La versión digital de esta tesis está protegida por la Ley de Derechos de Autor del Ecuador.

Los derechos de autor han sido entregados a la "ESCUELA POLITÉCNICA NACIONAL" bajo el libre consentimiento del (los) autor(es).

Al consultar esta tesis deberá acatar con las disposiciones de la Ley y las siguientes condiciones de uso:

- Cualquier uso que haga de estos documentos o imágenes deben ser sólo para efectos de investigación o estudio académico, y usted no puede ponerlos a disposición de otra persona.

- Usted deberá reconocer el derecho del autor a ser identificado y citado como el autor de esta tesis.

- No se podrá obtener ningún beneficio comercial y las obras derivadas tienen que estar bajo los mismos términos de licencia que el trabajo original.

El Libre Acceso a la información, promueve el reconocimiento de la originalidad de las ideas de los demás, respetando las normas de presentación y de citación de autores con el fin de no incurrir en actos ilegítimos de copiar y hacer pasar como propias las creaciones de terceras personas.

Respeto hacia sí mismo y hacia los demás.

ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE ALGORITMOS PARA EL SEGUIMIENTO DE TRAYECTORIAS DE UN QUADROTOR UTILIZANDO TÉCNICAS MODERNAS DE CONTROL CON ÁLGEBRA LINEAL

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y CONTROL

LUIS MIGUEL VARGAS FONSECA
email: lmiguelvargasf@gmail.com

DIRECTOR: GUSTAVO JUAN EDUARDO SCAGLIA
email:gscaglia@unsj.edu.ar

CODIRECTOR: JORGE ANDRÉS ROSALES ACOSTA

Quito, Septiembre 2015

DECLARACIÓN

Yo , LUIS MIGUEL VARGAS FONSECA, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedo mis derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

L. Miguel Vargas F.

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Luis Miguel Vargas Fonseca, bajo mi supervisión.

Gustavo Scaglia	**Andrés Rosales**
DIRECTOR DEL PROYECTO	**CODIRECTOR DEL PROYECTO**

AGRADECIMIENTO

Quiero agradecerle a mi madre, Mirian Fonseca, porque gracias a su esfuerzo, lucha y paciencia constante he podido alcanzar todas las metas propuestas en mi vida. Quiero agradecer a mi abuelita, Melania Lugo, por todo el cariño y apoyo que me brindó mientras vivía. Agradezco a mi tía, Patricia Fonseca, por haber sido no solo una segunda madre, sino también una amiga con la que siempre he podido contar.

Quiero expresar un profundo agradecimiento a Gustavo Scaglia quien de manera desinteresada y carismática me transmitió parte de sus vastos conocimientos, demostrándome al mismo tiempo el gran potencial de la rama de control. Gracias Gus, por toda la paciencia y tu tiempo, y gracias sobretodo por siempre haber confiado en que era capaz de culminar este proyecto.

Quiero también agradecer a Andrés Rosales por su apoyo y confianza incondicionales. Gracias, Doc, por su forma particular y divertida de conducir una clase, y gracias especialmente porque siempre al solicitar su ayuda la respuesta era un sí automático.

Quiero agradecer a Cesar Salazar por darme la oportunidad de aprender sobre el desarrollo de software, pero sobretodo por ser una grandiosa persona siempre dispuesta a tender una mano a quien la necesite.

Expreso mi más sincera gratitud y reconocimiento a David J. Malan por permitir que cualquier persona tenga acceso gratuitamente a CS50, un curso introductorio a Ciencias de la Computación que ha cambiado la vida de muchas personas, incluyendo la mía.

Finalmente, doy las gracias a todos mis amigos por haber estado conmigo en los buenos y malos momentos, especialmente a David Ochoa, Sika Abarca, Diego Vaca y Juan Casco.

Miguel

DEDICATORIA

A mi madre Mirian a quien le debo todo en mi vida y a mi abuelita Melania quien tras una larga lucha hoy descansa en paz.

Miguel

CONTENIDO

1 MARCO TEÓRICO **1**

1.1 ¿QUÉ ES UN QUADROTOR? 1

1.1.1 UAVs, UASs y DRONES 1

1.1.1.1 UAVs 1

1.1.1.2 UASs 1

1.1.1.3 Drones 2

1.2 HISTORIA DE LOS DRONES 2

1.2.1 PRIMEROS INTENTOS 2

1.2.1.1 Nikola Tesla y el primer bote a radio control 2

1.2.1.2 Gyroplane No.I, No.II y No. IIbis 2

1.2.1.3 Oehmichen No.2 y el helicóptero de Bothezat 2

1.2.1.4 Covertawings Model A Quadrotor y Curtiss-Wright VZ-7 3

1.2.2 UAVs MILITARES 3

1.2.2.1 Drones en la Primera Guerra Mundial 3

1.2.2.2 Drones en la Segunda Guerra Mundial 4

1.2.2.3 UAVs durante la Guerra Fría 4

1.2.3 DRONES ACTUALES 5

1.2.3.1 Drones de Uso Doméstico 5

1.3 CLASIFICACIÓN DE LOS MULTICÓPTEROS 7

1.3.1 TRICÓPTERO 8

1.3.2 CUADRICÓPTERO 8

1.3.3 Y4 8

1.3.4 HEXACÓPTERO 8

1.3.5 Y6 9

1.3.6 MULTICÓPTERO 9

1.4 APLICACIONES DE LOS DRONES 10

1.4.1 FOTOGRAFÍA 10
1.4.2 AGRICULTURA 10
1.4.3 ALIVIO DE DESASTRES 11
1.4.3.1 Detección de Incendios Forestales 11
1.4.3.2 Búsqueda y Rescate 11
1.5 ÁNGULOS DE NAVEGACIÓN 12
1.5.1 ROLL (ϕ) 13
1.5.2 PITCH (θ) 13
1.5.3 YAW (ψ) 14
1.6 CONTROLADORES CON ÁLGEBRA LINEAL 14
1.6.1 MÉTODOS NUMÉRICOS 15
1.6.1.1 Planteamiento del Problema 16
1.6.1.2 Diseño del Controlador 16
1.6.1.3 Análisis de Error 20
1.6.2 INTERPOLACIÓN LINEAL 22

2 ANÁLISIS DE LA ESTRUCTURA Y PARÁMETROS 25
2.1 QUADROTOR PARROT AR.DRONE 2.0 25
2.1.1 HARDWARE 25
2.1.1.1 Unidad de Medición Inercial 26
2.1.1.2 Altímetro Ultrasónico 27
2.1.1.3 Cámaras 28
2.2 ESTRUCTURA 29
2.2.1 MOVIMIENTOS BÁSICOS 29
2.2.1.1 Throttle 30
2.2.1.2 Roll 30
2.2.1.3 Pitch 31
2.2.1.4 Yaw 31
2.3 PARÁMETROS 32
2.3.1 MASA E INERCIA 32
2.3.2 COEFICIENTES DE IMPULSO Y ARRASTRE 33

3 MODELADO E IDENTIFICACIÓN 34
3.1 MODELADO . . . 34
3.1.1 SISTEMAS DE REFERENCIA . . . 34
3.1.2 MATRIZ DE ROTACIÓN . . . 35
3.1.3 MODELO . . . 36
3.2 IDENTIFICACIÓN Y VALIDACIÓN . . . 38
3.2.1 IDENTIFICACIÓN . . . 38
3.2.2 VALIDACIÓN . . . 38
3.3 COMPARACIÓN DEL MODELO . . . 40
3.3.1 MODELO I . . . 41
3.3.2 MODELO II . . . 42

4 DISEÑO E IMPLEMENTACIÓN DEL CONTROLADOR 43
4.1 DISEÑO DEL CONTROLADOR . . . 43
4.2 SIMULACIÓN DEL CONTROLADOR . . . 46
4.2.1 SIMULACIÓN CONTINUA . . . 46
4.2.2 SIMULACIÓN DISCRETA . . . 48
4.3 IMPLEMENTACIÓN DEL CONTROLADOR . . . 49
4.3.1 DESCRIPCIÓN DEL ALGORITMO . . . 49
4.3.2 APLICACIÓN MÓVIL . . . 54
4.3.2.1 Descripción de la App . . . 54
4.3.3 INTERFAZ GRÁFICA DE USUARIO . . . 57

5 PRUEBAS Y RESULTADOS 59
5.1 SIMULACIONES . . . 59
5.1.1 TRAYECTORIA LINEAL . . . 59
5.1.2 TRAYECTORIA PARABÓLICA . . . 61
5.1.3 TRAYECTORIA SINUSOIDAL . . . 62
5.2 EXPERIMENTACIÓN . . . 64
5.2.1 TRAYECTORIA LINEAL . . . 65
5.2.2 TRAYECTORIA PARABÓLICA . . . 66
5.2.3 TRAYECTORIA SINUSOIDAL . . . 67
5.3 JUSTIFICACIÓN DE ERRORES . . . 69

5.3.1 SENSORES 70
5.3.2 COMUNICACIÓN 70

6 CONCLUSIONES Y RECOMENDACIONES 73
6.1 CONCLUSIONES 73
6.2 RECOMENDACIONES 74

Appendices 79

ANEXO A CÓDIGO DEL CONTROLADOR 80

ANEXO B CÓDIGO DEL MODELO DISCRETO 82

ANEXO C SCRIPT PARA GRAFICAR RESULTADOS 84

ANEXO D FOTOGRAFÍAS DE LA EXPERIMENTACIÓN 87

RESUMEN

En las últimas décadas se ha dado un desarrollo tecnológico a pasos agigantados. Campos considerados ficción como la inteligencia artificial y la robótica han pasado a ser ramas de investigación y gran interés no solo para académicos sino también para aficionados. En particular, en la última década se han realizado una gran cantidad de proyectos con UAVs gracias a que el costo del hardware se ha ido reduciendo paulatinamente.

Generalmente, los proyectos con vehículos aéreos no tripulados están enfocados en tracking, procesamiento de imágenes y vuelos autónomos. En este último tópico se han implementado varios algoritmos de control, principalmente para el seguimiento de trayectorias. Entre los diferentes algoritmos de control se encuentran aquellos que utilizan el álgebra lineal como fundamento, los cuales se caracterizan porque poseen una implementación bastante sencilla; pero además, los resultados que se obtienen tras implementarlos son sobresalientes.

Entre los UAVs, los quadrotores han ganado gran popularidad gracias a su costo moderado en comparación con otros drones y a la facilidad de implementar controladores. En el mercado es posible encontrar varios quadrotores comerciales cuyos precios son asequibles como el AR.Drone 2.0.

El AR.Drone 2.0 posee varios frameworks y APIs para enviar comandos y recibir datos de navegación desde un computador. En este proyecto se ha creado una interfaz gráfica de usuario para ejecutar el seguimiento de trayectorias y el despliegue de resultados. Además, se ha elaborado una aplicación móvil con la finalidad de utilizar la pantalla táctil de un dispositivo móvil para dibujar la trayectoria que el quadrotor tiene que seguir.

Todas estas tecnologías se han integrado para realizar el seguimiento de trayectorias del AR.Drone 2.0 en el plano XY a una altura fija. Además, se ha mostrado la facilidad de simular e implementar controladores con álgebra lineal. Por lo tanto, todos los objetivos planteados se han cumplido cabalmente.

PRESENTACIÓN

El objetivo principal de este trabajo es presentar el diseño y la implementación de un controlador con álgebra lineal en un quadrotor para el seguimiento de una trayectoria preestablecida por el usuario. Este trabajo consta de seis capítulos y cinco anexos los cuales han sido repartidos de la siguiente forma:

En el primer capítulo se describen conceptos fundamentales, para este proyecto, como UAV o quadrotor, y se hace un breve resumen de la historia de los UAVs. Además, se presentan los diferentes tipos de drones, en función de su número de hélices, las aplicaciones con UAVs más populares actualmente y se desarrolla un marco teórico acerca de los controladores con álgebra lineal.

En el segundo capítulo se presenta el hardware del AR.Drone 2.0, como los sensores, motores, cámaras y baterías que este utiliza. Además, se analiza la estructura del drone y sus movimientos básicos. Finalmente, se describen algunos de sus parámetros como la masa, la inercia y los coeficientes de impulso y arrastre.

En el tercer capítulo se realiza el modelado del AR.Drone 2.0, el cual posteriormente es identificado y validado. Además, se hace una comparación entre el modelo utilizado en este proyecto y otros modelos empleados anteriormente.

En el cuarto capítulo se realiza el diseño y también la implementación del controlador. Además, se describe el algoritmo implementado y se presenta una descripción tanto de la interfaz gráfica de usuario como de la aplicación móvil.

En el quinto capítulo se muestran los resultados de las simulaciones y las pruebas experimentales realizadas. Además, se justifican los errores obtenidos en las pruebas en base a información del desarrollador del API utilizado.

Finalmente, en el sexto capítulo se sintetiza lo más relevante del proyecto en las conclusiones y se realizan una serie de sugerencias para aquellas personas que estén interesadas en elaborar un proyecto similar.

En los anexos se presentan los códigos de varios scripts realizados en *MATLAB* y también fotografías de las pruebas realizadas con el AR.Drone 2.0.

CAPÍTULO 1
MARCO TEÓRICO

1.1 ¿QUÉ ES UN QUADROTOR?

Un quadrotor, también conocido como quadrirrotor o quadricóptero, es un helicóptero que posee cuatro propulsores; formados cada uno por un motor y una hélice, para su movimiento. Los quadrotores son considerados multicópteros o multirrotores, es decir, aeronaves de alas giratorias que poseen más de dos propulsores. En ocasiones, los quadrotores son también llamados UAVs, UASs o incluso drones.

1.1.1 UAVs, UASs y DRONES

1.1.1.1 UAVs

UAV proviene de las siglas en inglés de *Unmanned Aerial Vehicle* que quiere decir *Vehículo Aéreo no Tripulado*. Un UAV es una aeronave sin piloto humano a bordo; por lo tanto, el vuelo es controlado de manera autónoma a través de computadores o utilizando un control remoto que es operado por un usuario. [23]

1.1.1.2 UASs

UAS también proviene del inglés, de las siglas de *Unmanned Aerial System* que traducido al español es *Sistema Aéreo no Tripulado*. El término UAS enfatiza la importancia de otros elementos como:

- Aeronave no tripulada.
- Sistema de control, como una Estación de Control en Tierra.
- Link de control, un enlace de datos especializado.

El término UAS fue usado por primera vez por la Administración Federal de Aviación de los Estados Unidos, y fue adoptado por el Departamento de Defensa Americano y la Autoridad de Aviación Civil Británica (*Civil Aviation Authority*). [25]

1.1.1.3 Drones

Drone es el término popular por el cual un UAV o un UAS son conocidos. Etimológicamente, la palabra drone proviene del inglés antiguo *dran* que significa zángano (abeja macho), pero desde el año 1946 se asignó el término a aeronaves sin piloto a bordo. [20]

1.2 HISTORIA DE LOS DRONES

Los drones inicialmente fueron construidos para propósitos militares, y no se los podía adquirir debido a su elevado costo. Sin embargo, gracias al avance tecnológico, hoy en día existen una gran cantidad de drones que son comerciales y que se venden a precios asequibles. [9]

1.2.1 PRIMEROS INTENTOS

1.2.1.1 Nikola Tesla y el primer bote a radio control

Nikola Tesla , en una exhibición en el Madison Square Garden de Nueva York en 1898, utilizó una caja que contenía un transmisor de radio para maniobrar un barco pequeño que flotaba sobre una piscina. Todos los presentes en la exhibición se quedaron completamente sorprendidos ya que en aquel entonces muy pocas personas tenían conocimiento de la existencia de las ondas de radio. Tesla fue el primero en patentar un radiocontrol para vehículos no tripulados, el cual fue crucial para el desarrollo de los UAV actuales. [25]

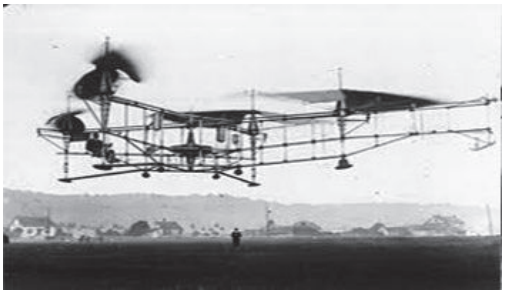
1.2.1.2 Gyroplane No.I, No.II y No. IIbis

Unos años más tarde de la exhibición de Tesla, para ser exactos en 1907, la empresa Breguet Aviation construyó el que podría considerarse el primer quadricóptero al que se bautizó como *Gyroplane No.I*. Esta aeronave fue volada por primera vez el 29 de septiembre de 1907, y alcanzó una elevación de 0.6 metros. [25]

1.2.1.3 Oehmichen No.2 y el helicóptero de Bothezat

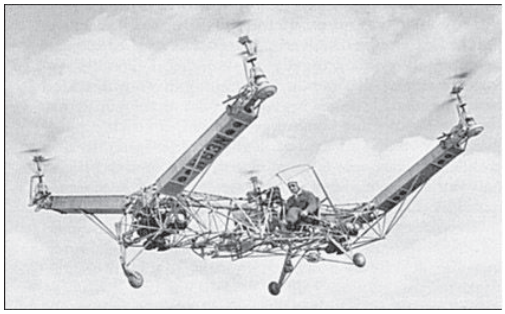
En 1920, Etienne Oehmichen diseñó el Oehmichen No.2, el cual tenía cuatro motores y ocho hélices. Más tarde en 1922, el Dr. George de Bothezat e Ivan Jerome desarrollaron el *helicóptero de Bothezat* el cuál alcanzó una altura de 5 metros y

realizó más de 100 vuelos. En la Figura 1.1 se puede apreciar al Oehmichen No.2 (Figura 1.1a) y al helicóptero de Bothezat (Figura 1.1b).

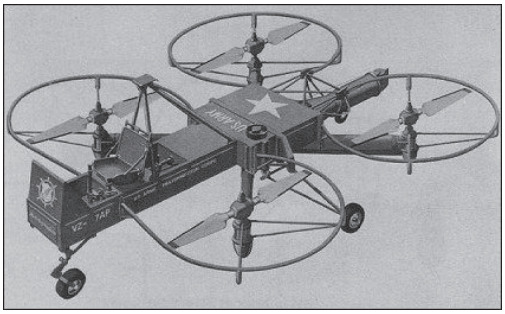


(a) Oehmichen No.2

(b) Helicóptero de Bothezat



(c) Convertawings Model A Quadrotor



(d) Curtiss-Wright VZ-7

Figura 1.1: Primeros intentos en el diseño de un quadrotor, tomados de [25].

1.2.1.4 Covertawings Model A Quadrotor y Curtiss-Wright VZ-7

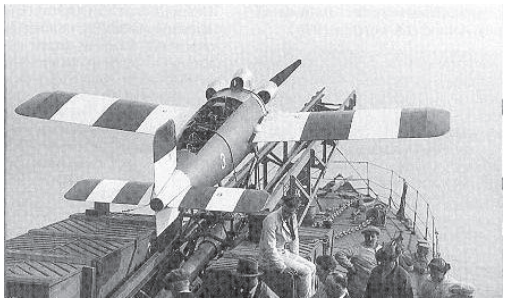
En el año de 1956, *Convertawings Model A Quadrotor* (Figura 1.1c) fue el primer helicóptero impulsado por cuatro motores que presentó un vuelo exitoso hacia adelante. Dos años más tarde, apareció el *Curtiss-Wright VZ-7* (Figura 1.1d), el cual fue diseñado para la Armada Americana por la compañía *Curtiss-Wright*. El*VZ-7* era controlado cambiando la velocidad a la que giraba cada uno de los cuatro motores propulsores. [24]

1.2.2 UAVs MILITARES

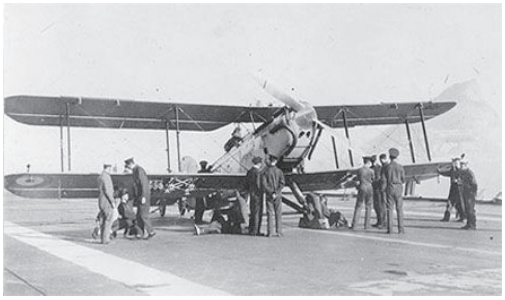
1.2.2.1 Drones en la Primera Guerra Mundial

Las primeras aeronaves sin piloto se construyeron durante la Primera Guerra Mundial. Un ejemplo de estas aeronaves es el aeroplano automático Hewitt-Sperry, también conocido como *la bomba voladora*. Otros ejemplos de UAV creados en esta etapa

son Larynx y Fairey III, desarrollados por la Armada Real y las Fuerzas Británicas respectivamente. [25] En la Figura 1.2 se muestran fotografías de Larynx (Figura 1.2a) y Fairey III (Figura 1.2b).



(a) Larynx.



(b) Fairey III.

Figura 1.2: Drones en la Primera Guerra Mundial, tomados de [25].

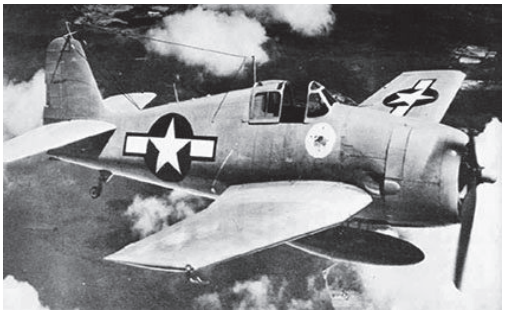
1.2.2.2 Drones en la Segunda Guerra Mundial

Durante la Segunda Guerra Mundial, Industrias Reginald Denny manufacturó más de quince mil drones para la Armada Americana. Otra aplicación que se le dió a los drones en esta etapa fue como pulsorreactores. Un pulsorreactor es un jet que utiliza un motor que descarga un chorro de fluido a gran velocidad para generar un empuje. [26] Pulsorreactores que desarrolló la Marina Americana son el T2D2-1 Katydid, KDD-1 y KDH-1.

1.2.2.3 UAVs durante la Guerra Fría

En esta etapa, una de las aplicaciones de los drones fue utilizarlos con el fin de llamar la atención mientras se hacían operaciones en cubierto. El primer drone que se utilizó como señuelo fue el *ADM-20 Quail*. Este UAV era transportado por el bombardero estratégico *Boeing B-52 Stratofortress*, con el fin de ayudar a penetrar espacio aéreo defendido. Drones como el *B-17 Flying Fortresses* (Figura 1.3a) y el *Grumman F6F Hellcat* (Figura 1.3b) se emplearon para recolección de datos radioactivos en pruebas nucleares. [25]

(a) B-17 Flying Fortresses.



(b) Grumman F6F Hellcat.

Figura 1.3: Drones durante la Guerra Fría, tomado de [25].

1.2.3 DRONES ACTUALES

En las últimas tres décadas, el uso de drones a escala ha llegado a ser cotidiano para un sin número de aplicaciones. Aunque muchas de las tareas que realizan los drones actualmente todavía están ligadas con actividades militares, los drones están formando parte de la vida diaria de las personas. Además, lo más relevante es que hoy en día cualquier persona puede adquirir drones de diferentes tamaños y a precios desde $35. [4]

1.2.3.1 Drones de Uso Doméstico

Actualmente es posible encontrar información para la fabricación de un drone. Por ejemplo, existen páginas web, como DIY Drones, que ofrecen tutoriales y vídeos para la construcción y mantenimiento de drones. También es posible adquirir un drone listo para volarlo. Los precios de los quadrotores varían desde $35, los cuales son utilizados como juguete, hasta más de $8,000, los cuales se emplean en proyectos de investigación. Entre las empresas que más éxito han tenido en la comercialización de drones se encuentran:

- **Parrot:** es una empresa francesa ubicada en París. Su producto destacado es el Parrot A.R. Drone. [27] Este quadrotor (Figura 1.4a) tiene un precio de $300 y se lo puede adquirir en sitios como Amazon o e-Bay. Aunque su estabilidad no es suficientemente robusta, se lo pueden emplear en proyectos de pequeña escala.
- **DJI UAV:** es una empresa china que desarrolla micro vehículos aéreos. [28] Estos

drones son relativamente fáciles de utilizar, y han ganado popularidad entre fotógrafos debido a la capacidad de tomar fotos y grabar vídeos en alta resolución. El drone que más se ha popularizado es el modelo Phantom (Figura 1.4b) el cual es un cuadricóptero empleado principalmente para cinematografía y fotografía aérea.

- **3DRobotics:** es una empresa norteamericana que ofrece varios drones de 4, 6 y hasta 8 hélices. Los precios de estas aeronaves varían entre $500 hasta $1,200. Además, ofrecen la ventaja de poder utilizar el ArduPilot, una plataforma abierta para proyectos con UAVs. [29] En la Figura 1.4c se observa el drone 3DR ISIS.
- **AscTec:** es una empresa alemana que desarrolla drones para proyectos de investigación. Los precios varían entre los $5,000 y $8,000. Aunque estas aeronaves son costosas, ofrecen una amplia gama de aplicaciones y plataformas para su control. En la Figura 1.4d se observa el *ASTEC* Pelican que posee una estructura en forma de torre, la cual le permite acoplar sensores para una mejor respuesta.

(a) ARDrone 2.0, tomado de [27].

(b) Phantom, tomado de [28].

(c) 3DR ISIS, tomado de [29].

(d) ASTEC Pelican, tomado de [5].

Figura 1.4: Drones Comerciales.

1.3 CLASIFICACIÓN DE LOS MULTICÓPTEROS

Existen varias clasificaciones que se han establecido para los drones. Por ejemplo, dependiendo de su misión principal se los puede clasificar en drones *de blanco*, *de combate*, *de logística*, *de investigación o desarrollo*, y *de uso comercial*. También se los puede clasificar en función de su techo y alcance máximo como se indica en la Tabla 1.1, en la cual, *ND* significa que su valor es tan elevado no es necesario considerarlo.

Categoría	**Acrónimo**	**Alcance [km]**	**Altitud de vuelo [m]**	**Autonomía [horas]**	**Máxima carga [kg]**
Micro < 250 g	Micro	< 10	250	1	<5
Mini < 25 kg	Mini	< 10	150 - 300	<2	<30
Alcan. cercano	CR	10 - 30	3,000	2 - 4	150
Alcance corto	SR	30 - 70	3,000	3 - 6	200
Alcance medio	MR	70 - 200	5,000	6 - 10	1,250
Altitud baja	LADP	> 250	50 - 9,000	0.5 - 1	350
Auton. media	MRE	> 500	8000	10 - 18	1,250
Autonomía alta - Altitud baja	LALE	> 500	3000	> 24	< 30
Autonomía alta - Altitud media	MALE	> 500	14,000	24 - 28	1,500
Autonomía alta - Altitud alta	HALE	> 2000	20,000	24 - 28	12,000
Ofensivo	LETH	300	4,000	3 - 4	250
Señuelo	DEC	500	5,000	< 4	250
Estratosférico	STRATO	> 2000	20,000 - 30,000	> 48	ND
EXO-estratosférico	EXO	ND	> 30,000	ND	ND

Tabla 1.1: Clasificación de los UAVs en función de sus características de vuelo.

Una de las formas más prácticas de clasificar a los drones es por el número de hélices que poseen. La cantidad de hélices de un drone puede variar desde dos o tres hasta incluso ocho o más. Del número de hélices dependen características como la estabilidad, el consumo de energía y la cantidad de carga que puede transportar una aeronave.

1.3.1 TRICÓPTERO

Un tricóptero es un drone que posee tres motores. La forma usual de un tricóptero es en "Y", distribuyendo los motores uniformemente a 120 grados, aunque existen modelos cuya forma es en "T". Este drone no es muy robusto y su potencia de elevación sigue siendo reducida. Por lo tanto, tiene que girar rápidamente para mantenerse suspendido en el aire, consumiendo así mucha energía.

1.3.2 CUADRICÓPTERO

Un cuadricóptero, también denominado quadrotor, posee cuatro motores montados en una estructura de cuatro brazos, los cuales están distribuidos simétricamente (cada brazo separado 90 grados). El cuadricóptero es el drone más popular entre investigadores y aficionados y sus configuraciones más habituales son la "X" (equis) y la "+" (plus).

1.3.3 Y4

Esta configuración posee, al igual que el quadrotor, cuatro motores para su propulsión aunque luce como un tricóptero. Esta aeronave tiene dos hélices en la parte trasera, una a continuación de la otra, las cuales permiten controlar el giro sobre el eje perpendicular al piso, y en la parte delantera se ubican las otras dos hélices separadas a 120 grados.

1.3.4 HEXACÓPTERO

Un hexacóptero tiene 6 motores montados en 6 brazos a una separación de 120 grados cada uno. Tres motores del hexacóptero giran en sentido horario y los otros tres en sentido antihorario para poder mantenerse en el aire. Este drone es muy similar al quadotor, pero con mayor estabilidad y un control mucho más simple.

1.3.5 Y6

Un Y6 tiene 6 motores al igual que un hexacóptero. Sin embargo, posee una estructura en forma de "Y". De hecho, su forma es similar a la de un tricóptero, pero el Y6 posee 2 motores en cada brazo. Un motor está ubicado en la parte superior del brazo, mientras que el otro se encuentra en la parte inferior. Ambos motores giran en sentido opuesto el uno del otro. Un Y6 es menos eficiente que un hexacóptero debido a la distribución de los motores.

1.3.6 MULTICÓPTERO

En general, un multicóptero es una aeronáve con dos o más motores para su propulsión. Como se ha mencionado, dependiendo del número de hélices se llaman específicamente: bicóptero, tricóptero, cuadricópero, pentacóptero, hexacóptero u octocóptero. Existen drones con más de ocho hélices, pero usualmente se los denomina como *multicópteros* y se indica el número de hélices que poseen.

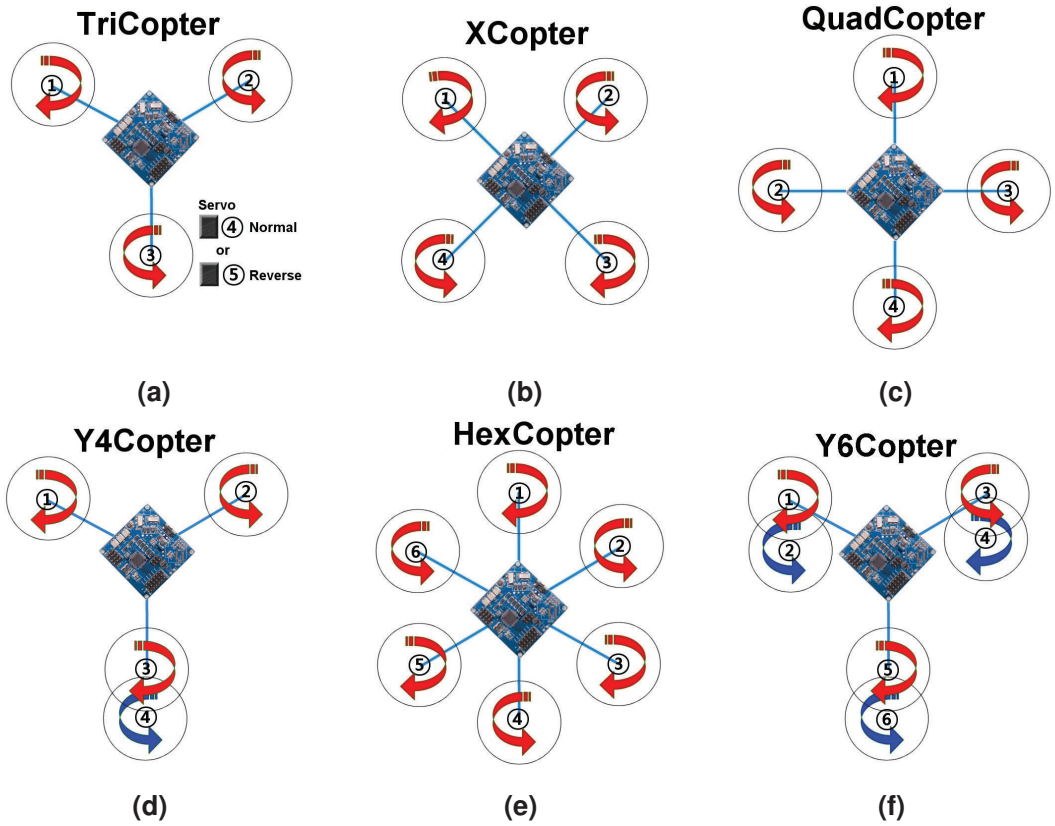


Figura 1.5: Configuración de hélices y mainboard de multicópteros, tomado de [17].

1.4 APLICACIONES DE LOS DRONES

Además de los usos militares, los UAVs poseen también varias aplicaciones civiles en las que los drones son empleados principalmente para actividades donde existe riesgo o peligro para el ser humano. Entre las aplicaciones civiles más populares se encuentran el uso de drones en la fotografía, la agricultura y el alivio de desastres.

1.4.1 FOTOGRAFÍA

La fotografía ha dado un cambio radical gracias a los drones pues estos permiten realizar tomas únicas. Además, la vida humana no se pone en peligro porque el fotógrafo puede estar a una distancia promedio de 15 metros desde donde se está haciendo la captura. Mientras la Figura 1.6a muestra a un fotógrafo utilizando un drone, la Figura 1.6b muestra una fotografía tomada con un drone.

(a) **(b)**

Figura 1.6: Drones empleados en la fotografía, tomado de [23].

1.4.2 AGRICULTURA

Aunque en un inicio parezca que entre los drones y la agricultura no existe ninguna relación, la industria del cultivo pueda obtener muchos beneficios del empleo de drones. Por ejemplo, las aeronaves pueden ser usadas para monitorear los campos, incrementando la producción y ahorrando dinero. Los drones son empleados también para echar sobre las plantaciones agua, fertilizantes y pesticidas con excelente precisión. La Figura 1.7 muestra drones usados en el campo agrícola.

(a) (b)

Figura 1.7: Drones empleados en la agricultura, tomado de [23].

1.4.3 ALIVIO DE DESASTRES

Los UAVs son capaces de transportar medicamentos, y traer de vuelta muestras médicas, hacia y desde regiones muy remotas y difíciles de acceder. Además, las aeronaves se pueden emplear para construir una imagen de la situación y dar recomendaciones de como la gente debe manejar sus recursos y salvar sus vidas.

Los drones tienen un amplio rango de aplicaciones para alivio de desastres, desde entrar en zonas llenas de radiación donde el acceso de personas sería peligroso hasta búsqueda de sobrevivientes a través de un paisaje lleno de escombros. [15]

1.4.3.1 Detección de Incendios Forestales

Los drones se pueden utilizar para la prevención y pronta detección de incendios forestales. La posibilidad de vuelo constante hace que los métodos empleados hasta ahora (helicópteros y observadores) queden obsoletos. Las cámaras y sensores, que proveen servicios de emergencia en tiempo real, son muy útiles para que los bomberos procedan a apagar el fuego.

1.4.3.2 Búsqueda y Rescate

Los drones probablemente jugarán un papel importante en la búsqueda y rescate de personas; de hecho, ya se ha empezado a usar drones en estos campos. Por ejemplo, en mayo de 2013, un herido en un accidente automovilístico fue rescatado por un drone en Saskatchewn, Canada. [19]

1.5 ÁNGULOS DE NAVEGACIÓN

Los ángulos de navegación de un UAV, también conocidos como ángulos de Euler [30], se utilizan para describir la orientación de un cuerpo rígido. Para realizar esta descripción en un espacio Euclidiano de tres dimensiones se necesitan tres parámetros. Los ángulos de Euler representan una secuencia de tres rotaciones elementales, es decir, rotaciones alrededor de los ejes de un sistema de coordenadas, ya que cualquier orientación se puede alcanzar mediante la composición de tres rotaciones. Estas rotaciones pueden ser extrínsecas, si se las hacen al rededor de un sistema coordenado fijo, o intrínsecas, si se las hace alrededor de un sistema coordenado que rota, el cual es inicialmente alineado con uno fijo y modifica su orientación después de cada rotación elemental.

El sistema de coordenadas rotativo está anexado a un cuerpo rígido, y en este caso es llamado sistema coordenado local. Sin considerar la posibilidad de usar dos diferentes convenciones para la definición de ejes de rotación, existen doce posibles secuencias para la rotación de ejes, las cuales están divididas en dos grupos:

i. **Ángulos de Euler:** (z-x-z, x-y-x, y-z-y, z-y-z, x-z-x, y-x-y)

ii. **Ángulos de Tait-Bryan:** (x-y-z, y-z-x, z-x-y, x-z-y, z-y-x, y-x-z)

Los ángulos de Tait-Bryan también son llamados **ángulos de Cardan**, **ángulos naúticos**, **heading**, **elevación** y **bank** o **yaw**, **pitch** y **roll**. [30] En ocasiones, ambas secuencias de ángulos son llamadas ángulos de Euler; en tal caso, las secuencias del primer grupo son denominadas ángulos de Euler propios o clásicos.

Un UAV en vuelo es libre de rotar en tres dimensiones: *pitch* (θ), nariz arriba o abajo alrededor de un eje que va de ala a ala; *yaw* (ψ), nariz a la izquierda o a la derecha alrededor de un eje que va de arriba hacia abajo; y *roll* (ϕ), rotación alrededor del eje que va desde la nariz a la cola. En la Figura 1.8 se muestra los ejes de rotación de estos ángulos de navegación. Estos ejes alternativamente se los conoce como *lateral*, *vertical* y *longitudinal*.

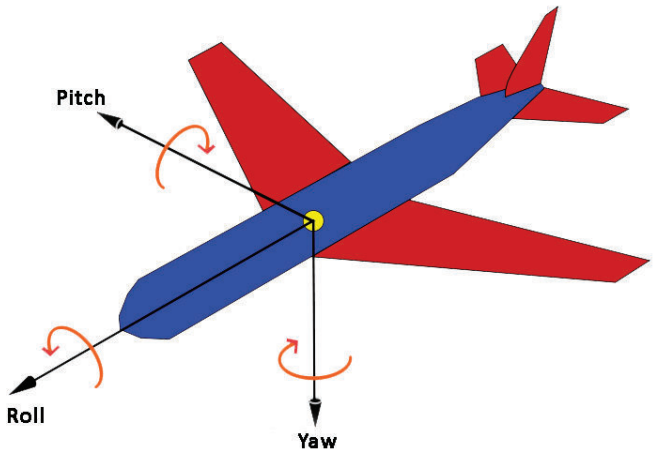


Figura 1.8: Posición de los tres ejes de rotación de una aeronave, tomado de [30].

1.5.1 ROLL (ϕ)

Roll es una rotación alrededor del eje del fuselaje. Este efecto es causado por la operación de los alerones. Para hacer un roll a al izquierda, el alerón izquierdo es elevado y el alerón derecho es descendido. La operación es inversa para hacer un roll a la derecha.

Figura 1.9: Rotación Roll, tomado de [30].

1.5.2 PITCH (θ)

Pitch cambia la elevación o descenso de la nariz de la aeronave. Este efecto es causado por la operación de un elevador. Cuando el elevador es levantado, la fuerza de el flujo de aire empuja la cola hacia abajo, rotando la aeronave alrededor del punto de balance y elevando la nariz.

Figura 1.10: Rotación Pitch, tomado de [30].

1.5.3 YAW (ψ)

Yaw es la rotación horizontal alrededor del eje vertical de la aeronave, y se realiza a través del timón. Si el timón es desviado a la izquierda o derecha, la presión del flujo de aire hace que la nave rote alrededor del eje vertical a la izquierda o derecha respectivamente.

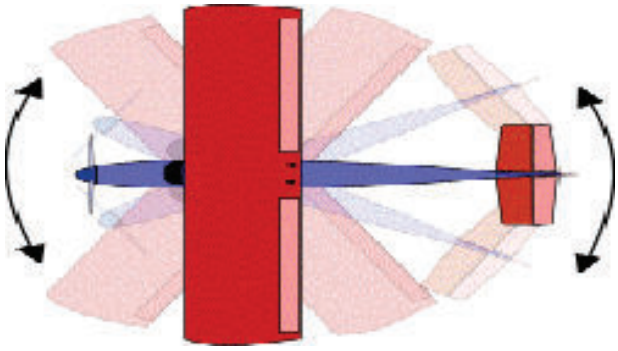


Figura 1.11: Rotación Yaw, tomado de [30].

1.6 CONTROLADORES CON ÁLGEBRA LINEAL

Un controlador con álgebra lineal es aquel que utiliza operaciones definidas en la teoría del álgebra lineal para simplificar procesos matemáticos. Los controladores que emplean métodos numéricos y aquellos basados en interpolación lineal usan álgebra lineal ya que facilita el diseño y simplifica la implementación de los mismos. Estos controladores son utilizados principalmente para resolver problemas de seguimiento de trayectorias debido a su sencillez y buen desempeño.

El proceso para el diseño de un controlador con álgebra lineal, tanto si emplea métodos numéricos como si utiliza interpolación lineal es el siguiente:

1. Aproximación de las variables de estado.
2. Cálculo de las señales de control con el objetivo de minimizar la distancia entre los estados deseados y los del sistema.
3. Análisis de las condiciones bajo las cuales el sistema tiene solución exacta, y posteriormente obtener la trayectoria deseada de algunos estados.
4. Minimización de propiedades adicionales del sistema.

1.6.1 MÉTODOS NUMÉRICOS

Los métodos numéricos se utilizan principalmente para encontrar aproximaciones numéricas de las soluciones de ecuaciones diferenciales ordinarias. Para muchas ecuaciones que no pueden ser resultas analíticamente, una aproximación numérica de la solución es suficiente. Entre los métodos numéricos más populares están el método de Euler y el método de Runge-Kutta de segundo orden.

$$\dot{y} = f(t, y(t)); y(t_0) = y_0 \tag{1.1}$$

donde f es una función definida para todos los números reales y con una condición inicial y_0 para $t = t_0$. Si se considera T_0 como la separación entre dos puntos en la secuencia t_0, $t_1 = t_0 + T_0$, $t_2 = t_0 + 2T_0$, ..., la solución numérica estimada para $y(t_n) = y_n$ está definida por,

$$y_{n+1} = y_n + \int_{nT_0}^{(n+1)T_0} f(y, t)dt \tag{1.2}$$

Aplicando independientemente el método de Euler o el método de Runge-Kutta de segundo orden a la ecuación 1.2 se obtienen los resultados mostrados en las ecuaciones 1.3a y 1.3b respectivammente.

$$y_{n+1} \approx y_n + T_0 \cdot f(y_n, t_n) \tag{1.3a}$$

$$y_{n+1} \approx y_n + \frac{T_0}{2}(f(y_n, t_n) + f(y_{n+1}, t_{n+1})) \tag{1.3b}$$

1.6.1.1 Planteamiento del Problema

A continuación se presenta el diseño de un controlador, para un robot móvil, que emplea métodos numéricos y álgebra lineal. Consideremos la siguiente ecuación diferencial,

$$\dot{y} = f(y, u, t); y(0) = y_0 \tag{1.4}$$

donde y representa la salida del sistema que va a ser controlado, u es la acción de control y t es el tiempo. Los valores de $y(t)$ para tiempo discreto son $t = nT_0$, donde T_0 es el tiempo de muestreo y $n \in \mathbb{Z}^+ \cup \{0\}$, serán denotados como y_n. Entonces, cuando se requiera calcular y_{n+1} conociendo el valor de y_n, la ecuación 1.4 debe ser integrada en el periodo de tiempo $nT_0 \leq t \leq (n+1)T_0$ como se expone a continuación,

$$y_{n+1} = y_n + \int_{nT_0}^{(n+1)T_0} f(y, u, t)dt \tag{1.5}$$

Usando la aproximación de Euler se tiene,

$$y_{n+1} \approx y_n + T_0 \cdot f(y_n, u_n, t_n) \tag{1.6}$$

El empleo de métodos numéricos en la simulación del sistema se basa principalmente en la posibilidad de determinar el estado de un sistema en el instante $n+1$ partiendo del estado, las acciones de control y otras variables en el instante n. [13] Por lo tanto, y_{n+1} puede ser sustituido por el valor de la trayectoria deseada y la acción de control para hacer que la salida del sistema evolucione del valor actual y_n hasta el valor deseado.

1.6.1.2 Diseño del Controlador

Se utilizará un modelo cinemático no lineal para un robot móvil como el de la Figura 1.12, el cuál está representado por el siguiente conjunto de ecuaciones:

$$\begin{cases} \dot{x} = V \cdot \cos\theta \\ \dot{y} = V \cdot \sin\theta \\ \dot{\theta} = \omega \end{cases} \tag{1.7}$$

donde V es la velocidad lineal del robot móvil, ω es la velocidad angular, x y y son las posiciones cartesianas, θ es la orientación del robót móvil, $\{R\}$ es el sistema de referencia inercial, mientras que $\{R_C\}$ es el sistema de referencia ligado al robot móvil. Entonces el objetivo es encontrar los valores de V y ω con los cuales el robot móvil es capaz de seguir una trayectoria preestablecida. Se asume que el robot móvil se mueve sobre un plano horizontal sin resbalar. Además, se considera que $|\theta_{n+1} - \theta_n| < \lambda$ siendo λ un ángulo suficientemente pequeño. [13]

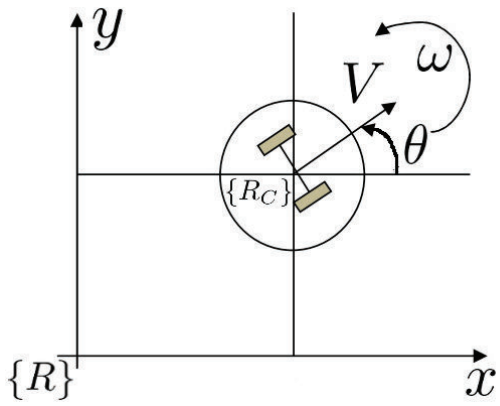


Figura 1.12: Descripción geométrica del robot móvil, tomado de [13].

Aplicando la aproximación de Euler al modelo cinemático de la ecuación 1.7, se obtiene el siguiente conjunto de ecuaciones:

$$\begin{cases} x_{n+1} \approx x_n + T_0 \cdot V_n \cdot \cos\theta_n \\ y_{n+1} \approx y_n + T_0 \cdot V_n \cdot \sin\theta_n \\ \theta_{n+1} \approx \theta_n + T_0 \cdot \omega_n \end{cases} \tag{1.8}$$

Esto puede ser expresado en forma vectorial como:

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \\ \theta_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \\ \theta_n \end{bmatrix} + T_0 \cdot \begin{bmatrix} \cos\theta_n & 0 \\ \sin\theta_n & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} V_n \\ \omega_n \end{bmatrix} \tag{1.9}$$

Si la trayectoria deseada $[x_{ref_{n+1}} \quad y_{ref_{n+1}} \quad \theta_{ref_{n+1}}]^T$ es conocida, entonces se puede sustituir $[x_{n+1} \quad y_{n+1} \quad \theta_{n+1}]^T$ en la ecuación 1.9 por $[x_{ref_{n+1}} \quad y_{ref_{n+1}} \quad \theta_{ref_{n+1}}]^T$. Por lo tanto, es posible calcular las acciones de control V_n y ω_n, las cuales son necesarias

para hacer que el robot móvil pase de el estado actual $[x_n \quad y_n \quad \theta_n]^T$ al estado deseado $[x_{ref_{n+1}} \quad y_{ref_{n+1}} \quad \theta_{ref_{n+1}}]^T$. Por definición:

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} = \begin{bmatrix} x_{ref_{n+1}} - x_n \\ y_{ref_{n+1}} - y_n \\ \theta_{ref_{n+1}} - \theta_n \end{bmatrix} \tag{1.10a}$$

$$B = \begin{bmatrix} \cos\theta_n & 0 \\ \sin\theta_n & 0 \\ 0 & 1 \end{bmatrix} \tag{1.10b}$$

Reemplazando las ecuaciones 1.10a y 1.10b en la ecuación 1.9, se obtiene:

$$B \cdot \begin{bmatrix} V_n \\ \omega_n \end{bmatrix} = \frac{1}{T_0} \cdot \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} \tag{1.11}$$

La solución óptima para el sistema de ecuaciones anterior es,

$$B^T \cdot B \cdot \begin{bmatrix} V_n \\ \omega_n \end{bmatrix} = \frac{1}{T_0} \cdot B^T \cdot \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} \tag{1.12}$$

$$\begin{bmatrix} \cos\theta_n & 0 \\ \sin\theta_n & 0 \\ 0 & 1 \end{bmatrix}^T \cdot \begin{bmatrix} \cos\theta_n & 0 \\ \sin\theta_n & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} V_n \\ \omega_n \end{bmatrix} = \frac{1}{T_0} \cdot \begin{bmatrix} \cos\theta_n & 0 \\ \sin\theta_n & 0 \\ 0 & 1 \end{bmatrix}^T \cdot \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} \tag{1.13}$$

$$\begin{bmatrix} \cos\theta_n & \sin\theta_n & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta_n & 0 \\ \sin\theta_n & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} V_n \\ \omega_n \end{bmatrix} = \frac{1}{T_0} \cdot \begin{bmatrix} \cos\theta_n & \sin\theta_n & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} \tag{1.14}$$

$$\begin{bmatrix} \cos^2\theta_n + \sin^2\theta_n & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} V_n \\ \omega_n \end{bmatrix} = \frac{1}{T_0} \cdot \begin{bmatrix} \cos\theta_n \cdot \Delta x + \sin\theta_n \cdot \Delta y \\ \Delta\theta \end{bmatrix} \cdot \tag{1.15}$$

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} V_n \\ \omega_n \end{bmatrix} = \frac{1}{T_0} \cdot \begin{bmatrix} \cos\theta_n \cdot \Delta x + \sin\theta_n \cdot \Delta y \\ \Delta\theta \end{bmatrix} \cdot \tag{1.16}$$

$$\begin{bmatrix} V_n \\ \omega_n \end{bmatrix} = \begin{bmatrix} \frac{\Delta x}{T_0} \cdot \cos\theta_n + \frac{\Delta y}{T_0} \cdot \sin\theta_n \\ \frac{\Delta\theta}{T_0} \end{bmatrix} \tag{1.17}$$

donde V_n y ω_n son las velocidades angulares y lineales necesarias para hacer que el robot móvil vaya de el estado actual al deseado.

Para encontrar la solución más cercana para el sistema de la ecuación 1.11, es necesario que las constantes reales a_1 y a_2 existan, tal que,

$$a_1 \cdot \begin{bmatrix} \cos\theta_n \\ \sin\theta_n \\ 0 \end{bmatrix} + a_2 \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{bmatrix} \tag{1.18}$$

donde a_1 y $a_2 \in \mathbb{R}$. Entonces se tiene que:

$$a_1 \cdot \begin{bmatrix} \cos\theta_n \\ \sin\theta_n \\ 0 \end{bmatrix} = \begin{bmatrix} \Delta x \\ \Delta y \\ 0 \end{bmatrix} \wedge a_2 = \Delta\theta \tag{1.19}$$

y debe cumplirse que:

$$\frac{\sin\theta_n}{\cos\theta_n} = \frac{\Delta y}{\Delta x} \tag{1.20a}$$

$$\tan\theta_n = \frac{\Delta y}{\Delta x} \tag{1.20b}$$

Por lo tanto, la orientación deseada está definida por:

$$\theta_{ezn} = \tan^{-1}\left(\frac{\Delta y}{\Delta x}\right) \tag{1.21}$$

donde θ_{ref_n} representa la orientación necesaria para el tiempo n, para hacer que el robot móvil siga la trayectoria de referencia. El subíndice ez proviene del inglés (*error zero*) y representa la orientación necesaria para que los errores de seguimiento tiendan a cero. Entonces, el controlador propuesto para el robot móvil está dado por las ecuaciones 1.22a y 1.22b:

$$\theta_{ezn} = \tan^{-1}\left(\frac{y_{ref_{n+1}} - k_V \cdot (y_{ref_n} - y_n) - y_n}{x_{ref_{n+1}} - k_V \cdot (x_{ref_n} - x_n) - x_n}\right) \tag{1.22a}$$

$$\begin{bmatrix} V_n \\ \omega_n \end{bmatrix} = \begin{bmatrix} \frac{1}{T_0} \cdot ((x_{ref_{n+1}} - k_V \cdot (x_{ref_{n+1}} - x_n) - x_n) \cdot \cos\theta_{ezn} + \\ (y_{ref_{n+1}} - k_V \cdot (y_{ref_{n+1}} - y_n) - y_n) \cdot \sin\theta_{ezn}) \\ \frac{1}{T_0} \cdot (\theta_{ezn} - k_\omega \cdot (\theta_{ezn-1} - \theta_{n-1}) - \theta_{n-1}) \end{bmatrix} \tag{1.22b}$$

En la ecuación 1.22b, el valor de θ_{ez_n} es usado en lugar de θ_{ref_n}. Los términos $k_V \cdot (y_{ref_{n+1}} - y_n)$, $k_V \cdot (x_{ref_{n+1}} - x_n)$ y $k_\omega \cdot (\theta_{ezn+1} - \theta_{n-1})$ han sido añadidos para que el error tienda a cero en forma gradual. Sin la presencia de estos términos, el valor de las constantes k_V y k_ω sería de cero, con lo que se llegaría a la referencia en un solo periodo de muestreo, lo cual implica que el valor de las acciones de control sea muy grande. Se debe satisfacer que $0 < k_V < 1 \quad \wedge \quad 0 < k_w < 1. \quad /$

1.6.1.3 Análisis del Error

Considerando el robot móvil descrito en [1], se tiene el siguiente modelo dinámico:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{u} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} u\cos\psi - a\omega\sin\psi \\ u\sin\psi + a\omega\cos\psi \\ \omega \\ \frac{\theta_3}{\theta_1}\omega^2 - \frac{\theta_4}{\theta_1}u \\ \frac{\theta_5}{\theta_2}u \cdot \omega - \frac{\theta_6}{\theta_2}\omega \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1/\theta_1 & 0 \\ 0 & 0 \\ 0 & 1/\theta_2 \end{bmatrix} \cdot \begin{bmatrix} u_c \\ \omega_c \end{bmatrix} \tag{1.23}$$

Si consideramos las siguientes ecuaciones,

$$\mathbf{f}(\mathbf{x}(t)) = \begin{bmatrix} u\cos\psi - a\omega\sin\psi \\ u\sin\psi + a\omega\cos\psi \\ \omega \\ \frac{\theta_3}{\theta_1}\omega^2 - \frac{\theta_4}{\theta_1}u \\ \frac{\theta_5}{\theta_2}u \cdot \omega - \frac{\theta_6}{\theta_2}\omega \end{bmatrix} \tag{1.24a}$$

$$\mathbf{g}(\mathbf{v}(t)) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1/\theta_1 & 0 \\ 0 & 0 \\ 0 & 1/\theta_2 \end{bmatrix} \cdot \begin{bmatrix} u_c \\ \omega_c \end{bmatrix} \tag{1.24b}$$

$$\mathbf{x}(t) = [x \;\; y \;\; \psi \; u \;\; \omega]^T \tag{1.24c}$$

$$\mathbf{v}(t) = [u_c \;\; \omega_c]^T \tag{1.24d}$$

podemos escribir la ecuación 1.23 de forma más compacta como se indica a continuación,

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t)) + \mathbf{g}(\mathbf{v}(t)) \tag{1.25}$$

Utilizando la aproximación de Euler, y reemplazando $\mathbf{x}_{n+1}$ por $\mathbf{x}_{d_{n+1}}$, se tiene que,

$$\mathbf{x}_{d_{n+1}} = \mathbf{x}_n + T_0[\mathbf{f}(\mathbf{x}_n) + \mathbf{g}(\mathbf{v}_n)] \tag{1.26}$$

como se ha mencionado uno de los objetivos de los controladores con ágebra lineal es encontrar las acciones de control; por lo tanto, de la ecuación anterior tenemos,

$$\mathbf{g}(\mathbf{v}_n) = \frac{1}{T_0}(\mathbf{x}_{d_{n+1}} - \mathbf{x}_{n+1}) - \tilde{\mathbf{f}}(\mathbf{x}_n) \tag{1.27}$$

donde $\tilde{\mathbf{f}}(\mathbf{x}_n)$ es la matriz $\mathbf{f}$ estimada en función de los estados del sistema, entonces,

$$\tilde{\mathbf{f}}(\mathbf{x}_n) = \frac{1}{T_0}(\mathbf{x}_{d_{n+1}} - \mathbf{x}_{n+1}) - \mathbf{g}(\mathbf{v}_n) \tag{1.28}$$

La ecuación anterior representa la estimación del modelo en el instante n y además la mejor estimación del modelo estará dada por el valor de la matriz de funciones en el instante anterior, es decir,

$$\tilde{\mathbf{f}}(\mathbf{x}_n) = \mathbf{f}(\mathbf{x}n - 1) \tag{1.29}$$

A partir de las ecuaciones 1.25 y 1.29 se tiene que,

$$\tilde{\mathbf{f}}(\mathbf{x}_n) = \frac{1}{T_0}(\mathbf{x}_{d_n} - \mathbf{x}_{n-1}) - \mathbf{g}(\mathbf{v}_{n-1}) \tag{1.30}$$

Por lo tanto, al sustituir la ecuación anterior en la ecuación 1.26

$$\mathbf{g}(\mathbf{v}_n) = \frac{1}{T_0}(\mathbf{x}_{d_{n+1}} - 2\mathbf{x}_n - \mathbf{x_{n-1}}) - \mathbf{g}(\mathbf{v}_{n-1}) \tag{1.31}$$

Ahora, la ecuación en lazo cerrado del sistema, incorporando la ley de control de la ecuación 1.31 estará expresada por,

$$\mathbf{x}_{n+1} = \mathbf{x}_n + T_0 \left[\mathbf{f}(\mathbf{x}_n) + \left(\frac{1}{T_0}(\mathbf{x}_{d_{n+1}} - 2\mathbf{x}_n + \mathbf{x}_{n-1}) + \mathbf{g}(\mathbf{v}_{k-1}) \right) \right] \tag{1.32}$$

Seguidamente, mediante la agrupación de términos y definiendo al error de estado por, $\mathbf{e}_k = \mathbf{x}_{d_n} - \mathbf{x}_n$, se sigue que,

$$\mathbf{e}_{k+1} = [(\mathbf{x}_{n-1} - \mathbf{x}_n) + T_0\mathbf{g}(\mathbf{v}_{n-1})] + T_0\mathbf{f}_(\mathbf{x}_n) \tag{1.33}$$

Por lo tanto, a partir de las ecuaciones 1.30 y 1.33, el error del sistema estará dado por,

$$\mathbf{e}_{n+1} = T_0[\mathbf{f}(\mathbf{x}_n) - \tilde{\mathbf{f}}(\mathbf{x}_n)] \tag{1.34}$$

esto quiere decir, que el error del sistema no es acumulativo y que depende de la aproximación del modelo solamente en el instante n.

1.6.2 INTERPOLACIÓN LINEAL

En matemáticas, se denomina interpolación a la obtención de nuevos puntos partiendo del conocimiento de un conjunto discreto de puntos. En ingeniería es usual disponer de un cierto número de puntos obtenidos por muestreo y estimar una función que los ajuste. La interpolación lineal es uno de los métodos de interpolación más simples. En esta interpolación se utilizan dos puntos (x_a, y_a) y (x_b, y_b), para la obtención de un tercer punto interpolado (x, y) a patir de la siguiente fórmula,

$$y = y_a + (x - x_a)\frac{y_b - y_a}{x_b - x_a} \tag{1.35}$$

Aunque la interpolación lineal es rápida y sencilla, en muchos casos no es muy precisa. Para obtener mejores resultados se pude utilizar la interpolación polinómica, pero su proceso es un poco más complejo que el de la interpolación lineal.

Ahora se procede a diseñar un controlador, para el robot móvil cuyo modelo está definido por la ecuación 1.7, utilizando el método de interpolación lineal. Para este modelo se tiene que,

$$\begin{cases} x_{n+1} = x_n + \int_{nT_0}^{(n+1)T_0} V \cdot \cos\theta dt \\ y_{n+1} = y_n + \int_{nT_0}^{(n+1)T_0} V \cdot \sin\theta dt \\ \theta_{n+1} = \theta_n + \int_{nT_0}^{(n+1)T_0} \omega dt \end{cases} \tag{1.36}$$

Aplicando la definición de interpolación lineal, dada por la ecuación 1.35, en la ecuación que define a la orientación θ y considerando que del método de Euler se tiene que $\theta_{n+1} = \theta_n + \omega_n \cdot T_0$, obtenemos

$$\theta(t) = \theta_n + \omega_n(t - nT_0) \tag{1.37}$$

Reemplazando 1.37 en la ecuación 1.36 se tiene que,

$$\begin{cases} x_{n+1} = x_n + V_n \int_{nT_0}^{(n+1)T_0} \cos[\theta_n + \omega_n(t - nT_0)]dt \\ y_{n+1} = y_n + V_n \int_{nT_0}^{(n+1)T_0} \sin[\theta_n + \omega_n(t - nT_0)]dt \end{cases} \tag{1.38}$$

A continuación se procede a integrar con respecto a t la ecuación 1.38, con lo cual se obtiene que,

$$\begin{cases} x_{n+1} = x_n + \dfrac{V_n}{\omega_n}(\sin\theta_{n+1} - \sin\theta_n) \\ y_{n+1} = y_n + \dfrac{V_n}{\omega_n}(\cos\theta_n - \cos\theta_{n+1}) \end{cases} \tag{1.39}$$

Dado que, al igual que el controlador por métodos numéricos, el controlador por interpolación se facilita empleando álgebra lineal, se pasará el sistema descrito por

la ecuación 1.39 a una forma matricial la cual se muestra en la ecuación 1.40.

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \\ \theta_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \\ \theta_n \end{bmatrix} + \begin{bmatrix} \frac{1}{\omega_n}(\sin\theta_{n+1} - \sin\theta_n) & 0 \\ \frac{1}{\omega_n}(\cos\theta_n - \cos\theta_{n+1}) & 0 \\ 0 & T_0 \end{bmatrix} \cdot \begin{bmatrix} V_n \\ \omega_n \end{bmatrix} \tag{1.40}$$

Como se indica en el procedimiento para el diseño de un controlador que emplea álgebra lineal, el siguiente paso consiste en calcular las acciones. Por lo tanto, si despejamos V_n y ω_n de la ecuación 1.40 llegamos a la ecuación 1.41d.

$$\Delta\theta = \theta_{n+1} - \theta_n \tag{1.41a}$$

$$\Delta x = x_{n+1} - x_n \tag{1.41b}$$

$$\Delta y = y_{n+1} - y_n \tag{1.41c}$$

$$\begin{bmatrix} V_n \\ \omega_n \end{bmatrix} = \frac{1}{T_0} \begin{bmatrix} \frac{\Delta\theta[\Delta x \cdot (\sin\theta_{n+1} - \sin\theta_n) + \Delta y \cdot (\cos\theta_n - \cos\theta_{n+1})]}{(\sin\theta_{n+1} - \sin\theta_n)^2 + (\cos\theta_n - \cos\theta_{n+1})^2} \\ \frac{\theta_{n+1} - \theta_n}{T_0} \end{bmatrix} \tag{1.41d}$$

Dado que el estado deseado es igual al estado en $n+1$ se puede reemplazar $\theta_{d_{n+1}}$ por θ_{n+1}. Las condiciones que deben satisfacerse para encontrar la solución más cercana son las siguientes,

$$\sin\theta_{n+1} = \frac{\omega_n}{V_n}\Delta x + \sin\theta_n \tag{1.42a}$$

$$\cos\theta_{n+1} = \cos\theta_n - \frac{\omega_n}{V_n}\Delta y \tag{1.42b}$$

$$\frac{\omega_n}{V_n} = \frac{2\Delta y\cos\theta_n - 2\Delta x\sin\theta_n}{\Delta x^2 + \Delta y^2} \tag{1.42c}$$

$$\theta_{d_{n+1}} = \arctan\left(\frac{\frac{\omega_n}{V_n}\Delta x + \sin\theta_n}{\cos\theta_n - \frac{\omega_n}{V_n}\Delta y}\right) \tag{1.42d}$$

CAPÍTULO 2

ANÁLISIS DE LA ESTRUCTURA Y PARÁMETROS

En este capítulo se indicarán los componentes que constituyen el hardware del quadrotor AR.Drone 2.0. Además, se describirán la estructura y parámetros de este quadrotor.

2.1 QUADROTOR PARROT AR.DRONE 2.0

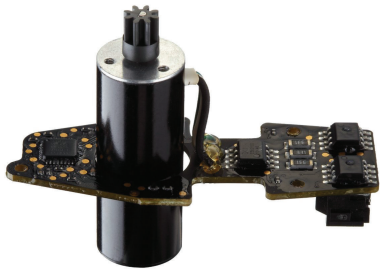
Un paso fundamental de este proyecto fue la selección del UAV con el que se elaboró el proyecto. Existen varios drones comerciales, pero los más populares son los quadrotores debido a su costo asequible y a que no requieren un control muy complejo. En este trabajo se ha utilizado el quadrotor comercial AR.Drone 2.0. El movimiento de este drone es controlado por comandos enviados a través de una conexión Wi-Fi.

2.1.1 HARDWARE

El AR.Drone 2.0, ver Figura 2.1, posee cuatro motores brushless de 35,000 [RPM] y 15 [W]. El suministro de energía está a cargo de una batería de polímero de litio con una capacidad de 1000 [mAh] la cual permite un vuelo de aproximadamente 10 minutos. Exiten también baterías de 1500 y 2000 [mAh] con las cuales se pueden realizar vuelos de 15 y 20 minutos respectivamente. [18]

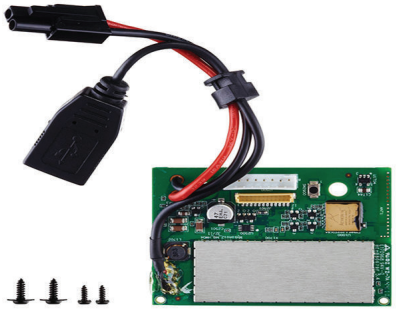
Figura 2.1: AR.Drone 2.0 con el case para interiores, tomado de [21].

Además, este drone posee un computador interno, con un procesador de 468 [MHz] y 128 [MB] de RAM, cuyo sistema operativo es Linux. Un conector USB ha sido incluido con al cual se le puede conectar una memoria para guardar imágenes y vídeos. Una tarjeta wireless 802.11g permite la comunicación con el drone. Es posible controlar a este quadrotor desde un computador o con una aplicación móvil. En la Figura 2.2 se muestran partes del hardware y la aplicación móvil del quadrotor.



(a) Motor Brushless

(b) Batería de 1500 [mAh].



(c) Mainboard.

(d) Application movil.

Figura 2.2: Hardware y aplicación del AR.Drone 2.0, tomados de [18].

2.1.1.1 Unidad de Medición Inercial

La unidad de medición inercial del AR.Drone 2.0 posee 6 grados de libertad. Esto permite realizar las mediciones de los ángulos de navegación *pitch*, *roll* y *yaw*. [18] Esta IMU es un sistema electromecánico o MEMS (del inglés, *micro electro-mechanical system*) que contiene 3 acelerómetros (uno para cada eje), un giroscopio de dos ejes para el roll y el pitch y un giroscopio de un solo eje para el yaw.

El acelerómetro del ARDrone2.0 es un *BA150* fabricado por Bosch Sensortec. [18] Este acelerómetro posee tres ejes perpendiculares. En cada eje se mide solamente la aceleración en esta dirección. La precisión del acelerómetro es de $\pm$ 50 $[\text{mg}]$

El AR.Drone 2.0 posee dos giroscopios, el primero es un giroscopio de 2 ejes para el *pitch* y el *roll*, mientras que el segundo es un giroscopio de un solo eje para el *yaw*. El giroscopio de dos ejes es el *IDG-500 Dual-Axis* el cual posee un rango de $500[°/s]$ y una sensibilidad de $2.0[mV/°/s]$ para movimientos con altas velocidades, mientras que un rango de $110[°/s]$ y una sensibilidad de $9.1[mV/°/s]$ para movimientos precisos a bajas velocidades. El giroscopio de un solo eje es el *Epson Toyocom XV-3500CB* el cual posee un rango de $100[°/s]$ y una sensibilidad de $0.67[mV/°/s]$ tanto movimientos con altas y bajas velocidades.

2.1.1.2 Altímetro Ultrasónico

Un sensor ultrasónico proporciona la medida de la altitud, la cual se puede usar para la estabilización de altura y el control vertical de velocidad. El sensor ultrasónico está ubicado en la parte inferior del quadrotor y apunta hacia abajo con el fin de medir la distancia entre el drone y el piso. Un paquete de ondas sonoras ultrasónicas es transmitido hacia el piso, las cuales son reflejadas de vuelta al sensor. El sistema mide el tiempo t que tarda el eco en regresar al sensor y calcula la distancia d utilizando la velocidad del sonido:

$$d = \frac{c \cdot t}{2} \tag{2.1}$$

donde $c \approx 343\ m/s$ es la velocidad del sonido en el aire. El sensor ultrasónico del AR.Drone 2.0 tiene un rango efectivo entre 20 [cm] y 6 [m]. Se debe considerar que el sonido se propaga en forma de cono donde los angulos varían entre 20 y 40 grados. Por lo tanto, el sensor ultrasónico puede informar si existe un objeto solamente si se encuentra en el cono de medición. También hay que tomar en cuenta la limitada frecuencia de operación. Antes de que una onda sonora pueda ser emitida, la onda que fue emitida previamente tiene que ser detectada por el sensor, esto reduce la velocidad de operación del sensor a $300/(2 \cdot 6) = 25$ [Hz]. En la Figura 2.3 se muestra el sensor ultrasónico utilizado por el AR.Drone 2.0.

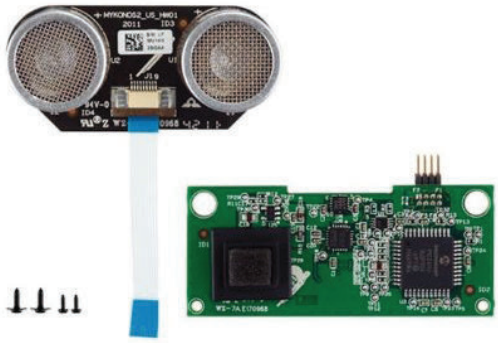


Figura 2.3: Sensor ultrasónico del AR.Drone 2.0, tomado de [3].

2.1.1.3 Cámaras

El AR.Drone 2.0 está equipado con dos cámaras CMOS: una cámara frontal y otra inferior. [18] Ambas cámaras pueden transmitir vídeo streaming a 30 y 15 imágenes por segundo respectivamente. La cámara frontal es una cámara HD de 720p. Esta es usada para detectar otros drones en juegos multijugador o para proporcionar video feedback en una pantalla, por ejemplo un teléfono inteligente, tablet o pantalla de un computador. La cámara inferior es una cámara QVGA la cual juega un papel importante siministrando información sobre el piso, con lo cual se puede tener estabilización horizontal y estimar la velocidad del drone. La Figura 2.4 muestra la cámara frontal (Figura 2.4a) y la cámara inferior (Figura 2.4b) del quadrotor.

(a) Cámara frontal. **(b)** Cámamara inferior.

Figura 2.4: Cámaras del AR.Drone 2.0, tomados de [2].

2.2 ESTRUCTURA

La estructura del AR.Drone está formada por barras de fibra de carbono y una cubierta de plástico de alta resistencia. En la Figura 2.5 se puede observar la vista superior de la configuración del quadrotor. Los cuatro motores del drone están distribuidos simétricamente a 90 grados de separación el uno del otro. En la parte interior de la estructura es donde se encuentran el mainboard, los sensores y la batería. Para que el drone pueda mantenerse en el aire el centro de masa debe estar situado en el centro de la estructura. [6]

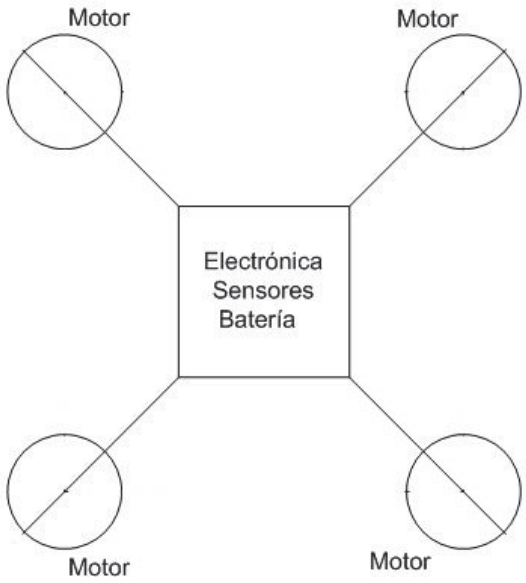


Figura 2.5: Estructura del AR.Drone 2.0

El ARDrone2.0 tiene 2 cubiertas, una para interiores y otra para exteriores. La cubierta para exteriores se utiliza con el fin de proteger las hélices cuando se realiza un vuelo en espacio cerrado, mientras que la cubierta de exteriores se la usa en sitios abiertos. Sin embargo, debido al uso, la cubierta para interiores se suele deteriorar causando que el centro de masa se desplace.

2.2.1 MOVIMIENTOS BÁSICOS

Aunque el quadrotor tiene seis grados de libertad, este está equipado solo con cuatro hélices; por lo tanto no es posible alcanzar un set-point deseado para todos los grados de libertad, sino como máximo cuatro. Sin embargo, gracias a la estructura,

es bastante fácil escoger las cuatro mejores variables y desacoplarlas con el fin de implementar un controlador fácilmente. Estas variables están relacionadas con los cuatro movimientos que se describen a continuación.

2.2.1.1 Throttle

Este movimiento se ejecuta al incrementar o decrementar la velocidad de todos los motores en la misma cantidad. Esto genera una fuerza que permite que el cuadrotor se eleve o descienda. La Figura 2.6 muestra el movimiento *throttle* en un quadrotor. En color azul se especifica la velocidad angular de las hélices la cual, en este caso, es igual a $\Omega_H + \Delta_A$ para cada una, donde Δ_A, en $[rad/s]$, representa un incremento respecto a Ω_H la cual es constante. Δ_A no puede ser muy grande dado que el modelo dejaría de ser lineal. [22]

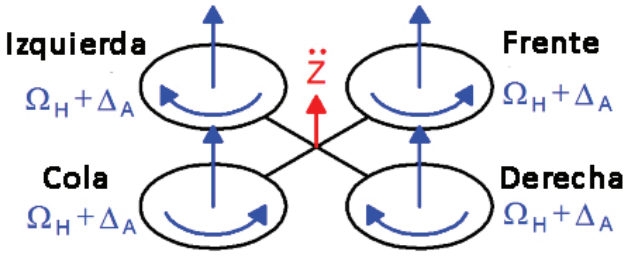


Figura 2.6: Movimiento Throttle, tomado de [14].

2.2.1.2 Roll

Este comando se lo realiza incrementando (o disminuyendo) la velocidad angular de la hélice izquierda y decrementando (o aumentando) la hélice derecha. Esto genera un torque con respecto al eje x en el sistema de referencia fijo al drone, lo cual hace que el quadrotor gire. El impulso vertical sigue siendo el mismo dado que este comando hace que el ángulo de navegación *roll* cambie. La Figura 2.7 muestra el comando *roll* en el quadrotor. Tanto Δ_A como Δ_B son variables positivas, las cuales son escogidas para mantener el impulso vertical invariable. Es posible demostrar que para valores pequeños $\Delta_A \approx \Delta_B$. Como en el caso del *throttle*, estas variables no pueden ser valores altos dado que el sitema dejaría de ser lineal. [22]

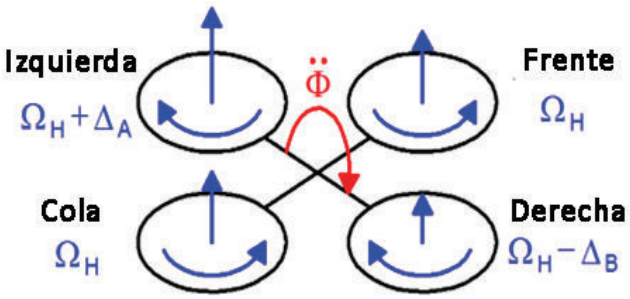


Figura 2.7: Movimiento Roll, tomado de [14].

2.2.1.3 Pitch

Esta maniobra es muy similar al roll y se ejecuta incrementando (o decrementando) la velocidad a la que gira el motor en la cola y decrementado (o aumentando) el motor delantero. Esto genera un torque con respecto al eje y del sistema de referencia fijo al drone, lo cual hace que el quadrotor gire. La velocidad vertical se mantiene constante dado que este comando solo cambia el ángulo de navegación *pitch*. La Figura 2.8 muestra el movimiento *pitch* en el quadrotor. Como en el caso anterior, tanto Δ_A como Δ_B son variables positivas las cuales se escogen para mantener el impulso vertical constante, y no pueden tener valores altos. En este caso también para valores pequeños se tiene que $\Delta_A \approx \Delta_B$. [14]

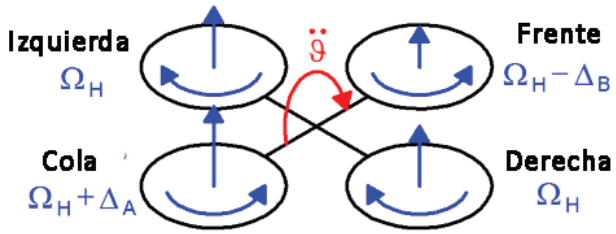


Figura 2.8: Movimiento Pitch, tomado de [14].

2.2.1.4 Yaw

Este movimiento se lo realiza incrementando (o decrementado) los motores tanto de la cola como de la parte delantera, y decrementado (o incrementando) los motores

derecho e izquierdo. Esto genera un torque con respecto al eje z respecto al sistema de referencia fijo al drone, lo cual hace que el quadrotor gire. El movimiento *yaw* es generado gracias al hecho de que tanto los motores izquierdo como derecho giran en sentido antihorario, mientras que los motores del frente y la cola giran en sentido horario. Por lo tanto, el torque total es desbalanceado y el drone gira alrededor del eje z_B. El impulso vertical total es el mismo ya que este comando solamente cambia el ángulo de navegación *yaw*. La Figura 2.9 muestra el movimiento *yaw* en el quadrotor. Como en los casos anteriores, Δ_A y Δ_B son elegidos de tal forma que el impulso vertical se mantenga invariable, y estos valores no pueden ser elevados. Además, $\Delta_A \approx \Delta_B$ cuando estos valores son pequeños. [18]

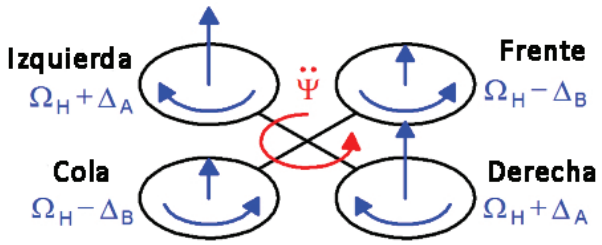


Figura 2.9: Movimiento Yaw, tomado de [14]

2.3 PARÁMETROS

2.3.1 MASA E INERCIA

Tanto la masa como la inercia son parámetros clave de un cuerpo rígido con seis grados de libertad. La masa promedio del quadrotor es de $433[\mathrm{g}]$ según [31]. Este promedio ha sido calculado luego de haber pesado por cinco ocasiones dos AR. Drone 2.0 diferentes. En estas mediciones se ha incluido la batería del drone.

Para estimar el momento de inercia en [31] se ha utilizado SolidWorks, un programa CAD. Los motores se han estimado utilizando cuatro cilindros en la posición en la que corresponden en el drone con la masa que poseen en la realidad. Con esto se ha estimado que la matriz de inercia es:

$$I_m = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} = \begin{bmatrix} 2.04016 & 0 & 0 \\ 0 & 1.56771 & 0 \\ 0 & 0 & 3.51779 \end{bmatrix} \times 10^4 \ [\mathrm{g \cdot cm^3}] \tag{2.2}$$

2.3.2 COEFICIENTES DE IMPULSO AERODINÁMICO Y ARRASTRE

Generalmente, es difícil medir la fuerza arodinámica y el torque. Se puede hacer que el AR.Drone permanezca suspendido en el aire con diferentes masas para estimar la fuerza neta generada por los propulsores. Además, es posible proporcionar un torque conocido y balancear al drone para estimar el torque neto de los propulsores.

En [31], se ha utilizado sensores de efecto Hall, sensores magnéticos y también un osciloscopio para realizar estos experimentos. Finalmente, se ha obtenido que el coeficiente de arrastre en las hélices es aproximadamente 2.38×10^{-9} $[\mathrm{N \cdot m/RPM^2}]$, mientras que el valor del coeficiente de impulso aerodinámico en las hélices es 9.14×10^{-6} $[\mathrm{g/RPM^2}]$. Estos valores corresponden a 3.29×10^{-11} $[\mathrm{N \cdot m/RPM^2}]$ y 1.27×10^{-7} $[\mathrm{g/RPM^2}]$ respectivamente en los motores con una relación de 8:68 entre los engranajes.

En la Tabla 2.1 se muestran los parámetros para el AR.Drone 2.0 los cuales han sido tomados de [18] y [31].

Descripción	**Valor [unidades]**
Momento de inercia en el eje x	2.04016 $[\mathrm{g \cdot cm^3}]$
Momento de inercia en el eje y	1.56771 $[\mathrm{g \cdot cm^3}]$
Momento de inercia en el eje z	3.51779 $[\mathrm{g \cdot cm^3}]$
Masa del quadrotor (motores y batería incluidos)	433 $[\mathrm{g}]$
Coeficiente de Impulso Aerodinámico (motores)	1.27×10^{-7} $[\mathrm{g/RPM^2}]$
Coeficiente de Arrastre (motores)	3.29×10^{-11} $[\mathrm{N \cdot m/RPM^2}]$

Tabla 2.1: Parámetros para el quadrotor AR.Drone 2.0.

CAPÍTULO 3

MODELADO E IDENTIFICACIÓN

En este capítulo se modelará, identificará y validará al AR.Drone 2.0. Además, se realizará una comparación del modelo obtenido con otros modelos utilizados en proyectos anteriores.

3.1 MODELADO

Antes de realizar el modelamiento del AR.Drone 2.0, es necesario definir los sistemas de referencia que se considerán para el quadrotor, y también la matriz de rotación utilizada para rotar un vector de un sistema a otro.

3.1.1 SISTEMAS DE REFERENCIA

Para modelar un sistema es necesario tener dos sistemas de referencia. El primer sistema de referencia es uno fijo a tierra, mientras que el segundo está centrado en el centro de masa del drone. En la Figura 3.1 se muestran ambos sistemas de referencia, E es el sistema fijo a tierra y B es el sistema ubicado en el centro de masa del quadrotor. Los dos sistemas siguen la convención de la mano derecha.

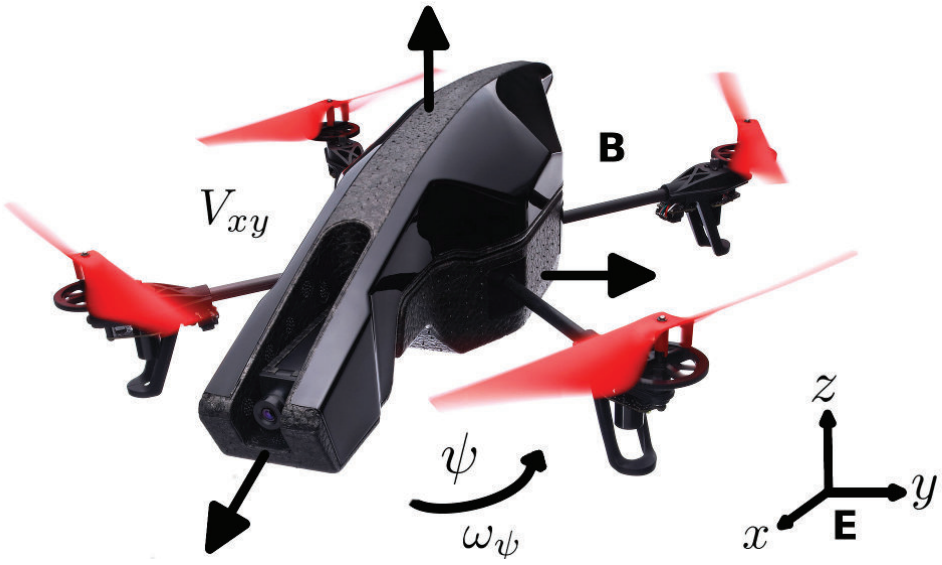


Figura 3.1: Sistemas de referencia del AR.Drone 2.0.

Una vez definidos los sistemas de referencia, es necesario un conjunto de transformaciones entre ambos, con el fin de que un vector en el sistema de referencia en tierra pueda ser expresado en el sistema de referencia fijo al cuerpo y viceversa.

3.1.2 MATRIZ DE ROTACIÓN

Dados dos sistemas de referencia cartesianos con el mismo origen, el primer sistema de referencia puede ser rotado para coincidir con la rotación del segundo aplicando tres rotaciones alrededor de los ejes en un orden dado. [30] Esta secuencia de rotaciones define de forma única la orientación del segundo sistema de referencia respecto al primero. Para evitar ambigüedades, la secuencia de rotaciones debe seguir una convención común. En este proyecto se utiliza la convención $z-y-x$, la cual se utiliza por convención en ingeniería aeroespacial. Las rotaciones alrededor de los ejes de un sistema de referencia cartesiano son descritas analíticamente por las siguientes ecuaciones:

$$\boldsymbol{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \tag{3.1a}$$

$$\boldsymbol{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \tag{3.1b}$$

$$\boldsymbol{R}_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1c}$$

Donde $\boldsymbol{R}_x(\phi)$, $\boldsymbol{R}_y(\theta)$ y $\boldsymbol{R}_z(\psi)$ son las matrices de rotación de los ángulos de navegación *roll*, *pitch* y *yaw* respectivamente, es decir, rotaciones alrededor de los ejes x, y y z respectivamente. Por lo tanto, la matriz de rotación completa $\boldsymbol{R}(\psi,\theta,\phi)$ se calcula multiplicando las matrices de las ecuaciones 3.1a, 3.1b y 3.1c. Hay que recalcar que el orden en el que se realiza la multiplicación es importante ya que la multiplicación de matrices no es conmutativa. Además, previamente se ha elegido la convención $z-y-x$ de lo cual resulta:

$$\boldsymbol{R}(\psi,\theta,\phi) = \boldsymbol{R}_{\Theta} = \boldsymbol{R}_z(\psi)\cdot\boldsymbol{R}_y(\theta)\cdot\boldsymbol{R}_x(\phi) =$$
$$\begin{bmatrix}\cos\psi & -\sin\psi & 0\\ \sin\psi & \cos\psi & 0\\ 0 & 0 & 1\end{bmatrix}\cdot\begin{bmatrix}\cos\theta & 0 & \sin\theta\\ 0 & 1 & 0\\ -\sin\theta & 0 & \cos\theta\end{bmatrix}\cdot\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\phi & -\sin\phi\\ 0 & \sin\phi & \cos\phi\end{bmatrix} = \tag{3.2}$$
$$\begin{bmatrix}\cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi-\sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi+\sin\psi\sin\phi\\ \sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi+\cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi-\cos\psi\sin\phi\\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi\end{bmatrix}$$

La matriz $\boldsymbol{R}(\psi,\theta,\phi)$ y su inversa, la cual es su matriz transpuesta debido a que $\boldsymbol{R}$ es una matriz ortogonal, se pueden utilizar para rotar un vector desde el sistema de referencia fijo a tierra hacia el sistema fijo al centro de masa y viceversa.

3.1.3 MODELO

De acuerdo con la documentación provista por el fabricante, se pueden enviar al AR.Drone 2.0 comandos para moverlo con respecto a un sistema de referencia fijo al centro de masa del drone. [21] Estos son comandos AT los cuales se envían vía WiFi utilizando varios APIs disponibles. Básicamente se puede mover al AR.Drone 2.0 a través de los ejes x_E, y_E y z_E en forma independiente o combinada. Por lo tanto, el modelo propuesto por *Parrot* está definido por la ecuación 3.3.

$$\begin{cases}\dot{x}_B = V_x\\ \dot{y}_B = V_y\\ \dot{z}_B = V_z\end{cases} \tag{3.3}$$

donde V_x, V_y y V_z son las entradas del sistema o acciones de control cuyos valores pueden variar desde 200 $[\mathrm{mm/s}]$ hasta 2000 $[\mathrm{mm/s}]$. $\dot{x_B}$, $\dot{y_B}$ y $\dot{z_B}$ son las velocidades del quadrotor para los ejes x_B, y_B y z_B respectivamente. Pasando el sistema a forma matricial se tiene:

$$\begin{bmatrix}\dot{x}_B\\ \dot{y}_B\\ \dot{z}_B\end{bmatrix} = \begin{bmatrix}V_x\\ V_y\\ V_z\end{bmatrix} \tag{3.4}$$

Es necesario utilizar la matriz de rotación con el fin de tener la posición en el sistema de referencia fijo a tierra. Por lo tanto, tras aplicar una rotación se tiene que el modelo del sistema está dada por la siguiente ecuación:

$$\begin{bmatrix} \dot{x}_E \\ \dot{y}_E \\ \dot{z}_E \end{bmatrix} = \boldsymbol{R}(\psi, \theta, \phi) \begin{bmatrix} \dot{x}_B \\ \dot{y}_B \\ \dot{z}_B \end{bmatrix} \tag{3.5}$$

Dado que los ángulos de navegación tanto θ (pitch) como ϕ (roll) puden solamente tomar valores entre $\pm$5 grados, se puede aproximar $\cos\theta$ y $\cos\phi$ a 1, mientras que $\sin\theta$ y $\sin\phi$ a 0. Esta configuración se la realiza a través de software en un archivo xml donde se indican todas las configuraciones iniciales. Reemplazando 3.2 en 3.5 tenemos:

$$\begin{bmatrix} \dot{x}_E \\ \dot{y}_E \\ \dot{z}_E \end{bmatrix} = \begin{bmatrix} \cos\psi\cos\theta & \cos\psi\sin\theta\sin\phi - \sin\psi\cos\phi & \cos\psi\sin\theta\cos\phi + \sin\psi\sin\phi \\ \sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix} \begin{bmatrix} \dot{x}_B \\ \dot{y}_B \\ \dot{z}_B \end{bmatrix} \tag{3.6}$$

Si reemplazamos en la ecuación 3.6 el hecho que los senos de θ y ϕ tienden a 0 y que los cosenos de estos mismos ángulos tienden a 1, como anteriomente se ha manifestado, se tiene que:

$$\begin{bmatrix} \dot{x}_E \\ \dot{y}_E \\ \dot{z}_E \end{bmatrix} \approx \begin{bmatrix} \cos\psi\cdot 1 & \cos\psi\cdot 0\cdot 0 - \sin\psi\cdot 1 & \cos\psi\cdot 0\cdot 1 + \sin\psi\cdot 0 \\ \sin\psi\cdot 1 & \sin\psi\cdot 0\cdot 0 + \cos\psi\cdot 1 & \sin\psi\cdot 0\cdot 1 - \cos\psi\cdot 0 \\ 0 & 1\cdot 0 & 1\cdot 1 \end{bmatrix} \begin{bmatrix} \dot{x}_B \\ \dot{y}_B \\ \dot{z}_B \end{bmatrix} \tag{3.7a}$$

$$\begin{bmatrix} \dot{x}_E \\ \dot{y}_E \\ \dot{z}_E \end{bmatrix} \approx \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{x}_B \\ \dot{y}_B \\ \dot{z}_B \end{bmatrix} \tag{3.7b}$$

Finalmente, reemplazando 3.4 en 3.7b, tenemos el modelo del AR.Drone 2.0 que se utilizará para el diseño del controlador.

$$\begin{bmatrix} \dot{x}_E \\ \dot{y}_E \\ \dot{z}_E \end{bmatrix} \approx \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V_x \\ V_y \\ V_z \end{bmatrix} \tag{3.8}$$

3.2 IDENTIFICACIÓN Y VALIDACIÓN

3.2.1 IDENTIFICACIÓN

La identificación es un campo de la modelación matemática de sistemas a partir de datos experimentales. [1] Dado que el modelo utilizado no presenta parámetros, para la identificación se puede utilizar cualquiera de los métodos no paramétricos como el análisis de la respuesta transitoria, de la respuesta en frecuencia, de la correlación, de Fourier entre otros. [16]

Para la identificación de este modelo se ha escogido el método de identificación no paramétrica en el dominio del tiempo. Esta técnica consiste en la respuesta transitoria de un sistema ante una entrada conocida. [10] Las señales de excitación para la identificación son:

$$V_{x_{ref}} = 0.9 \cdot \sin(0.7 \cdot t) + 0.6 \cdot \sin(0.6 \cdot t) + 0.25 \cdot \sin(0.5 \cdot t) \tag{3.9a}$$

$$V_{y_{ref}} = -0.8 \cdot \sin(0.4 \cdot t) - 0.8 \cdot \sin(0.3 \cdot t) + 0.35 \cdot \sin(0.2 \cdot t) \tag{3.9b}$$

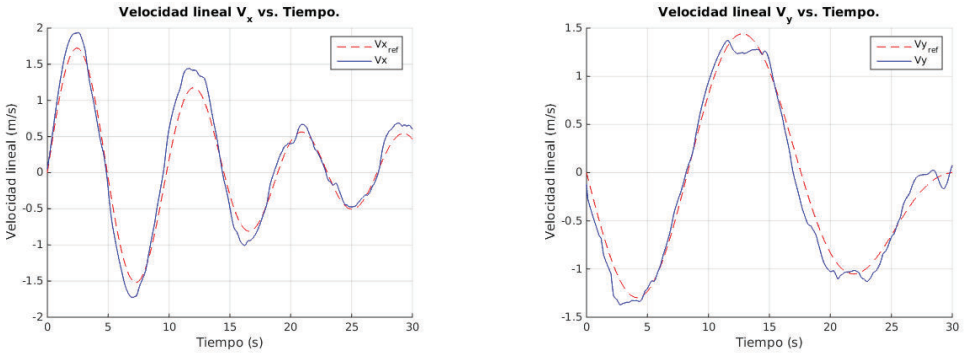
$$V_{z_{ref}} = 0.17 \cdot \sin(0.4 \cdot t) + 0.08 \cdot \sin(0.3 \cdot t) + 0.10 \cdot \sin(0.5 \cdot t) \tag{3.9c}$$

Las Figuras 3.2a, 3.2b y 3.2c muestran los resultados obtenidos en la identificación.

3.2.2 VALIDACIÓN

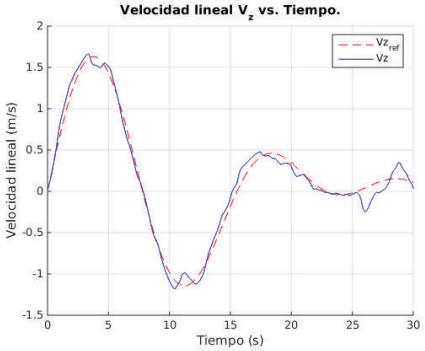
El proceso de validación trata de determinar cuándo un determinado modelo es lo suficientemente exacto para la aplicación requerida. En general, la mayoría de los métodos de validación tratan de determinar si la respuesta del modelo se ajusta con suficiente exactitud a los datos de entrada-salida obtenidos mediante experimentación. Para la validación del modelo se ha empleado el método de coherencia con el comportamiento de entrada-salida. [16]

Este método consite en determinar si el comportamiento de entrada-salida está suficientemente caracterizado, para lo cual se puede comprar los resultados obtenidos



(a) Identificación de $V_{x_{ref}}$.

(b) Identificación de $V_{y_{ref}}$.



(c) Identificación de $V_{z_{ref}}$.

Figura 3.2: Acciones de control para la identificación.

en la identificación no paramétrica en el dominio del tiempo para diferentes señales de excitación. Por lo tanto, el proceso consiste en comparar que tan bien se ajusta la señal de salida con respecto a la entrada. Las señales de excitación escogidas para la validación están definidas por:

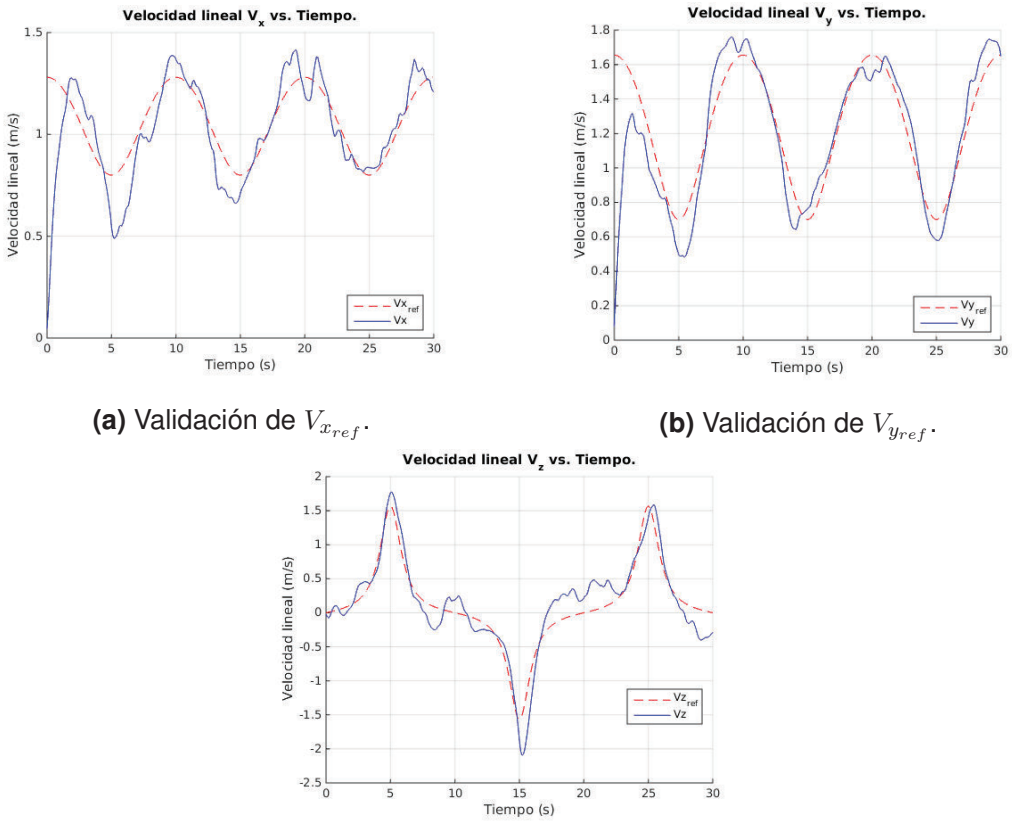
$$V_{x_{ref}} = \sqrt{0.8^2 + (1.0 \cdot \cos(0.1 \cdot \pi \cdot t))^2} \tag{3.10a}$$

$$V_{y_{ref}} = \sqrt{0.7^2 + (1.5 \cdot \cos(0.1 \cdot \pi \cdot t))^2)} \tag{3.10b}$$

$$V_{z_{ref}} = \frac{0.50 \cdot \pi \cdot \sin(0.1 \cdot \pi \cdot t)}{1 + \pi^2 \cdot \cos^2(0.1 \cdot \pi \cdot t)} \tag{3.10c}$$

Los resultados de la validación se pueden observar en las Figuras 3.3a, 3.3b y 3.3c, para las señales de entrada $V_{x_{ref}}$, $V_{y_{ref}}$ y $V_{z_{ref}}$ respectivamente.

Aunque claramente los resultados de la identificación y la validación del modelo indican que no representan de buena forma al quadrotor real existente, se utilizará



(a) Validación de $V_{x_{ref}}$.

(b) Validación de $V_{y_{ref}}$.

(c) Validación de $V_{z_{ref}}$.

Figura 3.3: Acciones de control para la validación.

en el proyecto al AR.Drone 2.0 debido a su costo. Adquirir un drone robusto y equipado con un buen sistema de sensores no está al alcance de un estudiante universitario pues su costo es de aproximadamente $\$7.000$ incluyendo gastos de importación y pago de impuestos.

3.3 COMPARACIÓN DEL MODELO

A continuación se procederá a hacer una comparación entre el modelo que se va a utilizar para el diseño del controlador en este proyecto y otros modelos que han sido usados en otros proyectos. Además, se indicará porque han sido descartados estos otros modelos.

3.3.1 MODELO I

El primero modelo con el que se realizará la comparación el es modelo más general de un quadrotor el cuál se presenta en trabajos como [31] y [7]. Este modelo está definido por el siguiente conjunto de ecuaciones:

$$\begin{cases} \ddot{x} = \dfrac{u}{m} \cdot (\cos\psi \cdot \sin\theta \cdot \cos\phi + \sin\psi \cdot \sin\phi) \\ \ddot{y} = \dfrac{u}{m} \cdot (\sin\psi \cdot \sin\theta \cdot \cos\psi - \cos\psi \cdot \sin\phi) \\ \ddot{z} = \dfrac{u}{m} \cdot \cos\theta \cdot \cos\phi - g \\ \ddot{\phi} = \tilde{\tau_\phi} \\ \ddot{\theta} = \tilde{\tau_\theta} \\ \ddot{\psi} = \tilde{\tau_\psi} \end{cases} \tag{3.11}$$

donde x, y y z son las coordenadas del sistema de referencia fijo a tierra. $\tilde{\tau_\phi}$, $\tilde{\tau_\theta}$ y $\tilde{\tau_\psi}$ son los momentos de *roll*, *pitch* y *yaw* respectivamente. Las entradas de este sistema son u, ϕ, θ y ψ que corresponden al impulso vertical del quadrotor, y los ángulos de navegación *roll*, *pitch* y *yaw* respectivamente.

Es notorio que este modelo es mucho más complejo del que estamos utilizando. Primeramente se tiene la presencia de parámetros como la masa del quadrotor y la aceleración de la gravedad. Segundo, este modelo utiliza la matriz de rotación para tres dimensiones y no se puede simplificar, como en el modelo de este proyecto, dado que de las entradas ϕ y θ depende el movimiento del quadrotor. Tercero, las ecuaciones del sistema son de segundo grado, es decir, las variables que queremos controlar están en forma de sus segundas derivadas.

El hecho de tener un modelo más complejo significa que tanto la simulación del sistema como el diseño y la implementación del controlador van a ser tareas más difíciles. Las ecuaciones del modelo que se utilizó para el proyecto dependen de la primera derivada de las variables que se desea controlar, lo cual facilita mucho todas estas tareas. Además, es posible simplificar la dependencia de matriz de rotación

pues los ángulos ϕ y θ no son entradas del sistema.

3.3.2 MODELO II

El segundo modelo con el que se comparará al modelo usado en el proyecto, es una extensión del modelo del robot móvil utilizado en [13], el cual está dado por el siguiente grupo de ecuaciones:

$$\begin{cases} \dot{x} = V_{xy} \cdot \cos\psi \\ \dot{y} = V_{xy} \cdot \sin\psi \\ \dot{z} = V_z \\ \dot{\psi} = \Psi \end{cases} \tag{3.12}$$

donde x, y y z son las coordenadas del sistema de referencia fijo a tierra. V_{xy}, V_z y Ψ son las entradas del sistema las cuales corresponde a la velocidad para el plano XY, la velocidad vertical y el ángulo de navegación *yaw* respectivamente.

Este modelo, así como el que se usó en el proyecto, tiene sus salidas en función de las primeras derivadas, lo cual simplifica bastante los procesos de simulación del sistema y diseño e implementación del controlador. Sin embargo, controlar el ángulo de navegación ψ es un poco más difícil que controlar los movimientos en los ejes del sistema de referencia fijo al cuerpo. Esta ha sido la razón por la que se ha descartado este modelo.

CAPÍTULO 4

DISEÑO E IMPLEMENTACIÓN DEL CONTROLADOR

En este capítulo se realizará la implementación del controlador utilizando el modelo del AR.Drone 2.0 descrito por la ecuación 3.8. Este controlador, el cual se basa en principios de álgebra lineal y métodos numéricos, permitirá que el quadrotor pueda seguir una trayectoria previamente establecida.

4.1 DISEÑO DEL CONTROLADOR

Aplicando la aproximación de Euler a la ecuación 3.8, se obtiene:

$$\begin{bmatrix} x_{E_{n+1}} \\ y_{E_{n+1}} \\ z_{E_{n+1}} \end{bmatrix} \approx \begin{bmatrix} x_{E_n} \\ y_{E_n} \\ z_{E_n} \end{bmatrix} + T_0 \begin{bmatrix} \cos\psi_n & -\sin\psi_n & 0 \\ \sin\psi_n & \cos\psi_n & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V_{x_n} \\ V_{y_n} \\ V_{z_n} \end{bmatrix} \tag{4.1}$$

Si la conocemos la trayectoria deseada, $[x_{E_{ref_{n+1}}} \quad y_{E_{ref_{n+1}}} \quad z_{E_{ref_{n+1}}}]^T$, entonces se puede sustituir $[x_{E_{n+1}} \quad y_{E_{n+1}} \quad z_{E_{n+1}}]^T$ por este vector, y entonces es posible calcular las acciones de control V_{x_n}, V_{y_n} y V_{z_n}, las cuales permiten que el quadrotor pase del estado actual al deseado.

Para facilitar el proceso, definimos la siguiente ecuación:

$$\begin{bmatrix} \Delta x_{E_n} \\ \Delta y_{E_n} \\ \Delta z_{E_n} \end{bmatrix} \triangleq \begin{bmatrix} x_{E_{ref_{n+1}}} - x_{E_n} \\ y_{E_{ref_{n+1}}} - y_{E_n} \\ z_{E_{ref_{n+1}}} - z_{E_n} \end{bmatrix} \tag{4.2}$$

Restando $[x_{E_n} \quad y_{E_n} \quad z_{E_n}]^T$ a ambos miembros de la ecuación 4.1 tenemos:

$$\begin{bmatrix} x_{E_{n+1}} \\ y_{E_{n+1}} \\ z_{E_{n+1}} \end{bmatrix} - \begin{bmatrix} x_{E_n} \\ y_{E_n} \\ z_{E_n} \end{bmatrix} \approx T_0 \begin{bmatrix} \cos\psi_n & -\sin\psi_n & 0 \\ \sin\psi_n & \cos\psi_n & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V_{x_n} \\ V_{y_n} \\ V_{z_n} \end{bmatrix} \tag{4.3}$$

Sustituyendo $[x_{E_{m+1}} \quad y_{E_{m+1}} \quad z_{E_{m+1}}]^T$ por $[x_{E_{ref_{n+1}}} \quad y_{E_{ref_{n+1}}} \quad z_{E_{ref_{n+1}}}]^T$ en 4.3.

$$\begin{bmatrix} x_{E_{ref_{n+1}}} \\ y_{E_{ref_{n+1}}} \\ z_{E_{ref_{n+1}}} \end{bmatrix} - \begin{bmatrix} x_{E_n} \\ y_{E_n} \\ z_{E_n} \end{bmatrix} \approx T_0 \begin{bmatrix} \cos\psi_n & -\sin\psi_n & 0 \\ \sin\psi_n & \cos\psi_n & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V_{x_n} \\ V_{y_n} \\ V_{z_n} \end{bmatrix} \tag{4.4a}$$

$$\begin{bmatrix} x_{E_{ref_{n+1}}} - x_{E_n} \\ y_{E_{ref_{n+1}}} - y_{E_n} \\ z_{E_{ref_{n+1}}} - z_{E_n} \end{bmatrix} \approx T_0 \begin{bmatrix} \cos\psi_n & -\sin\psi_n & 0 \\ \sin\psi_n & \cos\psi_n & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V_{x_n} \\ V_{y_n} \\ V_{z_n} \end{bmatrix} \tag{4.4b}$$

Posteriormente se procede a reemplazar 4.2 en 4.4b, con lo cual se tiene:

$$\begin{bmatrix} \Delta x_{E_n} \\ \Delta y_{E_n} \\ \Delta z_{E_n} \end{bmatrix} \approx T_0 \begin{bmatrix} \cos\psi_n & -\sin\psi_n & 0 \\ \sin\psi_n & \cos\psi_n & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V_{x_n} \\ V_{y_n} \\ V_{z_n} \end{bmatrix} \tag{4.5}$$

Dividiendo ambos miembros de la ecuación anterior por T_0 se llega a:

$$\frac{1}{T_0} \cdot \begin{bmatrix} \Delta x_{E_n} \\ \Delta y_{E_n} \\ \Delta z_{E_n} \end{bmatrix} \approx \begin{bmatrix} \cos\psi_n & -\sin\psi_n & 0 \\ \sin\psi_n & \cos\psi_n & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V_{x_n} \\ V_{y_n} \\ V_{z_n} \end{bmatrix} \tag{4.6}$$

El objetivo es despejar $[V_{x_n} \quad V_{y_n} \quad V_{z_n}]^T$ para lo cual debemos multiplicar a ambos miembros de la ecuación 4.6 por la matriz inversa de 3.1c. Afortunadamente, dado que esta es una matriz de rotación, se tiene que su inversa es su matriz transpuesta, por lo tanto:

$$\begin{bmatrix} \cos\psi_n & \sin\psi_n & 0 \\ -\sin\psi_n & \cos\psi_n & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \frac{1}{T_0} \cdot \begin{bmatrix} \Delta x_{E_n} \\ \Delta y_{E_n} \\ \Delta z_{E_n} \end{bmatrix} \approx \begin{bmatrix} \cos\psi_n & \sin\psi_n & 0 \\ -\sin\psi_n & \cos\psi_n & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\psi_n & -\sin\psi_n & 0 \\ \sin\psi_n & \cos\psi_n & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V_{x_n} \\ V_{y_n} \\ V_{z_n} \end{bmatrix} \tag{4.7}$$

Simplificando la ecuación anterior:

$$\frac{1}{T_0}\begin{bmatrix}\cos\psi_n & \sin\psi_n & 0\\ -\sin\psi_n & \cos\psi_n & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\Delta x_{E_n}\\ \Delta y_{E_n}\\ \Delta z_{E_n}\end{bmatrix} \approx \begin{bmatrix}\cos^2\psi_n + \sin^2\psi_n & 0 & 0\\ 0 & \sin^2\psi_n + \cos^2\psi_n & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}V_{x_n}\\ V_{y_n}\\ V_{z_n}\end{bmatrix} \tag{4.8a}$$

$$\frac{1}{T_0}\begin{bmatrix}\cos\psi_n \cdot \Delta x_{E_n} + \sin\psi_n \cdot \Delta y_{E_n}\\ -\sin\psi_n \cdot \Delta x_{E_n} + \cos\psi_n \cdot \Delta y_{E_n}\\ \Delta z_{E_n}\end{bmatrix} \approx \begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}V_{x_n}\\ V_{y_n}\\ V_{z_n}\end{bmatrix} \tag{4.8b}$$

$$\begin{bmatrix}\dfrac{\cos\psi_n \cdot \Delta x_{E_n} + \sin\psi_n \cdot \Delta y_{E_n}}{T_0}\\ \dfrac{-\sin\psi_n \cdot \Delta x_{E_n} + \cos\psi_n \cdot \Delta y_{E_n}}{T_0}\\ \dfrac{\Delta z_{E_n}}{T_0}\end{bmatrix} \approx \begin{bmatrix}V_{x_n}\\ V_{y_n}\\ V_{z_n}\end{bmatrix} \tag{4.8c}$$

$$\begin{bmatrix}V_{x_n}\\ V_{y_n}\\ V_{z_n}\end{bmatrix} \approx \begin{bmatrix}\dfrac{\cos\psi_n \cdot \Delta x_{E_n} + \sin\psi_n \cdot \Delta y_{E_n}}{T_0}\\ \dfrac{-\sin\psi_n \cdot \Delta x_{E_n} + \cos\psi_n \cdot \Delta y_{E_n}}{T_0}\\ \dfrac{\Delta z_{E_n}}{T_0}\end{bmatrix} \tag{4.8d}$$

Luego de realizar las respetivas operaciones y simplificaciones se ha llegado a la ecuación 4.8d, la cual expresa los valores que deben tomar las acciones de control en cada instante n para que el drone siga la trayectoria preestablecida. Además, estableciendo una analogía con el procedimiento que se hizo con el robot móvil, se tiene que los valores para las acciones de control V_x, V_y y V_z están dados por:

$$V_{x_n} = \frac{1}{T_0} \cdot (\cos\psi_n \cdot (x_{E_{ref_{n+1}}} - k_x \cdot (x_{E_{ref_n}} - x_{E_n}) - x_{E_n}) + \sin\psi_n \cdot (y_{E_{ref_{n+1}}} - k_y \cdot (y_{E_{ref_n}} - y_{E_n}) - y_{E_n})) \tag{4.9a}$$

$$V_{y_n} = \frac{1}{T_0} \cdot (-\sin\psi_n \cdot (x_{E_{ref_{n+1}}} - k_x \cdot (x_{E_{ref_n}} - x_{E_n}) - x_{E_n}) + \cos\psi_n \cdot (y_{E_{ref_{n+1}}} - k_y \cdot (y_{E_{ref_n}} - y_{E_n}) - y_{E_n})) \tag{4.9b}$$

$$V_{y_n} = \frac{1}{T_0} \cdot (z_{E_{ref_{n+1}}} - k_z \cdot (z_{E_{ref_n}} - z_{E_n}) - z_{E_n}) \tag{4.9c}$$

k_x, k_y y k_z son constantes positivas que permiten ajustar la función del sistema de control propuesto, las cuales deben satisfacer $0 < k_x < 1 \wedge 0 < k_y < 1 \wedge 0 < k_z < 1$. Como se ha indicado en el primer capítulo, la aparición de estas constantes y sus factores hacen que el error tienda a cero de forma gradual, asegurando que el valor de las acciones de control no sean muy altos.

4.2 SIMULACIÓN DEL CONTROLADOR

Con el fin de poner a prueba el modelo del AR.Drone 2.0 utilizado en este proyecto se ha realizado una simulación continua y una discreta del sistema utilizando *Simulink*. Además, estas simulaciones permiten probar el controlador y determinar si es factible o no la implementación del mismo. Para ambas simulaciones se ha escogido $100\ [ms]$ como tiempo de muestreo.

4.2.1 SIMULACIÓN CONTINUA

En la Figura 4.1 se muestra una captura de pantalla de la simulación continua hecha en *Simulink*. Como se puede ver el modelo del quadrotor ha sido abstraído en el bloque *AR.Drone 2.0 Model*. El motivo de abstraer el modelo en un solo bloque es para facilitar la visualización de los elementos de la simulación. El contenido de este bloque se muestra en la Figura 4.2, el cual se ha construido en base a la ecuación 3.8. Luego que la simulación ha sido ejecutada, se exportan al *Workspace* de *MATLAB* el tiempo t, las posiciones para los ejes x, y y z y las referencias para los mismos ejes para cada instante n.

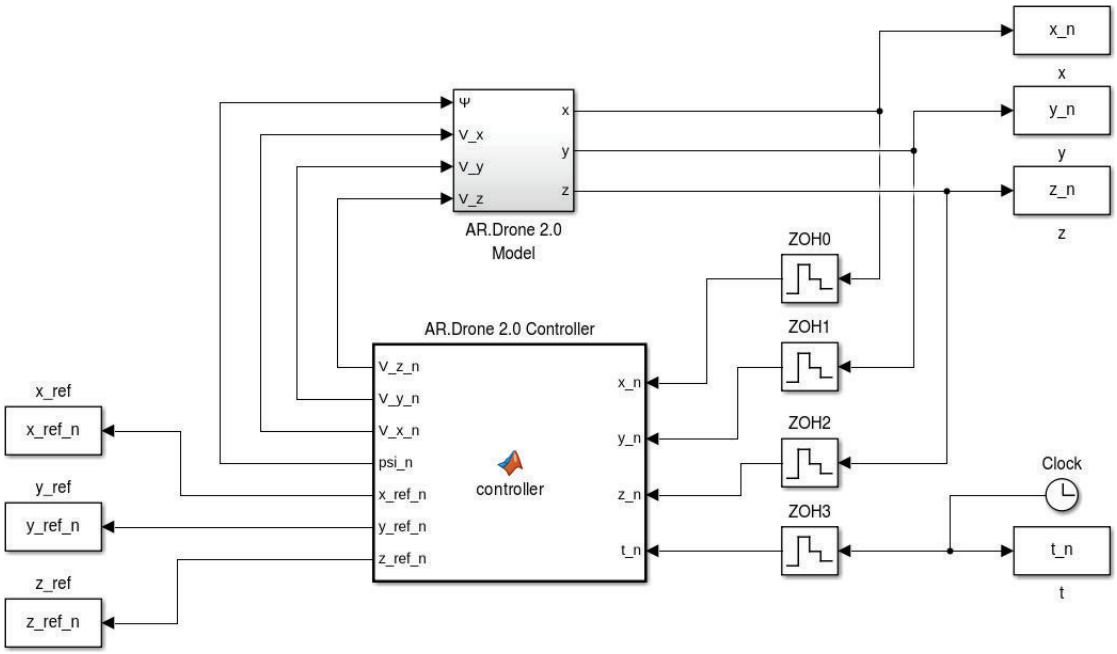


Figura 4.1: Simulación continua del controlador para el AR.Drone 2.0.

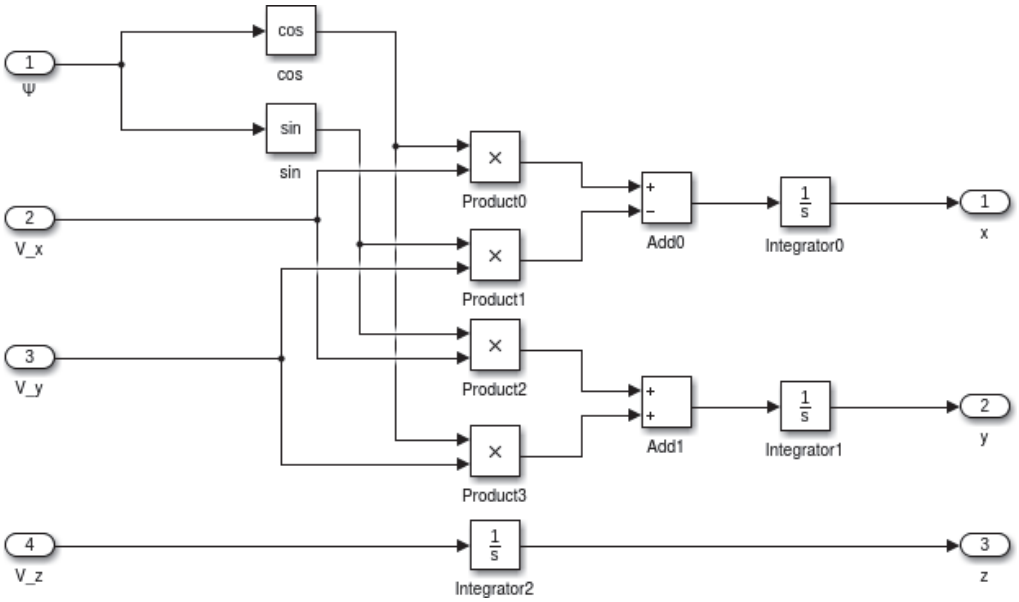


Figura 4.2: Contenido del bloque *AR.Drone 2.0 Model*.

Nótese que en la simulación se usan retenedores de orden cero para muestrear datos del modelo a un periodo de muestreo determinado. Las pruebas y resultados de esta simulación se presentan en el siguiente capítulo. El código del controlador se ha adjuntado en el anexo A.

4.2.2 SIMULACIÓN DISCRETA

Una vez implementada la simulación continua, se procede a realizar la discreta pues al momento de implementar el controlador lo que realmente se tiene es un modelo discreto. En la Figura 4.3, se muestra la simulación discreta implementada en *Simulink*.

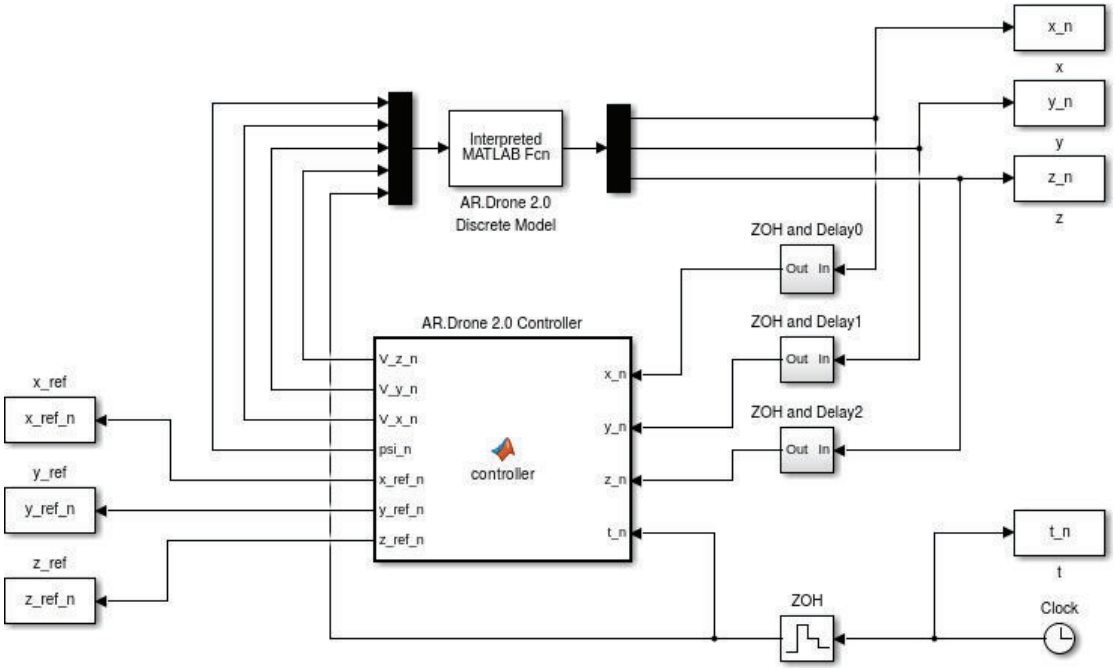


Figura 4.3: Simulación discreta del AR.Drone 2.0

En la Figura 4.3 se observa un bloque que no es propio de *Simulink*, al cual se lo ha nombrado *ZOH and Delay*. La Figura 4.4 muestra el contenido interno de este bloque. La razón de usar este bloque es poder arrancar la simulación ya que caso contrario *Simulink* no tiene idea de como empezarla y al momento de correrla se obtienen mensajes de error. Además, debido a que $\frac{1}{0.2s+1}$ es un filtro permite suavizar las señales x_n, y_n y z_n.

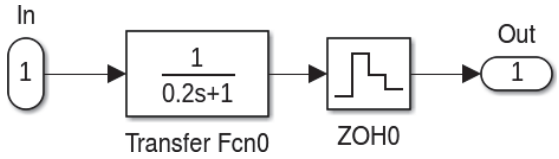


Figura 4.4: Contenido del bloque *ZOH and Delay*

El controlador empleado para esta simulación es el mismo, sin ningún cambio en la lógica, que se ha utilizado en la simulación continua. El código del modelo discreto del AR.Drone 2.0 se ha adjuntado en el anexo B, mientras que las pruebas y resultados se muestran en el siguiente capítulo.

4.3 IMPLEMENTACIÓN DEL CONTROLADOR

En esta sección, se explicará el algoritmo implementado para que el quadrotor sea capaz de seguir trayectorias autónomamente. Además, se hará una breve descripción de la aplicación móvil utilizada para dibujar la trayectoria a seguir por el drone. Finalmente, se explicará la interfaz gráfica de usuario (*GUI*) que permite al usuario inicializar las herramientas necesarias para la ejecución del programa.

4.3.1 DESCRIPCIÓN DEL ALGORITMO

Con el fin de facilitar la programación y la implementación del controlador, en este proyecto se ha utilizado el paradigma de programación orientada a objetos o *POO*. *POO* permite crear modelos basados en objetos reales o abstractos y las interacciones entre ellos. Además, la *POO* evita la repetición del código lo cual favorece el mantenimiento y corrección de errores. Debido a la extensión del código, este no se incluirá en el trabajo escrito, sino que será adjuntado en un CD.

En la Figura 4.5 se presenta el diagrama de flujo de la implementación del algoritmo. Al inicio del programa, lo primero que se hace es importar todos los módulos y datos al programa principal. Los módulos contienen principalmente las clases que se han definido para el proyecto, pero también contienen funciones que aportan utilidad al programa principal. Los datos que son importados o cargados son las trayectorias de referencias para los ejes x, y y z, también se importan valores para constantes como las constantes de control $k_{V_{xy}}$, k_{V_z} y k_{ω_ψ} o el tiempo de muestreo T_0.

Luego se procede a calibrar los sensores de la IMU enviando un comando desde el computador hacia el quadrotor. Este comando calibra específicamente los ángulos de navegación *roll* y *pitch*, haciendo que la inclinación de la superficie donde se encuentra el quadrotor en ese momento se tome como valor de cero grados para ambos ángulos. La calibración de los sensores se realiza dado que generalmente

luego de completar una trayectoria estos sensores se han descalibrado y la posición actual del drone es calculada en base a los datos leídos de estos sensores. Por lo tanto, es crucial que estos sensores entreguen una medida tan aproximada a la real como sea posible.

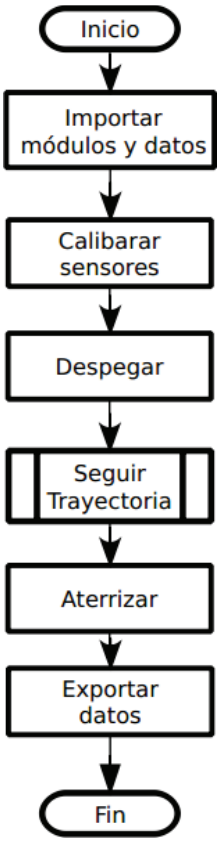


Figura 4.5: Diagrama de flujo del algoritmo implementado.

Todos los comandos que se envían desde el computador hacia el drone y viceversa se los hace a través de WiFi. Por ende, se tienen todas las ventajas y limitantes de esta conexión. Por ejemplo, una ventaja es que la comunicación es inalámbrica, lo cual permite que el quadrotor se mueva libremente. Por otra parte, una desventaja es la interferencia que se presenta debido a otras redes inalámbricas, las cuales influyen en la comunicación entre el drone y el computador.

Luego que los sensores han sido calibrados se procede a enviar un comando para que el quadrotor despegue. Una vez el quadrotor se encuentre en el aire se realiza la ejecución del seguimiento de la trayectoria. Este proceso será descrito en detalle

más adelante. Cuando el drone ha terminado el seguimiento de la trayectoria, se envía un comando para hacer aterrizar al quadrotor. Finalmente, luego de aterrizar se exportan los datos leídos de los sensores y los valores de las posiciones calculados mientras el drone seguía la trayectoria con el fin de graficarlos y poder analizar los resultados que se han obtenido experimentalmente.

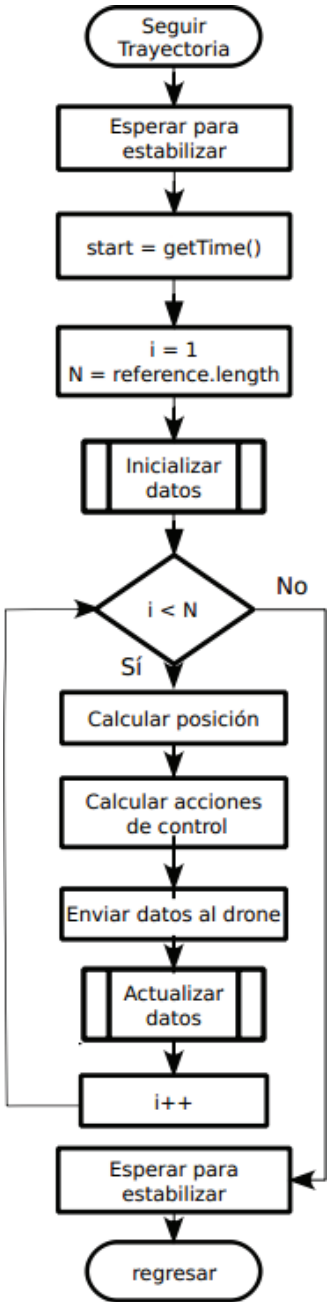


Figura 4.6: Diagrama de flujo del proceso *Seguir Trayectoria*.

Ahora se procede a explicar el subproceso *Seguir Trayectoria* cuyo diagrama de flujo se muestra en la Figura 4.6. Posteriormente al despegue, es necesario esperar entre 3 a 5 segundos para que la comunicación entre el drone y el computador se estabilice. Este tiempo de espera permite que no se pierda información y que los datos leídos de los sensores no sean aquellos leídos mientras el quadrotor está despegando. Luego se inicializa la variable *start* con un valor igual tiempo en aquel instante. Este paso es sumamente importante ya que este es el tiempo de referencia para la trayectoria, es decir, el valor de *start* se tomará como tiempo $t = 0$ para la ejecución de la trayectoria. El valor que retorna la función *getTime()* es el tiempo en milisegundos desde el *Unix Epoch* (00:00:00 del 01/01/1970). Después se definen dos variables i y N las cuales representan un índice y el número de puntos de la trayectoria de referencia. Ulteriormente, se tiene el proceso *Inicializar datos* el cual se muestra en la Figura 4.7.

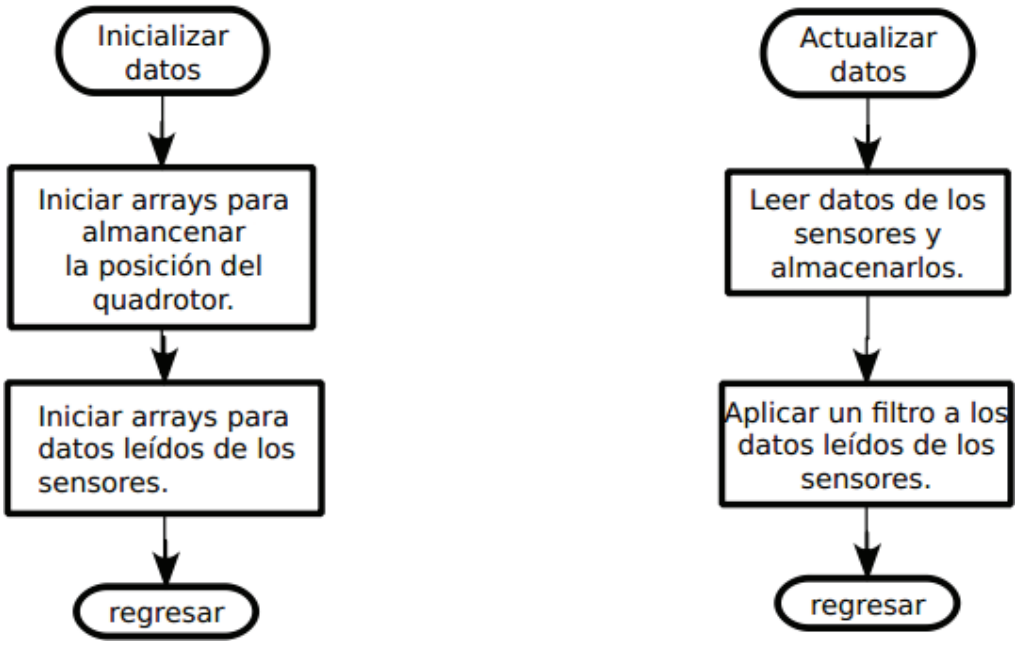


Figura 4.7: Diagramas de flujo de los procesos *Inicializar datos* y *Actualizar datos*.

El proceso de *Inicializar datos* consiste en la creación e inicialización los arrays que se usarán para almacenar la posición del drone para los ejes x, y y z. Para los ejes x y y estos valores iniciales son igual a 0.0 dado que el quadrotor parte de la posición $(x, y) = (0, 0)$ con respecto al sistema de referencia fijo a tierra, mientras

que para el eje z este valor es la altura medida con el sensor ultrasónico en aquel instante. En este proceso también se inicializan los arrays que almacenan los datos que son leídos de los sensores. Cada uno de estos arrays se inicializa con el valor correspondiente a la lectura del sensor en aquel instante.

Luego que el proceso *Inicializar datos* es completado, el flujo del programa continua en el proceso *Seguir Trayectoria* en el cual se entra a un lazo. Este lazo se ejecuta mientras el valor del índice i sea menor al número de puntos que tiene la trayectoria, es decir, mientras i sea menor que N. Para cada iteración del lazo primero se calcula la posición del quadrotor utilizando los datos que han sido previamente leídos de los sensores. Después, con las posiciones para cada uno de los ejes se procede a calcular las acciones de control que permiten que el drone siga la trayectoria preestablecida. Luego estas acciones de control son enviadas al quadrotor.

Posteriormente se ejecuta el subproceso *Actualizar datos* cuyo diagrama de flujo se muestra en la Figura 4.7. Aquí primeramente se leen los datos de los sensores y a continuación se aplica un filtro a cada uno de estos datos. El uso del filtro elimina perturbaciones generadas por ruido en la lectura de los sensores. Tras terminar el subproceso *Actualizar datos*, el flujo del programa vuelve al proceso *Seguir Trayectoria*, donde finalmente, dentro del lazo se procede a incrementar el índice i, lo cual garantiza que en algún momento el lazo terminará. Cuando el lazo termina, nuevamente se espera entre 3 a 5 segundos para que el quadrotor se estabilice antes de terminar le subproceso *Seguir Trayectoria* y volver al diagrama de flujo mostrado en la Figura 4.5.

El diagrama de flujo descrito puede ser traducido a cualquier lenguaje de programación, y lo más importante en este caso es escoger un API apropiado para el desarrollo del proyecto. Es muy importante que al escoger el API se investigue si es estable y la cantidad de proyectos en que ha sido utilizado. Probablemente, uno puede verse tentado a trabajar con un API cuyo lenguaje de programación es más familiar. Sin embargo, no tener un API estable o confiar en que a futuro lo estabilizarán es muy probablemente una pérdida de tiempo y esfuerzo.

4.3.2 APLICACIÓN MÓVIL

Se ha desarrollado una aplicación móvil la cual se puede instalar en un dispositivo cuyo sistema operativo sea *Android*. Esta aplicación es utilizada por el usuario para dibujar la trayectoria que el drone va a seguir y también establecer los valores de las constantes de control. Todos estos datos son posteriormente enviados a un correo electrónico desde el dispositivo móvil.

El aumento de la popularidad de las aplicaciones móviles y la facilidad para utilizarlas han sido las razones por las que se ha diseñado una aplicación para dibujar la trayectoria. Hay que añadir que *Android* es un sistema operativo completamente gratuito y su *IDE*, llamado *Android Studio*, también es gratuito. Existen otros sistemas operativos para dispositivos móviles también populares como *iOS* y *Windows Mobile*. Sin embargo, no son tan abiertos para el desarrollo y es necesario pagar para utilizarlos.

4.3.2.1 Descripción de la App

Con el fin de evitar un manual de usuario, al diseñar esta aplicación se ha tratado de hacerla lo más intuitiva posible. El nombre que se le ha dado a la aplicación es **Drone App**. Una vez instalada en un dispositivo móvil, se procede a abrir la aplicación haciendo *tap* sobre el ícono de la aplicación (Figura 4.8).

Figura 4.8: Ícono de la aplicación móvil **Drone App**.

Una vez abierta la aplicación al usuario se le pregunta la altura fija a la que el drone va a realizar el seguimiento de la trayectoria. La Figura 4.9 muestra la pantalla donde se ingresa la altura. Después de haber ingresado el valor de la altura en centímetros se hace un *tap* en *OK* para continuar con el flujo de la aplicación.

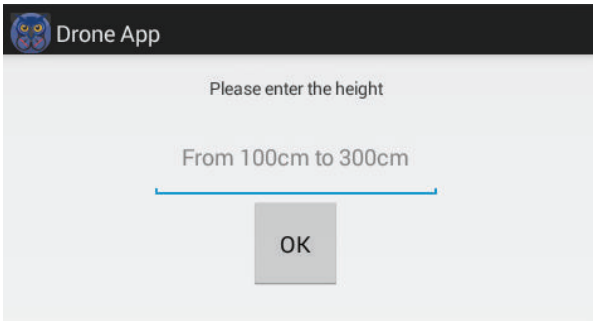


Figura 4.9: Pantalla para ingresar la altura a la que el drone volará.

Luego de haber hecho un *tap* en *OK* se visualiza la pantalla en la que el usuario debe dibujar la trayectoria que el quadrotor va a seguir en el plano XY. El eje x se ha ajustado para que tenga un valor máximo de 7.5 [m], mientras que ele eje y tiene un valor máximo de 5.0 [m]. Estos valores máximos se pueden cambiar por código si fuera menester adaptarlo a otro espacio más grande o más pequeño. En la Figura 4.10 se muestra la pantalla para dibujar la trayectoria con una trayectoria dibujada.

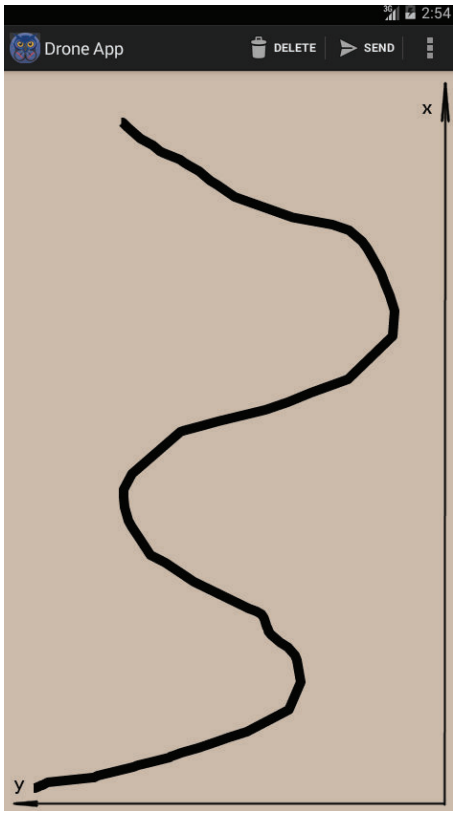


Figura 4.10: Pantalla para dibujar la trayectoria deseada.

Una vez dibujada la trayectoria, el usuario puede borrarla haciendo un tap en *Delete*, puede enviarla haciendo tap en *Send* o puede presionar el botón de *Configuración* (en la esquina superior derecha) para configurar los valores de las constantes de control. Si el usuario escoge la opción de cambiar los valores de las constantes de control, una nueva pantalla es mostrada. Aquí el usuario debe escoger un valor en el intervalo $]0, 1[$ para cada constante y hacer un *tap* en *Set* para definir estos valores y volver a la pantalla donde se dibuja la trayectoria. La Figura 4.11 muestra la pantalla donde se configuran los valores de las constantes de control.

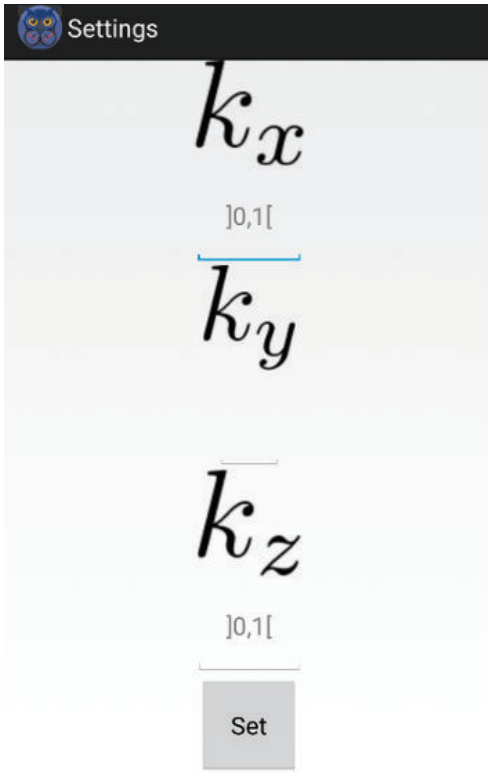


Figura 4.11: Pantalla para ajustar las constantes de control.

Cuando el usuario presiona el botón *Send*, envía los puntos que describen la trayectoria en tres archivos, uno para cada eje x, y y z. Además, envía en un archivo los valores de las constantes de control que se han ajustado. Si las constantes no se ajustaron, envía el archivo con todas las constantes con valor igual a 0.90. Estos cuatro archivos son enviados a un correo electrónico establecido en el código de la app. El hecho de permitir al usuario definir las constantes de control en la aplicación evita que tenga que entrar al código para ajustarlos.

4.3.3 INTERFAZ GRÁFICA DE USUARIO

Una interfaz gráfica o *GUI* (del inglés *Graphical User Interface*) permite a un usuario interactuar con dispositivos electrónicos a través de herramientas gráficas como indicadores visuales o íconos. En este proyecto, la interfaz gráfica permitirá al usuario inicializar la comunicación con el quadrotor, ejecutar el seguimiento de la trayectoria y graficar los resultados. En la Figura 4.12 se muestra una captura de pantalla de la interfaz gráfica utilizada en el proyecto.

Figura 4.12: Interfaz gráfica de usuario.

El uso de la interfaz es muy simple. Tras haber enviado la trayectoria dibujada y las constantes ajustadas en la aplicación móvil, se procede a descargar estos datos presionando el botón *Save Attached Files*. Mientras estos archivos están siendo descargados, la interfaz gráfica queda inhabilitada.

Luego de haber descargado los archivos, se debe presionar el botón *Start ROS* para inicializar la comunicación entre el drone y el computador. El botón *Draw Trajectory* permite visualizar previamente la trayectoria que el drone va a seguir.

Posteriormente se debe presionar el botón *Follow Trajectory* el cual ejecuta el algoritmo del diagrama de flujo presentado en la Figura 4.5, en el cual se envía una serie de comandos para que el drone realice el seguimiento de la trayectoria previamente dibujada en la *app*.

Finalmente, tras haber terminado el proceso de seguimiento de trayectorias, se debe presionar el botón *Plot Results* el cual muestra los resultados obtenidos para cada eje y sus errores correspondientes. Aunque no se planteó la elaboración de una interfaz gráfica en el alcance del proyecto, se la ha realizado para facilitar el entendimiento del proceso a las personas que no están familiarizadas con los comandos de consola en *Linux*.

CAPÍTULO 5

PRUEBAS Y RESULTADOS

En este capítulo se presentan las pruebas y resultados obtenidos en el proyecto. Estos resultados se han dividido en dos partes las cuales se centran en las simulaciones realizas en *MATLAB* y las pruebas de la implementación del controlador en el quadrotor con sus respectivos resultados. Finalmente, se justifican los errores obtenidos en la implementación del controlador en el AR.Drone 2.0.

5.1 SIMULACIONES

5.1.1 TRAYECTORIA LINEAL

La primera simulación realizada es el seguimiento de una trayectoria lineal, es decir, una línea recta. Por facilidad esta trayectoria se encuentra ubicada en el primer octante, y está definida por el siguiente conjunto de ecuaciones:

$$\begin{cases} x = x_0 + V_x \cdot t \\ y = y_0 + V_y \cdot t \\ z = Z \end{cases} \tag{5.1}$$

donde x_0 y y_0 son las posiciones iniciales para el eje x y y respectivamente, V_x y V_y son velocidades constantes para los ejes x y y respectivamente y Z es una constante positiva que representa la altura fija a la que el drone sigue la trayectoria. Los valores que se emplearán para estás constantes son:

$x_0 = 0.25\ [m]$, $y_0 = 0.25\ [m]$, $Z = 1.25\ [m]$, $V_x = 0.50\ [m/s]$ y $V_y = 0.25\ [m/s]$.

El tiempo de muestreo T_0 es de $100\ [ms]$, mientras que las constantes de control k_x, k_y y k_z toman los valores 0.97, 0.95 y 0.91 respectivamente. El tiempo de duración de la simulación es de veinte segundos. Los resutados obtenidos tras correr la simulación se presentan en la Figura 5.1. También se ha hecho una gráfica en tres dimesiones de la trayectoria seguida por el quadrotor tanto en la simulación continua como en la discreta, la cual se muestra en la Figura 5.2.

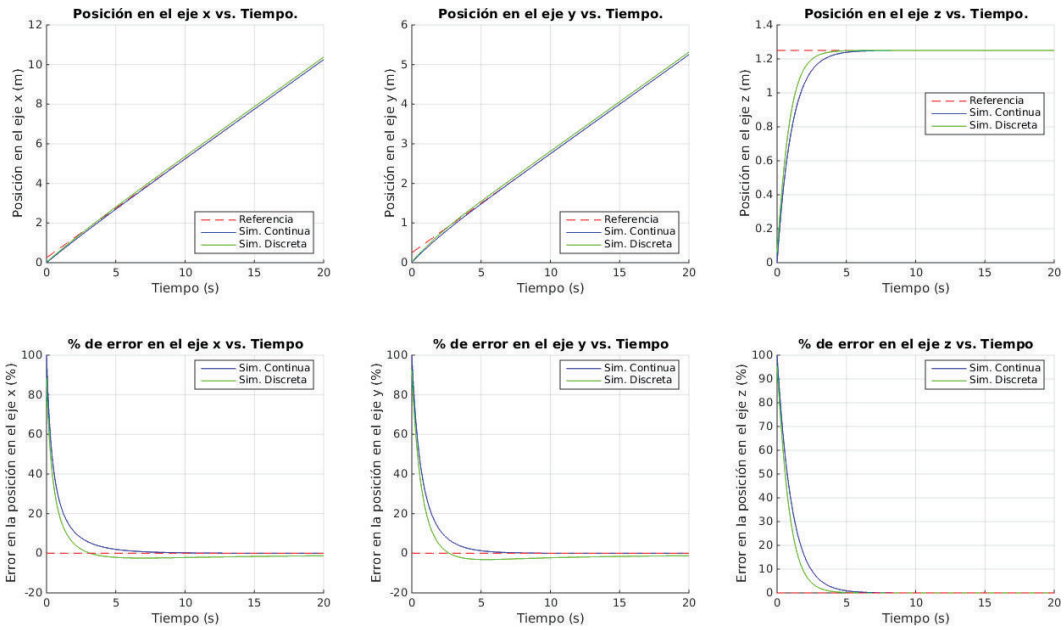


Figura 5.1: Resultados de la simulación de una trayectoria lineal en el primer octante.

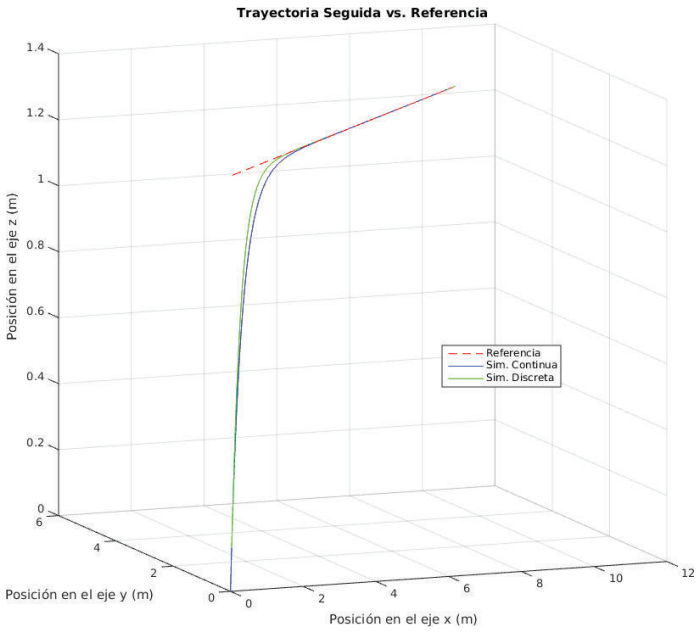


Figura 5.2: Gráfica en tres dimensiones de la trayectoria lineal en el primer octante.

Los resultados logrados en esta primera simulación son muy satisfactorios porque para los tres ejes x, y y z luego de 5 segundos se consigue la estabilización tanto para el modelo continuo como para el discreto. Nótese también cómo el error va tendiendo a cero paulatinamente en los ejes x y y.

5.1.2 TRAYECTORIA PARABÓLICA

La siguiente simulación será de una trayetoria parabólica en el primer octante. En el eje x se tendrá un movimiento rectilíneo uniforme, mientras que en el eje y se tendrá un movimiento acelerado. De igual manera que en la simulación anterior, el movimiento en el eje z se mantiene a una altura constante Z. Las ecuaciones para el moviento del quadrotor en esta simulación están dadas a continuación:

$$\begin{cases} x = x_0 + V_x \cdot t \\ y = y_0 + \dfrac{A_y}{2} \cdot t^2 \\ z = Z \end{cases} \tag{5.2}$$

Haciendo un breve análisis de las ecuaciones anteriores se tiene que el movimiento en el eje x es rectilíneo uniforme, cuya velocidad constante es V_x. Por otra parte, en el eje y el movimiento es rectilíneo uniformemente variado cuya aceleración es V_y. Los valores de las constantes que se han escogido son los siguientes:

$x_0 = 0.35\ [m]$, $y_0 = 0.50\ [m]$, $Z = 1.5\ [m]$, $V_x = 0.35\ [m/s]$ y $A_y = 0.01\ [m/s^2]$.

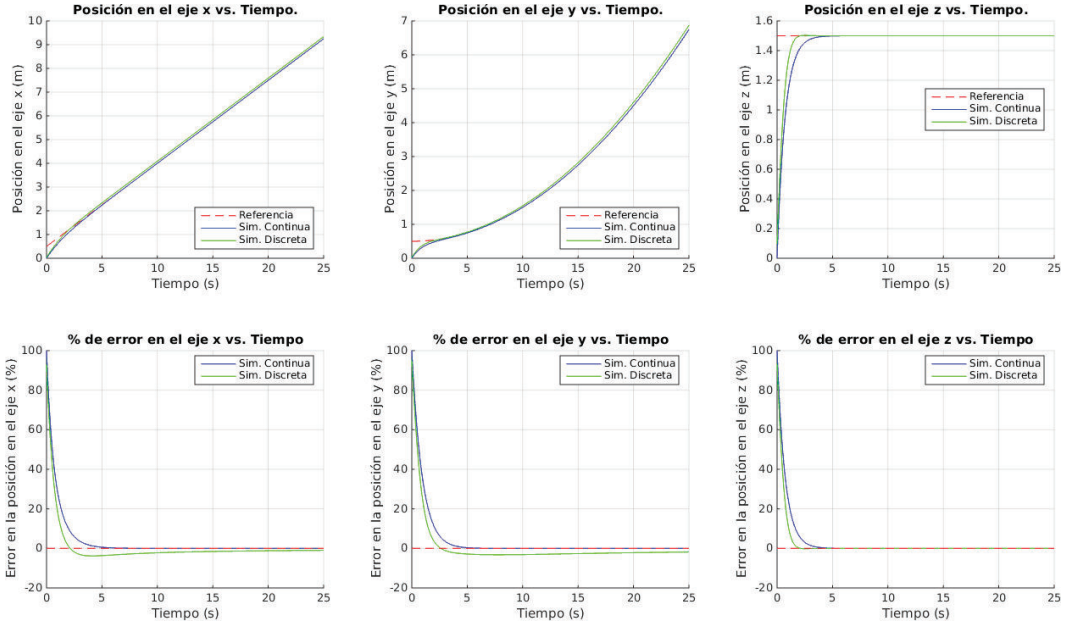


Figura 5.3: Resultados de la simulación de una trayectoria parabólica en el primer octante.

El tiempo de duración de esta simulación es de 25 segundos. En la Figura 5.3 se

presentan los resultados que se han obtenido luego de haber ejecutado la simulación. Los valores de las constantes de control son 0.93 para k_x, 0.90 para k_y y 0.87 para k_z. El tiempo de muestreo es nuevamente 100 milisegundos. Por otra parte, en la Figura 5.4 se muestra una gráfica de la trayectoria parabólica en tres dimensiones.

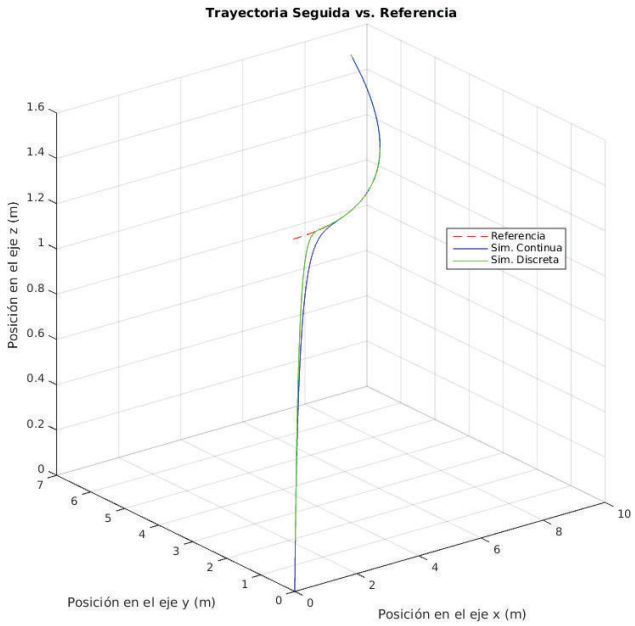


Figura 5.4: Gráfica en tres dimensiones de la trayectoria parabólica en el primer octante.

De igual forma que con la simulación anterior, los resultados obtenidos han sido muy satisfactorios pues se observa claramente que luego de 3 segundos de haber empezado la simulación los errores para los tres ejes x, y y z se aproximan a cero.

5.1.3 TRAYECTORIA SINUSOIDAL

Finalmente, se simulará el seguimiento de una trayectoria sinusoidal. Esta trayectoria está descrita por el siguiente conjunto de ecuaciones paramétricas:

$$\begin{cases} x = x_0 + V_x \cdot t \\ y = y_0 + Am_y \cdot \sin(2 \cdot \pi \cdot f \cdot t) \\ z = Z \end{cases} \tag{5.3}$$

Haciendo un breve análisis, en el eje x tenemos un movimiento rectilíneo uniforme cuya velocidad constante es V_x y cuya posición inicial es x_0. En el eje y se tiene un

movimiento sinusoidad cuya amplitud, frecuencia y posición inicial son Am_y, f y y_0 respectivamente. Una vez más se mantiene una altura constante Z. La frecuencia f nos indica el número de oscilaciones que se realizan cada segundo, es decir, $1/f$ es el tiempo necesario para completar una oscilación. En esta simulación los valores que se han escogido son los siguientes:

$$x_0 = 0.25\ [m],\ y_0 = 0.50\ [m],\ Z = 3\ [m],\ V_x = 0.40\ [m/s],\ Am_y = 3\ [m] \text{ y } f = \frac{1}{30}\ [Hz].$$

El valor del tiempo de muestreo T_0 sigue siendo $100\ [ms]$, mientras que el valor de las constantes de control es de 0.93 para cada una. El tiempo de simulación es 30 segundos, tiempo necesario para que se complete una oscilación. Los resultados de esta simulación se muestran en las Figura 5.5.

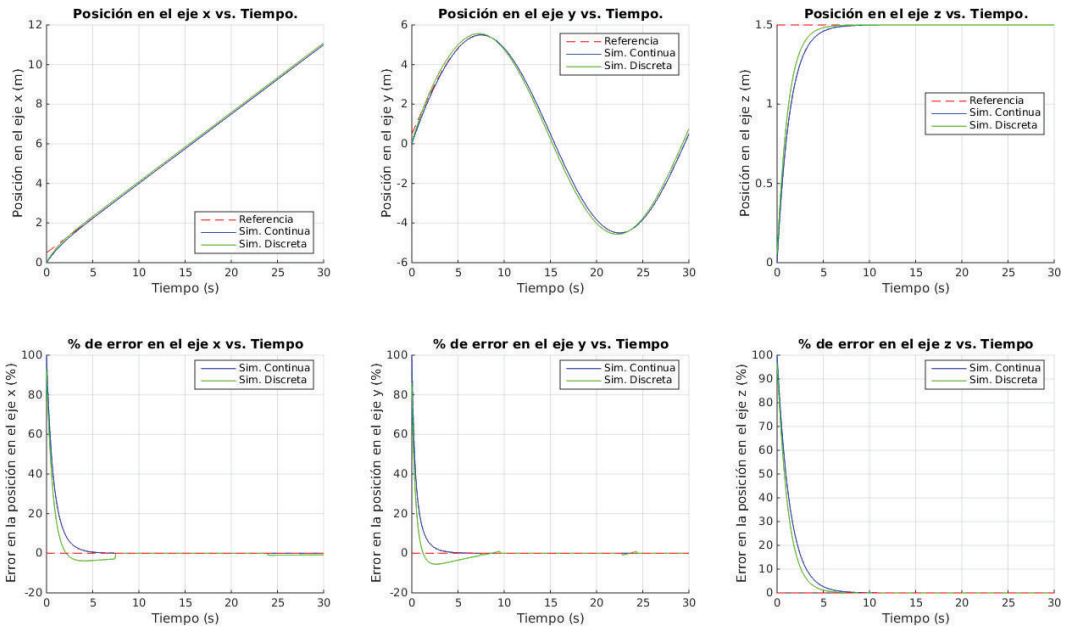


Figura 5.5: Resultados de la simulación de una trayectoria sinusoidal en el primer octante.

A pesar de cambiar la trayectoria de una lineal a una parabólica o a una sinusoidal los resultados que se obtienen siguen siendo satisfactorios. En esta simulación se tiene un error menor al 5% luego de 5 segundos en los tres ejes. En la Figura 5.6 se puede observar la trayectoria seguida en la simulación, la cual ha sido graficada en tres dimensiones.

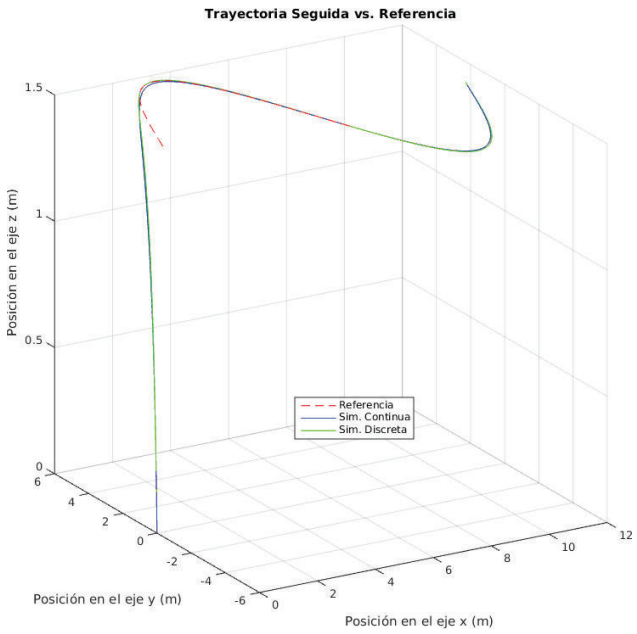


Figura 5.6: Gráfica en tres dimensiones de la trayectoria sinusoidal.

Tras haber realizado las simulaciones, se puede concluir si es factible o no implementar el controlador en el quadrotor real. En base a los resultados obtenidos, se concluye que es posible la implementación de este controlador, basado en métodos numéricos, pues los errores son menores a un 5% en aproximadamente 5 segundos luego de haber arrancado la simulación.

5.2 EXPERIMENTACIÓN

En la parte experimental se utiliza la aplicación móvil para la generación de la trayectoria que el quadrotor va a seguir, y se usa la interfaz gráfica de usuario para ejecutar el seguimiento de trayectorias y la visualización de resultados. Estas pruebas experimentales han sido realizadas en la cancha de fútbol del edificio de doctorado de la FLACSO, el cual se ubica en el sector La Pradera de la ciudad de Quito.

Dado que los resultados experimentales no han sido tan sobresalientes como los de la simulación, se han adjuntado tres resultados para cada una de las trayectorias con el fin de mostrar la evolución de los mismos. Estos tres resultados corresponden a los mejores, peores e intermedios resultados obtenidos.

Para cada trayectoria se presentan dos figuras. La primera figura muestra los resul-

tados para cada eje con sus respectivos errores. En una misma figura se presentan los resultados de las tres pruebas para la trayectoria. En la segunda figura se muestran dos capturas de la gráfica en tres dimensiones de la trayectoria seguida por el quadrotor.

Para cada resultado se utiliza la misma trayectoria, pero se cambian los valores de las constantes de control k_x, k_y y k_z y también el valor máximo de la velocidad en cada eje V_X, V_Y y V_Z. El valor que puede tomar cada constante de control está en el intervalo entre 0 y 1, mientras que el valor máximo de la velocidad para cada eje está en el rango de 200 [mm/s] y 2000 [mm/s].

Los valores de las constantes de control se pueden ajustar usando la aplicación móvil o entrando directamente al código, mientras que los valores de las velocidades máximas para cada eje solamente se pueden ajustar entrando al código. Algunas fotografías de esta parte experimental se han adjuntado en el Anexo D donde se puede observar al quadrotor con las cubiertas de interiores y de exteriores en la cancha mencionada.

5.2.1 TRAYECTORIA LINEAL

La primera trayectoria para las pruebas será una trayectoria lineal en el primer octante. La altura que se ha escogido para esta prueba ha sido de 100 [cm], y posteriormente se ha dibujado una trayectoria lo más lineal posible. A continuación se presentan el valor de las constantes utilizadas para cada prueba:

- **Mejores Resultados:** $k_x = 0.92$, $k_y = 0.93$, $k_z = 0.88$, $V_X = 1850$ [mm/s], $V_Y = 1800$ [mm/s] y $V_Z = 1700$ [mm/s].
- **Resultados Intermedios:** $k_x = 0.90$, $k_y = 0.90$, $k_z = 0.92$, $V_X = 1700$ [m/s], $V_Y = 1900$ [m/s] y $V_Z = 2000$ [m/s].
- **Peores Resultados:** $k_x = 0.85$, $k_y = 0.85$, $k_z = 0.92$, $V_X = 2000$ [m/s], $V_Y = 2000$ [m/s] y $V_Z = 2000$ [m/s].

En la Figura 5.7 se muestran los resultados para cada eje, mientras que, en la Figura 5.8 se muestran dos tomas de la gráfica en tres dimesiones de la trayectoria seguida por el quadrotor.

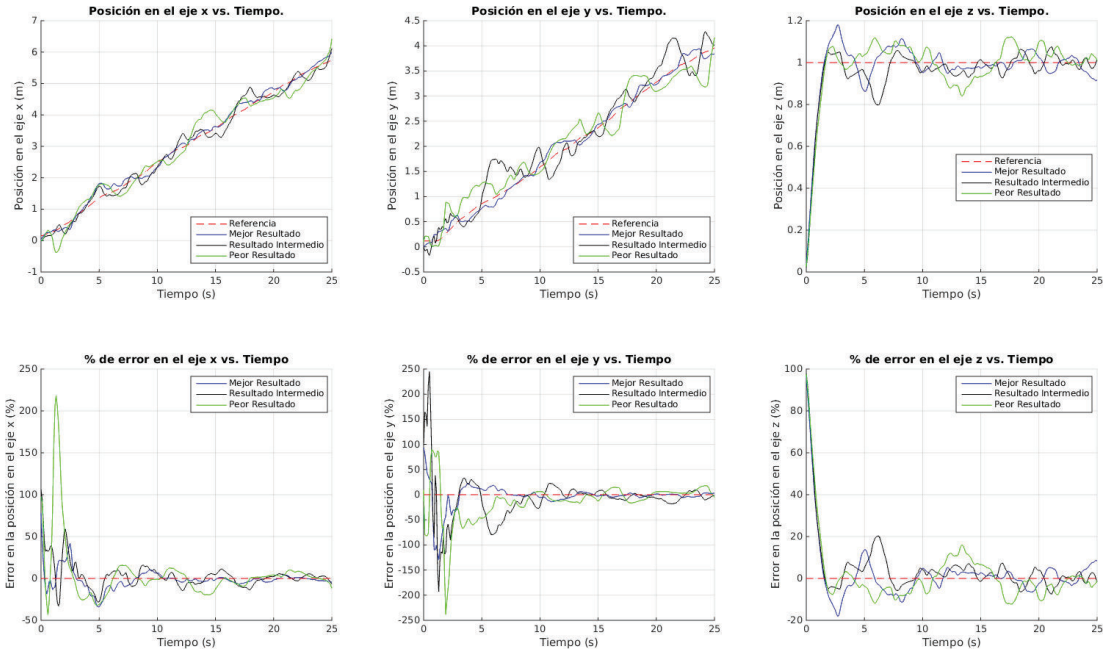


Figura 5.7: Resultados experimentales de la trayectoria lineal.

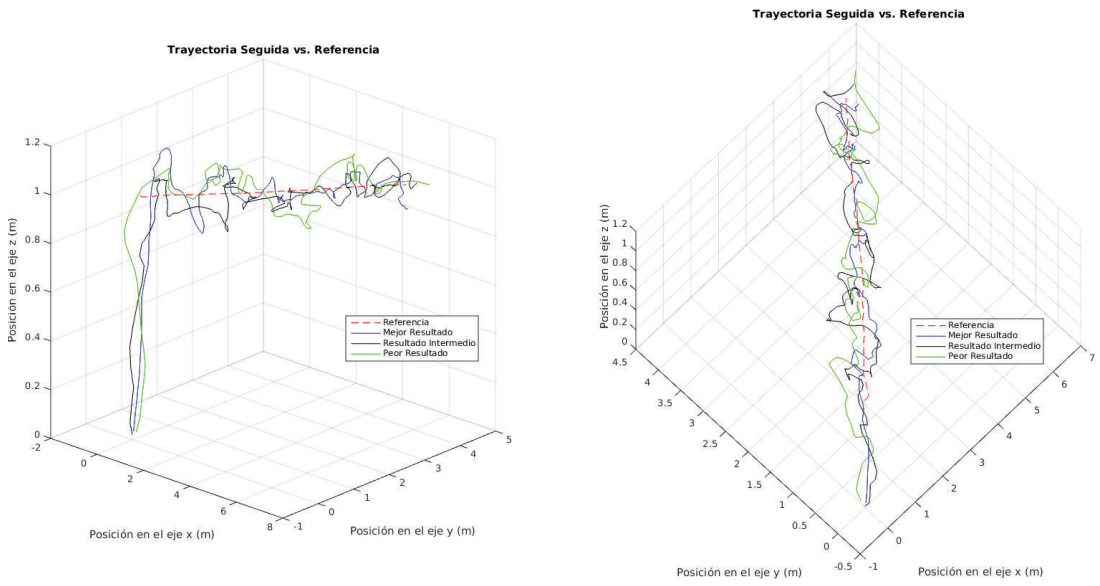


Figura 5.8: Gráfica en tres dimensiones de la trayectoria lineal.

5.2.2 TRAYECTORIA PARABÓLICA

La segunda trayectoria es una trayectoria parabólica en el primer octante. La altura que se ha escogido para esta prueba ha sido de 150 [cm]. A continuación se presentan el valor de las constantes utilizadas para cada prueba:

- **Mejores Resultados:** $k_x = 0.91$, $k_y = 0.90$, $k_z = 0.92$, $V_X = 1600$ [mm/s], $V_Y = 1700$ [mm/s] y $V_Z = 1800$ [mm/s].
- **Resultados Intermedios:** $k_x = 0.94$, $k_y = 0.91$, $k_z = 0.91$, $V_X = 1800$ [m/s], $V_Y = 1700$ [m/s] y $V_Z = 1900$ [m/s].
- **Peores Resultados:** $k_x = 0.87$, $k_y = 0.89$, $k_z = 0.93$, $V_X = 1950$ [m/s], $V_Y = 1950$ [m/s] y $V_Z = 2000$ [m/s].

En la Figura 5.9 se muestran los resultados para cada eje, mientras que, en la Figura 5.10 se muestran dos tomas de la gráfica en tres dimesiones de la trayectoria.

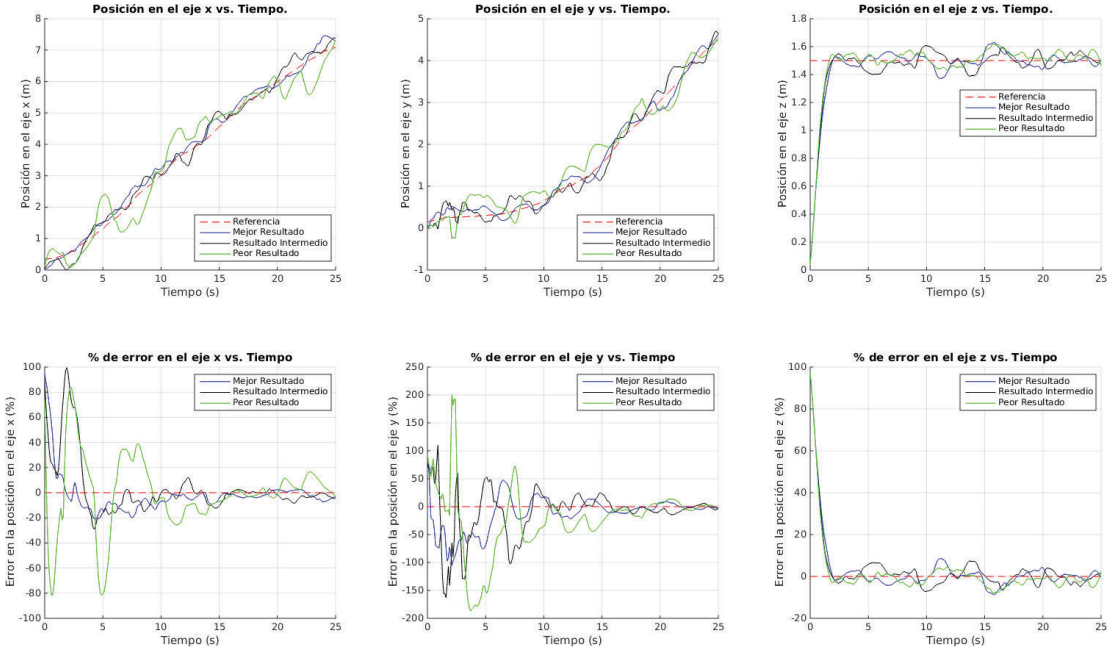


Figura 5.9: Resultados experimentales de la trayectoria parabólica.

5.2.3 TRAYECTORIA SINUSOIDAL

La tercera y última trayectoria es una trayectoria sinusoidal. La altura que se ha escogido para esta prueba ha sido de 200 [cm]. A continuación se presentan el valor de las constantes utilizadas para cada prueba:

- **Mejores Resultados:** $k_x = 0.91$, $k_y = 0.90$, $k_z = 0.91$, $V_X = 1800$ [mm/s], $V_Y = 1875$ [mm/s] y $V_Z = 1850$ [mm/s].
- **Resultados Intermedios:** $k_x = 0.89$, $k_y = 0.89$, $k_z = 0.91$, $V_X = 1900$ [m/s], $V_Y = 1950$ [m/s] y $V_Z = 1850$ [m/s].

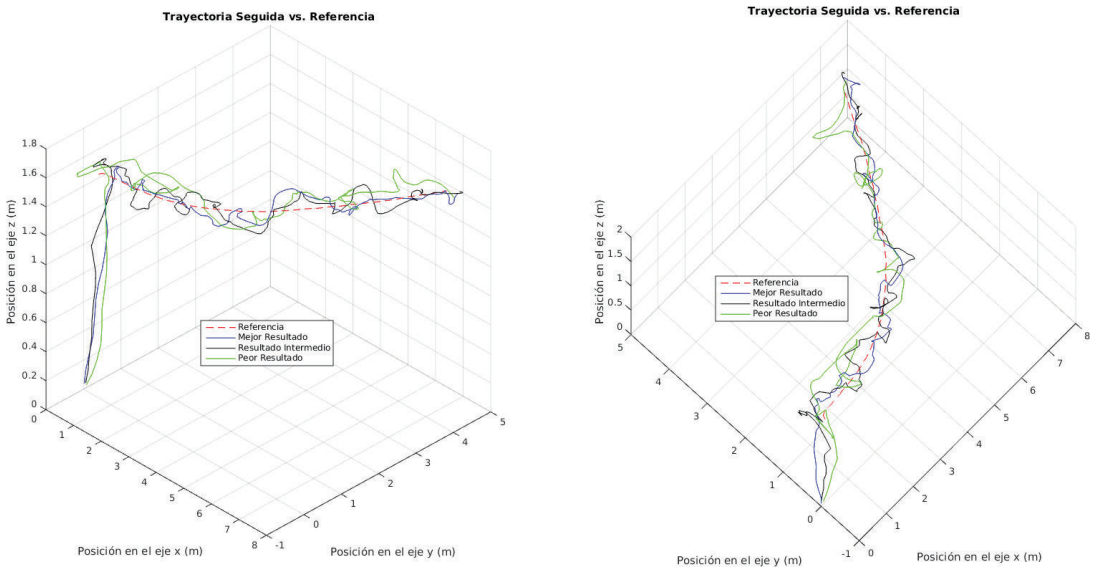


Figura 5.10: Gráfica en tres dimensiones de la trayectoria parabólica.

- **Peores Resultados:** $k_x = 0.86$, $k_y = 0.88$, $k_z = 0.91$, $V_X = 2000$ [m/s], $V_Y = 2000$ [m/s] y $V_Z = 1850$ [m/s].

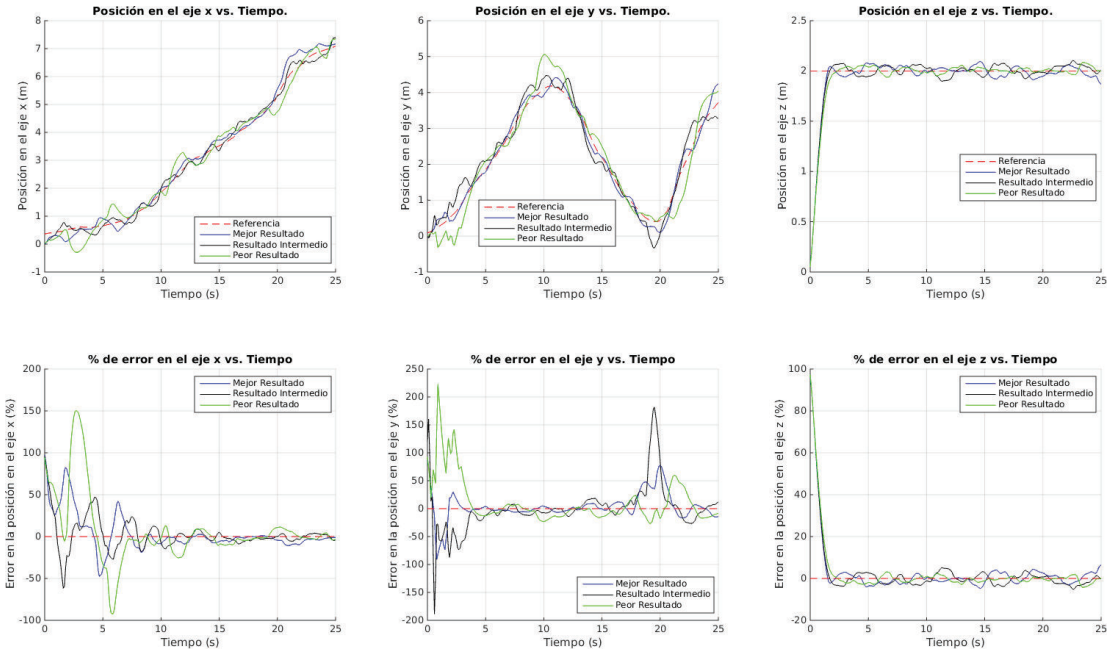


Figura 5.11: Resultados experimentales de la trayectoria sinusoidal.

En la Figura 5.11 se muestran los resultados para cada eje, mientras que, en la Figura 5.12 se muestran dos tomas de la gráfica en tres dimesiones de la trayectoria.

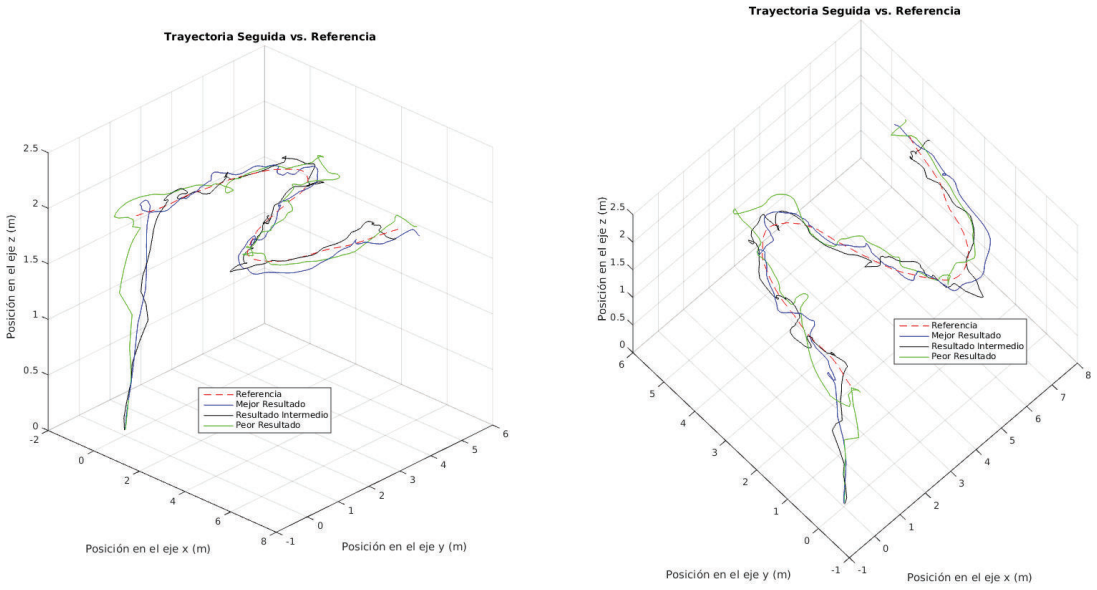


Figura 5.12: Gráfica en tres dimensiones de la trayectoria sinusoidal.

Se puede notar que cada una de las pruebas dura 25 segundos. Esto se debe a que en la aplicación móvil se ha realizado una distribución de los puntos que forman la trayectoria. El número de puntos que la trayectoria tiene se lo puede escoger y en este caso es 251. Si la trayectoria que el usuario ha dibujado tiene más de 251 puntos, el arreglo para cada eje pasa por una función que reduce los puntos y toma uniformente la cantidad de puntos deseados. Si la trayectoria tiene 251 puntos o un número menor, no se aplica esta reducción y distribución.

Este proceso se aplica para limitar el número de puntos de la trayectoria pues de no existir, el número de puntos dependería del tiempo en que el usuario tardo en dibujar la trayectoria. La frecuencia de muestreo de la pantalla tiene por defecto un valor de 60 [Hz]. Las trayectorias linear, parabólica y sinusoidal presentadas tenían originalmente 307, 264 y 261 puntos respectivamente. Tras ser aplicada la reducción y distribución de puntos, todas tenían 251 puntos lo cual hace que la prueba dure 25 segundos.

5.3 JUSTIFICACIÓN DE ERRORES

Los errores que se han obtenido en la parte experimental se deben principalmente a los sensores que posee el AR.Drone y a lo inestable que es la comunicación entre

el computador y el drone. Sin embargo, tener un quadrotor robusto, con buenos sensores y cuya comunicación sea estable, supone una inversión superior a $7,000, lo cual generalmente no está al alcance de un estudiante.

5.3.1 SENSORES

El AR.Drone 2.0 fue fabricado para ser un drone de bajo costo que podía ser utilizado como juguete sofisticado por niños y aficionados a los UAVs. Los sensores de este quadrotor son de bajo costo; por ejemplo, la *IMU* tiene un valor menor a $10. Tener una *IMU* de bajo costo significa que es necesaria una calibración de los acelerómetros y giroscopios, el uso de sensores externos o la incorporación de sensores adicionales en el quadrotor.

Dado que el objetivo de este proyecto era la implementación de un controlador basado en métodos numéricos usando álgebra lineal, no se ha realizado ninguna de las alternativas anteriores. Además, la implementación de estas tres alternativas constituye en sí un proyecto de sensado lo cual va más allá del alcance establecido en el proyecto.

En [8] se tiene el mismo problema que ha sido mencionado con los sensores del drone. En este paper se manifiesta que los datos leídos de los sensores afectan a la odometría. También se sugiere que la forma de arreglar esto es utilizando un sensor GPS o un sistema de visión artificial.

Es importante manifestar que los errores de la *IMU* se ven reflejados en los ejes x y y ya que en el eje z se tiene una mejor respuesta en todas las pruebas realizadas. Esto se debe a que el dato de la posición en z es leído directamente del sensor ultrasónico colocado en la parte inferior del quadrotor. Por lo tanto, no es necesario integrar los datos leídos de los sensores como se hace para obtener las posiciones en los ejes x y y.

5.3.2 COMUNICACIÓN

La comunicación entre el quadrotor y la computadora es a través de conexión Wi-Fi. Esto implica que la comunicación se puede ver afectada por la presencia de otras redes Wi-Fi porque muchos otros dispositivos usan la misma frecuencia para comu-

nicarse e incluso el mismo canal. [32] Además, es necesario que no se presenten obstácuos, como muros gruesos, pues estos debilitan la intensidad de la señal.

El API que se ha utilizado en el proyecto es una libería abierta pubicada en GitHub a la que ocasionalmente envían reportes con problemas presentados, los cuales son atendidos y en la mayoría de los casos solucionados. Uno de los problemas reportados es la falla en la comunicación.

Por ejemplo, el 14 de julio de 2015 se reportó un problema con el tiempo de muestreo. Este tiempo tiene un valor por defecto de 200 [Hz]. Sin embargo, como se indica en [11] cada uno o dos minutos esta frecuencia de muestreo se reduce a un valor entre 8-20 [Hz]. Esta condición dura alrededor de 6 segundos antes de volver a la frecuencia normal. En la Figura 5.13 se puede ver una captura de pantalla de este fenómeno.

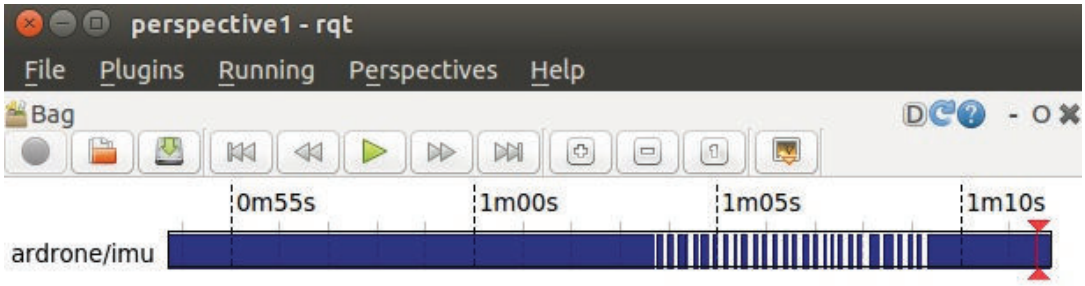


Figura 5.13: Fenómeno en el tiempo de muestreo.

Cuando se realiza seguimiento de trayectorias, el tiempo de muestreo es crucial y siempre debe respetarse. El efecto que tiene este fenómeno es que muchos de los datos se están perdiendo. Por lo tanto, el computador está recibiendo valores incorrectos y realiza los cálculos de posición y acciones de control con estos datos. Si a este fenómeno se le suma el problema que presentan los sensores, se puede entender claramente porque se llega a tener errores que varían desde el 30% hasta incluso un 200%.

Otro problema reportado el 26 de julio de 2014 fue la pérdida total de a comunicación la cual se daba aleatoriamente. Como se indica en [12] este problema se debe a que la conexión Wi-Fi no es muy estable. Una posible solución dada por el creador del

API es utilizar un router externo para la comunicación y no directamente la tarjeta WiFi de una laptop. En la Figura 5.14 se puede ver una captura de pantalla de los mensajes impresos en consola de este fenómeno.

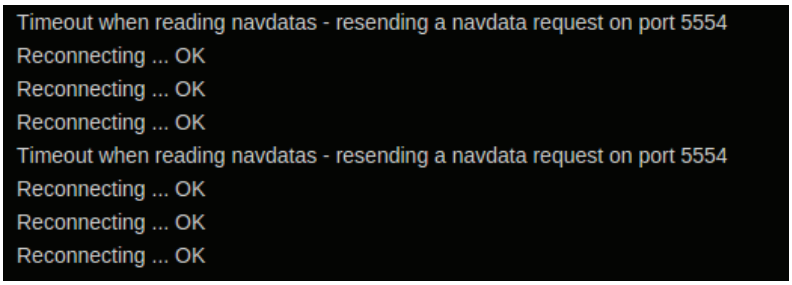


Figura 5.14: Problema en la conexión entre el AR.Drone 2.0 y la computadora.

En el proyecto se han experimentado ambos problemas indicados. Sin embargo, el segundo problema se ha dado raramente y generalmente ha sido en el momento de empezar la comunicación entre la laptop y el quadrotor. Por lo tanto, no se ha visto la necesidad de usar un router aunque lo apropiado sería utilizarlo. El primer problema se ha podido notar claramente y se ve reflejado en las pruebas y resultados presentados. Lastimosamente el creador del API todavía no ha dado una solución a este problema.

CAPÍTULO 6

CONCLUSIONES Y RECOMENDACIONES

Finalmente, para concluir con este trabajo escrito, en este capítulo se expondrán las conclusiones y recomendaciones a las que se han llegado con este proyecto.

6.1 CONCLUSIONES

- Se realizó el diseño de un controlador, basado en métodos numéricos que emplea álgebra lineal, el cual ha sido implementado en el quadrotor AR.Drone 2.0 para el seguimiento de trayectorias preestablecidas. Tanto el diseño como la implementación del controlador fueron bastante sencillos por el hecho de utilizar álgebra lineal ya que los procesos matemáticos y su codificación se facilitan.

- Se ha elaborado un marco teórico sobre el diseño de controladores que usan álgebra lineal, el cual muestra el diseño de un controlador para un robot móvil. Aquí se ha evidenciado la facilidad del diseño de estos controladores y también se ha presenciado la inclusión de las constantes de control las cuales permiten que el error tienda a cero paulatinamente haciendo que los valores de las acciones de control no sean elevados.

- Se realizó el análisis de la estructura del quadrotor y sus parámetros más importantes. Sin embargo, en el modelo del drone dado por el fabricante, todos estos parámetros han sido abstraídos y las ecuaciones dependen directamente de las acciones de control. Aunque se pudo haber utilizado otro modelo, en el que estos parámetros estén presentes, por facilidad se terminó usando el modelo del fabricante.

- Pese a que el fabricante sugiere el modelo utilizado, durante la identificación y validación del modelo se pudo notar que este no representaba de buena forma al quadrotor real. En esta etapa del proyecto se pudo haber optado por utilizar otro drone, pero debido a la falta de financiamiento se siguió usando el AR.Drone 2.0. Tras el diseño del controlador y su implementación en el AR.Drone, se vieron

reflejados errores bastante altos los cuales se deben principalmente a que los sensores internos del quadrotor no son de buena calidad lo cual afecta a la odometría. Estos errores altos se deben también a que la comunicación Wi-Fi entre el computador y el drone no es estable y en ocasiones no se está respetando el tiempo de muestreo.

- Se evidenciaron perturbaciones debido al viento en las pruebas experimentales. Sin embargo, en cada ocasión que una perturbación se presentaba, el controlador corregía este error haciendo que tienda a cero de forma gradual y logrando que el quadrotor pueda continuar con el seguimiento de la trayectoria preestablecida.

- La programación orientada a objetos permite realizar el modelado de problemas reales con el objetivo de ser resueltos o analizados utilizando un computador. Además, utilizar programación orientada a objetos permite la reutilización de código lo que facilita el mantenimiento y la depuración del programa desarrollado. El uso de POO en este proyecto ha permitido abstraer conceptos como sensor, posición y referencia los cuales juegan un papel esencial en el seguimiento de trayectorias. El desarrollo de clases para estos conceptos ha evitado repetición continua en el código y también ha facilitado todos los cambios que se han hecho en el modelo.

6.2 RECOMENDACIONES

- Si es posible, escoger un drone robusto para realizar cualquier proyecto con UAVs pues esto elimina la dependencia de sensores externos o un sistema de visión artificial para obtener la posición en la que se encuentra. Además, un drone robusto tiene una respuesta suave y la comunicación es estable. Por lo tanto, uno se puede enfocar en la implementación de controladores o algoritmos de control.

- Es muy importante elegir adecuadamente el framework o API con el que se vaya a trabajar. Originalmente para este proyecto se escogió un API llamado *AutoFlight*, el cual estaba en un estado *alpha* desde enero de 2014, e ingenuamente se creyó que pasaría al menos a una versión *beta* más estable. Sin embargo, actualmente sigue en este estado y sin ningún reporte acerca de una futura actualización.

Luego se utilizó NodeCopter como API, el cual es multiplataforma y tiene un conjunto de servicios que se pueden consumir para controlar al AR.Drone. Sin embargo, los problemas de comunicación entre el drone y la computadora eran bastante frecuentes, y la falta de conocimiento sobre programación asíncrona hizo que se descarte este API. Finalmente, se terminó escogiendo ROS el cual había sido utilizado en algunos proyectos anteriormente y que ofrecía una gran cantidad de librerías predefinidas para el envío de comandos y la recepción de datos. Por lo tanto, se recomienda consultar proyectos elaborados anteriormente, sus resultados y problemas que presenten antes de ponerse a trabajar con un API.

- Se recomienda utilizar programación orientada a objetos (POO) porque permite resolver problemas mediante el modelado de un sistema. Esto significa que se pueden abstraer conceptos reales y sus interacciones con los seres humanos. POO permite también la reutilización de código, lo cual facilita el mantenimiento del programa y la corrección de errores. Probablemente, dado que en la carrera de Control no se ofrece la materia de POO, al estudiante le tome tiempo aprender este paradigma de programación, pero por los beneficios a largo plazo es una buena inversión de tiempo.

- Recomiendo encarecidamente realizar los experimentos con el quadrotor en un espacio cerrado y amplio; de tal forma que el viento exterior no influya en el seguimiento de la trayectoria y no se tenga tantas limitaciones de espacio en las trayectorias a definir. Además, generalmente se recomienda no usar la cubierta para interiores dado que tiende también a desestabilizar al quadrotor durante el vuelo aunque al no usarlo se corre un alto riesgo de dañar las hélices en caso de existir un impacto.

- Si se quiere tomar este proyecto como punto de partida para realizar uno nuevo, recomiendo utilizar uno o varios Kinect como sensores externos para calcular la posición del drone. También sugiero implementar un sistema de sensores de buena calidad en el mismo quadrotor. Finalmente, recomiendo el uso de un router en lugar de la tarjeta de comunicación inalámbrica de la laptop para tener una comunicación más robusta entre el computador y el drone.

REFERENCIAS BIBLIOGRÁFICAS

[1] A. Rosales, V. Andaluz, y G. Andaluz, "Modelación, Identificación y Control de Robots Móviles", Facultad de Ingeniería en Sistemas, Electrónica e Industrial, Universidad Técnica de Ambato, Ambato, Ecuador, 2012.

[2] Amazon (2012). Parrot AR Drone 2.0 EPP Bottom Structure. [Online]. Available: http://www.amazon.com/Parrot-Drone-2-0-Bottom-Structure/dp/B007HZLT4W/

[3] Amazon (2013). Parrot AR Drone 2.0 Navigation board. [Online]. Available: http://www.amazon.com/Parrot-Drone-2-0-Navigation-board/dp/B007HZLOSS/

[4] Amazon (2012). Syma X1 4 Channel 2.4G RC Quad Copter. [Online]. Available: http://www.amazon.com/Syma-Channel-2-4G-Quad-Copter/dp/B00906PKQ4

[5] AscTec (2013). Falcon 8. [Online]. Available: http://www.asctec.de/en/uav-uas-drones-rpas-roav/asctec-pelican/

[6] C. Nadales, "Control de un Quadrotor mediante la plataforma Arduino", tesis de masterado, Departamento de Automática, Universitat Politècnica de Catalunya, Barcelona, España, 2009.

[7] C. Rosales, G. Scaglia, M. Sarcinelli, y R. Carelli, "Control de trayectoria de un helicóptero de cuatro rotores basado en métodos numéricos", Instituto de Automática, Universidad de San Juan Argentina, San Juan, Argentina, 2006.

[8] C. Rosales, G. Scaglia, A. Brandão, M. Sarcinelli y R. Carelli, "Trajectory Tracking For A Four Rotor Mini-Quadrotor", Automation Institute, University of San Juan, San Juan, Argentina, 2008.

[9] DIYdrones (2010). Drones: a history of flying robots. [Online]. Available: http://www.nesta.org.uk/node/753

[10] F. Ruiz, "Identificación de Sistemas, Identificación no paramétrica", Maestría en Ingeniería Electrónica, Pontifica Universidad Javeriana, Bogotá, Colombia, 2013.

[11] GitHub (2015). Periodically reduced sampling time. [Online]. Available: https://github.com/AutonomyLab/ardrone_autonomy/issues/153

[12] GitHub (2014). Timeout when reading navdatas. [Online]. Available: https://github.com/AutonomyLab/ardrone_autonomy/issues/118

[13] G. Scaglia, L. Quintero, V. Mut, y F. Di Sciascio, "Numerical methods based controller design for mobile robots", Automation Institute, University of San Juan, San Juan Argentina, 2010.

[14] J. Van der Spek, and M. Voorsluys, "AR.Drone Autonomous Control and Position Determination", Bachelor Thesis, Faculty of Electrical Engineering, Delft University of Technology, Delft, Netherlands, 2012.

[15] Livescience (2013). 9 Totally Cool Uses for Drones. [Online]. Available: http://www.livescience.com/28137-cool-uses-for-drones.html

[16] M. Guillén, "Identificación de Sistemas", Departamento de Electrónica, Instituto Tecnológico de Costa Rica, Costa Rica, 2013.

[17] MultiWii (2012). Multicopter Types. [Online]. Available: http://www.multiwii.com/wiki/index.php?title=Multicopter_Types

[18] N. Dijkshoorn, "Simultaneous localization and mapping with the AR.Drone". M.S thesis, Computer Science Department, Universiteit van Amsterdam, Amsterdam, Netherlands, 2012.

[19] National Geographic News (2013). 5 Surprising Drone Uses. [Online]. Available: http://goo.gl/GJOrVl

[20] Online Etymology Dictionary (2001). Drone. [Online]. Available: http://www.etymonline.com/index.php?term=drone

[21] Piskorski N., Brulez N., Eline P. y D'Haeyer F. AR.Drone. Developer Guide SDK 2.0. [Online]. Available: http://www.msh-tools.com/ardrone/ARDrone_Developer_Guide.pdf

[22] T. Krajník, V. Vonásek, D. Fišer, J. Faigl, "AR-Drone as a Platform for Robotic Research and Education", The Gerstner Laboratory for Intelligent Decision Making and Control Department of Cybernetics, Czech Technical University in Prague, Prague, Czech Republic, 2012.

[23] Wikipedia (2010). Unmanned Aerial Vehicle. [Online]. Available: http://en.wikipedia.org/wiki/UAV

[24] Wikipedia (2011). Quadcopter. [Online]. Available: http://en.wikipedia.org/wiki/Quadcopter

[25] Wikipedia (2010). History of unmanned aerial vehicles. [Online]. Available: http://en.wikipedia.org/wiki/History_of_unmanned_aerial_vehicles

[26] Wikipedia (2009). Pulsejet. [Online]. Available: http://en.wikipedia.org/wiki/Pulsejet

[27] Wikipedia (2013). Parrot(company). [Online]. Available: http://en.wikipedia.org/wiki/Parrot_AR.Drone

[28] Wikipedia (2013). DJI UAV. [Online]. Available: http://en.wikipedia.org/wiki/DJI_UAV

[29] Wikipedia (2012). ArduPilot. [Online]. Available: https://en.wikipedia.org/wiki/Ardupilot

[30] Wikipedia (2005). Euler angles. [Online]. Available: http://en.wikipedia.org/wiki/Euler_angles

[31] Y. Sun, "Modeling, Identification and Control of a Quadrotor Drone Using Low-Resolution Sensing", M.S. thesis, Mechanical Engineering, University of Illinois at Urbana-Champaign, Champaign, IL, United States, 2012.

[32] Youtube. Los Problemas más comunes de las redes Wi-Fi y cómo solucionarlos. [Online]. Available: https://www.youtube.com/watch?v=kVJ_CRKgRmU

ANEXOS

ANEXO A

CÓDIGO DEL CONTROLADOR

Este código ha sido usado tanto en la simulación continua como en la simulación discreta.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%% AR.DRONE 2.0 CONTROLLER %%%%%%%%%%%%%%%%%%%%%%%%%%%
function [V_z_n, V_y_n, V_x_n, PSI, x_ref_n, y_ref_n, z_ref_n] = ...
    controller(x_n, y_n, z_n, t_n)
%% Constants
T0 = getSampleTime();
Tf = 2 * T0;
[K_V_X, K_V_Y, K_V_Z] = getControlConstants();
PSI = getPsi();
%% Controller Logic
% Reference Trajectory at n
[x_ref_n, y_ref_n, z_ref_n]  = getReferences(t_n);
% Reference Trajectory at n + 1
[x_ref_np1, y_ref_np1, z_ref_np1] = getReferences(t_n + T0);
% Control Actions
V_x_n = (1 / T0) * ((x_ref_np1 - K_V_X * (x_ref_n - x_n) - x_n) * ...
    cos(PSI) + (y_ref_np1 - K_V_Y * (y_ref_n - y_n) - y_n) * sin(PSI));

V_y_n = (1 / T0) * (-(x_ref_np1 - K_V_X * (x_ref_n - x_n) - x_n) * ...
    sin(PSI) + (y_ref_np1 - K_V_Y * (y_ref_n - y_n) - y_n) * cos(PSI));

V_z_n = (1 / T0) * (z_ref_np1 - K_V_Z * (z_ref_n - z_n) - z_n);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Las funciones **getSampleTime**, **getPsi**, **getControlConstants**, y **getReferences** se definen a continuación. La razón de modularizar tanto como sea posible el código se debe a que facilita su lectura y la corrección de posibles errores.

getSampleTime.m

```
function T0 = getSampleTime()
% sample time T0 is defined in seconds
```

```
T0 = 0.1;
end
```

getPsi.m

```
% psi angle is given in radians
function psi = getPsi()
psi = 0;
end
```

getControlConstants.m

```
function [K_V_X, K_V_Y, K_V_Z] = getControlConstants()
% 0 < K < 1
K_V_X = 0.90;
K_V_Y = 0.90;
K_V_Z = 0.90;
end
```

getReferences.m

```
function [x_ref, y_ref, z_ref] = getReferences(t)
% define the reference for each axis as a parametric equation
x_ref = 0.25 + 0.1 * t;
y_ref = 0.25 + 0.2 * t;
z_ref = 1.5;
end
```

ANEXO B

CÓDIGO DEL MODELO DISCRETO

A continuación se presenta el código del modelo discreto del AR.Drone 2.0 que se ha utilizado en las simulación discreta.

ARDrone2_Discrete_Model.m

```
%%%%%%%%%%%%%%%%%%%%%%%%AR.DRONE 2.0 DISCRETE MODEL%%%%%%%%%%%%%%%%%%%%%%%%%
function output = ARDrone2_Discrete_Model(input)
%% Constants
% Sampling time
T0 = getSampleTime();
%% Load Inputs
PSI = input(1);
V_x = input(2);
V_y = input(3);
V_z = input(4);
t = input(5);
%% Load values computed in the previous iteration
values_nm1 = load('values_nm1.txt', 'values_nm1');
%% Index
index = t / T0;
%% values at n - 1
x_nm1 = (index == 0) * 0 + ~(index == 0) * values_nm1(1);
y_nm1 = (index == 0) * 0 + ~(index == 0) * values_nm1(2);
z_nm1 = (index == 0) * 0 + ~(index == 0) * values_nm1(3);
V_x_nm1 = (index == 0) * 0 + ~(index == 0) * values_nm1(4);
V_y_nm1 = (index == 0) * 0 + ~(index == 0) * values_nm1(5);
V_z_nm1 = (index == 0) * 0 + ~(index == 0) * values_nm1(6);
%% AR.Drone 2.0 Model
% at n - 1
% Velocities
x_dot_nm1 = computeXDot(V_x_nm1, V_y_nm1, PSI);
y_dot_nm1 = computeYDot(V_x_nm1, V_y_nm1, PSI);
z_dot_nm1 = V_z_nm1;
```

```
% at n
% Velocities
x_dot_n = computeXDot(V_x, V_y, PSI);
y_dot_n = computeYDot(V_x, V_y, PSI);
z_dot_n = V_z;
% Position
x_n = trapezoidalIntegration(x_nm1, x_dot_nm1, x_dot_n, T0);
y_n = trapezoidalIntegration(y_nm1, y_dot_nm1, y_dot_n, T0);
z_n = trapezoidalIntegration(z_nm1, z_dot_nm1, z_dot_n, T0);
%% Outputs
output = [x_n, y_n, z_n];
%% Save variables in a file
values_nm1 = [x_n, y_n, z_n, V_x, V_y, V_z];
save('values_nm1.txt','-ascii', 'values_nm1');
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

La función **getSampleTime** es la misma cuyo código se muestra en el anexo A, mientras que las funciones **computeXDot**, **computeYDot** y **trapezoidalIntegration** se definen a continuación.

computeXDot.m

```
function xDot = computeXDot(V_x, V_y, psi)
xDot = V_x * cos(psi) - sin(psi) * V_y;
end
```

computeYDot.m

```
function yDot = computeYDot(V_x, V_y, psi)
yDot = V_x * sin(psi) + cos(psi) * V_y;
end
```

trapezoidalIntegration.m

```
function result = trapezoidalIntegration(data_nm1, ...
data_dot_nm1, data_dot_n, T0)
result = data_nm1 + (T0 / 2) * (data_dot_nm1 + data_dot_n);
end
```

ANEXO C

SCRIPT PARA GRAFICAR RESULTADOS

El siguiente código se ha utilizado para graficar los resultados de la parte experimental. En una misma gráfica se muestran los mejores, intermedios y peores resultados que se obtuvieron para cada eje. Además, se realiza una gráfica en tres dimensiones para visualizar la trayectoria seguida.

```
% This program is used to plot the experimentation results
clear;
clc;
%% Load arrays
trajectory = 0;

switch trajectory
    case 0
        trajectory = 'linear';
    case 1
        trajectory = 'parabolic';
    case 2
        trajectory = 'sinusoidal';
end

%% Arrays that contain the best results
x_n_best = loadDataFromStruct(strcat(trajectory, '/x_n_best'), 0);
y_n_best = loadDataFromStruct(strcat(trajectory, '/y_n_best'), 1);
z_n_best = loadDataFromStruct(strcat(trajectory, '/z_n_best'), 2);
%% Arrays that contain the middle quality results
x_n_mid = loadDataFromStruct(strcat(trajectory, '/x_n_mid'), 0);
y_n_mid = loadDataFromStruct(strcat(trajectory, '/y_n_mid'), 1);
z_n_mid = loadDataFromStruct(strcat(trajectory, '/z_n_mid'), 2);
%% Arrays that contain the worst results
x_n_worst = loadDataFromStruct(strcat(trajectory, '/x_n_worst'), 0);
y_n_worst = loadDataFromStruct(strcat(trajectory, '/y_n_worst'), 1);
z_n_worst = loadDataFromStruct(strcat(trajectory, '/z_n_worst'), 2);
%% Arrays that contain the reference
x_ref_n = loadReferenceWithSize(strcat('references/', trajectory, ...
```

```
'/x_ref_n.txt'), size(x_n_best, 1));
y_ref_n = loadReferenceWithSize(strcat('references/', trajectory, ...
'/y_ref_n.txt'), size(x_n_best, 1));
z_ref_n = loadReferenceWithSize(strcat('references/', trajectory, ...
'/z_ref_n.txt'), size(x_n_best, 1));

t_n = 0: getSampleTime() : getSampleTime() * (size(x_n_best) - 1);

figure;
%% Plotting x-axis results
subplot(2, 3, 1);
xAxisResultVectors = {t_n, x_ref_n, x_n_best, x_n_mid, x_n_worst};
xAxisResultPlotTitle = 'Posición en el eje x vs. Tiempo.';
axisResultLegends = {'Referencia', 'Mejor Resultado', ...
'Resultado Intermedio', 'Peor Resultado'};
xAxisResultLabels = {'Tiempo (s)', 'Posición en el eje x (m)'};
plotAxisComparison(xAxisResultVectors, axisResultLegends, ...
xAxisResultPlotTitle, xAxisResultLabels);
%% Plotting x-axis errors
subplot(2, 3, 4);
xAxisErrorLegends = {'Mejor Resultado', 'Resultado Intermedio', ...
'Peor Resultado'};
xAxisErrorPlotTitle = '% de error en el eje x vs. Tiempo';
xAxisErrorLabels = {'Tiempo (s)', 'Error en la posición en el eje x (%)'};
plotErrors(xAxisResultVectors, xAxisErrorLegends, xAxisErrorPlotTitle, ...
xAxisErrorLabels);
%% Plotting y-axis results
subplot(2, 3, 2);
yAxisResultVectors = {t_n, y_ref_n, y_n_best, y_n_mid, y_n_worst};
yAxisResultPlotTitle = 'Posición en el eje y vs. Tiempo.';
yAxisResultLabels = {'Tiempo (s)', 'Posición en el eje y (m)'};
plotAxisComparison(yAxisResultVectors, axisResultLegends, ...
yAxisResultPlotTitle, yAxisResultLabels);
%% Plotting y-axis errors
subplot(2, 3, 5);
yAxisErrorLegends = {'Mejor Resultado', 'Resultado Intermedio', ...
'Peor Resultado'};
yAxisErrorPlotTitle = '% de error en el eje y vs. Tiempo';
```

```
yAxisErrorLabels = {'Tiempo (s)', 'Error en la posición en el eje y (%)'};
plotErrors(yAxisResultVectors, yAxisErrorLegends, yAxisErrorPlotTitle, ...
yAxisErrorLabels);
%% Plotting z-axis results
subplot(2, 3, 3);
zAxisResultVectors = {t_n, z_ref_n, z_n_best, z_n_mid, z_n_worst};
zAxisResultPlotTitle = 'Posición en el eje z vs. Tiempo.';
zAxisResultLabels = {'Tiempo (s)', 'Posición en el eje z (m)'};
plotAxisComparison(zAxisResultVectors, axisResultLegends, ...
zAxisResultPlotTitle, zAxisResultLabels);
%% Plotting z-axis errors
subplot(2, 3, 6);
zAxisErrorLegends = {'Mejor Resultado', 'Resultado Intermedio', ...
'Peor Resultado'};
zAxisErrorPlotTitle = '% de error en el eje z vs. Tiempo';
zAxisErrorLabels = {'Tiempo (s)', 'Error en la posición en el eje z (%)'};
plotErrors(zAxisResultVectors, zAxisErrorLegends, zAxisErrorPlotTitle, ...
zAxisErrorLabels);
%% Plotting trajectory
figure;
subplot(1, 2, 1);
trajectoryVectors = {x_ref_n, y_ref_n, z_ref_n, x_n_best, y_n_best, ...
z_n_best, x_n_mid, y_n_mid, z_n_mid, x_n_worst, y_n_worst, z_n_worst};
trajectoryLabels = {'Posición en el eje x (m)', ...
'Posición en el eje y (m)', 'Posición en el eje z (m)'};
trajectoryTitle = 'Trayectoria Seguida vs. Referencia';
plotTrajectoryComparison(trajectoryVectors, trajectoryLabels, ...
axisResultLegends, trajectoryTitle);
subplot(1, 2, 2);
plotTrajectoryComparison(trajectoryVectors, trajectoryLabels, ...
axisResultLegends, trajectoryTitle);
clc;
```

ANEXO D

FOTOGRAFÍAS DE LA EXPERIMENTACIÓN

En este anexo se muestran fotografías de pruebas en las que la trayectoria o ha sido generada con *MATLAB* o ha sido dibujada por el usuario usando la app. En la Figura D.1 se muestra el espacio físico en el que se han ejecutado las pruebas con el quadrotor.

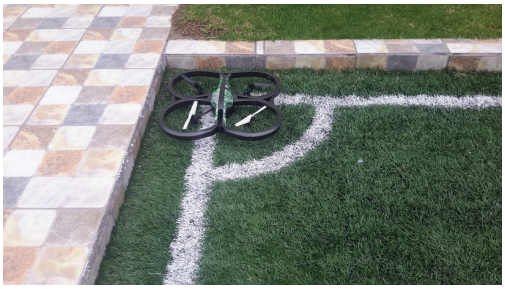
(a)

(b)

Figura D.1: Cancha de fútbol en la que se realizaron las pruebas experimentales.

El punto de partida para el seguimiento de trayectorias que se ha escogido se muestra en la Figura D.2 con el quadrotor con y sin la cubierta para vuelos en interiores.

(a)



(b)

Figura D.2: Punto de partida para el seguimiento de trayectorias.

La primera prueba que se realizó fue el seguimiento de una trayectoria lineal en el primer octante de la cual se muestran algunas imágenes en la Figura D.3.
El segundo tipo de pruebas que se han realizado son trayectorias parabólicas en el primer octante de cuyas pruebas se muestran algunas fotografías en la Figura D.4.

(a)

(b)

Figura D.3: AR.Drone siguiendo una trayectoria lineal definida en el primer octante.

(a)

(b)

Figura D.4: AR.Drone siguiendo una trayectoria parabólica definida en el primer octante.

Las siguientes fotografías (Figura D.5) muestran al AR.Drone 2.0 luego de haber terminado el seguimiento de una trayectoria preestablecida.

(a)

(b)

Figura D.5: AR.Drone luego de haber aterrizado.

La última y tercera clase de pruebas que se han realizado son trayectorias sinusoidales de las cuales se muestran unas fotografías (Figura D.6) con el AR.Drone

2.0 usando la cubierta para interiores.

Figura D.6: AR.Drone 2.0 con la cubierta para vuelos en interiores.

En la ejecución de las pruebas generalmente se ha evitado utilizar la cubierta para interiores dado que los resultados son mucho mejores porque esta cubierta desestabiliza al AR.Drone 2.0 durante el vuelo. Sin embargo, con la cubierta para interiores las hélices del quadrotor están mucho mejor protegidas, y en caso de una colisión, estas casi no sufren daño.