como además los primeros dos términos son casi paralelos debido a que los radios polares y ecuatoriales de la Tierra son muy similares, tenemos que la norma de $\frac{\partial\phi}{\partial\varphi}$ puede aproximarse por sumar las normas de los primeros dos términos, es decir que:

$$\Rightarrow \left\|\frac{\partial\phi}{\partial\varphi}\right\| \approx \sqrt{R_e^2cos^2(\varphi)+R_p^2sen^2(\varphi)}+h$$

Finalmente, es claro que como $\phi = \phi\mid_{h=0} + h\frac{N}{\|N\|} \Rightarrow \left\|\frac{\partial\phi}{\partial h}\right\| = \left\|\frac{N(\theta,\varphi)}{\|N(\theta,\varphi)\|}\right\| = 1.$

Se considera entonces $\left\{\widehat{\frac{\partial\phi}{\partial\theta}}, \widehat{\frac{\partial\phi}{\partial\varphi}}, \widehat{\frac{\partial\phi}{\partial h}}\right\}$ los vectores unitarios colineales con las derivadas parciales de ϕ, es decir:

$$\begin{cases} \frac{\partial\phi}{\partial\theta} = sen(\varphi)\left[R_e+h\right]\widehat{\frac{\partial\phi}{\partial\theta}} \\ \frac{\partial\phi}{\partial\varphi} = \left(\sqrt{R_e^2cos^2(\varphi)+R_p^2sen^2(\varphi)}+h\right)\widehat{\frac{\partial\phi}{\partial\varphi}} \\ \frac{\partial\phi}{\partial h} = \widehat{\frac{\partial\phi}{\partial h}} \end{cases}$$

La dirección de estas derivadas se bosqueja en la figura B,2

Se expresará el desarrollo de Taylor en la base $\left\{\widehat{\frac{\partial\phi}{\partial\theta}}\mid_{X_0}, -\widehat{\frac{\partial\phi}{\partial\varphi}}\mid_{X_0}, \widehat{\frac{\partial\phi}{\partial h}}\mid_{X_0}\right\}$ (donde $X_0 = (\theta_0, \varphi_0, h_0)$ es el centro del Taylor), de modo que en el sistema cartesiano resultante de coordenadas (x, y, z), se tendrá que el eje x crece en la dirección hacia el este, el eje y crece hacia el norte, y el eje z crece al aumentar la altura.

Usando el Taylor de orden 1 de ϕ en X_0, se puede escribir:

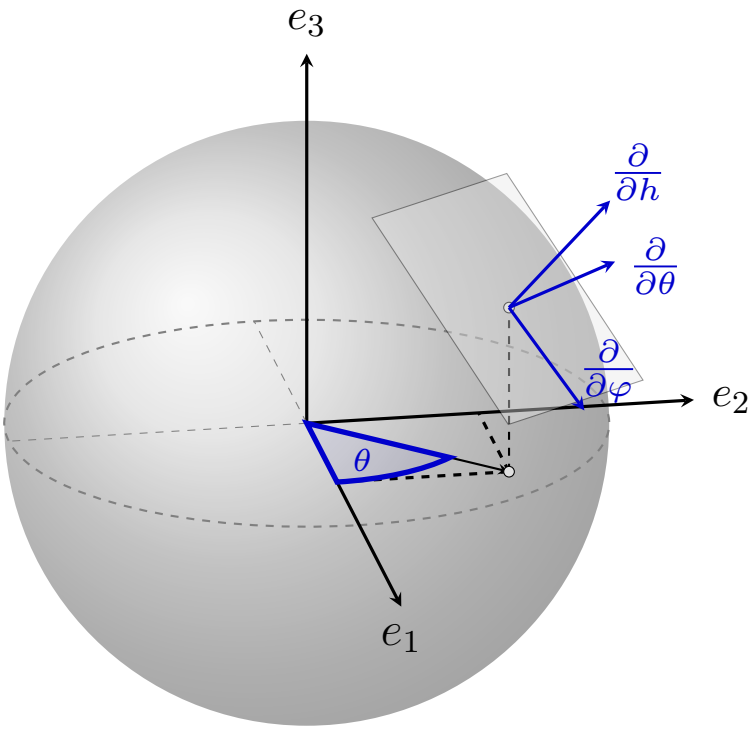


Figura B.2: Base definida por las derivadas parciales de la parametrización ϕ.

$$\phi(\theta,\varphi,h) \approx \frac{\partial\phi}{\partial\theta}\,|_{X_0}\,(\theta-\theta_0) + \frac{\partial\phi}{\partial\varphi}\,|_{X_0}\,(\varphi-\varphi_0) + \frac{\partial\phi}{\partial h}\,|_{X_0}\,(h-h_0)$$

Por lo tanto, en la base $\left\{\widehat{\frac{\partial\phi}{\partial\theta}}\,|_{X_0}, -\widehat{\frac{\partial\phi}{\partial\varphi}}\,|_{X_0}, \widehat{\frac{\partial\phi}{\partial h}}\,|_{X_0}\right\}$ se obtienen las siguientes coordenadas:

$$\boxed{\begin{cases} x = sen(\varphi_0)\,(R_e+h_0)\,(\theta-\theta_0) \\ y = -\left(\sqrt{R_e^2 cos^2(\varphi_0) + R_p^2 sen^2(\varphi_0)} + h_0\right)(\varphi-\varphi_0) \\ z = h - h_0 \end{cases}} \tag{B.2}$$

donde se recuerda que φ (y análogamente φ_0) se obtiene a partir de la latitud geodésica como:

$$\boxed{\varphi = cos^{-1}\left(\frac{R_p cos(\varphi_L)}{\sqrt{R_e^2 sen^2(\varphi_L) + R_p^2 cos^2(\varphi_L)}}\right)} \tag{B.3}$$

Apéndice C

Cuaterniones

C.1. Fundamentos Básicos

Se definen los símbolos $\{i, j, k\}$ y se identifica $\mathbb{R}^4$ con el conjunto de sumas formales $\mathbb{H} = \{a + bi + cj + dk : a, b, c, d \in \mathbb{R}\}$. Los elementos $\{i, j, k\}$ pueden interpretarse como vectores unitarios en 3 ejes espaciales.

En $\mathbb{H}$ se considera la suma componente a componente, es decir que

$$(a + bi + cj + dk) + (a' + b'i + c'j + d'k) = a + a' + i(b + b') + j(c + c') + k(d + d')$$

Se define en el conjunto $\mathbb{H}$ el siguiente producto $\cdot : \mathbb{H} \times \mathbb{H} \to \mathbb{H}$:

$$\begin{aligned}(a + bi + cj + dk) \cdot (a' + b'i + c'j + d'k) = \\ = aa' - bb' - cc' - dd' + \\ + i(ab' + ba' + cd' - dc') + \\ + j(ac' + ca' + db' - bd') + \\ + k(ad' + da' + bc' - cb')\end{aligned}$$

El conjunto $(\mathbb{H}, +, \cdot)$ con las operaciones asi definidas, se denomina **cuaterniones** [41].

Se puede ver que este producto verifica

- (asociatividad): si $\mathbf{q}_1$, $\mathbf{q}_2$, $\mathbf{q}_3 \in \mathbb{H}$, entonces

  $$(\mathbf{q}_1 \cdot \mathbf{q}_2) \cdot \mathbf{q}_3 = \mathbf{q}_1 \cdot (\mathbf{q}_2 \cdot \mathbf{q}_3)$$

  Esto permite omitir los paréntesis en productos triples de cuaterniones.

- (distributiva respecto de la suma): si $\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3 \in \mathbb{H}$, entonces

  $$\mathbf{q}_1 \cdot (\mathbf{q}_2 + \mathbf{q}_3) = \mathbf{q}_1 \cdot \mathbf{q}_2 + \mathbf{q}_1 \cdot \mathbf{q}_3$$
  $$(\mathbf{q}_1 + \mathbf{q}_2) \cdot \mathbf{q}_3 = \mathbf{q}_1 \cdot \mathbf{q}_3 + \mathbf{q}_2 \cdot \mathbf{q}_3$$

- (bilinealidad sobre $\mathbb{R}$): si $\alpha, \beta \in \mathbb{R}$ y $\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3 \in \mathbb{H}$, entonces

$$\mathbf{q}_1 \cdot (\alpha\mathbf{q}_2 + \beta\mathbf{q}_3) = \alpha\mathbf{q}_1 \cdot \mathbf{q}_2 + \beta\mathbf{q}_1 \cdot \mathbf{q}_3$$
$$(\alpha\mathbf{q}_1 + \beta\mathbf{q}_2) \cdot \mathbf{q}_3 = \alpha\mathbf{q}_1 \cdot \mathbf{q}_3 + \beta\mathbf{q}_2 \cdot \mathbf{q}_3$$

  donde el producto por un escalar en $\mathbb{H}$ es el calculado componente a componente (el cual también coincide con identificar $\alpha = \alpha + 0i + 0j + 0k \in \mathbb{H}$ y $\beta = \beta + 0i + 0j + 0k \in \mathbb{H}$ y usar el producto definido para $\mathbb{H}$).

- (productos entre elementos canónicos):

$$\begin{cases} i^2 = j^2 = k^2 = -1 \\ ij = -ji = k \\ jk = -kj = i \\ ki = -ik = j \end{cases}$$

Se define la norma $|\cdot| : \mathbb{H} \to \mathbb{R}^{\geq 0} = \{r \in \mathbb{R} : r \geq 0\}$ como la usual (euclídea) de $\mathbb{R}^4$, es decir que si $\mathbf{q} = a + bi + cj + dk \in \mathbb{H}$ entonces

$$|\mathbf{q}|^2 = a^2 + b^2 + c^2 + d^2$$

Se definen los cuaterniones unitarios como el conjunto

$$S^3 = \{\mathbf{q} \in \mathbb{H} : |\mathbf{q}| = 1\} \subset \mathbb{H}$$

Dado un cuaternión $\mathbf{q} \in \mathbb{H}$, se define su cuaternión conjugado $\overline{\mathbf{q}}$ como

$$\overline{\mathbf{q}} = a - bi - cj - dk$$

Se puede corroborar que

1. si $\mathbf{q}_1, \mathbf{q}_2 \in \mathbb{H}$, entonces
$$\overline{\mathbf{q}_1\mathbf{q}_2} = \overline{\mathbf{q}}_2\overline{\mathbf{q}}_1$$
2. si $\mathbf{q} \in \mathbb{H}$, entonces
$$\mathbf{q}\overline{\mathbf{q}} = |\mathbf{q}|^2$$

En particular, el segundo punto implica que si $|\mathbf{q}| = 1$ (es decir, $\mathbf{q} \in S^3$), entonces $\mathbf{q}^{-1} = \overline{\mathbf{q}}$.

Si $n \in \mathbb{N}$, se dice que una transformación lineal $T : \mathbb{R}^n \to \mathbb{R}^n$ preserva orientación si el determinante de su matriz asociada en alguna base $B = \{v_1, \ldots, v_n\}$ tiene determinante positivo, es decir $det({}_B(T)_B) > 0$.

Sea $O(3) = \{T : \mathbb{R}^3 \to \mathbb{R}^3 : T \text{ lineal}, |T(v)| = |v|\}$ el conjunto de transformaciones ortogonales en $\mathbb{R}^3$ y $SO(3) = \{T : \mathbb{R}^3 \to \mathbb{R}^3 : \ T \text{ lineal}, |T(v)| = |v|, \ T \text{ preserva el sentido}\}$ las transformaciones ortogonales que preservan orientación.

Identificando $\mathbb{R}^3$ con $Im(\mathbb{H}) = \{bi + cj + dk \in \mathbb{H}; b, c, d \in \mathbb{R}\}$, podemos definir entonces $p : S^3 \to SO(3)$, dada por

$$p(\mathbf{q})(v) = \mathbf{q}v\overline{\mathbf{q}} \quad \forall v \in Im(\mathbb{H}) \tag{C.1}$$

A este mapa $p : S^3 \mapsto SO(3)$ se lo denomina *acción por conjugación.*

A continuación, se verá que el mapa $p : S^3 \to SO(3)$ está bien definido.

Para eso, en primer lugar, podemos ver que si se fija $\mathbf{q} \in S^3$, entonces el mapa $\mathbb{H} \to \mathbb{H}$ tal que $v \mapsto \mathbf{q}v\overline{\mathbf{q}}$ es lineal.

Como $p(\mathbf{q})(v) = \mathbf{q}v\overline{\mathbf{q}}$ define bien un mapa si se toma como codominio de p a las transformacione lineales de $\mathbb{H}$ en $\mathbb{H}$, hay que ver que si se restringe $p(\mathbf{q})$ a $Im(\mathbb{H})$, se obtiene una transformación $p(\mathbf{q}) : Im(\mathbb{H}) \to Im(\mathbb{H})$ ortogonal que preserva el sentido.

Si $\mathbf{q} = a + bi + cj + dk \in S^3$, si calcularamos $p(\mathbf{q})(i)$, $p(\mathbf{q})(j)$, $p(\mathbf{q})(k)$ usando la expresión (C.1) se obtiene

- para i:

$$\begin{aligned}
\mathbf{q}i\overline{\mathbf{q}} = (a + bi + cj + dk)i(a - bi - cj - dk) = \\
= (ai - b - ck + dj)(a - bi - cj - dk) = \\
= ab - ba - cd + dc + \\
+i(a^2 + b^2 - c^2 - d^2) + \\
+j(ad + bc + cb + da) + \\
+k(-ac + bd - ca + db) = \\
= i(a^2 + b^2 - c^2 - d^2) + j2(ad + bc) + k2(bd - ac) \in Im(\mathbb{H})
\end{aligned} \tag{C.2}$$

- para j:

$$\begin{aligned}
\mathbf{q}j\overline{\mathbf{q}} = (a + bi + cj + dk)j(a - bi - cj - dk) = \\
= (aj + bk - c - di)(a - bi - cj - dk) = \\
= ac + bd - ac - db + \\
+i(-ad + bc + cb - da) + \\
+j(a^2 - b^2 + c^2 - d^2) + \\
+k(ab + ba + cd + dc) + \\
= i2(bc - ad) + j(a^2 - b^2 + c^2 - d^2) + k2(cd + ab) \in Im(\mathbb{H})
\end{aligned} \tag{C.3}$$

- para k:

$$\begin{aligned}
\mathbf{q}k\overline{\mathbf{q}} = (a + bi + cj + dk)k(a - bi - cj - dk) = \\
= (ak - bj + ci - d)(a - bi - cj - dk) = \\
= ad - bc + cb - da + \\
+i(ac + bd + ca + db) + \\
+j(-ab - ba + cd + dc) + \\
+k(a^2 - b^2 - c^2 + d^2) = \\
= i2(bd + ac) + j2(-ad - ba) + k(a^2 - b^2 - c^2 - d^2) \in Im(\mathbb{H})
\end{aligned} \tag{C.4}$$

Por lo que, usando linealidad del mapa $v \mapsto \mathbf{q}v\overline{\mathbf{q}}$, si $v = v_1 i + v_2 j + v_3 k \in Im(\mathbb{H})$, tendremos entonces que

$$p(\mathbf{q})(v) = v_1 \underbrace{p(\mathbf{q})(i)}_{\in Im(\mathbb{H})} + v_2 \underbrace{p(\mathbf{q})(j)}_{\in Im(\mathbb{H})} + v_3 \underbrace{p(\mathbf{q})(k)}_{\in Im(\mathbb{H})} \in Im(\mathbb{H})$$

Es decir que $p(\mathbf{q}) : Im(\mathbb{H}) \to Im(\mathbb{H})$ es una transformación bien definida, y resta ver que efectivamente $p(\mathbf{q})$ es ortogonal y preserva orientación.

Para ver que es ortogonal, observemos que si $\mathbf{q} \in S^3$ y $v \in Im(\mathbb{H})$, entonces

$$|p(\mathbf{q})(v)|^2 = p(\mathbf{q})(v) \cdot \overline{p(\mathbf{q})(v)} = \mathbf{q}v\overline{\mathbf{q}} \cdot \underbrace{\overline{\mathbf{q}v\overline{\mathbf{q}}}}_{\overline{\overline{\mathbf{q}}}\cdot\overline{\mathbf{q}v}} = \mathbf{q}v \cdot \underbrace{\overline{\mathbf{q}} \cdot \overline{\overline{\mathbf{q}}}}_{|\overline{\mathbf{q}}|^2 = |\mathbf{q}|^2 = 1} \cdot \underbrace{\overline{\mathbf{q}v}}_{\overline{v}\cdot\overline{\mathbf{q}}} =$$

$$= \mathbf{q} \cdot \underbrace{v \cdot \overline{v}}_{|v|^2 \in \mathbb{R}} \cdot \overline{\mathbf{q}} = |v|^2 \underbrace{\mathbf{q} \cdot \overline{\mathbf{q}}}_{|\mathbf{q}|^2 = 1} = |v|^2$$

$$\Rightarrow |p(\mathbf{q})(v)|^2 = |v|^2$$

Es decir que $p(\mathbf{q})$ es ortogonal ($p(\mathbf{q}) \in O(3)$).

Resta ver que $p(\mathbf{q})$ preserva orientación. Para eso, observemos que el mapa $p : S^3 \to O(3)$ es continuo (es decir, si se representa la transformación lineal $p(\mathbf{q})$ matricialmente, las componentes de esa matriz varían continuamente con las coordenadas de $\mathbf{q} \in S^3 \subset \mathbb{H}$), y además si se considera el cuaternión $\mathbf{q} = 1 = 1 + 0i + 0j + 0k \in S^3$ entonces $p(1) : Im(\mathbb{H}) \to Im(\mathbb{H})$ es la identidad pues

$$p(1)(v) = 1 \cdot v \cdot 1^{-1} = v$$

Por lo tanto, como el mapa $p : S^3 \to O(3), \mathbf{q} \mapsto p(\mathbf{q})$ es continuo y S^3 es conexo, entonces $p(S^3) \subset O(3)$ también es conexo, y como $O(3)$ tiene 2 componentes conexas (las transformaciones ortogonales de determinante positivo $SO(3)$, y las de determinante negativo $-SO(3)$), tendremos que $p(S^3)$ estará contenida en una de estas 2 componentes conexas. Como $p(1) = id \in SO(3)$, tendremos entonces que $p(S^3) \subset SO(3)$, es decir que el mapa $p : S^3 \mapsto SO(3)$ está bien definido así.

C.2. Utilización de Cuaterniones para Imponer Cambios en la Actitud de un Dron

Si se elige un vector unitario $u = (u_x, u_y, u_z) \in \mathbb{R}^3$ que representará un eje de giro, y un ángulo de giro θ, se define el cuaternión de rotación asociado como

$$\mathbf{q}_{u,\theta} = cos\left(\frac{\theta}{2}\right) + (u_x i + u_y j + u_z k) sen\left(\frac{\theta}{2}\right). \tag{C.5}$$

El giro según el vector u de ángulo θ (en el sentido dado por la regla de la mano derecha) corresponde con $p(\mathbf{q}_{u,\theta}) \in SO(3)$.

Se puede expresar el cuaternión de la ecuación (C.5) en términos de los ángulos que forma el vector u con los ejes coordenados. Para hacer esta representación, basta observar que las componentes del vector unitario $u = (u_x, u_y, u_z)$, son los cosenos de los ángulos que forma el eje de rotación con cada uno de los ejes coordenados, pues si $\beta_x, \beta_y, \beta_z$ son los ángulos que forma u con los vectores canónicos e_x, e_y, e_z respectivamente, entonces:

$$\begin{cases} u_x =< u, e_x >= \frac{<u,e_x>}{|u|.|e_x|} = cos(\beta_x) \\ u_y =< u, e_y >= \frac{<u,e_y>}{|u|.|e_y|} = cos(\beta_y) \\ u_z =< u, e_z >= \frac{<u,e_z>}{|u|.|e_z|} = cos(\beta_z) \end{cases}$$

Es decir que una rotación de ángulo θ, con respecto a un eje que forma ángulos $(\beta_x, \beta_y, \beta_z)$ respecto de los ejes coordenados, se obtiene calculando $p(\mathbf{q}_{u,\theta})$ siendo $\mathbf{q}_{u,\theta}$ el cuaternión dado por

$$\mathbf{q}_{u,\theta} = cos\left(\frac{\theta}{2}\right) + \Big(cos(\beta_x)i + cos(\beta_y)j + cos(\beta_z)k\Big) sen\left(\frac{\theta}{2}\right)$$

A partir de esto, se podrán imponer cambios en la actitud del dron usando cuaterniones, de modo tal que el cuaternión representa siempre un giro a partir de su acción por conjugación p [41].

Esto funciona considerando una base ortonormal fija orientada positivamente $\{\hat{e}_1, \hat{e}_2, \hat{e}_3\}$ de $\mathbb{R}^3$, y otra base ortonormal orientada positivamente $\{\hat{e}'_1, \hat{e}'_2, \hat{e}'_3\}$ (la cual se querrá que sea solidaria al dron), y se considera la transformación lineal $T : \mathbb{R}^3 \to \mathbb{R}^3$ que mapea $\hat{e}_i \mapsto \hat{e}'_i$ para todo $i \in \{1, 2, 3\}$. Considerando $\mathbf{q}$ tal que $p(\mathbf{q}) = T$, se le pasa al controlador de vuelo el cuaternión $\mathbf{q}$ como parámetro de actitud.

En la controladora de vuelo usada, los vectores de la base $\{\hat{e}_1, \hat{e}_2, \hat{e}_3\}$ apuntan hacia el Este, Norte y arriba respectivamente.

Al transmitirle este cuaternión $\mathbf{q}$ a la controladora de vuelo, el dron se orientará de manera que el drón oriente sus ejes principales según $\{\hat{e}'_1, \hat{e}'_2, \hat{e}'_3\}$

Esta página ha sido intencionalmente dejada en blanco.

Apéndice D

Constantes Elegidas

Constante	Valor
Estrategia de campos potenciales	
$k_{a_{min}}$	1
$k_{a_{max}}$	2
$d_{k_{a_{min}}}$	$6m$
$k_{r_{min}}$	2
$k_{r_{max}}$	100
p	2
θ_{rep}	$45°$
d_{rep}	$11m$
Estrategia de tres estados	
v_{riesgo}	$0{,}1m/s$
θ_{riesgo}	$50°$
R_{riesgo}	$15m$
R_{int}	$1{,}6m$
h_{min}	$1m$
d_{segura}	$2m$
Formación deseada	
d	2m
θ un seguidor	$0°$
θ (simulación) dos seguidores	$\pm 30°$
θ (campo) dos seguidores	$\pm 60°$
Cambio de coordenadas	
R_e	$6378137{,}0m$
R_p	$6356742{,}31424m$

Tabla D.1: Tabla de constantes utilizadas para el cálculo de la estrategia de campos potenciales, la estrategia de tres estados y el cambio de coordenadas.

Esta página ha sido intencionalmente dejada en blanco.

Apéndice E

Biblioteca ENDRA_lib

Se creó una biblioteca de funciones en lenguaje C++ que permitieron llevar a cabo las tareas necesarias, como generar los campos de potencias (atractores y repulsores), y hacer cambios de coordenadas para generar un sistema cartesiano absoluto a partir de la latitud, longitud y altura sobre el nivel del mar.

E.1. Descripción de las Funciones Definidas en la Biblioteca

En primera instancia, se generaron dos **estructuras**:

- **R2**: dispone de dos variables (x, y), de tipo **double** cada una.
- **R3**: dispone de tres variables (x, y, z), de tipo **double** cada una.

Las principales **funciones** declaradas en la biblioteca fueron:

- **double** phi_from_latitude(**double** latitude): A partir de la latitud, retorna φ siguiendo la ecuación (B.3).
- **double** latitude_from_phi(**double** phi): Función inversa de phi_from_latitude.
- **double** x_from_LLA(**double** theta_0, **double** phi_0, **double** h_0, **double** theta, **double** phi, **double** h): A partir de un punto sobre la Tierra con coordenadas según la parametrización (B), y un punto de operación (centro del Taylor) en las mismas coordenadas, se retorna la componente en x del sistema cartesiano generado a partir de la ecuación (B.2).

- **double** y_from_LLA(**double** theta_0, **double** phi_0, **double** h_0, **double** theta, **double** phi, **double** h): A partir de un punto sobre la Tierra con coordenadas según la parametrización (B), y un punto de operación (centro del Taylor) en las mismas coordenadas, se retorna la componente en y del sistema cartesiano generado a partir de la ecuación (B.2).

- **double** z_from_LLA(**double** theta_0, **double** phi_0, **double** h_0, **double** theta, **double** phi, **double** h): A partir de un punto sobre la Tierra con coordenadas según la parametrización (B), y un punto de operación (centro del Taylor) en las mismas coordenadas, se retorna la componente en z del sistema cartesiano generado a partir de la ecuación (B.2)

- **double** phi_from_cartesian(**double** theta_0, **double** phi_0, **double** h_0, **double** x, **double** y, **double** z): A partir del punto del centro del Taylor y de las coordenadas de un punto en el sistema cartesiano, devuelve la coordenada φ de ese punto (correspondiente a la parametrización (B)).

- **double** theta_from_cartesian(**double** theta_0, **double** phi_0, **double** h_0, **double** x, **double** y, **double** z): A partir del punto del centro del Taylor y de las coordenadas de un punto en el sistema cartesiano, devuelve la coordenada θ de ese punto (correspondiente a la parametrización (B)).

- **double** h_from_cartesian(**double** theta_0, **double** phi_0, **double** h_0, **double** x, **double** y, **double** z): A partir del punto del centro del Taylor y de las coordenadas de un punto en el sistema cartesiano, devuelve la coordenada h de ese punto (correspondiente a la parametrización (B)).

- R3 cartesian_from_LLA(sensor_msgs::NavSatFix coord, sensor_msgs::NavSatFix coord_0): A partir de un punto en coordenadas $(latitud, longitud, altura)$ y el centro del Taylor en las mismas coordenadas, devuelve la posición en el sistema cartesiano generado por la ecuación (B.2).

- geographic_msgs::GeoPoseStamped LLA_from_cartesian(R3 xyz, sensor_msgs::NavSatFix coord_0): A partir de un punto en el sistema cartesiano generado a partir de (B.2), y del centro del Taylor en $(latitud, longitud, altura)$, devuelve la $(latitud, longitud, altura)$ del punto en cuestión.

- R2 xy_projection(R3 xyz): Se queda con las componentes en x e y de un elemento de tipo R3.

- **double** norma(R2 vector): Devuelve la norma usual (euclídea) de un elemento de tipo R2.

- **double** interno(R2 vector1, R2 vector2): Realiza el producto interno usual entre dos elementos de tipo R2.

- R3 Geomsg_to_R3(geometry_msgs::PoseStamped pose): Se queda con la componente de posición de un elemento de tipo geometry_msgs::PoseStamped, transformándola a un elemento de clase R3.

- sumar(R2 vector1, R2 vector2): Suma dos elementos de clase R2 componente a componente.

- R2 restar(R2 vector1, R2 vector2): Hace la resta entre las componentes de dos vectores de clase R2.

- R2 vec_por_num(R2 vector, **double** a): Multiplica todas las componentes de un vector tipo R2 por un elemento de tipo double.
- R2 girar(R2 vector, **double** angulo): Realiza una rotación centrada en el origen en sentido anihorario, de un vector de clase R2.
- R2 atractor(R2 posicion, R2 objetivo, **double** Kat): Devuelve las componentes de un campo atractor siguiendo la ecuación (3.1).
- R2 repulsor_radial(R2 posicion, R2 obstaculo, **double** Krep): Devuelve las componentes de un campo repulsor radias ($\theta_{rep} = 0$) según la ecuación (3.2).
- **double** angulo_repulsion (R2 posicion, R2 obstaculo,
  R2 velocidad, R2 velocidad_obstaculo): Calcula el ángulo de giro correspondiente al campo repulsor, siguiendo la ecuación (3.7).
- R2 repulsor (R2 posicion, R2 obstaculo, R2 velocidad,
  R2 velocidad_obstaculo, **double** Krep): Genera el campo repulsor de la ecuación (3.2) y teniendo en cuenta el ángulo de repulsión (3.7).
- **double** get_ka(**double** xy_distance, **double** xy_velL, **double** d): Consigue el factor multiplicativo k_a de los campos atractores, respetando la ecuación (3.4),
- **double** get_kr(R2 xy, R2 obst_xy, R2 vel_xy, R2 obst_vel_xy): Consigue el factor multiplicativo k_r de los campos repulsores, a partir de la ecuación (3.6).

Esta página ha sido intencionalmente dejada en blanco.

Apéndice F

Planificación del Proyecto

La planificación inicial del proyecto se realizó a principios de marzo del 2020, previo a la pandemia ocasionada por el virus COVID-19. Este suceso causó que el avance del proyecto no siguiera la planificación inicial.

Al comienzo del proyecto, se dividieron las actividades a realizar en los siguientes módulos:

1. Aprendizaje de ROS
2. Ensamblado de los drones
3. Modelado Físico-Matemático
4. Diseño de acciones de control
5. Simulaciones en ROS
6. Pruebas de campo
7. Documentación

De estos módulos, el que se vio más afectado por la pandemia fue el del ensamblado de los drones ya que a comienzos de la misma, por razones de seguridad, se suspendió la asistencia a la facultad, causando demoras en el comienzo del ensamblado. Luego, con los permisos necesarios, se pudieron comenzar las reuniones en facultad, aunque sólo una vez por semana, por lo que esta actividad tuvo una duración mayor a la planeada en un principio. Para que no surgieran demoras en los módulos restantes, se paralelizó el armado de los drones con las actividades que no requerían de las aeronaves funcionando.

Por otro lado, para el módulo de las pruebas de campo, se invirtió una gran cantidad de tiempo por causa de diversos factores. Por un lado se debe contar con condiciones climáticas adecuadas, principalmente poco viento y sin lluvias. Luego, existe el factor de la duración de las baterías, los drones son capaces de realizar cierta cantidad de pruebas, pero luego se deben recargar las baterías, haciendo el proceso de pruebas un poco más lento. Y el factor más importante es que existe una gran diferencia entre las simulaciones y la realidad. En muchas ocasiones sucedió

que los algoritmos en las simulaciones funcionaban de buena manera, pero en las pruebas de campo los resultados eran muy distintos, haciendo que las iteraciones para corregir los mismo fueran un proceso lento y no tan sencillo.

Finalmente, la tarea de documentar todos los avances obtenidos a lo largo del proyecto, en varias oportunidades se vio suspendida, para priorizar el avance en tiempo y forma de otro módulos. En primera instancia, se había planificado escribir la documentación regularmente, esto no se logró cumplir, por lo que hubo meses en los cuales la prioridad era documentar. De todas formas se logró culminar con esta tarea sin mayores inconvenientes, obteniendo como resultado el presente documento.

A pesar de las demoras generadas por los factores mencionados, se logró finalizar el proyecto cumpliendo ampliamente los objetivos planteados en tiempo y forma.

Referencias

[1] Enable packet forwarding. `https://linuxconfig.org/how-to-turn-on-off-ip-forwarding-in-linux`. Página web visitada el 2 de febrero de 2021.

[2] I2C Interface Wiring. `https://docs.px4.io/master/en/sensor/lidar_lite.html`. Página web visitada el 2 de febrero de 2021.

[3] MAVLink Router Configuration. `https://github.com/mavlink-router/mavlink-router`. Página web visitada el 2 de febrero de 2021.

[4] Adnan Ashraf, Amin Majd, and Elena Troubitsyna. Towards a realtime, collision-free motion coordination and navigation system for a uav fleet. *Fifth European Conference on the Engineering of Computer-Based Systems (ECBS 2017)*, pages 1–9, 08 2017.

[5] Argel Bandala, Elmer Dadios, Ryan Vicerra, and Laurence Gan Lim. Swarming algorithm for unmanned aerial vehicle (uav) quadrotors: Swarm behavior for aggregation, foraging, formation, and tracking. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, 18:745–751, 09 2014.

[6] Josep Paradells Aspas Carles Gómez i Montenegro. Redes ad-hoc: El próximo reto. *Wireless Networks Group, Entel Dept., Technical University of Catalonia (UPC)*, 6 2004.

[7] CB Insights. Still Soaring: Deals To Drone Startups On Track For 6th Annual High. `https://www.cbinsights.com/research/drones-startup-funding/`, 2017. Página web visitada el 11 de enero de 2021.

[8] Xi Chen, Jun Tang, and Songyang Lao. Review of unmanned aerial vehicle swarm communication architectures and routing protocols. *Applied Sciences*, 10:3661, 05 2020.

[9] Daegyun Choi, Kyuman Lee, and Donghoon Kim. Enhanced Potential Fields-Based Collision Avoidance for Unmanned Aerial Vehicles in a Dynamic Enviroment. *AIAA Scitech Forum*, 01 2020.

[10] Amr Elhennawy and Maki Habib. Nonlinear robust control of a quadcopter: Implementation and evaluation. *IECON 2018 - 44th Annual Conference of the IEEE Industrial Electronics Society*, pages 3782–3787, 10 2018.

[11] Jang-Ho Cho, Dong-Sung Pae, Myo Taeg Lim, and Tae-Koo Kang. A Real-Time Obstacle Avoidance Method for Autonomous Vehicles Using an Obstacle-Dependent Gaussian Potential Field. *Advanced Transportation, August 2018*, 08 2018.

[12] Kai Chang, Yuanqing Xia, and Kaoli Huang. UAV formation control design with obstacle avoidance in dynamic three-dimensional environment. *Springer Plus, vol. 5, no. 1*, 07 2016.

[13] Rudolph Emil Kalman. New approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, pages 35–45, March 1, 1960.

[14] Steven M. Kay. Fundamentals of statistical signal processing, volume i: Estimation theory. In *Fundamentals of Statistical Signal Processing, Volume I: Estimation Theory*, pages 419–478. Prentice Hall, 1994.

[15] Fangzhen Lin. Analysis and verification of multi-rotors attitude control algorithms in pixhawk. In *Proceedings of the 2016 6th International Conference on Advanced Design and Manufacturing Engineering (ICADME 2017)*, pages 612–618. Atlantis Press, 2017/07.

[16] MAVLink. Developers Guide Introduction. `https://mavlink.io/en/`. Página web visitada el 25 de enero de 2021.

[17] MAVLink Router. Github Repository. `https://github.com/mavlink-router/mavlink-router#mavlink-router`. Página web visitada el 23 de febrero de 2021.

[18] L. Meier, D. Honegger, and M. Pollefeys. Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6235–6240, 2015.

[19] National Aeronautics and Space Administrations. NASA Armstrong Fact Sheet: Environmental Research Aircraft and Sensor Technology (ERAST. `https://www.nasa.gov/centers/armstrong/news/FactSheets/FS-020-DFRC.html`, 2014. Página web visitada el 11 de enero de 2021.

[20] L.R. Newcome. *Unmanned Aviation: A Brief History of Unmanned Aerial Vehicles*. EngineeringPro collection. American Institute of Aeronautics and Astronautics, 2004.

[21] Kenzo Nonami. Prospect and recent research & development for civil use autonomous unmanned aircraft as uav and mav. *Journal of system Design and Dynamics*, 1(2):120–128, 2007.

[22] José María Cabrera Peña, L.E.T. Medina, Ricardo Aguasca Colomo, and Blas José Galván González. Technology review, standards and applications of unmanned aerial vehicle in engineering (part 1). *Dyna (Bilbao)*, 91, 2016.

[23] PX4. Architectural Overview. https://docs.px4.io/master/en/concept/architecture.html#px4-architectural-overview. Página web visitada el 2 de febrero de 2021.

[24] PX4. Controller Diagrams. https://docs.px4.io/master/en/flight_stack/controller_diagrams.html#multicopter-control-architecture. Página web visitada el 2 de febrero de 2021.

[25] PX4. Failsafes Configuration. https://docs.px4.io/master/en/config/safety.html#safety-configuration-failsafes. Página web visitada el 2 de junio de 2021.

[26] PX4. Heading and Directions. https://docs.px4.io/master/en/getting_started/px4_basic_concepts.html#heading-and-directions. Página web visitada el 2 de febrero de 2021.

[27] PX4. MAVROS Installation. https://docs.px4.io/master/en/ros/mavros_installation.html. Página web visitada el 26 de enero de 2021.

[28] PX4. Pixhawk 1 Specifications. https://docs.px4.io/master/en/flight_controller/pixhawk.html. Página web visitada el 31 de enero de 2021.

[29] PX4. Quadrotor X Frame Reference. https://docs.px4.io/master/en/airframes/airframe_reference.html#quadrotor-x. Página web visitada el 17 de febrero de 2021.

[30] PX4. SITL Simulation Environment. https://docs.px4.io/master/en/simulation/#sitl-simulation-environment. Página web visitada el 17 de febrero de 2021.

[31] PX4. Using the ECL EKF. https://docs.px4.io/master/en/advanced_config/tuning_the_ecl_ekf.html. Página web visitada el 28 de Abril de 2021.

[32] QGroundControl. User Guide. https://docs.qgroundcontrol.com/master/en/index.html#qgroundcontrol-user-guide. Página web visitada el 25 de enero de 2021.

[33] RadioLink. R9DS Radio Reciever. http://www.radiolink.com.cn/doce/product-detail-120.html. Página web visitada el 31 de enero de 2021.

[34] ROS. Documentation. http://wiki.ros.org/Documentation. Página web visitada el 26 de enero de 2021.

[35] ROS. Main Concepts. http://wiki.ros.org/ROS/Concepts. Página web visitada el 26 de enero de 2021.

[36] ROS. MAVROS Wiki. http://wiki.ros.org/mavros. Página web visitada el 26 de enero de 2021.

[37] ROS. rqt_graph. `http://wiki.ros.org/rqt_graph`. Página web visitada el 30 de marzo de 2021.

[38] ROS. Running ROS across multiple machines. `http://wiki.ros.org/ROS/Tutorials/MultipleMachines`. Página web visitada el 29 de enero de 2021.

[39] ROS. Tutorials. `http://wiki.ros.org/ROS/Tutorials`. Página web visitada el 26 de enero de 2021.

[40] Anam Tahir, Jari Böling, Mohammad-Hashem Haghbayan, Hannu Toivonen, and Juha Plosila. Swarms of unmanned aerial vehicles – a survey. *Journal of Industrial Information Integration*, 16 (100106), 10 2019.

[41] Yan-Bin Jia. Quaternions and Rotations. `http://graphics.stanford.edu/courses/cs348a-17-winter/Papers/quaternion.pdf`. Página web visitada el 27 de marzo de 2021.

[42] Zhihao Xu, Robin Hess, and Klaus Schilling. Constraints of Potential Field for Obstacle Avoidance on Car-like Mobile Robots. *IFAC Proceedings Volumes, vol. 45, no. 4*, 2012.

Esta es la última página.
Compilado el miércoles 9 junio, 2021.
http://iie.fing.edu.uy/