Universidad de la República
Facultad de Ingeniería

uQuad IV: Diseño y construcción de un UAV autónomo utilizando algoritmos de redes neuronales para el control de actitud

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

Nicolás Cáceres, Rodrigo Moreira y Araceli Rodríguez

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

Tutor

Rafael Canetti ..... Universidad de la República

Tribunal

Rafael Canetti ..... Universidad de la República
Sebastián Fernández ..... Universidad de la República
Juan Pablo Oliver ..... Universidad de la República

Montevideo
jueves 31 diciembre, 2020

Esta página ha sido intencionalmente dejada en blanco.

Agradecimientos

Queremos agradecer a nuestro tutor Rafael Canetti por el apoyo y los conocimientos compartidos.

A Carla Ibarguren por la gestión de compras de los componentes, a Martha Delgado por los trámites en Aduana y a Roberto y Sergio del Taller del IIE por la ayuda en el diseño de algunos dispositivos que implementamos.

Por último, pero no menos importante, agradecer a nuestros familiares y amigos por el apoyo brindado a lo largo de la carrera y especialmente durante este proyecto.

Esta página ha sido intencionalmente dejada en blanco.

Resumen

El proyecto tiene por objeto principal el diseño y automatización de un cuadricóptero utilizando técnicas de control basadas en redes neuronales.

Se describen componentes seleccionados para la construcción del drone, así como también la caracterización mecánica y sensorial de los componentes fundamentales.

Luego, se presenta la automatización y el control de vuelo detallando los algoritmos diseñados e implementados junto con los resultados obtenidos en una plataforma de simulación y en el sistema físico.

Por último, las conclusiones del proyecto realizado y propuestas de mejoras a futuro.

Esta página ha sido intencionalmente dejada en blanco.

Tabla de contenidos

Resumen III

I Introducción 1

1. Introducción 3
1.1. Antecedentes . . . 3
1.2. Objetivos . . . 3
1.3. Organización del documento . . . 4

II Solución Adoptada 5

2. Descripción General 7
2.1. Esquema general del control de vuelo de un drone . . . 7
2.2. Solución funcional . . . 8
2.3. Fusión Sensorial . . . 9
2.4. Control de vuelo . . . 9
2.5. Autonomía de vuelo . . . 11
2.6. Sistema simulado . . . 11
2.7. Modificaciones en el marco de la emergencia sanitaria . . . 12

3. Elección de Componentes 13
3.1. Introducción . . . 13
3.2. Componentes básicos . . . 13
3.3. Sensores . . . 16
3.4. Inteligencia . . . 17
3.5. Comunicación . . . 18
3.6. Piezas adicionales . . . 18
3.7. Resumen de componentes . . . 19
3.8. Diagramas de Conexión . . . 19
3.9. Montaje del Drone . . . 20

III Modelo Físico y Caracterización 21

4. Modelo Físico, Dinámica y Cinemática del Sistema 23
- 4.1. Introducción . . . 23
- 4.2. Descripción del Sistema Físico . . . 23
- 4.3. Descripción Cinemática . . . 24
- 4.4. Descripción del Estado . . . 26
- 4.5. Descripción Dinámica . . . 26
  - 4.5.1. Primera Cardinal . . . 26
  - 4.5.2. Segunda Cardinal . . . 27

5. Caracterización Mecánica 29
- 5.1. Introducción . . . 29
- 5.2. Masa del drone . . . 29
- 5.3. Ensayo de Inercia . . . 29
  - 5.3.1. Experimento . . . 29
  - 5.3.2. Resultados y Análisis de datos . . . 31

6. Caracterización de Propulsores 33
- 6.1. Introducción . . . 33
- 6.2. Medida de empuje . . . 33
  - 6.2.1. Experimento . . . 33
  - 6.2.2. Resultados y Análisis de datos . . . 34
- 6.3. Medida de torque . . . 35
  - 6.3.1. Experimento . . . 35
  - 6.3.2. Resultados y análisis de datos . . . 36
- 6.4. Comparación de hélices y consumo . . . 37

IV Estimación del Estado 41

7. Fusión Sensorial 43
- 7.1. Introducción . . . 43
- 7.2. Filtro de Kalman Extendido . . . 43
- 7.3. Uso del filtro en el sistema . . . 45
- 7.4. Filtro complementario . . . 47

8. Caracterización de Sensores 49
- 8.1. Caracterización de sensores . . . 49
  - 8.1.1. IMU . . . 49
  - 8.1.2. GPS . . . 51

V Automatismo y Control con Inteligencia Artificial 53

9. Introducción 55
9.1. Fundamentos de Reinforcement Learning 55
9.1.1. Consideraciones previas 55
9.1.2. Planteo del Problema de Reinforcement Learning 57

10. Controlador de Actitud 59
10.1. Introducción 59
10.2. Controlador de posición angular 60
10.2.1. Controlador de velocidad angular 60
10.2.2. Control posición angular 65
10.3. Controlador de velocidad en el eje z 65
10.3.1. Arquitectura de la red neuronal 65
10.3.2. Entrenamiento 66
10.3.3. Resultados 67
10.4. Throttle Mixing 68

11. Controlador de Posición 71
11.1. Introducción 71
11.2. Generación estado angular 71
11.3. Generación velocidad en z deseada 73

12. Autonomía de Vuelo 75
12.1. Planificación de trayectorias 75
12.2. Seguimiento de trayectorias 77
12.2.1. Seguimiento de línea recta 78
12.2.2. Seguimiento de arco de circunferencia 79
12.3. Evasión de obstáculos 79
12.3.1. Introducción 79
12.3.2. El algoritmo DistBug 79

VI Implementación y Resultados 83

13. Introducción 85
13.1. ROS 85

14. Sistema Simulado 87
14.1. Simulador Gazebo 87
14.2. Modelo del drone 87
14.3. Estructura de nodos en simulación 88
14.4. Control de estabilización en Gazebo 90
14.5. Simulación del sistema completo 90
14.6. Interfaz de Usuario Web 93

15. Sistema Real **95**
15.1. Estructura de nodos en el sistema físico . . . 95
15.2. Ensayo de control de actitud . . . 96
15.3. Estudio analítico del sistema . . . 99

VII Conclusiones **103**

16. Conclusiones **105**
16.1. Conclusiones . . . 105
16.2. Mejoras a futuro . . . 106

VIII Apéndices **107**

A. Piezas adicionales de hardware **109**
A.1. Cúpula . . . 109
A.2. Soportes sensores ultrasonido . . . 110
A.3. Soporte NVidia . . . 110
A.4. Soporte IMU . . . 111

B. Cuaterniones **113**
B.1. Introducción . . . 113
B.2. Álgebra de Cuaterniones . . . 113
B.2.1. Suma y Multiplicación . . . 113
B.2.2. Conjugado, norma e inverso de un cuaternión . . . 114
B.2.3. El cuaternión como operador de rotaciones . . . 114

C. Calibración ESCs **117**
C.1. Configuración . . . 117
C.2. Calibración . . . 117

D. Software **119**
D.1. Paquetes ROS . . . 119
D.1.1. uquad4_communication . . . 119
D.1.2. uquad4_controller . . . 119
D.1.3. uquad4_description . . . 120
D.1.4. uquad4_logs . . . 121
D.1.5. uquad4_neural_network . . . 121
D.1.6. uquad4_path_planning . . . 121
D.1.7. uquad4_position_controller . . . 122
D.1.8. uquad4_sensor_fusion . . . 122
D.1.9. uquad4_obstacle_avoidance . . . 123
D.1.10. uquad4_state_machine . . . 123
D.1.11. uquad4_sensors . . . 124

E. Manual de Usuario **125**
E.1. Instalación del Software 125
E.2. Uso del sistema 125
E.2.1. Inicialización 125
E.2.2. Uso del Web Server 125
E.2.3. Configuración de Logs 127
E.3. Entrenamiento redes 128

Referencias **131**

Índice de tablas **134**

Índice de figuras **136**

Esta página ha sido intencionalmente dejada en blanco.

Parte I

Introducción

Capítulo 1

Introducción

1.1. Antecedentes

En el Instituto de Ingeniería Eléctrica de la Facultad de Ingeniería de la Universidad de la República, existe desde hace ya varios años una línea de investigación asociada a la robótica móvil y al desarrollo de técnicas de control. Dentro de la robótica móvil, se han desarrollado robots terrestres (SULLA y Rover), acuáticos (Pez Robot I y II) y aéreos no tripulados.

Es en esta última área en la que se ha enfocado más con distintas aplicaciones como son Termodron I y II, en donde se diseñó y construyó un drone autónomo con arquitectura de cuadricóptero que releva imágenes termo-gráficas.

Además se crearon los proyectos uQuad! [1], uQuad2 [2] y uQuad3 [3], donde el primero se enfocó en el diseño de control para adaptarlo a una plataforma comercial y el segundo se encargó de capacitar al drone para lograr un vuelo autónomo en cuanto al seguimiento de trayectorias.

El proyecto uQuad3 tuvo como objetivos el diseño y construcción de un drone utilizando no sólo técnicas clásicas de control sino también el uso de algoritmos de redes neuronales.

En el presente proyecto se continúa con la línea de investigación presentada, en particular se plantea como objetivo principal el diseño y construcción de un drone autónomo con arquitectura de cuadricóptero. Para el desarrollo del control del sistema se utilizarán algoritmos de redes neuronales; para la planeación, seguimiento de trayectorias y la evasión de obstáculos se trabajarán sobre mejoras de los algoritmos usados por los proyectos anteriores.

1.2. Objetivos

Se define como objetivo principal del proyecto diseñar, construir y programar un drone capaz de cumplir misiones de forma autónoma. Estas misiones consisten en que el drone despegue, pase por una cantidad finita de puntos en el aire (llamados waypoints) y aterrice en un punto específico, evitando los obstáculos que encuentre en el camino.

Además se busca poner en práctica técnicas de control basadas en redes neuronales, por lo que el control de la orientación y la altura será realizado de esa forma.

Para el cumplimiento del objetivo se plantean las siguientes especificaciones:

- Las misiones a cumplir se cargarán de forma remota.
- Las misiones están constituidas por uno a diez waypoints.
- La autonomía de vuelo debe superar los 10 minutos.

Otro objetivo importante del proyecto es la implementación de un entorno de trabajo virtual, generando una simulación del drone y su entorno para poder realizar pruebas experimentales que avalen el diseño del control y autonomía de vuelo. Con esto también se busca desarrollar un código modular y reutilizable, permitiendo agilizar el desarrollo de futuros proyectos que sigan una línea de trabajo similar.

1.3. Organización del documento

El documento se encuentra divido en seis partes que se describen a continuación:

- Solución Adoptada: presenta la descripción funcional del sistema y las soluciones implementadas. Además se presentan los componentes utilizados.
- Modelo Físico y Caracterización: describe la dinámica del drone junto con las ecuaciones que definen el sistema físico y los experimentos realizados para la caracterización de los componentes mecánicos.
- Estimación del Estado: presenta el algoritmo para implementar la fusión sensorial así como también los experimentos para caracterizar los sensores.
- Automatismo y Control con Inteligencia Artificial: muestra de forma detallada la implementación de los algoritmos utilizados para el control y la autonomía de vuelo.
- Implementación y Resultados: se describen los distintos softwares utilizados, una estructura global de los distintos paquetes implementados, la interfaz de usuario y los resultados de las simulaciones realizadas.
- Conclusiones: conclusiones generales de la tesis y presentación de mejoras a futuro.

Parte II

Solución Adoptada

Capítulo 2

Descripción General

2.1. Esquema general del control de vuelo de un drone

El drone diseñado es un cuadricóptero que consiste en una estructura simétrica en forma de cruz, con 4 propulsores ubicados equidistantes al centro de masa tal como se muestra en la Figura 2.1.

La orientación del drone se describe en términos de los ángulos de Euler, descomponiendo los ángulos en roll (ϕ), pitch (θ) y yaw (ψ).

Cada propulsor i ejerce una fuerza F_i y un torque M_i sobre el sistema en función de su velocidad angular $\dot{\Omega}_i$. Por lo tanto si:

- $\dot{\Omega}_1 + \dot{\Omega}_4 > \dot{\Omega}_2 + \dot{\Omega}_3 \Rightarrow$ Aumenta roll.
- $\dot{\Omega}_3 + \dot{\Omega}_4 > \dot{\Omega}_1 + \dot{\Omega}_2 \Rightarrow$ Aumenta pitch.
- $\dot{\Omega}_1 + \dot{\Omega}_3 > \dot{\Omega}_2 + \dot{\Omega}_4 \Rightarrow$ Aumenta yaw.

El control de vuelo de un drone se logra mediante el control de actitud y altura. El control de actitud consiste en lograr que el drone se oriente a cualquier vector de ángulos de Euler deseado. De forma análoga, el control de altura consiste en lograr

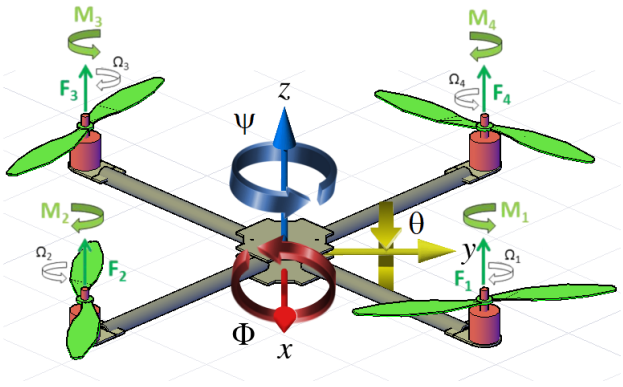


Figura 2.1: Diagrama físico del drone. [3]

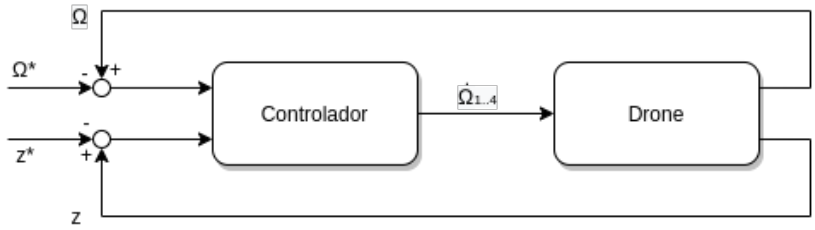


Figura 2.2: Diagrama simplificado de control del sistema. Donde Ω y Ω^* es la orientación del drone y la orientación deseada respectivamente, z y z^* la altura y altura deseada y $\dot{\Omega}_{1.,4}$ las velocidades angulares de los propulsores.

alcanzar cualquier altura deseada. La Figura 2.2 muestra un diagrama simplificado de control del sistema.

Para hacer posible este control, se utilizó una familia de algoritmos dentro de los de inteligencia artificial, llamado Reinforcement Learning, los cuales ajustan los pesos de las capas de una red neuronal, de forma tal que esta minimice una recompensa objetivo. Dada la forma en la que estos algoritmos funcionan, se optó por desarrollar una versión simulada del drone real, en la cual se pueda iterar los algoritmos sin riesgo de que el sistema físico sufra daños.

2.2. Solución funcional

El diagrama de la Figura 2.3 describe en grandes rasgos la solución implementada.

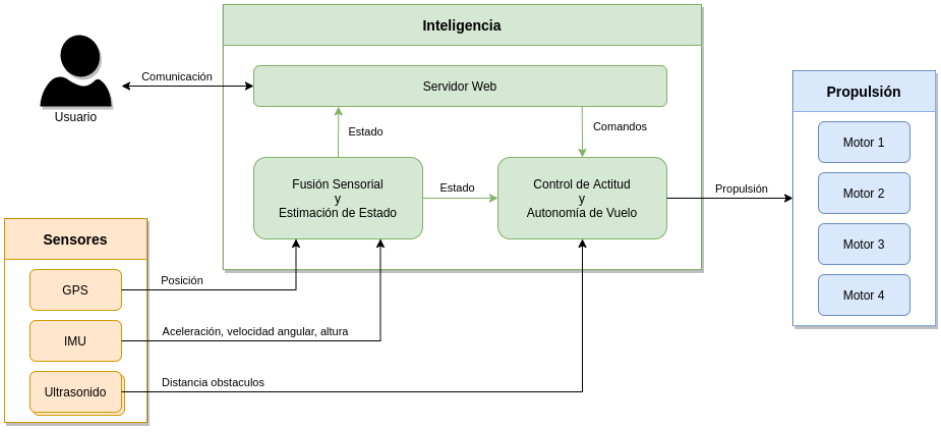


Figura 2.3: Diagrama de la solución funcional del sistema.

El usuario se conecta al drone remotamente por WiFi al servidor web hosteado en el mismo, accediendo a una página web en la cual se puede visualizar el estado del drone y programar las misiones que se desea que se lleven a cabo.

Para obtener el estado, el drone cuenta con un GPS y una IMU (Unidad de Medida Inercial). El GPS devuelve la posición en los ejes x e y. La IMU cuenta

con distintos sensores de los cuales se puede obtener la aceleración, la altura y la velocidad angular del drone. Debido a que sensores como el GPS poseen tiempos de muestreo lentos, para esta aplicación y para mejorar la precisión de la medidas, se realiza una estimación del estado utilizando algoritmos de fusión sensorial que se explicarán más adelante.

El sistema cuenta con distintos modos de vuelo que el usuario puede seleccionar. Estos modos son:

- Despegue: El drone realiza la secuencia de despegue y una vez alcanzada una altura segura, se pasa inmediatamente al modo hovering.
- Hovering: El drone permanece inmóvil en el aire.
- Misión: En este modo se habilita la recepción de la misión que debe cumplir el drone. Una vez el usuario ingresa los waypoints por los cual el drone debe pasar, se planean las trayectorias y el drone comienza el seguimiento hasta llegar al destino donde permanece inmóvil esperando un nuevo comando. Si en el camino se encuentra con un obstáculo detectado a través de los sensores de ultrasonido, este lo evade y continua la misión.
- Aterrizaje: Inicializa la secuencia de aterrizaje seguro.
- Emergencia: Se apagan los motores inmediatamente, sin importar el estado en que se encuentre.

2.3. Fusión Sensorial

Para poder estimar el conjunto de variables de estado del drone en cada paso de tiempo t_k, se realizó la denominada fusión sensorial. La misma se puede ver como un bloque donde se reciben las lecturas de los sensores (IMU, GPS y barómetro), se interpretan, agrupan y filtran obteniendo como resultado el vector de variables de estado estimadas.

Se estima la posición y velocidad utilizando el Filtro de Kalman Extendido [4], una extensión del Filtro de Kalman que aplica a sistemas no lineales. Para la estimación de la orientación se utilizó el Filtro Complementario [5], mientras que la velocidad angular se obtiene directamente del giróscopo de la IMU.

2.4. Control de vuelo

Los drones son sistemas no lineales donde generalmente para resolver el control del mismo, se utilizan controladores del tipo PID, realizando una linealización del sistema en el punto de trabajo. En este proyecto, se optó por trabajar con redes neuronales entrenadas con algoritmos de Reinforcement Learning.

Para resolver el control de vuelo, se divide el control de actitud del drone en dos grandes controladores, uno de posición angular y otro de velocidad en z. El diagrama del control de actitud se muestra en la Figura 2.4. El control se

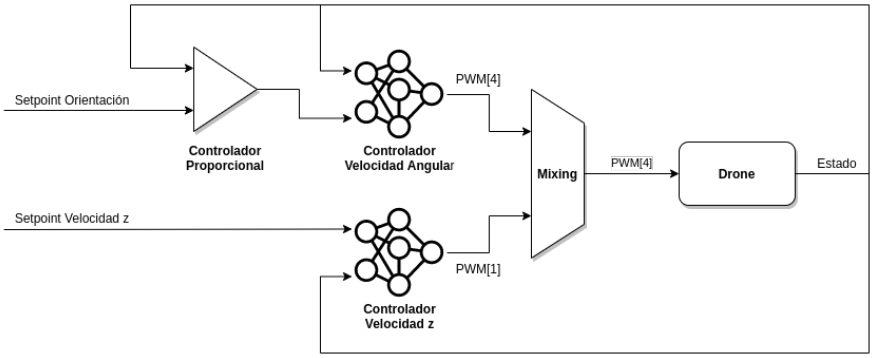


Figura 2.4: Diagrama simplificado de control de actitud.

considera exitoso si es capaz de alcanzar y mantener una orientación y altura deseada partiendo de un estado cualquiera.

El control de la posición angular se separó en dos controladores en serie. El primero busca controlar la velocidad angular del drone a partir del estado, devolviendo a su salida la propulsión necesaria. Se diseñó una arquitectura de una red neuronal multicapa cuya entrada es un vector de dimensión seis donde se contemplan los errores al setpoint deseado en las velocidades de roll, pitch y yaw y las diferencias de estos errores entre un tiempo t y $t-1$. Luego, se tienen dos capas ocultas de 8 neuronas y un vector de salida de dimensión cuatro que representa los valores de PWM a enviar a cada ESC. La recompensa utilizada para el entrenamiento de la red se descompone en tres factores, buscando minimizar el error a los setpoints, el consumo y las oscilaciones de los motores.

El segundo controlador es un bloque proporcional que tiene como entrada el error al setpoint de orientación y su salida es la velocidad angular deseada, de forma tal que al concatenar ambos controladores, se obtiene un controlador de posición angular.

Para la altitud del drone, se decidió realizar un controlador que dado el estado del drone determine los valores de PWM que se deben enviar a las ESCs para que se alcance la velocidad en z deseada. En este caso se armó una red con un vector de dimensión cuatro a la entrada, dos capas ocultas de 8 neuronas y una única salida. La entrada corresponde a los valores de roll, pitch, el error al setpoint deseado y la diferencia del error al setpoint en t y $t-1$, mientras que la salida toma valores entre 0 y 1, donde 0 significa que todos los motores están apagados y 1 girando a su máxima velocidad.

Por último, se combinan las salidas de los controladores de altitud y posición angular obteniendo como una única señal de PWM por cada motor, en base a las salidas de cada controlador. La Figura 2.4 muestra la conexión entren los distintos bloques y controladores.

Los controladores se ejecutan en un Nvidia Jetson Nano a bordo del drone, a una frecuencia de muestro de 175 Hz. Se utiliza esta frecuencia debido a que es la frecuencia máxima promedio a la que se es capaz de ejecutar la red.

En los Capítulos 10 y 11 se describen en detalle las soluciones adoptadas y los

resultados obtenidos.

2.5. Autonomía de vuelo

Se dividió el tema en tres partes: la planeación de trayectorias, el seguimiento de las mismas y la evasión de obstáculos. Para cada uno de ellos se utilizó un algoritmo diferente, que fusionados forman la autonomía de vuelo del drone.

La planeación de trayectorias se basó en la teoría de Dubins [6]. Allí se explica como dados dos puntos en un sistema de referencia inercial (x_1, y_1), (x_2, y_2) con una orientación θ_1 y θ_2 para cada punto y dado un radio mínimo de curvatura r, existe una curva continuamente diferenciable cuya longitud es mínima que une los dos puntos y es de la forma CSC o CCC, donde S representa una recta y C un arco de circunferencia de radio fijo r.

De esta forma se tienen seis posibles curvas que minimizan una trayectoria: RSR, LSL, LSR, RSL, LRL y RLR donde R es un arco de circunferencia recorrido en sentido horario y L un arco recorrido en sentido antihorario.

Luego, Shkel y Lumelsky [7] logran determinar qué curva de Dubins es la que corresponde con un par de waypoints de forma que la trayectoria sea mínima.

El seguimiento de las trayectorias se implementó con el algoritmo "Carrot Chasing" de Sousa y Saripali [8], donde a partir de la trayectoria, la ubicación del drone y el yaw, se busca tener un valor de yaw deseado para poder seguir la trayectoria satisfactoriamente. Este algoritmo se extendió a tres dimensiones de forma de poder también variar la altura de los waypoints y que el drone no se desplace siempre a un mismo valor de z.

Por último, la evasión de obstáculos se basó en el algoritmo Distbug, donde si se detecta la presencia de un obstáculo, el drone comenzará a bordearlo hasta que se detecte el objetivo o que se vuelva a encontrar con la trayectoria que se estaba siguiendo.

La implementación y profundización de estos temas se detallan en el Capítulo 12.

2.6. Sistema simulado

Se utiliza la plataforma Gazebo para simular la dinámica y cinemática del drone, teniendo en cuenta la masa e inercia del sistema junto con la propulsión y el torque generado por los motores. Se realizaron ensayos de laboratorio con el fin de determinar todas las características mencionadas los cuales se detallan en la Parte III.

También se pueden simular colisiones y los sensores utilizados (IMU, GPS y ultrasonido) permitiendo entrenar las redes neuronales y testear los algoritmos en un entorno seguro.

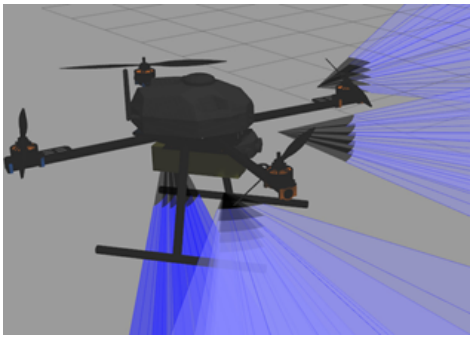


Figura 2.5: Modelo del drone simulado.

2.7. Modificaciones en el marco de la emergencia sanitaria

El 13 de marzo del 2020, en el país se declaró la emergencia sanitaria debido al COVID-19. En este contexto, la Universidad cerró sus puertas a alumnos y personal no esencial, debiendo ser suspendido el trabajo experimental en el laboratorio, de forma repentina sin una fecha de retorno estimada.

En ese momento nos encontrábamos comenzando la etapa de pruebas relacionadas con el control del sistema físico, en el laboratorio de la Facultad. Ante la incertidumbre, teniendo en cuenta las medidas tomadas en la región, y la cancelación de las actividades presenciales, se replanificó el proyecto dándole prioridad al trabajo simulado, suspendiendo todo el trabajo relacionado con actividades experimentales, en particular cancelando las pruebas de vuelo.

Meses después, cuando la Universidad reabrió sus puertas bajo ciertos protocolos sanitarios, se realizaron pruebas de control de actitud en instalaciones del Laboratorio, presentando los resultados de los mismos en el Capítulo 15. Una vez llegado a resultados satisfactorios, se dio por finalizado el proyecto.

Capítulo 3

Elección de Componentes

3.1. Introducción

Para la construcción del drone, se realizó una búsqueda de los componentes capaces de satisfacer los requisitos para poder cumplir con los objetivos del proyecto. Se presentan los componentes seleccionados para la construcción del drone.

3.2. Componentes básicos

Frame

La estructura, más comúnmente llamada frame, es la base principal del drone y donde se colocan todos los componentes que utilizará como los motores, procesadores, sensores, hélices, etc. En este caso se eligió:

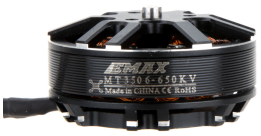
- Modelo: Tarot Iron Man 650
- Material: Fibra de carbono
- Tamaño: 650 mm
- Peso: 0,476 kg

Motores

Los motores sin escobillas o brushless motors, suelen ser utilizados en drones debido a su rendimiento, durabilidad y bajo peso. Además es capaz de controlar las revoluciones con precisión ya que se compone de un rotor de imán permanente y tres pares de bobinas en el estator controladas por una ESC.

The voltage (V)	Propeller size	current (A)	thrust (G)	power (W)	efficiency (G/W)	speed (RPM)
14.8	1340 Carbon fiber Propeller	1	170	14.8	11.5	2700
		2	310	29.6	10.5	3540
		3	420	44.4	9.5	4030
		4	510	59.2	8.6	4360
		5	600	74	8.1	4730
		6	660	88.8	7.4	4980
		7	760	103.6	7.3	5210
		8	830	118.4	7	5480
		9	890	133.2	6.7	5620
		10	950	148	6.41	5790
		11	1020	162.8	6.2	5940
		12	1090	177.6	6.1	6100

Figura 3.1: Especificaciones motores EMAX MT3506.



Para la selección de motores se tomó como criterio de diseño, que el empuje total proporcionado por los cuatro motores sea cercano al doble del peso del drone, ya que de esta manera la aceleración máxima vertical vale g tanto para ascender como para descender (con motores apagados).

De la elección de los demás componentes, se determinó que el peso aproximado del drone sería de 2 kg. Luego de analizar distintas opciones de motores en el mercado, se optó por utilizar el EMAX MT3506 por su peso, consumo y precio.

- Modelo: EMAX MT3506
- KV: 650 rpm/V
- Corriente máxima: 14 A
- Diámetro: 41,5 mm
- Alto: 24,2 mm
- Peso: 67 g

El fabricante brinda especificaciones de ensayos realizados con el motor para distintos tamaños de hélices utilizando baterías de tres y cuatro celdas. De acuerdo a la tabla proporcionada para el motor seleccionado se obtiene que para cumplir el requisito propuesto, junto a este motor se debían utilizar hélices de 13 pulgadas de diámetro y 4 pulgadas de paso, junto a una batería de 4 celdas (Figura 3.1).

Hélices

- Modelo: Quanum carbon fiber propeller
- Material: Fibra de carbono
- Tamaño: 13*4.0 in
- Peso: 7 g

ESC

Los ESCs, del inglés Electronic Speed Controller, son controladores de velocidad electrónicos. De acuerdo a los motores y hélices elegidas, se seleccionó:

- Modelo: Makefire 35 A BLHeli-32
- Corriente: 35 A
- Peso: 32 g

El Anexo C muestra la configuración elegida para las ESCs y cómo calibrarlas.

Batería

Uno de los requerimientos del drone es que este pueda volar por más de 10 minutos, intentando superar los 15 minutos. La primera limitación en la elección de la batería está impuesta por los motores seleccionados, la misma debe ser de 4 celdas.

Para el cálculo de autonomía se tomó como hipótesis que el consumo de los motores es mucho mayor al de los demás componentes juntos, dado que para mantener el drone suspendido en el aire, los motores consumirán 16 A a 14.8 V (237 W) mientras que el NVidia Jetson Nano consume un máximo de 6 A a 5 V (30 W) en su modo de mayor consumo. Además se supuso que en una misión, los motores estarán el 80 % del tiempo al 50 % de su empuje y el resto al 100 %, consumiendo 4 A y 12 A respectivamente. De esta forma se obtiene la siguiente ecuación:

$$T = \frac{C \times 60min}{4(4A \times 0,8 + 12A \times 0,2)} \tag{3.1}$$

donde T es el tiempo de vuelo en minutos y C la capacidad de la batería en Ah.

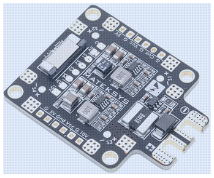
De acuerdo a las baterías disponibles en el mercado, se optó por utilizar la batería ZIPPY Compact de 6200 mAh 40 C, consiguiendo una autonomía de 16,6 minutos.

- Modelo: Zippy Compact 6200 mAh 4S 40C

- Capacidad: 6200 mAh
- Descarga: 40 C
- Peso: 589 g

Distribuidor

Distribuidor de energía donde se conectarán los ESCs.



- Modelo: Matek FCHUB-6S
- Corriente máxima: 184 A

3.3. Sensores

IMU

La Unidad de Medida Inercial (IMU: Inertial Measurement Unit) es un dispositivo formado por varios sensores que indica el estado del drone. En este caso se eligió una IMU de 10 grados de libertad que incluye acelerómetro, barómetro, giróscopo, magnetómetro y termómetro.

- Modelo: Berry IMULSM9DS0 10DOF
- IMU: LSM9DS1 (acelerómetro, magnetómetro y giróscopo)
- Barómetro: BMP280 (con sensor de temperatura)
- Protocolo de comunicación: I2C

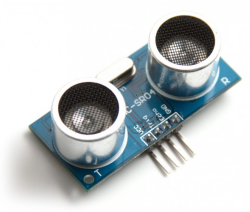
GPS

El GPS, en español Sistema de Posicionamiento Global, permite determinar la ubicación de un objeto determinado mediante satélites. El GPS utilizado es el siguiente:

- Modelo: Ublox NEO-M8N GPS
- Protocolo de comunicación: GNSS
- Peso: 40 g

Sensor ultrasonido

El sensor de ultrasonido permite calcular la distancia a determinado objeto. Se decidió utilizar dos sensores diferentes, el primero estará ubicado en la parte inferior del drone para poder realizar aterrizajes de forma segura, mientras que el segundo será para la parte frontal y laterales ya que posee un mayor alcance y precisión para la evasión de obstáculos.



- Modelo: HC-SR04
- Frecuencia: 10 Hz
- Rango de medición: 2 cm - 500 cm
- Ángulo de medición: 15°

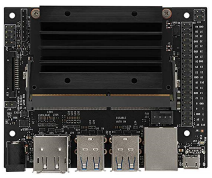
- Modelo: MB1010
- Frecuencia: 20 Hz
- Rango de medición: 30 cm - 650 cm

Cámara

- Modelo: RPi Camera Module V2
- Sensor: Sony Exmor IMX219
- Peso: 10 g

3.4. Inteligencia

NVidia Jetson Nano



El NVidia Jetson Nano es un procesador pensado para ser utilizado en robots y dispositivos con inteligencia artificial. Cuenta con una GPU de NVidia compatible con CUDA y ejecuta Ubuntu 18.04 como sistema operativo. Este será el procesador principal del drone y se encargará de procesar las redes neuronales, así como de realizar la planeación de trayectorias, seguimiento y evasión de obstáculos.

- Modelo: NVIDIA Jetson Nano Development Kit
- Procesador: Quad-core ARM Cortex-A57 MPCore
- GPU: 128-núcleos Maxwell GPU
- Memoria RAM: 4 GB
- Peso: 130 g

Teensy 3.5

El Teensy es un microcontrolador programable mediante USB, compatible con la plataforma Arduino y sus librerías. Será el procesador secundario y se encargará del accionar de los motores.

- Modelo: Teensy 3.5
- Procesador: ARM Cortex M-4 120 MHz
- Memoria RAM: 256 kB
- Peso: 20 g

3.5. Comunicación

Módulo de radiofrecuencia

La comunicación entre el usuario y el drone será mediante radiofrecuencia ya que se tiene mayor rango de comunicación frente a WiFi. El módulo elegido es el siguiente:

- Modelo: NRF24L01+ Wireless Module
- Rango: 800 m
- Peso: 15 g

3.6. Piezas adicionales

Con el fin de poder ensamblar el drone, y dado que el espacio que tiene el frame es reducido para la cantidad de componentes a colocar dentro, se realizaron una serie de impresiones 3D que sirvieron de soporte para los diversos sensores que se utilizaron. También se imprimió una cúpula para proteger el interior del drone.

Por otro lado, se realizó un PCB para conectar el Teensy con las ESCs con el objetivo de que las conexiones sean confiables y que quede prolijo.

En el Anexo A se adjuntan los distintos soportes diseñados.

3.7. Resumen de componentes

Se presenta la Tabla 5.1 con todos los componentes seleccionados a modo de resumen y mejor visualización.

Tabla 3.1: Listado de componentes utilizados.

Item	**Marca / Modelo**
Frame	Tarot Iron Man 650
Motor	EMAX MT3506
Hélice	Quanum 13x40
Batería	Zippy Compact 6200 mAh
Distribuidor	Matek FCHUB-6S
IMU	Berry Imu v2
Ultrasonido	HC-SR04
Sonar	MB1010
Microcontrolador	Teensy 3.5
Procesador	NVidia Jetson Nano
Cámara	Rasperry Module v2
Radiofrecuencia	NRF24L01+ Wireless

3.8. Diagramas de Conexión

En las Figuras 3.2 y 3.3 se muestran los diagramas de conexión de los componentes al NVidia y Teensy respectivamente.

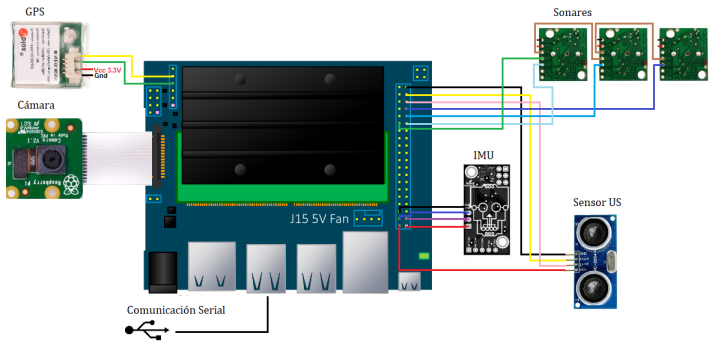


Figura 3.2: Diagrama de conexionado - NVidia Jeston Nano.

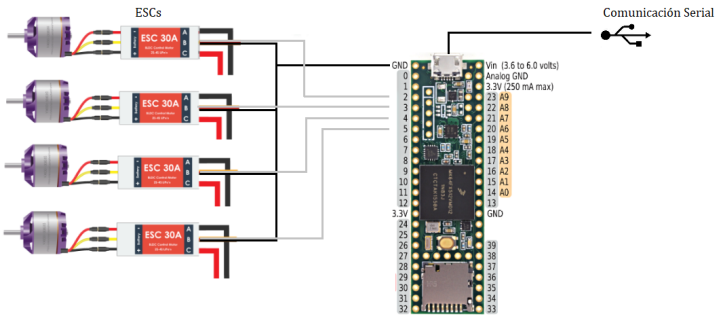


Figura 3.3: Diagrama de conexionado - Teensy 3.5.

3.9. Montaje del Drone

En las Figuras 3.4 y 3.5 se muestran fotografías del drone armado, utilizando los componentes mencionados anteriormente.

Figura 3.4: Fotografías del drone armado. Vista Frontal.

(a) Vista Lateral. (b) Vista Superior.

Figura 3.5: Fotografías del drone armado.

Parte III

Modelo Físico y Caracterización

Capítulo 4

Modelo Físico, Dinámica y Cinemática del Sistema

4.1. Introducción

Para poder abordar el problema del control de actitud y altura, es necesario conocer las relaciones entre las distintas variables dinámicas y cinemáticas. Luego de analizadas estas relaciones, se procederá a estudiar la primera y segunda cardinal, para deducir el efecto de las fuerzas y torques que cada motor produce sobre las diferentes variables de estado del drone.

4.2. Descripción del Sistema Físico

Para describir el sistema físico se sigue la línea teórica y de razonamiento ya utilizada por los proyectos uQuad2 [2] y uQuad3 [3]. Un drone cuadricóptero es un vehículo aéreo no tripulado (de sus siglas en inglés UAV: unmanned aerial vehicle) compuesto por una estructura y cuatro propulsores dispuestos en forma de cruz equidistante donde el centro de la cruz coincide con el centro de masa del drone.

En la Figura 4.1 se pueden observar los ejes cartesianos solidarios al vehículo $\{x, y, z\}$ y los ángulos de Euler Φ (roll), Θ (pitch) Ψ (yaw).

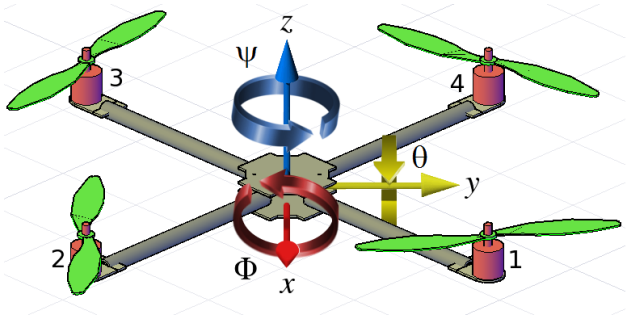


Figura 4.1: Diagrama físico del drone. [3]

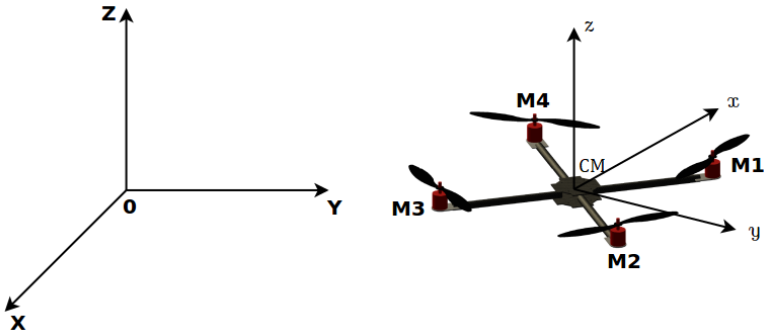


Figura 4.2: Sistemas de referencia.[3]

Cada uno de los sistemas de propulsión (motor y hélice en conjunto) pueden ejercer una fuerza denominada empuje y un torque sobre el robot. Los cuatro propulsores actúan como las variables de control del sistema físico de seis grados de libertad.

4.3. Descripción Cinemática

A lo largo del análisis se trabajará con dos sistemas de referencia cartesianos: uno fijo a la tierra llamado absoluto S_A $\{X, Y, Z, O\}$, el cual vamos a suponer que es inercial dadas las condiciones del problema. El otro, ya visto en la Figura 4.1, solidario al drone S_B, con origen en el centro de masa, y el eje x apuntando al frente tal como indica la Figura 4.2, $S_B = \{x, y, z, CM\}$.

Los ángulos de Euler determinan la orientación del vehículo y son los ángulos de las rotaciones que, aplicadas en forma sucesiva, permiten transformar los ejes absolutos en ejes relativos.

En este trabajo se adoptó la secuencia: primero se gira en torno al eje Z un ángulo ψ, luego se gira un ángulo θ respecto al eje intermedio y' y finalmente un ángulo ϕ en torno al eje x. En esta convención la matriz de rotación de cambio de coordenadas del sistema inercial (S_A) al solidario al cuadricóptero (S_B) es:

$$R_{A\mapsto B} = \begin{pmatrix} \cos\theta\cos\psi & \operatorname{sen}\psi\cos\theta & -\operatorname{sen}\theta \\ \cos\psi\operatorname{sen}\theta\operatorname{sen}\phi - \operatorname{sen}\psi\cos\phi & \operatorname{sen}\psi\operatorname{sen}\theta\operatorname{sen}\phi + \cos\psi\cos\phi & \cos\theta\operatorname{sen}\phi \\ \cos\psi\operatorname{sen}\theta\cos\phi - \operatorname{sen}\psi\operatorname{sen}\phi & \operatorname{sen}\psi\operatorname{sen}\theta\cos\phi - \cos\psi\operatorname{sen}\phi & \cos\theta\cos\phi \end{pmatrix}. \tag{4.1}$$

El problema de trabajar con ángulos de Euler, es que aparece el fenómeno llamado bloque de cardán (gimbal lock) debido a que no existe una descripción única para cada variable de posición. Además debido a su topología las ecuaciones del sistema suelen ser más complejas, lo que aumenta el costo computacional.

Es por esta razón que en este trabajo se utilizan algoritmos de estimación de la orientación basados en cuaterniones. Los mismos son presentados en el Anexo B.

Un cuaternion q se asocia a una rotación que transforma dos sistemas de coordenadas por su eje de giro $\hat{n} = (n_1, n_2, n_3)$ y un ángulo α de la siguiente forma:

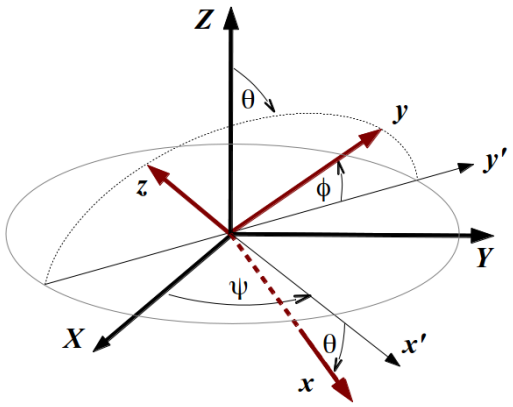


Figura 4.3: Cambio de coordenadas mediante rotaciones en los ángulos de Euler.

$$\begin{cases} lq_0 = \cos(\frac{\alpha}{2}) \\ q_1 = n_1 \operatorname{sen}(\frac{\alpha}{2}) \\ q_2 = n_2 \operatorname{sen}(\frac{\alpha}{2}) \\ q_3 = n_3 \operatorname{sen}(\frac{\alpha}{2}) \end{cases} . \tag{4.2}$$

Si la rotación lleva el sistema absoluto S_A a coincidir con el solidario al cuadricóptero S_B, el cuaternion asociado se denotará q_A^B y como se demostró en el Anexo B, se puede cambiar de sistema de coordenadas a cualquier vector $\vec{v}$ con el producto de cuaterniones $\bigotimes$:

$$\vec{v}_B = q^*_{A\mapsto B} \bigotimes \vec{v}_A \bigotimes q_{A\mapsto B}. \tag{4.3}$$

La matriz asociada a este cambio de coordenadas $R_{A\mapsto B}$ parametrizada con los componentes del cuaternión $q_{A\mapsto B}$ se escribe:

$$R_{A\mapsto B} = \begin{pmatrix} 2(q_0^2+q_1^2)-1 & 2(q_1q_2+q_0q_3) & 2(q_3q_1-q_0q_2) \\ 2(q_1q_2-q_0q_3) & 2(q_0^2+q_2^2)-1 & 2(q_3q_2+q_0q_1) \\ 2(q_3q_1+q_0q_2) & 2(q_3q_2-q_0q_1) & 2(q_0^2+q_3^2)-1 \end{pmatrix}. \tag{4.4}$$

Además si se parametriza la misma rotación mediante ángulos de Euler, la relación de estos entre el cuaternión $q_{A\mapsto B}$ (siempre que sea normalizado) es:

$$\begin{cases} \psi = \arctan^2(2(q_1q_2+q_0q_3), q_0^2+q_1^2-q_2^2-q_3^2) \\ \theta = \operatorname{arc\,sen}(2(q_0q_2-q_1q_3)) \\ \phi = \arctan^2(2(q_2q_3+q_0q_1), q_3^2-q_2^2-q_1^2+q_0^2) \end{cases} . \tag{4.5}$$

Finalmente la relación de las derivadas con los parámetros de velocidad angular para ángulos de Euler es:

$$\begin{cases} \dot{\psi} = w_2 \frac{\sin\phi}{\cos\theta} + w_3 \frac{\cos\phi}{\cos\theta} \\ \dot{\theta} = w_2 \cos\phi - w_3 \sin\phi \\ \dot{\phi} = w_1 + w_2 \frac{\sin^2\phi}{\cos\theta} + w_3 \frac{\sin\phi\cos\phi}{\cos\theta} \end{cases} . \tag{4.6}$$

La relación de derivadas con cuaterniones es:

$$\dot{q} = \frac{1}{2} q \bigoplus \vec{w} \tag{4.7}$$

con $\vec{w}$ la velocidad angular del cuadricóptero en el sistema no inercial.

4.4. Descripción del Estado

Para describir el estado del drone como sistema físico vamos a separar las variables en traslacionales y angulares. El estado traslacional especifica su posición y velocidad en el sistema absoluto S_A:

$$(X, Y, Z, v_X, v_Y, v_Z). \tag{4.8}$$

Para el estado angular vamos a tomar en cuenta la orientación y la velocidad con que cambia la misma. Para eso tomamos los ángulos de Euler como variables estado, junto con el vector de velocidad angular solidario al drone. Las relaciones entre $\vec{w}$ y las derivadas de los ángulos de Euler ya fueron determinadas.

$$(\psi, \theta, \phi, w_1, w_2, w_3) \tag{4.9}$$

Este último, parametrizado en cuaterniones sería:

$$(q, \dot{q}) \tag{4.10}$$

4.5. Descripción Dinámica

El problema de la dinámica del sistema se abordará por el lado de la mecánica Newtoniana, separando el problema en una primera cardinal para las aceleraciones lineales, y una segunda cardinal para el momento angular. En la Figura 4.4 se ve el diagrama de cuerpo libre del drone con todas las fuerzas involucradas. La fuerza de empuje de cada motor está representada como F_i, los torques por M_i y la velocidad de cada motor con ω_i.

4.5.1. Primera Cardinal

La primera cardinal va a describir la traslación del centro de masa, para esto se analiza la segunda ley de Newton que relaciona las fuerzas con la aceleración del centro de masa:

$$m\vec{a_{CM}} = \vec{F_1} + \vec{F_2} + \vec{F_3} + \vec{F_4} + m\vec{g}. \tag{4.11}$$

El vector de fuerzas de empuje de cada motor tiene la dirección del eje z del sistema S_B solidario al drone. El peso opuesto a Z del sistema absoluto S_A.

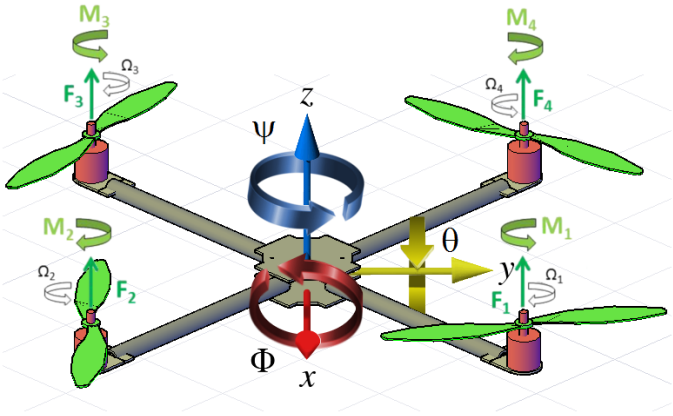


Figura 4.4: Diagrama de cuerpo libre. [3]

4.5.2. Segunda Cardinal

Se plantea el momento angular en el centro de masa CM del vehículo y se tiene:

$$\vec{L_{CM}} = \vec{L_{CM}^{q}} + \vec{L_{CM}^{r}} \tag{4.12}$$

con $\vec{L_{CM}^{q}}$ el momento angular del drone, y $\vec{L_{CM}^{q}}$ el momento angular de los propulsores.

Se procede a calcular cada uno de los momentos expresados en el sistema solidario drone.

- **Momento angular del drone:**

$$\vec{L^{q}} = I_{CM}^{q}\vec{\omega} = \begin{pmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{pmatrix} \begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix} \tag{4.13}$$

  con (I_x, I_y, I_z) los momentos de inercia del robot respecto a los ejes solidarios al mismo, y $(\omega_1, \omega_2, \omega_3)$ las componentes de la velocidad angular del cuadricóptero en el sistema no inercial.

- **Momento angular de los motores:** Serán la suma de cada uno, se desarrolla para el primer motor y los demás serán análogos:

$$\vec{L_1^{r}} = I_{B1}^{r}(\Omega_1\vec{z} + \vec{\omega}) + m_1(\vec{CM - B_1}) \times \vec{v_{B1}} \tag{4.14}$$

  donde I_{B1}^{r} el momento de inercia del drone con centro en la base del motor 1, Ω_1 la velocidad angular del motor y $\vec{v_{B1}}$ la velocidad de la base del motor.

  Se tiene en cuenta que $\Omega_1 \gg \mid \vec{\omega} \mid$ y que el segundo sumando es también despreciable frente a Ω_1, por lo que se llega a la siguiente expresión para la suma de momentos angulares de los motores:

$$\vec{L^{r}} \cong I_r(-\Omega_1 + \Omega_2 - \Omega_3 + \Omega_4)\vec{z} = I_r\Omega_{res}\vec{z}. \tag{4.15}$$

Sumando los dos momentos se obtiene:

$$\vec{L^q} + \vec{L^r} \cong I^q_{CM}\vec{\omega} + I_r(-\Omega_1 + \Omega_2 - \Omega_3 + \Omega_4)\vec{z}. \tag{4.16}$$

Se plantea la segunda cardinal derivando el momento angular total, despreciando la derivada de $\vec{L^r}$ en el sistema no inercial:

$$\dot{\vec{L}} = \dot{\vec{L^q}} + \dot{\vec{L^r}} \cong I^q_{CM}\dot{\vec{\omega}} + \vec{\omega} \times I_{CM}\vec{\omega} + \vec{\omega} \times \vec{L^r} = \sum M_{CM} \tag{4.17}$$

con $\sum M_{CM}$ la suma de todos los torques con centro en el centro de masas. Se escribe en función de las componentes de $\vec{\omega}$ en el sistema solidario al drone:

$$\dot{\vec{L}} = \begin{cases} I_x\dot{\omega_1} + (I_z - I_y)\omega_2\omega_3 + I_r\Omega_{res}\omega_2 \\ I_y\dot{\omega_2} + (I_x - I_z)\omega_1\omega_3 - I_r\Omega_{res}\omega_1 \\ I_x\dot{\omega_3} + (I_y - I_x)\omega_1\omega_2 \end{cases} . \tag{4.18}$$

Por otro lado, el momento total está compuesto por los torques propios de cada motor M_i y los generados por los empujes multiplicados vectorialmente por el brazo del drone $F_i \times l$ (desde la base del motor al centro de masas) y valen:

$$\begin{cases} M_x = l(F_1 - F_2 - F_3 + F_4) \\ M_y = l(-F_1 - F_2 + F_3 + F_4) \\ M_z = M_1 - M_2 + M_3 - M_4 \end{cases} , \tag{4.19}$$

de esta forma las ecuaciones para la segunda cardinal resultan:

$$\begin{cases} I_x\dot{\omega_1} = (I_y - I_z)\omega_2\omega_3 - I_r\Omega_{res}\omega_2 + M_x \\ I_y\dot{\omega_2} = (I_z - I_x)\omega_1\omega_3 + I_r\Omega_{res}\omega_1 + M_y \\ I_x\dot{\omega_3} = (I_x - I_y)\omega_1\omega_2 + M_z \end{cases} . \tag{4.20}$$

En los siguientes capítulos se ven los experimentos que caracterizan al drone. Entre estos están la determinación de los momentos de inercia (I_x, I_y, I_z) y la función que relaciona la fuerza de empuje y el torque producido por cada motor.

Capítulo 5

Caracterización Mecánica

5.1. Introducción

Para poder modelar las características mecánicas del drone, es decir la masa y los momentos de inercia, se realizaron una serie de ensayos experimentales.

5.2. Masa del drone

Una vez armado el drone se midió la masa con una balanza de 1 gramo de apreciación, obteniendo $m = 2,141$ kg.

5.3. Ensayo de Inercia

Para la realización de este experimento, se utilizó el modelo del péndulo físico para pequeñas oscilaciones.

5.3.1. Experimento

Elementos utilizados

- Drone armado
- Cronómetro
- Tanza

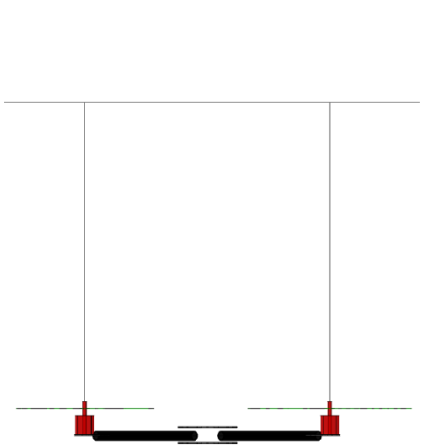
Descripción

Se cuelga el drone mediante tanza a un eje cilíndrico con el fin de que pueda oscilar sin rozamiento. Para relevar el período de oscilación, se pone a oscilar el drone y se comienza a tomar el tiempo hasta que se cumplen 15 oscilaciones.

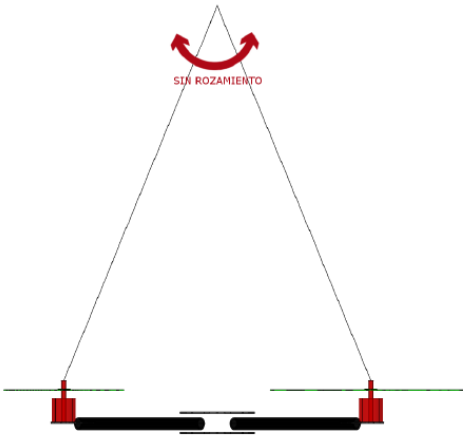
Se repite el experimento 5 veces. Luego se calcula el período de oscilación promediando las medidas tomadas.

Se realiza el experimento para las tres posiciones del drone correspondientes para relevar la inercia según los ejes (x, y, z).

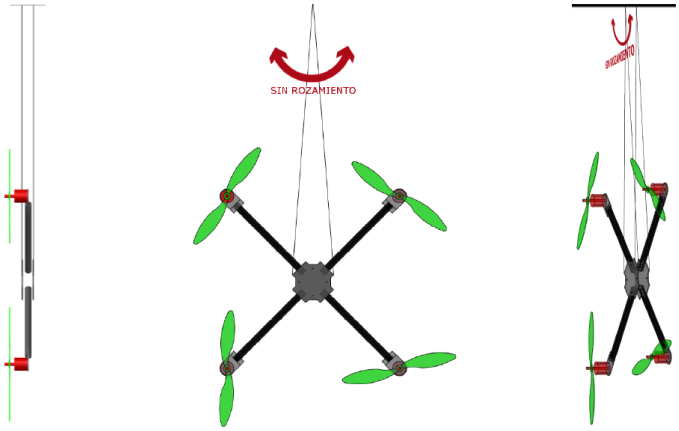
Esquemas de conexión



(a) Momento de Inercia I_x e I_y. Vista Frontal.



(b) Momento de Inercia I_x e I_y. Vista Lateral.



(c) Momento de Inercia I_z.

Figura 5.1: Esquema de conexión.[3]

5.3.2. Resultados y Análisis de datos

Tabla 5.1: Datos para el período en cada uno de los ejes.

	I_x		I_y		I_z	
Prueba	t_i (s)	T_i (s)	t_i (s)	T_i (s)	t_i (s)	T_i (s)
1	21,04	1,402	21,07	1,404	22,29	1,486
2	21,07	1,405	21,06	1,404	22,31	1,487
3	20,99	1,399	21,12	1,408	22,72	1,515
4	20,98	1,398	21,14	1,409	22,49	1,499
5	21,14	1,409	21,05	1,403	22,56	1,504

Para obtener el período de oscilación para cada eje, primero se calculó T_i de la forma:

$$T_i = \frac{t_i}{N} \tag{5.1}$$

donde N es la cantidad de oscilaciones, en este caso 15.

Luego promediando los T_i en cada caso se obtuvo que:

$$\boxed{T_x = 1,403 \text{ s}} \quad \boxed{T_y = 1,406 \text{ s}} \quad \boxed{T_z = 1,498 \text{ s}}$$

Para poder calcular la inercia del drone, además de medir el período de oscilación, se midió la distancia desde el eje cilíndrico al centro de masa.

Para eso primero se halló el centro de masa usando la técnica de intersección de verticales como se muestra en la Figura 5.2. Sabiendo que el drone es simétrico, se halla la distancia en z entre el centro de masa y la estructura del drone.

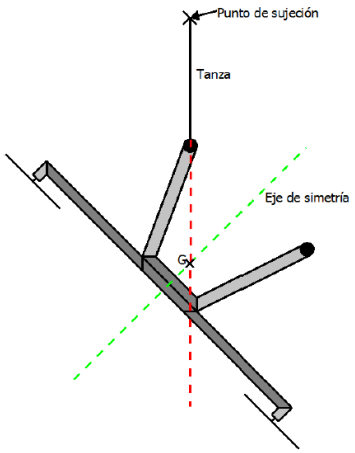


Figura 5.2: Técnica de intersección de verticales.[3]

Por último usando la ecuación del péndulo físico en pequeñas oscilaciones y el Teorema de Steiner tenemos que:

$$T = 2\pi\sqrt{\frac{I_0}{mgl}} I_0 = I_G + ml^2 \tag{5.2}$$

donde l es la distancia al centro de masa, e I_0 la inercia respecto al eje de oscilación.

Finalmente se tiene que:

$$\boxed{I_{Gx} = 0,041 \text{ kgm}^2} \quad \boxed{I_{Gy} = 0,043 \text{ kgm}^2} \quad \boxed{I_{Gz} = 0,075 \text{ kgm}^2}$$

Capítulo 6

Caracterización de Propulsores

6.1. Introducción

El presente capítulo detalla los experimentos realizados para la caracterización de los propulsores. Un propulsor se compone del motor, el controlador ESC y la hélice. Se relevaron medidas de empuje y torque para dos tipos de hélice distinta con el fin de determinar cuál de ellas era la más adecuada a utilizar en función del consumo y condiciones de vuelo.

6.2. Medida de empuje

6.2.1. Experimento

Elementos utilizados

- Motor EMAX MT3506 650 kV
- Hélice fibra de carbono 12x55 y 13x40
- ESC 35 A Makerfire
- Batería ZIPPY Compact 6200 mAh
- Generador de señales SIGLENT SDG805
- Fuente SIGLENT para 5 V
- Osciloscopio SIGLENT SDS1102CML+
- Conjunto fotodiodo (LED IR TSAL6200 y Detector IR TSOP38256)
- Balanza electrónica CAMRY EK9320

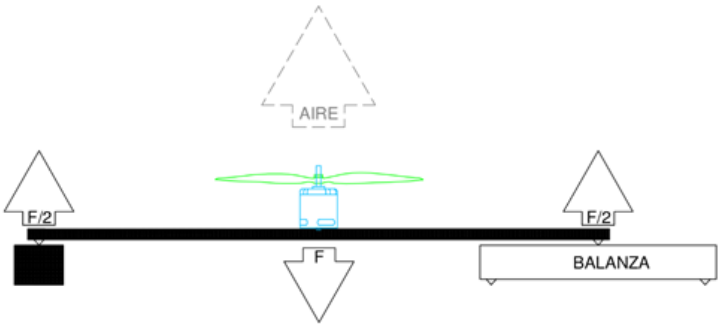


Figura 6.1: Diagrama de armado mecánico.[3]

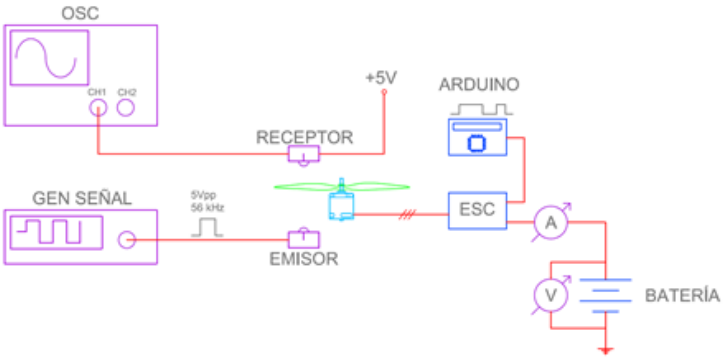


Figura 6.2: Diagrama de armado eléctrico.[3]

Descripción

Se fija el motor a una barra metálica apoyada en sus extremos, como se puede observar en la Figura 6.1. Se conecta el motor de forma tal que la fuerza realizada a la barra sea hacia abajo. El esquema de la conexión eléctrica se muestra en la Figura 6.2.

La velocidad angular del motor se controla con el ancho de pulso de la señal PWM que se envía al controlador ESC. Esta señal se varía de 1000 μs a 2000 μs con pasos de 100 μs.

Para relevar el empuje, fuerza realizada por el motor, se registra el peso que marca la balanza. Además se mide corriente consumida y la frecuencia de la señal cuadrada por el detector IR para luego poder obtener la velocidad angular del motor.

La señal enviada con el generador de señales es una onda cuadrada de 5 Vpp y 56 kHz, frecuencia a la cual funciona el conjunto fotodiodo.

6.2.2. Resultados y Análisis de datos

La Tabla 6.1 muestra las medidas relevadas para el ensayo del empuje.

Tabla 6.1: Datos relevados para hélice 13x40.

PWM (μs)	Balanza (g)	Corriente (A)	Voltaje (V)	Frecuencia (Hz)
1100	3	0,20	14,88	22,23
1200	23	0,51	14,86	53,10
1300	61	0,81	14,85	80,13
1400	110	1,48	14,83	104,6
1500	172	2,34	14,80	127,8
1600	241	3,79	14,76	148,6
1700	319	5,40	14,69	168,9
1800	398	7,40	14,61	186,3
1900	473	9,80	14,52	202,6
2000	512	11,51	14,43	211,7

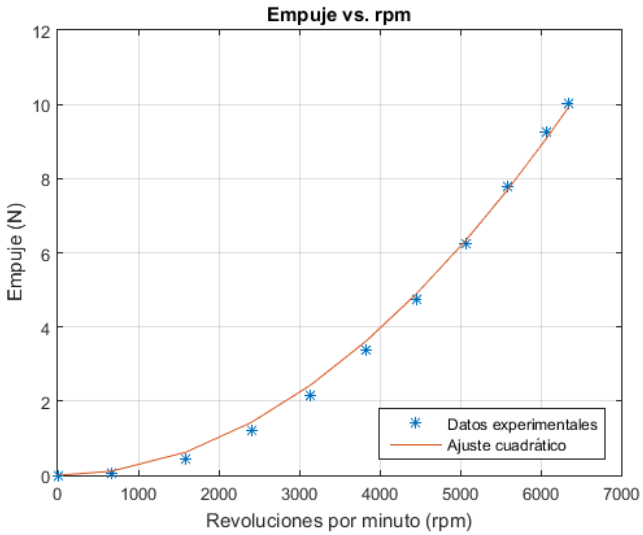


Figura 6.3: Empuje vs. rpm hélice 13x40.

La Figura 6.3 muestra la gráfica del empuje en función de las rpm para la hélice 13x40. Se muestran los datos relevados junto con un ajuste cuadrático de la forma E(N)=$a\times$rpm^2 donde $a = 2{,}464 \times 10^{-7}$ N/rpm^2.

6.3. Medida de torque

6.3.1. Experimento

Elementos utilizados

- Motor EMAX MT3506 650 kV

- Hélice fibra de carbono 12x55 y 13x40
- ESC 35 A Makerfire
- Batería ZIPPY Compact 6200 mAh
- Generador de señales SIGLENT SDG805
- Fuente SIGLENT para 5 V
- Osciloscopio SIGLENT SDS1102CML+
- Conjunto fotodiodo (LED IR TSAL6200 y Detector IR TSOP38256)
- Balanza electrónica CAMRY EK9320

Descripción

El motor se monta fijo en una barra articulada sin fricción. El esquema mecánico del experimento se muestra en la Figura 6.4. Se arma el mismo circuito eléctrico que el utilizado para la medida del empuje (Figura 6.2).

La balanza mide la fuerza del torque que el motor ejerce sobre la barra.

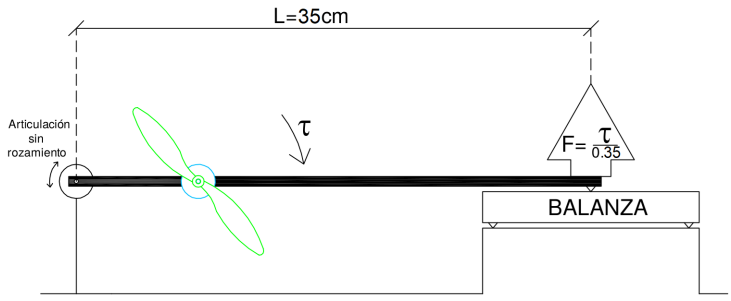


Figura 6.4: Diagrama de armado mecánico.[3]

6.3.2. Resultados y análisis de datos

La Tabla 6.2 muestra las medidas relevadas para el ensayo de torque.

Tabla 6.2: Datos relevados para hélice 13x40.

PWM (μs)	Balanza (g)	Corriente (A)	Voltaje (V)	Frecuencia (Hz)
1100	0	0	0	22,23
1200	4	0,58	16,98	53,10
1300	9	0,92	16,57	80,13
1400	13	1,52	16,08	104,6
1500	19	2,40	15,96	127,8
1600	28	3,98	15,46	148,6
1700	37	5,80	15,34	168,9
1800	46	8,10	15,32	186,3
1900	53	10,10	15,14	202,6
2000	58	12,67	15,06	211,7

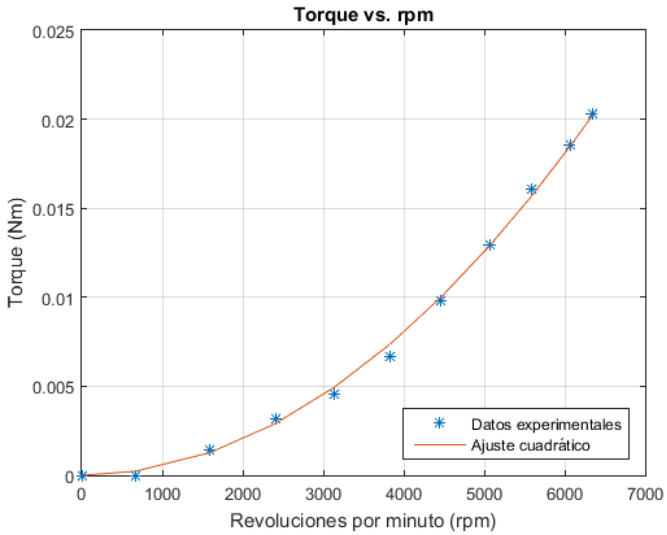


Figura 6.5: Torque vs. rpm para la hélice 13x40.

La Figura 6.5 muestra la gráfica del torque en función de las rpm para la hélice 13x40. Se muestran los datos relevados junto con un ajuste cuadrático de la forma $\tau(\text{Nm}) = a \times \text{rpm}^2$ donde $a = 5{,}021 \times 10^{10}$ Nm/rpm^2.

6.4. Comparación de hélices y consumo

Realizando los mismos estudios a las hélices de 12x55, se procedió a hacer una análisis comparativo del empuje y el torque realizado para cada hélice (Figura 6.6 y Figura 6.7).

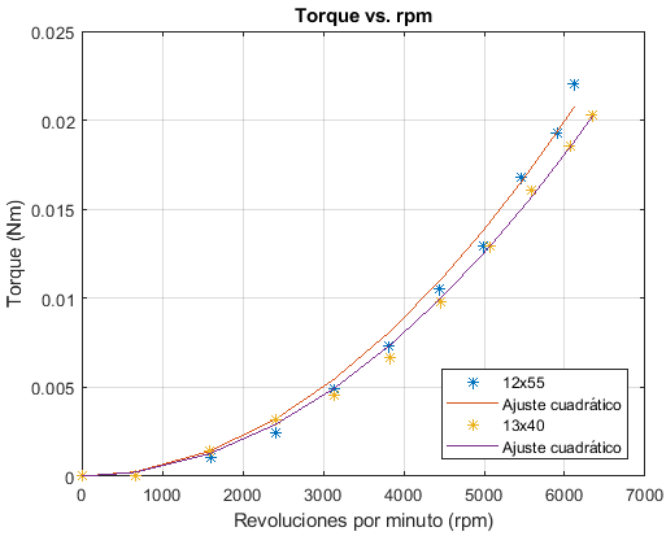


Figura 6.7: Torque vs. rpm hélices 12x55 y 13x40.

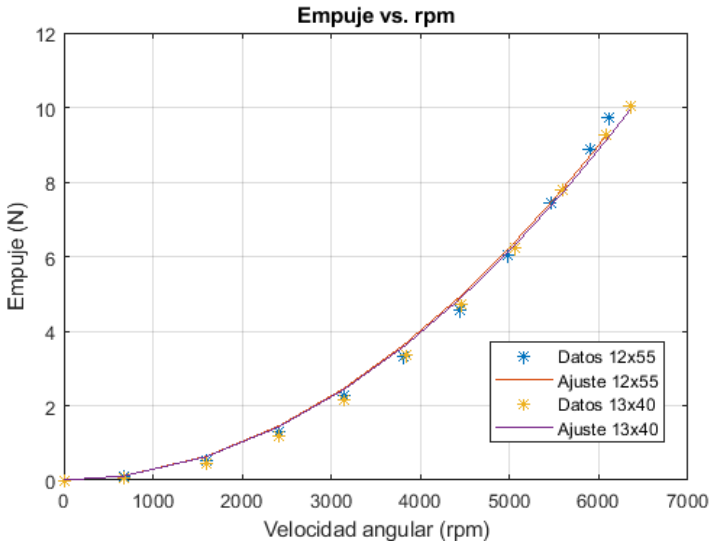


Figura 6.6: Empuje vs. rpm hélices 12x55 y 13x40.

Como se obtuvieron resultados muy similares para los dos tipos de hélices, se pasó a estudiar su consumo para elegir la hélice óptima. Para esto se evaluó el consumo en cada una de ellas frente al empuje ejercido. Los resultados se observan en la Figura 6.8.

Nuevamente se obtuvieron resultados muy similares entre las dos hélices, notando un mejor desempeño en empujes altos con el modelo 13x40. Por lo tanto ésta fue la hélice elegida.

Resulta interesante conocer también la eficiencia de la hélice en todo su rango de funcionamiento. Una forma de visualizarlo es ver como cambia la relación N/W

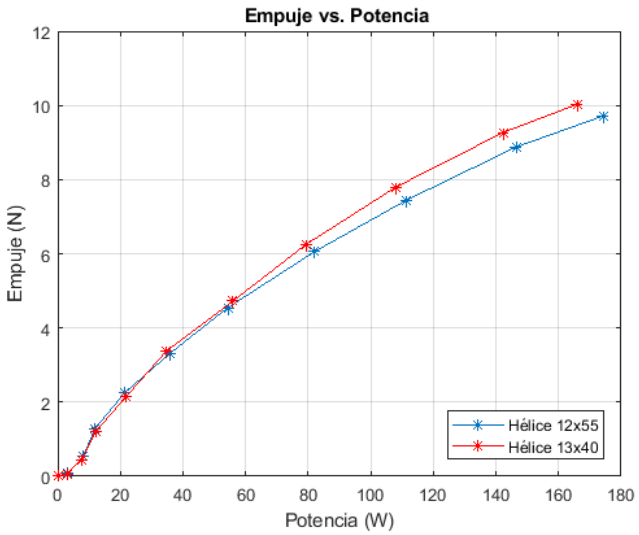


Figura 6.8: Empuje vs. potencia hélices 12x55 y 13x40.

(cuántos Watts son necesarios para producir 1 N de empuje) con los distintos empujes que es capaz de producir la hélice.

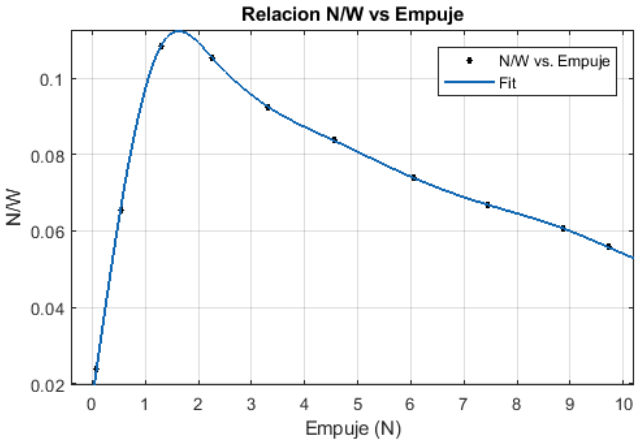


Figura 6.9: Eficiencia de la hélice 13x40.

El punto de rendimiento máximo está en los 2 N de empuje. Para poder compensar el peso del drone, cada hélice deberá producir aproximadamente 5,25 N de empuje. En este punto de funcionamiento, el rendimiento será de un 70 % relativo al rendimiento máximo.

Esta página ha sido intencionalmente dejada en blanco.

Parte IV

Estimación del Estado

Capítulo 7

Fusión Sensorial

7.1. Introducción

El estado del drone se puede dividir en dos, el estado traslacional (posición y velocidad) y el estado rotacional (orientación y velocidad angular). El primero, se obtiene directamente de la IMU, mientras que el segundo es posible estimarlo utilizando la posición, la altura y la aceleración dada por el GPS, el barómetro y la IMU respectivamente. Pero dado que las frecuencias de muestreo de las magnitudes con las que trabajan son distintas, es necesario combinar estas medidas, utilizando la fusión sensorial.

Para realizar la fusión sensorial se decidió utilizar dos filtros, el Filtro de Kalman Extendido (EKF) [4], extensión del algoritmo presentado por Rudolf E. Kalman [9] en 1960 y el Filtro Complementario [5]. El Filtro de Kalman Extendido es capaz de estimar una serie de variables de estado junto con sus varianzas, usando las mediciones de estados y las leyes que rigen al sistema para encontrar la estimación de estado óptima. Asumiendo como hipótesis que el ruido sigue distribución gaussiana, el filtro expande las ecuaciones de estado y observaciones usando la fórmula de Taylor, obteniendo así una linealización de primer orden. La mejora que introduce el EKF es que puede utilizarse en sistemas no lineales. Mientras que el Filtro Complementario es una simplificación del Filtro de Kalman que no tiene en cuenta las varianzas de las medidas.

El estado traslacional se estima utilizando el Filtro de Kalman, mientras que para el estado angular se usa el Filtro Complementario ya que el costo computacional de este es más bajo.

7.2. Filtro de Kalman Extendido

La representación en tiempo discreto del espacio de estados de un sistema no lineal está dado por:

$$x_k = f(x_{k-1}, u_{k-1}) + w_{k-1} \tag{7.1}$$

$$y_k = h(x_k, u_k) + v_k \tag{7.2}$$

donde $x_k \in R^{N_X}$ representa el vector de estados y k el paso de tiempo; $y_k \in R^{N_Y}$ representa el vector de observaciones. N_X y N_Y representan la dimensión de los vectores de estados y observaciones respectivamente, u_k representa el vector de entradas. $f : R^{N_X} \rightarrow R^{N_Y}$ es la función de transición de estado no lineal que refleja la dinámica del sistema y $h : R^{N_X} \rightarrow R^{N_Y}$ es la función de observación no lineal que relaciona el vector de estados x_k con el vector de medidas y_k.

Se toma como hipótesis que las funciones f y h son C_1. w_{k-1} es el vector de ruido relacionado al proceso; $v_k \in R^{N_Y}$ es el vector de ruido asociado a la observación. v_k y w_{k-1} son independientes entre sí, asumiendo que ambos son ruido blanco Gaussiano con media cero, las definiciones de sus covarianzas son:

$$E\{w_i w_j^T\} = Q_i \delta_{ij} \tag{7.3}$$

$$E\{v_i v_j^T\} = R_i \delta_{ij} \tag{7.4}$$

con δ_{ij} la función de Kronecker (valor unitario cuando $i = j$, nula en los demás casos). Q_i y R_i representan los elementos de la matriz de covarianza del ruido asociado al proceso, y la matriz de covarianza del ruido asociado a la observación respectivamente. Basados en que w y v no están correlacionados, Q y R son matrices diagonales con valores asumidos constantes para simplificar el problema.

Como las ecuaciones que describen el espacio de estados son diferenciables, se pueden linealizar con la ecuación de Taylor. Este proceso de linealización se expresa como:

$$F_{k|k-1} = [F_{i,j}] = \left[\frac{\partial f_i}{\partial x_j} |_{\hat{x}_{k-1}} \right] \tag{7.5}$$

$$H_k = [H_{i,j}] = \left[\frac{\partial h_i}{\partial x_j} |_{\hat{x}_{k|k-1}} \right] \tag{7.6}$$

con $F_{k|k-1}$ y H_k las matrices Jacobianas de $f(\cdot)$ y $h(\cdot)$ respectivamente. $\hat{x}_{k-1}$ es el valor estimado a posteriori del vector de estados en el paso de tiempo $k-1$, $\hat{x}_{k|k-1}$ es el valor estimado a priori del vector de estados en el paso k.

El algoritmo consta de dos loops de cálculo, el de ganancia y el de filtrado, ambos conectados entre sí. Esta conexión se logra a través del factor K_k (ganancia de Kalman), que intenta minimizar P_k (covarianza del error del vector de estimación de estado). La ganancia de Kalman está definida por la siguiente ecuación:

$$K_k = P_{k|k-1}.H_k^T (H_k P_{k|k-1} H_k^T + R)^{-1}. \tag{7.7}$$

En el loop de filtrado se realizan dos procesos de actualización, primero en el tiempo y luego en la medida. La actualización en el tiempo, actualiza el vector de estimación de estado utilizando la función de transición

$$\hat{x}_{k|k-1} = f(\hat{x}_{k-1}, u_{k-1}), \tag{7.8}$$

mientras que la actualización de la medida, actualiza la covarianza del error del vector de estado

$$P_{k|k-1} = F_{k|k-1} P_{k-1} F_{k|k-1}^T + Q. \tag{7.9}$$

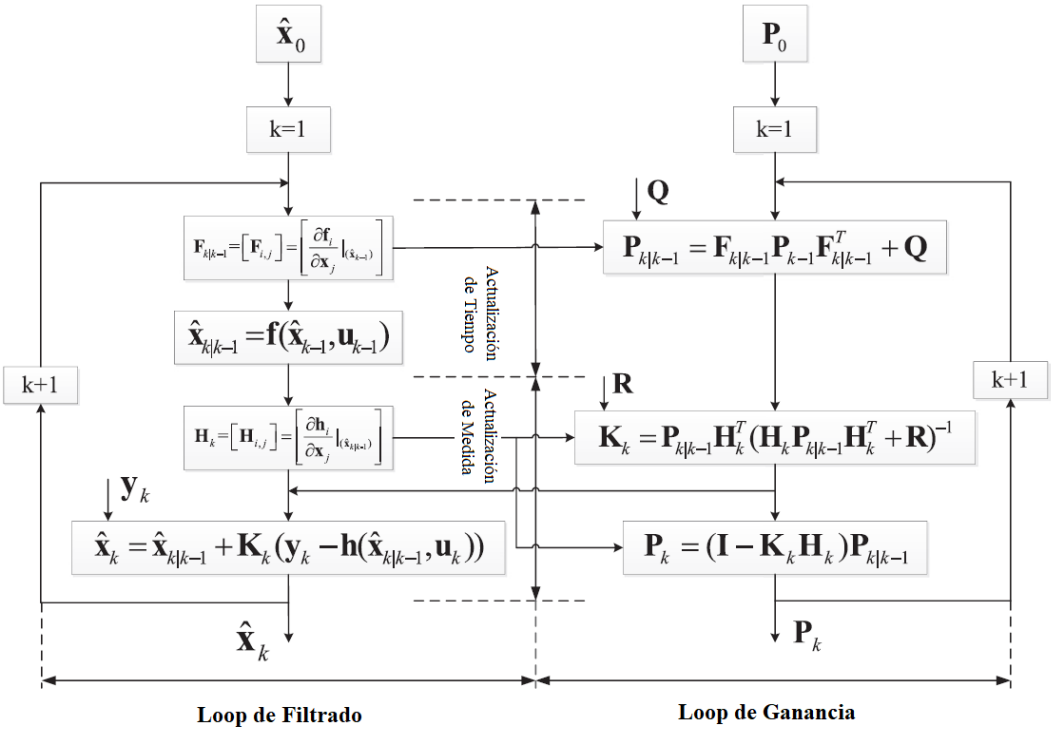


Figura 7.1: Loop de cálculo del algoritmo EKF.[4]

Se puede calcular ahora K_k y el vector de estados estimados a posteriori se obtiene en el proceso de actualización de medida como

$$\hat{x}_k = \hat{x}_{k|k-1} + K_k(y_k - h(\hat{x}_{k|k-1}, u_k)) \tag{7.10}$$

$$P_k = (Id - K_k H_k)P_{k|k-1}. \tag{7.11}$$

Y el vector de observaciones actualizado es

$$\hat{y_k} = h(\hat{x}_k, u_k) \tag{7.12}$$

Para mejor entendimiento, la Figura 7.1 muestra los pasos de calculo del algoritmo EKF.

7.3. Uso del filtro en el sistema

En nuestro sistema, el filtro es utilizado para adecuar las señales de los distintos sensores que describen las variables del estado traslacional del drone ya vistas en la Sección 4.4. El vector de estados x_k es un vector de dimensión 9 el cual posee la posición, velocidad y aceleración del sistema, siendo de la forma:

$$x_k = (x(k\Delta t), y(k\Delta t), z(k\Delta t), \dot{x}(k\Delta t), \dot{y}(k\Delta t), \dot{z}(k\Delta t), \ddot{x}(k\Delta t), \ddot{y}(k\Delta t), \ddot{z}(k\Delta t)) \tag{7.13}$$

donde x, y, z son las posiciones en sus respectivos ejes y Δt el tiempo de muestro del algoritmo.

Utilizando que la velocidad es la integral de la aceleración y la posición la integral de la velocidad, se obtiene que $f(x_{k-1}, u_{k-1})$ es una función independiente de u_{k-1} y lineal en x_{k-1}, por lo cual la matriz $F_{k|k-1}$ es una matriz constante de la forma:

$$F_{k|k-1} = \begin{pmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{\Delta t^2}{2} & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{\Delta t^2}{2} & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 & \frac{\Delta t^2}{2} \\ 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{7.14}$$

Ya que los sensores involucrados para estimar este estado obtienen la medida de forma directa, la matriz H_k tiene todos sus valores 0 exceptuando el correspondiente a la magnitud que se está midiendo del vector x, el cual vale 1. Por ejemplo para el GPS se obtiene:

$$H_{GPS_k} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}. \tag{7.15}$$

La matriz R representa el ruido de los diferentes sensores y sus correlaciones. Si se suponen independientes entre sí, la matriz es una diagonal con valores de la varianza al cuadrado de cada sensor. Para el GPS esta vale:

$$R_{GPS_k} = \begin{pmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{pmatrix} \tag{7.16}$$

donde σ_x y σ_y es la varianza de la medida en los ejes x e y respectivamente del GPS.

La matriz Q, en cambio y de acuerdo a varios autores [10], es el parámetro de diseño más difícil de seleccionar en el filtro de Kalman extendido, ya que esta matriz relaciona el ruido y la covarianza entre los ruidos de las distintas variables de estado, haciendo que se deba ajustar empíricamente.

En nuestro caso el modelo de planta del drone (relaciones entre entradas, salidas y variables de estado del sistema) va a tener diferencias con el modelo real, ya sea por errores en la identificación de parámetros (medida del peso e inercia del drone, medida de las funciones que caracterizan al sistema de propulsión) como errores del modelado físico, errores de medida, efecto de la discretización y aproximaciones en la derivación de variables.

Para simplificar el problema, se toma como punto de partida la matriz Q del proyecto de software libre Robot Localization[11], el cual implementa nodos de ros para realizar la fusión sensorial.

$$Q = \begin{pmatrix} 0{,}05 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0{,}05 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0{,}06 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0{,}025 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0{,}025 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0{,}04 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0{,}01 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0{,}01 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0{,}01 \end{pmatrix} \tag{7.17}$$

7.4. Filtro complementario

Dada la alta demanda de computo que requiere el Filtro de Kalman, se decidió usar un algoritmo más intuitivo para la obtención del estado angular. Un filtro complementario es en sí un filtro de Kalman de estado estacionario simplificado, donde no se considera ninguna descripción estadística del ruido y es obtenido solamente por un análisis en el dominio de la frecuencia. El filtro resulta sencillo de tratar matemáticamente y su implementación consume pocos recursos computacionales.

La idea básica de este filtro es combinar la salida del acelerómetro y del giróscopio para obtener una buena estimación del ángulo de orientación del drone, compensando los errores del giróscopo con la dinámica lenta del acelerómetro [5].

El filtro complementario propuesto es el que se muestra en la Figura 7.2 donde θ_a es el ángulo medido por el acelerómetro cuya señal está afectada por ruidos de alta frecuencia provenientes de las vibraciones, por lo que es filtrado por un pasa bajos. θ_g es el ángulo medido por el giróscopo afectado por el offset y por lo tanto es filtrado por un pasa altos y $\widehat{\theta}$ es el ángulo estimado.

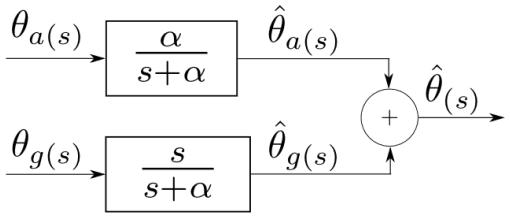


Figura 7.2: Diagrama de bloques del filtro complementario.[5]

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 8

Caracterización de Sensores

8.1. Caracterización de sensores

Como se vio en el capítulo anterior, para la implementación del filtro de Kalman se necesitan conocer las varianzas de los distintos sensores. En esta capítulo se muestra como se obtuvieron las medidas.

8.1.1. IMU

Descripción y caracterización

La IMU (Inertial Measurement Unit) es un dispositivo electrónico capaz de medir velocidad, orientación y fuerzas gravitacionales en torno a 3 ejes cartesianos. Esto lo logra integrando un giroscopio, acelerómetro y magnetómetro dentro de un mismo sistema.

En adición, la IMU adquirida también integra un barómetro y un termómetro para obtener una medida de altitud respecto a un punto de altitud y presión atmosférica conocida, en nuestro caso, presión a nivel del mar.

Las especificaciones brindadas por el fabricante[12] son:

Tabla 8.1: Especificaciones de la IMU.

Marca/modelo	BerryIMU v2
Acelerómetro	Rango ±2g / ±4g / ±8g / ±16g
Giroscopio	Rango ±245/±500/±2000 dps
Magnetómetro	Rango ±4 / ±8 / ±12 / ±16 gauss
Barómetro	Rango 300 a 1100 hPa

Para obtener la varianza de las distintas variables se tomaron una serie de medidas estáticas. Los resultados fueron los siguientes:

$$\boxed{\sigma^2_{roll} = 2{,}30 \times 10^{-6}\ \text{rad}^2} \qquad \boxed{\sigma^2_{pitch} = 4{,}64 \times 10^{-6}\ \text{rad}^2}$$

$$\boxed{\sigma^2_{yaw} = 1{,}64 \times 10^{-3}\ \text{rad}^2}$$

La Figura 8.1 muestra el campo magnético relevado por la IMU sin ser calibrada, variando su orientación. Se puede observar como esta tiene forma de circunferencia pero está corrida del eje $(0, 0)$, incluyendo el eje z se obtiene una esfera corrida. Esto se debe a distorsiones provocadas por objetos que generan un campo magnético, denominado "Hard Iron Distorsion". Esta distorsión es evitable al calibrar el magnetómetro.

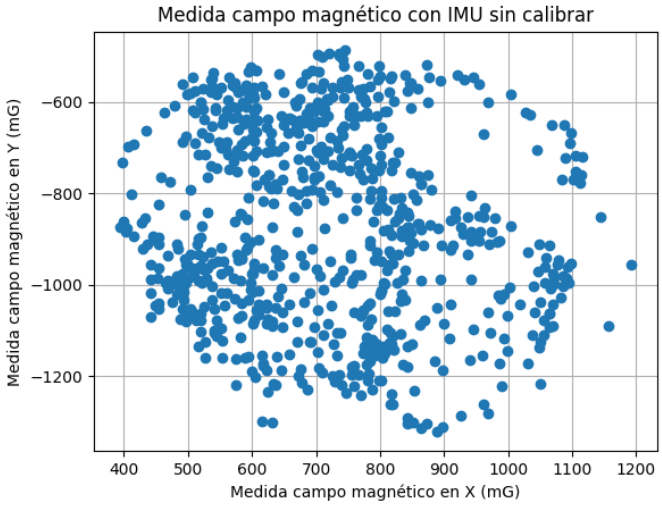


Figura 8.1: Campo magnético de la IMU sin calibrar.

Una vez calibrada la IMU, se generó la gráfica que se muestra en la Figura 8.2 comprobando ahora que la esfera está centrada en el $(0, 0)$ y por tanto la calibración se hizo de forma satisfactoria.

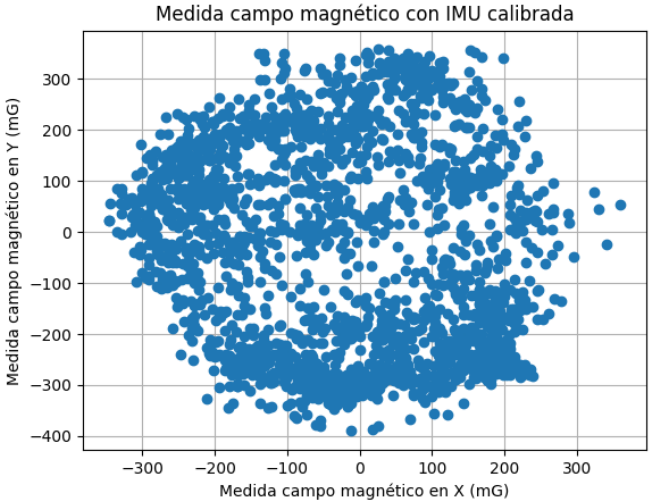


Figura 8.2: Campo magnético una vez calibrada la IMU.

8.1.2. GPS

Descripción

El GPS (Global Positioning System) es un sistema de radio navegación global basado en la recepción de señales provenientes de satélites. El concepto de su funcionamiento es el siguiente: cada satélite lleva consigo un reloj atómico sincronizado uno con otro y a su vez con los relojes en la tierra. Por otra parte las locaciones de los satélites se conocen de manera exacta.

Los satélites transmiten continuamente una señal de radio conteniendo datos de su posición y su reloj. Como la velocidad de las ondas de radio es constante y conocida, la diferencia de tiempo entre la señal recibida y el tiempo local del receptor es proporcional a la distancia entre el satélite y el receptor. Este último, conectado a varios satélites, resuelve ecuaciones para triangular una posición precisa.

El módulo utilizado fue el Ublox NEO-M8N [13] cuyas especificaciones se muestran en la Tabla 8.2.

Tabla 8.2: Especificaciones del GPS.

Marca/modelo		Ublox NEO-M8N
Límites operacionales	Dinámica	$\leq$4g
	Altitud	50000 m
	Velocidad	500 m/s
Precisión	Velocidad	$\pm$0,05 m/s
	Orientación	$\pm$0,3°

Para obtener los datos en latitud y longitud se utilizó la librería NEO-GPS. La misma se introduce como una librería totalmente configurable, que minimiza el uso de RAM (10 bytes en total) y CPU.

Para la conversión de ángulos a coordenadas cartesianas se utilizó el modelo del geoide WGS84, también utilizado por uQuad3 [3].

Caracterización

Para poder implementar el filtro de Kalman es necesario medir la varianza de una serie de medidas realizadas por el GPS. Gracias a resultados de proyectos uQuad anteriores sabemos que el GPS mejora sustancialmente su performance cuando se encuentra en movimiento. Es por esto que se hace un análisis a un set de medidas estáticas.

La muestra se realiza dejando el GPS quieto en una posición durante 30 minutos, siendo este un tiempo mayor al máximo que se espera que el drone vuele.

En la Figura 8.3 puede observarse la muestra realizada. Los círculos concéntricos marcan las medidas comprendidas entre 1, 2 y 3 metros de radio (circunferencias roja, amarilla y verde respectivamente).

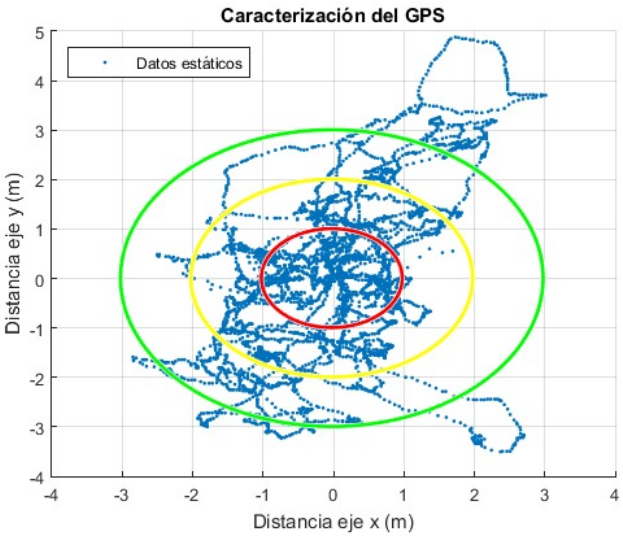


Figura 8.3: Experimento estático GPS.

De las medidas tomadas se calculó la varianza en el eje x y eje y obteniendo los siguientes resultados para el GPS elegido:

$$\boxed{\sigma_x^2 = 1{,}08 \text{ m}^2} \qquad \boxed{\sigma_y^2 = 2{,}64 \text{ m}^2}$$

Parte V

Automatismo y Control con Inteligencia Artificial

Capítulo 9

Introducción

9.1. Fundamentos de Reinforcement Learning

9.1.1. Consideraciones previas

El Reinforcement Learning (RL) [14] es una técnica del aprendizaje automático, compuesta por un agente y un entorno, donde se busca enseñar al agente qué acción debe tomar en el entorno para maximizar una recompensa.

El entorno es el mundo donde vive e interactúa el agente. En cada paso de iteración el agente observa el entorno y realiza una acción en consecuencia; el entorno cambia con la acción del agente y puede cambiar por su cuenta.

Para que el agente sea capaz de cumplir un objetivo, este recibe una señal de recompensa del entorno al realizar una acción. Por ejemplo si se busca llegar a un punto, la recompensa puede crecer a medida que la distancia al punto sea menor. La Figura 9.1 ejemplifica la relación descrita.

Estados y observaciones

Un estado s es una descripción completa del entorno, todo lo que se sabe del mismo es proporcionado por s. Una observación es una descripción parcial del estado, que puede omitir información.

Distintos entornos permiten diferentes tipos de acciones. El conjunto de todas las acciones válidas es llamado espacio de acciones. Los hay discretos y continuos.

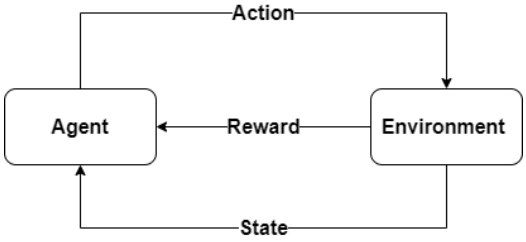


Figura 9.1: Diagrama de Reinforcement Learning.

En nuestro caso nos vamos a enfocar en este último, en los espacios continuos las acciones son vectores de números reales.

Políticas

Una política es una regla usada por un agente para decidir qué acciones tomar, es esencialmente el cerebro del agente.

Pueden ser deterministas

$$\mu : a_t \mu(s_t),$$

y también pueden ser estocásticas,

$$\pi : a_t \sim \pi(\cdot|s_t).$$

En RL se utiliza políticas parametrizadas, aquellas cuya salida está compuesta por funciones computables que dependen de un set de parámetros los cuales podemos ajustar para cambiar el comportamiento mediante el uso de algún algoritmo de optimización. Están funciones serán las redes neuronales.

Los parámetros de una política los denotamos con θ o ϕ, y los escribimos como un subscript en la política para resaltar la conexión:

$$a_t = \mu_\theta(s_t)$$

$$a_t \sim \pi_\theta(\cdot|s_t).$$

Trayectorias

Una trayectoria τ es una secuencia de estados y acciones en el environment

$$\tau = (s_0, a_0, s_1, a_1, ...).$$

El primer estado es muestreado de manera aleatoria desde la distribución de estado inicial denotado ρ_0 :

$$s_0 \sim \rho_0(\cdot).$$

Las transiciones de estado (lo que pasa entre el estado a tiempo t, s_t, y el estado a $t+1$, s_{t+1}) son gobernadas por las leyes del entorno y dependen de la acción más reciente a_t. Pueden ser deterministas: $s_{t+1} = f(s_t, a_t)$ o estocásticas: $s_{t+1} \sim P(\cdot|s_t, a_t)$.

Recompensa y Retorno

La función recompensa R depende del estado actual, la acción recién tomada y el próximo estado, $r_t = R(s_t, a_t, s_{t+1})$.

El objetivo del agente es maximizar la recompensa acumulativa sobre una trayectoria $R(\tau)$, a esta función la llamaremos retorno (return en ingles). Existen dos tipos de retornos, "finite-horizon undiscounted return" y "infinite-horizon discounted return".

El tipo "finite-horizon undiscounted return", es la suma de recompensas obtenidas en una ventana fija de pasos:

$$R(\tau) = \sum_{t=0}^{T} r_t.$$

Y el tipo "infinite-horizon discounted return", es la suma de todas las recompensas obtenidas por el agente, pero descontadas dependiendo de qué tan lejos en el futuro fueron obtenidas, utilizando un factor de descuento $\gamma \in (0, 1)$:

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t.$$

9.1.2. Planteo del Problema de Reinforcement Learning

Cualquiera sea el tipo de retorno elegido, y cualquiera sea la política elegida, el objetivo de RL es seleccionar una política que maximice el retorno cuando el agente actúa acuerdo a ella.

Supongamos que las transiciones del entorno y la política son estocásticas. En este caso, la probabilidad de una trayectoria de T-pasos es:

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t).$$

El retorno esperado $J(\pi)$, es entonces:

$$J(\pi) = \int_{\tau} P(\tau|\pi)R(\tau) = \underset{\tau\sim\pi}{E}[R(\tau)].$$

El problema central de optimización en RL puede ser expresado por

$$\pi^* = \arg\max_{\pi} J(\pi)$$

con π^* la política óptima.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 10

Controlador de Actitud

10.1. Introducción

Dada las ecuaciones del sistema que describen al drone, se puede observar que la posición y orientación pueden ser controladas completamente variando las velocidades angulares de los motores. Además, si se quiere que el drone quede estático a una altura y orientación deseada, la aceleración en x e y en un sistema de referencia en la Tierra, queda determinado. Por lo tanto se busca diseñar un controlador capaz de controlar la actitud del drone. Por control de actitud se entiende como mantener las variables de posiciones y velocidades angulares en torno a un set-point deseado. Para esto se decidió utilizar redes neuronales entrenadas utilizando Reinforcement Learning.

En primer lugar se entrenó una red que, a partir de la diferencia entre el vector orientación concatenado a la altura del drone y un vector compuesto por la orientación y altura deseada, determine cuáles deben ser las señales de PWM a enviar a la ESC, para que el mismo alcance el estado deseado utilizando el algoritmo DDPG[15] y el simulador Gazebo. Con este enfoque no se logró que la red convergiera, por lo que se decidió entrenar dos redes que actúen como controladores para simplificar el problema en el entrenamiento.

El primer controlador se diseñó para controlar la velocidad angular, y el segundo para controlar la velocidad en el eje z independientemente de la posición angular. Para completar el control de actitud se deberán combinar las salidas de ambos controladores de forma tal que se logren satisfacer los dos objetivos.

Para el entrenamiento de ambos controladores, se utilizaron los algoritmos basados en PPO [16] proporcionado por la librería Stable-Baseline [17] en Python (realizándole pequeñas modificaciones que se mencionarán más adelante). Mientras que para la simulación, se programó un simulador usando las ecuaciones de dinámica del drone para acelerar el entrenamiento ya que los controladores deben tomar acciones en un intervalo de tiempo muy corto y es necesario reanudar y detener la simulación en cada paso de acción. A partir de este simulador se generaron entornos basados en gym [18] de OpenAI, para hacerlo compatible con la librería Stable-Baseline y otras librerías de aprendizaje por refuerzo.

Estos algoritmos se ejecutan en la CPU y GPU del nVidia Jetson Nano. A partir de pruebas realizadas, se obtuvo que la frecuencia de control en una primera versión del controlador eran cercanas a los 100 Hz, siendo esta una frecuencia demasiado baja como para asegurar el control. Para mejorar esto se trabajó sobre los algoritmos de programación. El cambio principal fue el pasaje de Python a C++ y la simplificación de algunas funciones recursivas, logrando así una frecuencia máxima de 175 Hz.

10.2. Controlador de posición angular

Para controlar la posición angular se optó por dividir el problema en dos, primero entrenar una red neuronal capaz de controlar la velocidad angular del drone dada una velocidad deseada. Luego, utilizando un controlador proporcional, poder controlar la posición angular haciendo que este retorne en su salida la velocidad angular necesaria para llegar a dicho estado.

10.2.1. Controlador de velocidad angular

El objetivo del controlador de velocidad angular es, dado un setpoint de velocidad angular y la velocidad angular actual del sistema, obtener los valores de PWM que se necesitan enviar a las ESCs, para llegar al setpoint.

Este controlador se basó principalmente en la implementación realizada en la tesis de Neuroflight[19].

Arquitectura de la red neuronal

Para reducir la dimensión de la entradas en la red, se calcula previamente el error al setpoint deseado de la forma:

$$e(t) = (w^*_\phi(t) - w_\phi(t), w^*_\theta(t) - w_\theta(t), w^*_\psi(t) - w_\psi(t)) \tag{10.1}$$

donde $w_\phi(t), w_\theta(t), w_\psi(t)$ son las velocidades angulares en roll, pitch y yaw respectivamente en el tiempo t y $w^*_\phi(t)$, $w^*_\theta(t)$ y $w^*_\psi(t)$ el setpoint. Este vector se concatena con la variación del error en t y $t-1$, obteniendo como entrada a la red un vector de dimensión seis de la forma $(e(t), \Delta e(t))$.

La salida de la misma corresponde al valor de PWM de las ESCs, por lo tanto se define un vector de dimensión cuatro de la forma $(y_1(t), y_2(t), y_3(t), y_4(t))$ donde cada elemento $y_i(t)$, corresponde al valor de PWM a enviar al ESC i, como salida de la red. Para no apagar los motores y no utilizar todo el ancho de acción disponible en PWM ([1000 μs; 2000 μs]), se restringe que la acción de control mínima sea de 1100 μs y se entrena para que el máximo no supere los 1450 μs.

Finalmente, para la arquitectura de la red se utiliza una red multicapa con una capa de entrada de 6 nodos, dos capas ocultas de 8 nodos y una capa de salida de 4 nodos (Figura 10.1). La función de activación en todas las capas es una tangente hiperbólica.

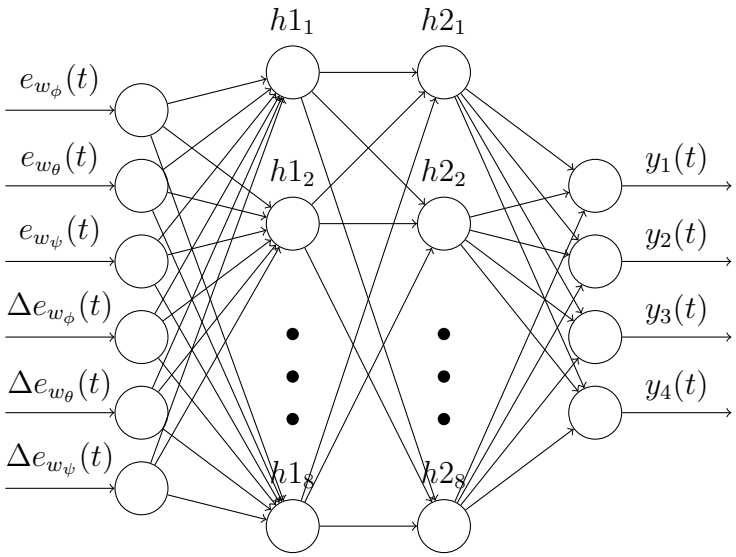


Figura 10.1: Arquitectura red neuronal controlador velocidad angular.

Entrenamiento

Para entrenar el controlador se programó un entorno[1] utilizando la librería gym de OpenAI y las ecuaciones que describen el movimiento del drone, obteniendo de esta forma un simulador compatible con distintas librerías capaces de entrenar redes utilizando algoritmos de Reinforcement Learning.

Este entorno se programó para que cada 4 segundos de vuelo, se reinicie la simulación, inicializando el setpoint Ω^* y la velocidad angular Ω en $(0, 0, 0)$. Cuando la simulación llega a los 0,5 s, se cambia el setpoint por uno aleatorio $\sim N(0,$ 1.4) para las componentes de roll y pitch, y un setpoint $\sim N(0,0.7)$ para el yaw. Finalmente cuando se llega a los 2,5 segundos de simulación se vuelve a cambiar el setpoint por $(0, 0, 0)$. De esta forma se logra que la red sea capaz de responder a distintos setpoints y que se intensifique el entrenamiento en velocidades cercanas al 0, ya que es donde el drone pasará más tiempo.

Como la salida de la red posee una distribución gaussiana, se configuró el entorno para que las acciones esperadas fueran valores entre $[-1, 1]$. Internamente, si la acción está fuera de estos rangos, se limitará, para luego pasarlo a valores entre $[0, 1]$ correspondientes al [1100 μs, 2000 μs] de los PWM que se aplicarán a las ESCs, con las siguientes ecuaciones:

$$y_{clip} = clip(a, a_{min}, a_{max}) \tag{10.2}$$

$$u = \frac{y_{clip} - a_{min}}{a_{max} - a_{min}} \tag{10.3}$$

donde a es la salida de la red, u la salida mapeada a valores ente $[0, 1]$ y a_{min} y

[1] Archivo "tesis_uquad4/src/uquad4_neural_networks/src/environments/control_vel_ang.py"

a_{max} los valores mínimos y máximos que permitimos que tome la salida de la red respectivamente, los cuales valen -1 y 1.

El objetivo de este entrenamiento es, no solo conseguir que el drone sea capaz de converger a una velocidad angular deseada, sino que también lo haga con el menor consumo posible. Las razones principales de esto, es que la salida de todos los motores apagados y la salida de todos los motores a su máxima rpm, producen el mismo efecto en la velocidad angular, pero la segunda además de generar un consumo excesivo, reduciría el rango que tiene el controlador de altura para controlar la misma debido al algoritmo de mixing de controladores que se explicará más adelante.

El Algoritmo 1, se diseñó para lograr cumplir dichos objetivos. Se comienza fijando la recompensa a cero para luego sumarle la recompensa basada en cuánto mejoró respecto al eje que más error poseía (líneas 2 y 3), debido a que si se busca minimizar el error en todos los ejes utilizando la media cuadrática, el error en yaw suele no converger a cero dado que este eje es el más lento respecto a los otros. Para minimizar las oscilaciones, se resta el máximo de la diferencia entre la acción realizada en la actualidad (t) y la acción pasada ($t-1$), línea 4. El bajo consumo se recompensa si el error en todos los ejes es menor al 15 % respecto al setpoint actual, recompensando qué tan bajo es el promedio de las acciones (líneas 5 y 6). Finalmente para que la salida de la red no sature en uno, se agrega una penalización (líneas 7 y 8) que penaliza qué tan grande es la salida de la red (sin clipear) respecto a la acción máxima permitida. No se penalizan los valores bajos para no afectar la convergencia.

Algoritmo 1: Función de recompensa controlador velocidad angular

Result: Recompensa r en el tiempo t

1 $r \leftarrow 0$
2 $i_{emax}(t) \leftarrow i \in \{\psi; \theta : \phi\} : e_i(t) = \text{máx}(|\Omega(t) - \Omega^*(t)|)$
3 $r \leftarrow r + 3(-e_{i_{emax}(t)}(t) + e_{i_{emax}(t)}(t-1))$
4 $r \leftarrow r - \beta\,\text{máx}(|u(t) - u(t-1)|)$
5 **if** $|e(t)| \leq 0{,}15|\Omega^*|$ **then**
6 $\quad r \leftarrow r + \gamma(1 - \overline{u}(t))$
7 **for** $a_i \in a$ **do**
8 $\quad r \leftarrow r - \text{PENALTY}\,\text{máx}(a_i - 1{,}05, 0)$

Para el entrenamiento se utilizó el algoritmo PPO2 [20] de la librería Stable-Baseline, el cual se modificó para que no realizara un clip sobre la salida de la red y así poder aplicar la penalización. Esta implementación de PPO es capaz de entrenar la red corriendo varias simulaciones al mismo tiempo, por lo que para acelerar los tiempos de entrenamiento se configuró para correr 16 entornos al mismo tiempo. La Tabla 10.1 muestra los hiper-parámetros elegidos para el entrenamiento.

Durante el entrenamiento se realizaron simulaciones, obteniendo las recompensas con el Algoritmo 1, para evaluar y ajustar las constantes y los hiper-parámetros.

Tabla 10.1: Hiper-parámetros entrenamiento controlador velocidad angular, ρ es una variable que corresponde al progreso del entrenamiento (de 1 a 0).

Hiper-parámetro	Valor
Número de entornos	16
Epochs	5
Steps por entorno	256
Minibach	32
Learning rate	ρ*1e-4
Discount (γ)	0,99
GAE parameter (λ)	0,95

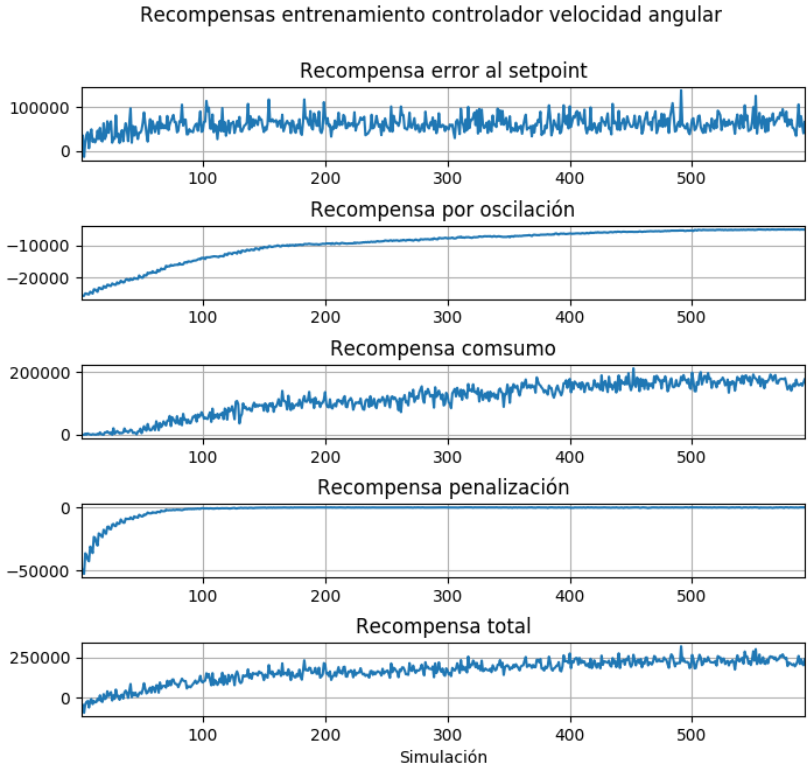


Figura 10.2: Recompensas obtenidas en simulaciones realizadas durante el entrenamiento de red neuronal, capaz de controlar la velocidad angular.

En la Figura 10.2, se muestran las recompensas obtenidas del modelo final. Las constantes obtenidas fueron $\beta = 100$, $\gamma = 1000$ y $PENALTY = 100$.

Resultados

En la Figura 10.3 se observan las pruebas realizadas con la red entrenada. Las velocidades angulares convergen rápidamente a los setpoints deseados, sin errores

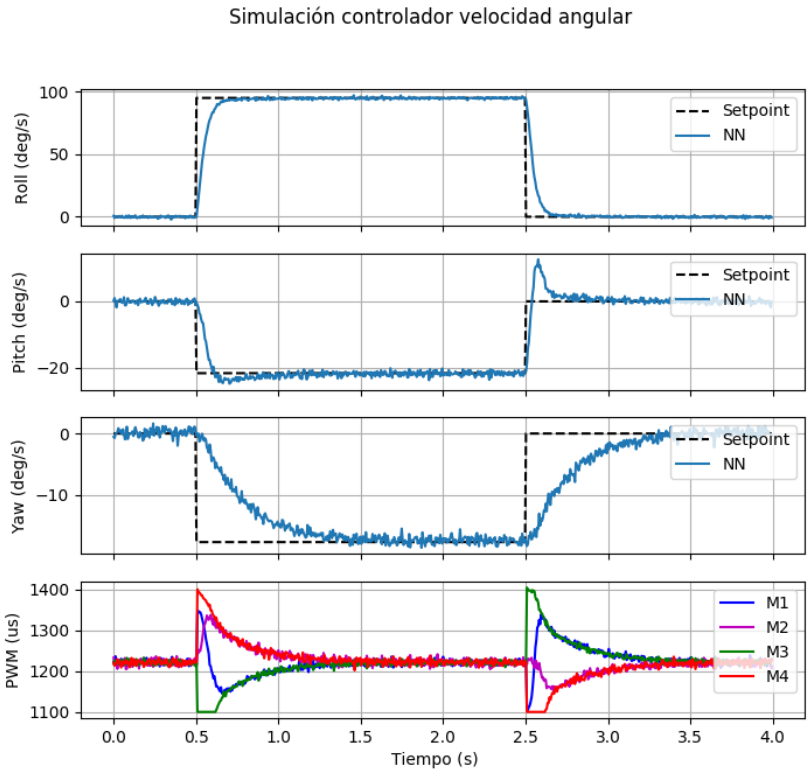


Figura 10.3: Simulación controlador velocidad angular entrenado en entorno de entrenamiento. Las primeras tres gráficas muestran la velocidad angular del drone por curvas continuas azules, el setpoint deseado por las curvas ralladas negras. La última gráfica muestra los valores de PWM aplicado a las ESCs.

asintóticos apreciables en ninguno de los ejes. En algunas simulaciones se observan sobretiros en roll o pitch al aplicar un escalón.

Observando los valores de PWM al inicio de la simulación y en la convergencia a los setpoints, se concluye que se logró entrenar la red para que el consumo fuera bajo, ya que en esos intervalos el valor promedio de las PWM es de 1200 μs.

Respecto a las oscilaciones de los motores, se observa un buen comportamiento en toda la simulación excepto en los tiempos que se aplica el escalón a la entrada ($t = 500$ ms y $t = 2500$ ms). En estos tiempos se tiene una discontinuidad a la salida que no se pudo corregir en los entrenamientos. Ya que esta discontinuidad se produce en un intervalo de tiempo muy corto, se concluyó que no debería afectar la vida útil de los motores, pero por precaución se deberá prestar atención en los valores de PWM y temperatura de los motores en las primeras pruebas con el sistema real.

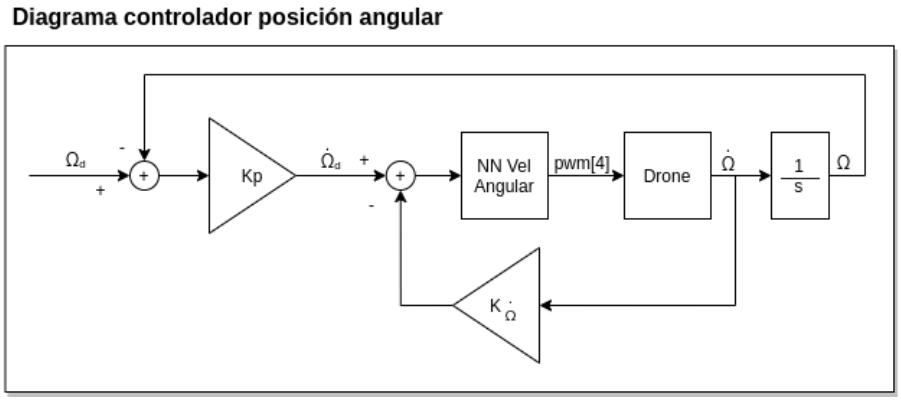


Figura 10.4: Diagrama de bloques control de orientación con controlador proporcional, junto a red neuronal de control de velocidad angular.

10.2.2. Control posición angular

Finalmente, para controlar la posición angular, se implementó un controlador proporcional como se muestra en la Figura 10.4. Allí, el bloque K_P es un proporcional, el cual recibe como entrada el error a un setpoint de orientación y a la velocidad angular deseada para que el drone converja al setpoint. El bloque "NN Vel Angular" es la red neuronal mencionada anteriormente.

El bloque $K_{\dot{\Omega}}$ es un proporcional en la realimentación de la velocidad angular. Los motivos por los que se agregó este proporcional se verán en la Sección 15.2.

El error de orientación en yaw (e_{o_ϕ}) se calcula de forma tal de obtener el camino más corto a este error. Como esta medida está en radianes entre $-\pi$ y $-\pi$ se calcula de la forma:

$$e_{o_\phi}(t) = mod(r_{o_\phi}(t) - y_{o_\phi}(t) + \pi, 2\pi) - \pi. \tag{10.4}$$

Los parámetros de roll y pitch no se utilizan en esta fórmula, dado que al hacerlo permitiría que el drone realice giros completos en estos ejes, los cuales podrían causar que choque contra el piso.

10.3. Controlador de velocidad en el eje z

Para controlar la altitud del drone se diseña otra red neuronal que dado el estado del drone devuelva los valores de PWM a aplicar a las ESCs, para que pueda alcanzar una velocidad en z deseada.

10.3.1. Arquitectura de la red neuronal

Como entrada a la red se utiliza el mismo principio que para el controlador de velocidad angular, simplificando la entrada a un vector de dimensión 4, donde sus primeros dos elementos son los valores en roll y en pitch del drone respectivamente,

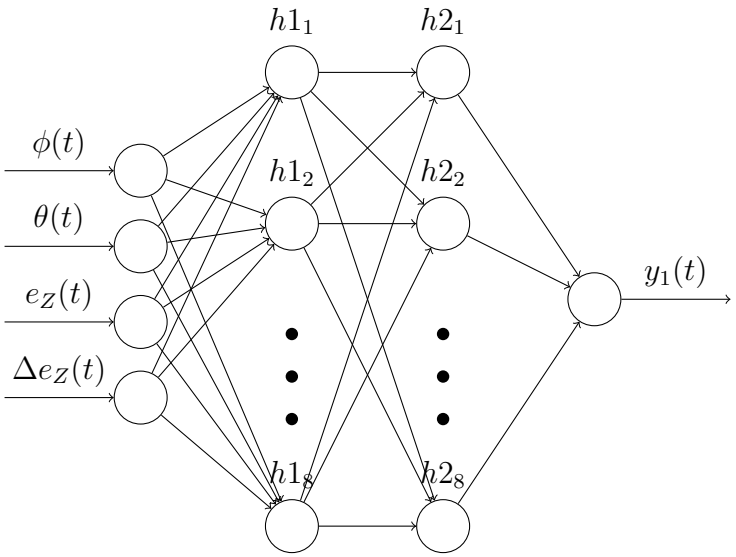


Figura 10.5: Arquitectura red neuronal controlador velocidad en eje z.

el tercer valor es el error al setpoint deseado y el último la diferencia del error al setpoint en t y $t-1$:

$$(\phi(t), \theta(t), e_z(t), e_z(t) - e_z(t-1)). \tag{10.5}$$

Dado que para controlar la altura sin alterar la posición angular todos los motores deben girar a la misma velocidad, la dimensión de la salida de la red es 1, y toma valores entre cero y uno correspondientes a todos los motores apagados y todos girando a su máxima velocidad respectivamente.

Para las capas ocultas se optó por dos capas de 8 neuronas cada una tal como muestra la Figura 10.5.

10.3.2. Entrenamiento

Para entrenar esta red se programó un nuevo entorno, en el cual se simuló por 5 segundos el drone, con pasos de 1 ms, partiendo desde una velocidad en z de 0 m/s, y una orientación en pitch y roll aleatoria entre valores de $-55°$ y $+55°$. Al reiniciar la simulación, el setpoint z^* se fija en un valor aleatorio entre
$[-3$ m/s, $+3$ m/s].

Dado que el drone debe ser capaz de mantener una velocidad en z independientemente de la posición angular que se encuentre, además de iniciar en una orientación aleatoria, al realizar el entrenamiento se convierte la salida de la red a un vector de dimensión cuatro y se le suma un ruido Gaussiano de media nula y desviación estándar 0,001, a cada componente. De esta forma se permite que el drone cambie de orientación de forma aleatoria durante la simulación, para que el agente sea capaz de controlar la velocidad sin importar la orientación del mismo.

La función de recompensa se obtiene a partir del Algoritmo 2, donde en cada paso de simulación, se obtiene la recompensa como el error acumulado a lo largo de la simulación filtrado por un filtro pasa bajos (línea 2). Para lograr una convergencia rápida, se aplica una recompensa por bajo consumo solo si el error en z es menor a un metro (líneas 3 y 4) y una recompensa por bajas oscilaciones en los motores solo si el error actual es menor a un valor proporcional a z^* (líneas 5 y 6).

Algoritmo 2: Función de recompensa del controlador velocidad en el eje z

Result: Recompensa r en el tiempo t

1 $r \leftarrow 0$
2 $r \leftarrow -[0{,}2 \times e_z(t) + 0{,}8 \times \sum_{k=0}^{t-1} |e_z(K)|]$
3 **if** $|e_z(t)| \leq 0{,}1$ **then**
4 $\quad r \leftarrow r - \beta|y(t) - y(t-1)|$
5 **if** $|e_z(t)| \leq 0{,}15|z^*|$ **then**
6 $\quad r \leftarrow r + \gamma(1 - y(t))$

Se realizaron entrenamientos de un total de 2 500 000 pasos para buscar las contantes e hiper-parámetros de entrenamiento, para que el mismo converja a un solución capaz de cumplir con los objetivos, obteniendo $\beta = 0{,}01$, $\gamma = 0{,}001$ y los hiper-parámetros, que se muestran en la Tabla 10.2, como buenos candidatos.

Tabla 10.2: Hiper-parámetros entrenamiento controlador velocidad en eje z.

Hiper-parámetro	Valor
Steps	2 500 000
Epochs	4
Minibach size	64
Adam epsilon	1e-5
Discount (γ)	0,99
GAE parameter (λ)	0,95

10.3.3. Resultados

Para validar el entrenamiento se observó el resultado de distintas simulaciones. Las Figuras 10.6 y 10.7 muestran los resultados de dos de las simulaciones realizadas.

Se concluye que el entrenamiento fue satisfactorio, ya que el mismo es capaz de llegar a la velocidad deseada con un error asintótico y consumo bajo.

Aún falta realizar pruebas con el drone corriendo este algoritmo, para validar completamente el entrenamiento.

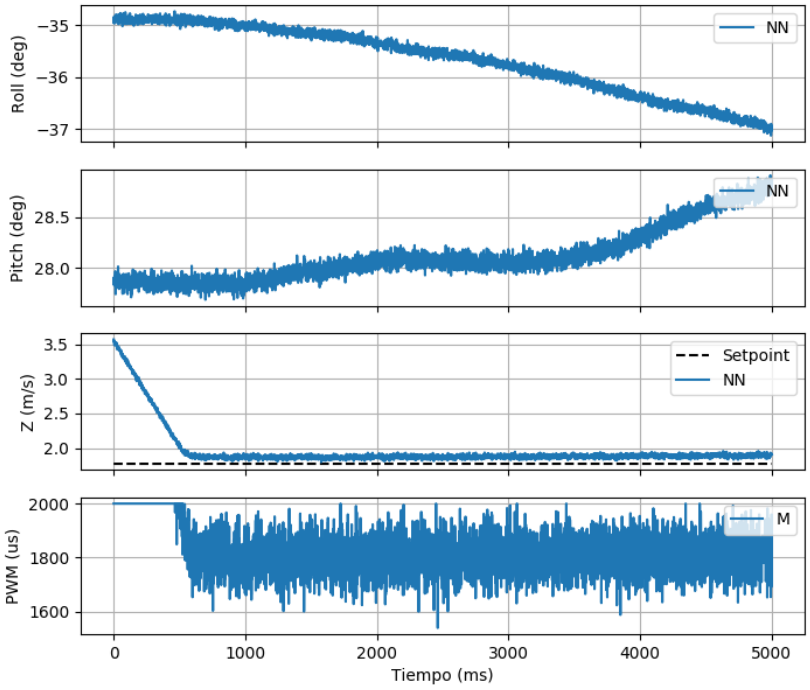


Figura 10.6: Simulación controlador velocidad en eje z.

10.4. Throttle Mixing

Para realizar el mixing de las salidas de las redes, se utiliza lo implementado en Neuroflight[19]. Para esto, primero se limita la salida del controlador de velocidad en el eje z para darle prioridad al otro controlador, redefiniéndola como:

$$\hat{y_z}(t) = y_z(t) \times (1 - max_i(y_{ang_i}(t))) \tag{10.6}$$

donde $y_z(t)$ es la salida de la red del controlador de velocidad en z e $y_{ang}(t)$ la salida del controlador de velocidad angular.

Finalmente, la salida combinada se define como:

$$y_i(t) = \hat{y_z}(t) + y_{ang_i}(t). \tag{10.7}$$

Se obtiene una salida que toma valores entre 0 y 1, los cuales se pasan a valores de PWM, definiendo el valor de PWM del motor i como:

$$PWM_{M_i}(t) = 900 \times y_i(t) + 1100. \tag{10.8}$$

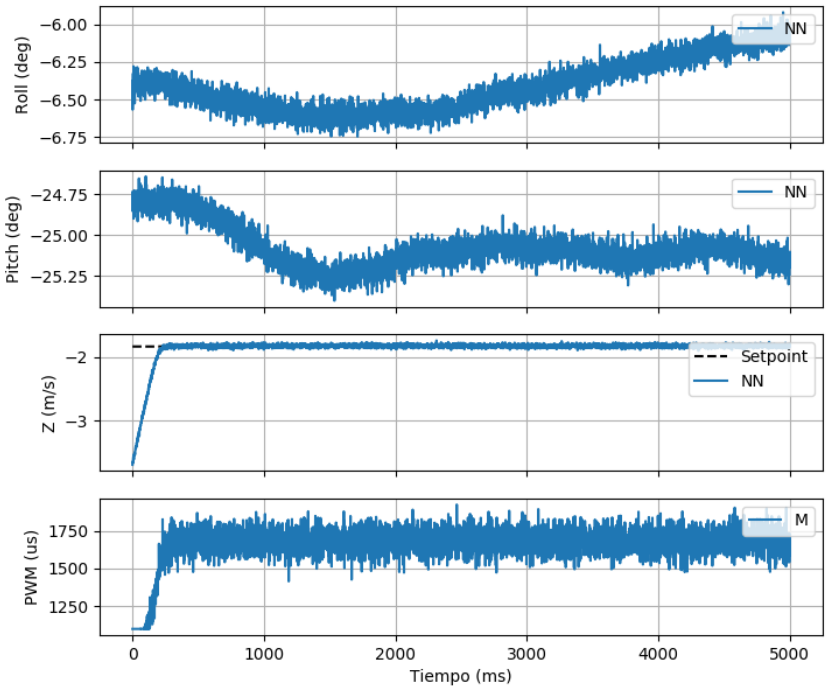


Figura 10.7: Simulación controlador velocidad en eje z.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 11

Controlador de Posición

11.1. Introducción

Para poder seguir trayectorias es necesario poder controlar la ubicación del drone, por este motivo se desarrolló un segundo controlador que llamaremos "controlador de posición". Este se encarga de obtener el estado angular y velocidad en z deseada (entradas del controlador de actitud), necesarios para llegar a un determinada posición.

Por compatibilidad con otros paquetes de ROS, se optó por definir como entrada al controlador, la velocidad en x relativa al drone, la velocidad en z y la velocidad en yaw.

Debido al ruido en la medida de velocidad y errores asintóticos en los controladores basados en redes neuronales, al controlar solamente la velocidad, no se podría entrar en un estado de hovering (estar quieto en una posición), por lo cual si la velocidad en x se desea que sea cero, se utiliza un sub controlador interno que se encarga de mantener la última posición en x e y desde que se mandó cero en alguno de estos ejes. En el caso de la velocidad deseada en z sea cero, se utiliza una lógica similar.

En la Figura 11.1 se muestra el diagrama implementado, el cual posee dos ramas, una que genera la orientación y otra la velocidad en z deseada.

11.2. Generación estado angular

El estado angular se determina a partir de la velocidad en x deseada y la velocidad en yaw deseada.

La orientación en yaw, es independiente al control de velocidad en x, pudiendo obtenerse multiplicando la velocidad en yaw deseada (w_{d_ψ}) por una constante (K_ψ) más la orientación actual en este eje (ψ). Es decir:

$$\psi_d(t) = K_\psi \times w_{d_\psi}(t-1) + \psi(t-1). \tag{11.1}$$

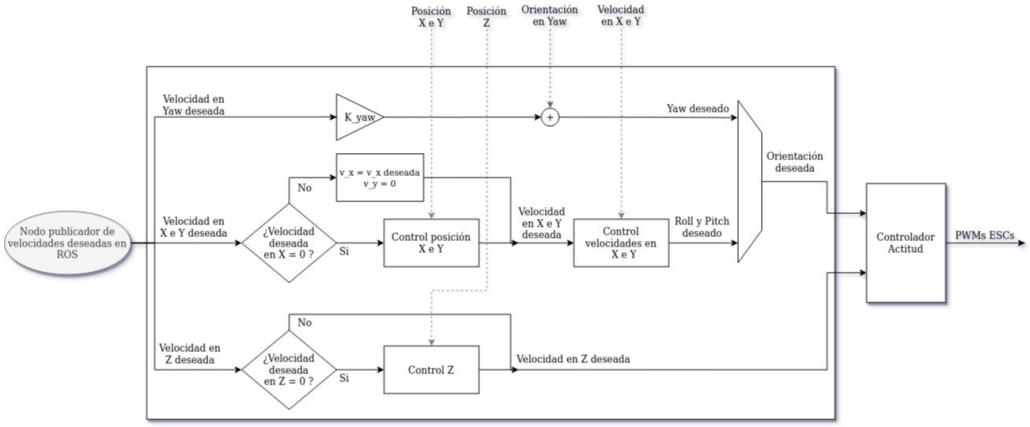


Figura 11.1: Diagrama de bloques controlador de posición.

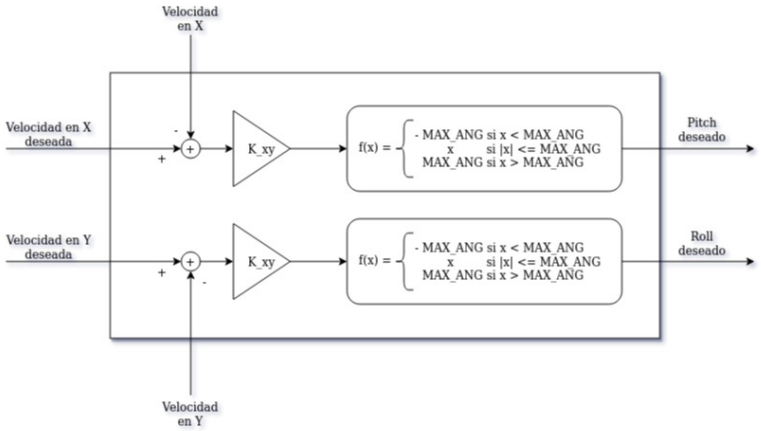


Figura 11.2: Diagrama de bloques controlador de velocidades en x e y.

Para que el drone se mueva a una velocidad constante en un eje relativo al mismo (x o y), se puede variar el pitch en caso del eje x y el roll en caso de y, obteniendo una fuerza. La variación necesaria se obtiene a partir de un controlador proporcional para cada eje, amplificando el error a la velocidad deseada y la actual. El diagrama de la implementación se muestra en la Figura 11.2.

Dada la simetría del drone en estos ejes, se utilizan proporcionales con el mismo valor y para evitar que alcance ángulos muy empinados (provocando que no se pueda controlar la altura), se limitan estos estados.

Cuando el controlador recibe que la velocidad deseada en x es cero, se asume que se busca estar quieto en ese punto, activando un controlador que genera las velocidades en x e y necesarias para mantenerse cercano a ese punto. Para eso se utiliza un controlador proporcional, con la misma estructura que el controlador anterior, sustituyendo velocidades por posiciones.

11.3. Generación velocidad en z deseada

En caso de que se busque tener una velocidad en z distinto de cero, ya que el controlador de actitud está diseñado para recibir esta entrada, la salida del controlador de posición es la misma que la entrada.

En caso de ser cero, utilizando la idea del controlador anterior para cuando las velocidades son 0, se utiliza un controlador proporcional para mantenerse a una altura fija.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 12

Autonomía de Vuelo

12.1. Planificación de trayectorias

Para la planeación de trayectorias, se siguió en la línea de lo que implementó uQuad3 [3], utilizando la teoría de Dubins [6] ya mencionada en el Capítulo 2.

Siguiendo la teoría de Dubins, para determinar qué tipo de curva es la que minimiza el recorrido dado un par de waypoints, se tendrían que calcular todas las posibles curvas y luego ver cuál es la de menor longitud. Sin embargo, mediante el trabajo de Shkel y Lumelsky [7] se evita tener que calcular las seis posibilidades de curvas y luego tener que elegir la de menor trayectoria. Además, se menciona que si la distancia entre dos puntos es mayor a cuatro veces el radio de curvatura, entonces la solución es una curva de la forma CSC.

A partir de estos tres autores, es que se determinan los cálculos para poder hallar la curva mínima entre un par de waypoints y su orientación.

Primero se normaliza la distancia entre dos puntos, de ahora en más denominados waypoints, tomando el radio r como unidad. Es decir

$$d = \frac{D}{r} \tag{12.1}$$

donde D es la distancia real entre los dos waypoints.

Luego, para cada trayectoria L, S y R, existe una transformación en $\mathbb{R}^3$ que mapea un waypoint inicial (x, y, ϕ) en $\mathbb{R}^3$ obteniendo su imagen en $\mathbb{R}^3$:

$$\begin{cases} L_v(x,y,\phi) = (x + \sin(\phi+v) - \sin\phi, y - \cos(\phi+v) + \cos\phi, \phi+v) \\ R_v(x,y,\phi) = (x - \sin(\phi-v) + \sin\phi, y + \cos(\phi-v) - \cos\phi, \phi-v) \\ S_v(x,y,\phi) = (x + v\cos(\phi) + v\sin\phi, \phi) \end{cases} \tag{12.2}$$

donde v representa el largo de la curva S o C.

En el sistema de coordenadas elegido, la configuración inicial de una trayectoria que comienza en $(0,0,\alpha)$ y termina en $(d,0,\beta)$ y es de la forma R, S, L con longitudes t, p y q respectivamente debe terminar en

$$L_q(S_p(R_t(0,0,\alpha))) = (d,0,\beta) \tag{12.3}$$

y tendrá una longitud igual a la suma de las longitudes de las curvas R, S y L:

$$Largo = t + p + q. \tag{12.4}$$

A partir de la ecuación (12.2) se pueden obtener las distintas opciones de trayectorias mencionadas previamente.

1. $L_q(S_p(L_t(0,0,\alpha))) = (d,0,\beta)$

$$\begin{cases} p\cos(\alpha+t) - \sin(\alpha) + \sin(\beta) = d \\ p\sin(\alpha+t) + \cos(\alpha) - \cos(\beta) = 0 \\ \alpha + t + q = \beta \end{cases} \tag{12.5}$$

cuya solución es

$$\begin{cases} t_{lsl} = -\alpha + \arctan(\frac{\cos\beta - \cos\alpha}{d + \sin\alpha - \sin\beta}) \\ p_{lsl} = \sqrt{2 + d^2 - 2\cos(\alpha-\beta) + 2d(\sin\alpha - \sin\beta)} \\ q_{lsl} = \beta - \arctan(\frac{\cos\beta - \cos\alpha}{d + \sin\alpha - \sin\beta}) \end{cases} . \tag{12.6}$$

De la misma forma se llega a los resultados de las siguientes curvas:

2. $R_q(S_p(R_t(0,0,\alpha))) = (d,0,\beta)$

$$\begin{cases} t_{rsr} = \alpha - \arctan(\frac{\cos\alpha - \cos\beta}{d - \sin\alpha + \sin\beta}) \\ p_{rsr} = \sqrt{2 + d^2 - 2\cos(\alpha-\beta) + 2d(\sin\beta - \sin\alpha)} \\ q_{rsr} = -\beta + \arctan(\frac{\cos\alpha - \cos\beta}{d - \sin\alpha + \sin\beta}) \end{cases} \tag{12.7}$$

3. $R_q(S_p(L_t(0,0,\alpha))) = (d,0,\beta)$

$$\begin{cases} t_{lsr} = -\alpha + \arctan(\frac{-\cos\alpha - \cos\beta}{d + \sin\alpha + \sin\beta}) - \arctan(\frac{-2}{p_{lsr}}) \\ p_{lsr} = \sqrt{-2 + d^2 + 2\cos(\alpha-\beta) + 2d(\sin\alpha + \sin\beta)} \\ q_{lsr} = -\beta + \arctan(\frac{-\cos\alpha - \cos\beta}{d + \sin\alpha + \sin\beta}) - \arctan(\frac{-2}{p_{lsr}}) \end{cases} \tag{12.8}$$

4. $L_q(S_p(R_t(0,0,\alpha))) = (d,0,\beta)$

$$\begin{cases} t_{rsl} = \alpha - \arctan(\frac{\cos\alpha + \cos\beta}{d - \sin\alpha - \sin\beta}) + \arctan(\frac{2}{p_{rsl}}) \\ p_{rsl} = \sqrt{-2 + d^2 + 2\cos(\alpha-\beta)} - 2d(\sin\alpha + \sin\beta) \\ q_{rsl} = \beta - \arctan(\frac{\cos\alpha + \cos\beta}{d - \sin\alpha - \sin\beta}) + \arctan(\frac{2}{p_{rsl}}) \end{cases} \tag{12.9}$$

Luego, si se divide el eje cartesiano en cuatro cuadrantes, se pasan a ubicar las orientaciones iniciales y finales α y β para poder determinar qué curva es la que corresponde utilizar sin tener que calcular todas las posibilidades.

Final Quadrant / Initial Quadrant	1	2	3	4
1	RSL	if $S_{12} < 0$ then RSR if $S_{12} > 0$ then RSL	if $S_{13} < 0$ then RSR if $S_{13} > 0$ then LSR	if $S_{14}^1 > 0$ then LSR if $S_{14}^2 > 0$ then RSL else RSR
2	if $S_{21} < 0$ then LSL if $S_{21} > 0$ then RSL	if $S_{22}^1 < 0$ then LSL if $S_{22}^1 > 0$ then RSL if $S_{22}^2 < 0$ then RSR if $S_{22}^2 > 0$ then RSL	RSR	if $S_{24} < 0$ then RSR if $S_{24} > 0$ then RSL
3	if $S_{31} < 0$ then LSL if $S_{31} > 0$ then LSR	LSL	if $S_{33}^1 < 0$ then RSR if $S_{33}^1 > 0$ then LSR if $S_{33}^2 < 0$ then LSL if $S_{33}^2 > 0$ then LSR	if $S_{34} < 0$ then RSR if $S_{34} > 0$ then LSR
4	if $S_{41}^1 > 0$ then RSL if $S_{41}^2 > 0$ then LSR else LSL	if $S_{42} < 0$ then LSL if $S_{42} > 0$ then RSL	if $S_{43} < 0$ then LSL if $S_{43} > 0$ then LSR	LSR

Figura 12.1: Tabla de decisión sobre curva mínima.[7]

La Figura 12.1 muestra las curvas que minimizan la trayectoria según la ubicación de las orientaciones inicial y final que definieron Shkel y Lulinsky, de donde

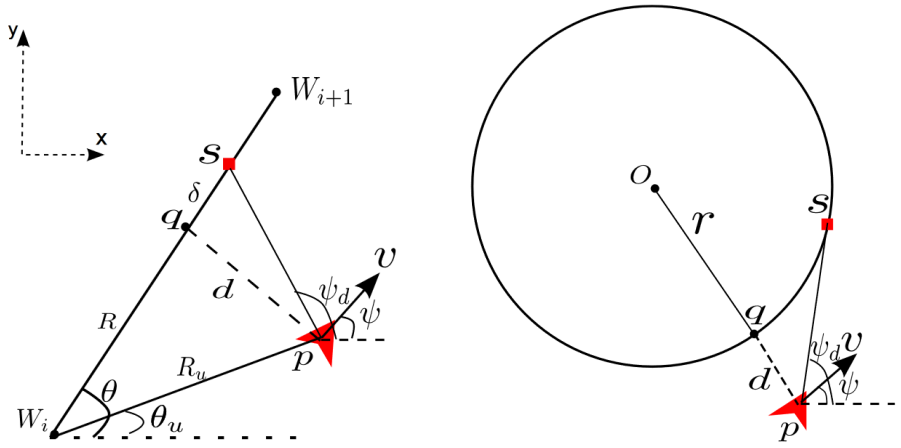
$$\begin{cases} S_{12} = p_{rsr} - prsl - 2(q_{rsl} - \pi) \\ S_{13} = t_{rsr} - \pi \\ S_{14}^1 = t_{rsr} - \pi \quad S_{14}^2 = t_{rsr} - \pi \\ S_{21} = p_{lsl} - prsl - 2(t_{rsl} - \pi) \\ \begin{cases} \text{if } \alpha > \beta, \quad S_{22}^1 = p_{lsl} - prsl - 2(t_{rsl} - \pi) \\ \text{if } \alpha < \beta, \quad S_{22}^2 = p_{rsr} - prsl - 2(q_{rsl} - \pi) \end{cases} \\ S_{24} = q_{rsr} - \pi \\ S_{31} = q_{lsl} - \pi \\ \begin{cases} \text{if } \alpha > \beta, \quad S_{33}^1 = p_{rsr} - plsr - 2(q_{lsr} - \pi) \\ \text{if } \alpha < \beta, \quad S_{33}^2 1 =_{lsl} - plsr - 2(t_{lsr} - \pi) \end{cases} \\ S_{34} = p_{rsr} - plsr - 2(t_{lsr} - \pi) \\ S_{41}^1 = t_{lsl} - \pi \quad S_{41}^2 = q_{lsl} - \pi \\ S_{42}^1 = t_{lsl} - \pi \\ S_{12} = p_{lsl} - plsr - 2(q_{lsr} - \pi) \end{cases} . \tag{12.10}$$

Con esta información queda definida la curva que minimiza el recorrido entre dos waypoints. Luego resta concatenar todas las curvas para obtener la trayectoria del drone en una misión que contemple más de dos waypoints.

12.2. Seguimiento de trayectorias

El algoritmo elegido para seguir las curvas de Dubins obtenidas, es el "Carrot Chasing" de Sousa y Saripali[8].

Dada una trayectoria, la localización del drone $p = (x, y)$ y el yaw (ψ), el seguimiento se basa en obtener un yaw deseado (ψ_d) de forma tal que el drone se dirija al waypoint siguiendo la trayectoria y evitando movimientos bruscos. Ya



(a) Seguimiento de línea. (b) Seguimiento arco de circunferencia.

Figura 12.2: Carrot chasing.[8]

que la trayectoria está compuesta por rectas y arcos de circunferencia, se utiliza un algoritmo distinto dependiendo del tipo de curva.

Los algoritmos planteados a continuación se basan en un movimiento en el plano, para extenderlo a 3 dimensiones haciendo que la altura objetivo del drone sea:

$$z_d(p) = z_i + (z_{i+1} - z_i)p \tag{12.11}$$

donde $p \in [0, 1]$ es el progreso de la distancia recorrida sobre la distancia total entre waypoints en el plano, z_i y z_{i+1} la altura en el waypoint inicial y final de la curva que se está siguiendo respectivamente.

12.2.1. Seguimiento de línea recta

Dada una recta se definen $W_i = (x_i, y_i)$ y $W_{i+1} = (x_{i+1}, y_{i+1})$ como los waypoints de inicio y fin de la recta y el punto auxiliar s como el objetivo del drone (ver Figura 12.2(a)), de forma tal que si el drone sigue este punto a lo largo del tiempo, se termina pasando por el waypoint W_{i+1}.

Definiéndose s como un punto de la recta obtenido a partir de sumar una distancia δ al punto q, tal que q es la proyección de p en la recta.

Algoritmo 3: Seguimiento de línea recta.

Result: Yaw deseado (ψ_d)

1 $R_u \leftarrow ||W_i - p||$, $\theta \leftarrow \arctan(y_{i+1} - y_i, x_{i+1} - x_i)^2$

2 $\theta_u \leftarrow \arctan^2(y - y_i, x - x_i)$, $\beta \leftarrow \theta - \theta_u$

3 $R \leftarrow \sqrt{R_u^2 - (R_u \sin\beta)^2}$

4 $s = (x'_t, y'_t) \leftarrow ((R + \delta)\cos\theta, (R + \delta)\sin\theta)$

5 $\psi_d \leftarrow \arctan(y'_t - y, x'_t - x)$

12.2.2. Seguimiento de arco de circunferencia

Para un arco de circunferencia se define $O = (x_t, y_t)$ como el centro de la circunferencia y r su radio. Como en el caso de la recta, se utiliza el punto auxiliar s (Figura 12.2(b)) perteneciente a la circunferencia, tal que s es el punto q rotado un ángulo λ respecto al centro de la circunferencia.

Algoritmo 4: Seguimiento de curva.

Result: Yaw deseado (ψ_d)

1 $d \leftarrow ||O - p|| - r$
2 $\theta \leftarrow \arctan^2(y - y_t, x - x_t)$
3 $s = (x'_t, y'_t) \leftarrow (r\cos(\theta + \lambda), r\sin(\theta + \lambda))$
4 $\psi_d \leftarrow \arctan^2(y'_t - y, x'_t - x)$

Una vez obtenido ψ_d se obtiene el $\dot{\psi_d} = k(\psi_d - \psi)$, donde $\dot{\psi_d}$ es la velocidad deseada en yaw y k una constante de control para ajustar la velocidad de giro de forma de evitar movimientos bruscos o una convergencia lenta.

12.3. Evasión de obstáculos

12.3.1. Introducción

Dentro de la familia de los algoritmos de navegación, existen los algoritmos de navegación reactivos. En estos el entorno no es conocido, y su función está limitada a crear un camino para que el robot llegue a un punto de destino, independientemente de que este camino sea o no el óptimo. La ventaja de estos algoritmos es su simplicidad.

El estado del arte de esta familia de algoritmos fue alcanzada por los algoritmos Bug, para planificación de trayectorias locales. Los Bugs se basan en tres supuestos [21]:

1. El robot es considerado un punto en el espacio.
2. El robot conoce su estado, en especial su localización.
3. El robot tiene sensores ideales.

Aun sabiendo que no se cumplen los supuestos ideales, los algoritmos consiguen buenos resultados y son considerados como un primer paso hacia la solución al problema del cumplimiento de misiones. Los más usados en robots móviles son el Bug1, Bug2 y Distbug; siendo el último el que presenta mejor desempeño [22].

12.3.2. El algoritmo DistBug

El algoritmo Distbug[23] fue diseñado para hacer posible la navegación de un robot en un plano bidimensional desconocido, con obstáculos estáticos de cualquier forma. Idealmente el robot debe ser equipado con un sensor de rango en toda una circunferencia con un alcance de lectura igual a R.

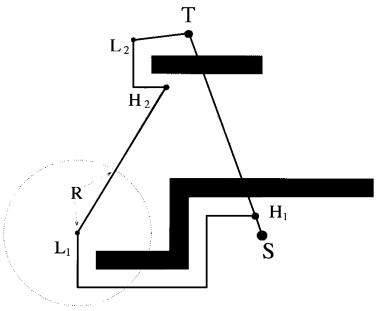


Figura 12.3: Puntos notables del Distbug. [21]

El algoritmo se basa en el uso de dos tipos de movimiento, uno hacia el destino, y otro que bordea el obstáculo, y sigue la siguiente estructura (tomar la Figura 12.3 como referencia):

1. El robot inicia su movimiento directamente hacia el objetivo hasta que uno de los siguientes eventos sucedan:

   a) El objetivo es alcanzado. El robot se detiene y se sale del loop.

   b) Un obstáculo es alcanzado, a ese lugar del plano se le llamará "hit-point". En este caso ir al punto 2 del algoritmo.

2. Llegado al hitpoint se guarda en memoria la distancia mínima hacia el objetivo $d_{min}(T)$ siendo T el punto objetivo. Se inicia la etapa de bordeado del obstáculo, mientras se tendrá en cuenta el espacio libre F que es la distancia desde la posición actual X hasta el obstáculo más cercano en dirección al destino T (si no hay obstáculo, $F = R$). El bordeado del obstáculo sigue hasta que uno de los siguientes eventos ocurren:

   a) El objetivo es visible por el robot ($d(X,T) - F \leq 0$). En ese caso se va al punto 1.

   b) Sea conveniente salir del bordeado y dirigirse hacia el objetivo por más que haya otro obstáculo. Eso se cumple si $d(X,T) - F \leq d_{min}(T) - K$ con K una constante predefinida. En este caso se vuelve al punto 1.

   c) El robot cumple una vuelta entera alrededor del obstáculo. En este caso el punto objetivo T no se puede alcanzar. El robot se detiene.

Análisis del algoritmo

Siendo un hecho que el movimiento en línea recta es más rápido y seguro que el bordeado de un obstáculo, la condición de salida del bordeado fue diseñada para ser abandonada apenas la convergencia esté asegurada.

Esta condición de salida se cumple cuando uno de los siguientes términos es verdadero: $d(X,T) - F \leq 0$ o $d(X,T) - F \leq d_{min}(T) - K$.

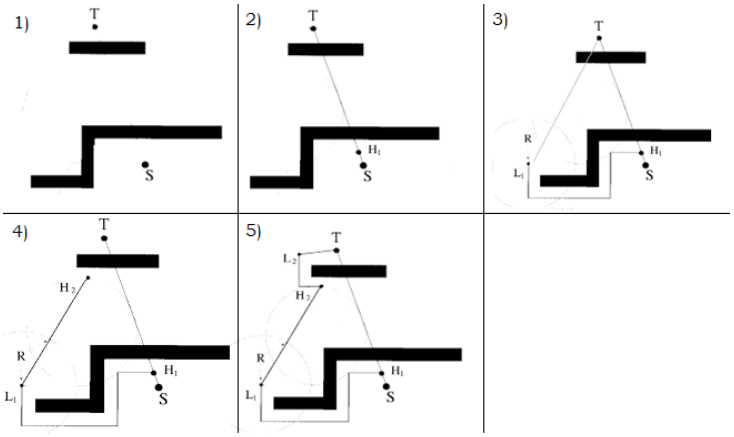


Figura 12.4: Ejemplo de uso del algoritmo. [21]

El primer término es verdadero cuando el objetivo es directamente visible, y en este caso el robot puede ir a su destino directamente.

La segunda condición garantiza que la distancia hacia el objetivo disminuya al menos una distancia K entre dos hitpoints sucesivos, el parámetro K en nuestro caso se elije igual a R.

La convergencia del algoritmo fue estudiada y probada por Ishay Kamon y Ehud Rivlin [21] en los siguientes teoremas:

Lema 1 *Si el punto de destino se puede alcanzar directamente desde un hitpoint, la condición de salida va a hacer que el robot deje de bordear el obstáculo luego de seguir un camino de largo finito.*

Teorema 1 *El algoritmo siempre termina luego de seguir un camino de largo finito.*

Teorema 2 *El algoritmo siempre llega a destino si este es alcanzable.*

Ejemplo de uso del algoritmo

En la Figura 12.4 se plantea un problema de evasión con dos obstáculos y con S como punto de partida y T el de llegada. A continuación se estudia la traza del algoritmo.

1. Se plantea el problema.
2. Se entra en el primer loop. Se dirige en línea recta hacia T hasta encontrar un obstáculo.
3. Luego de encontrado un obstáculo se entra al segundo loop: el bordeado de obstáculo.
4. Se sale del segundo loop cuando se cumple la condición $2b$ ya descrita. Se entra en el primer loop.

5. Se sigue en este hasta encontrar un nuevo obstáculo.
6. Se repite el procedimiento nuevamente hasta llegar a T.

Parte VI

Implementación y Resultados

Capítulo 13

Introducción

13.1. ROS

Para el desarrollo de la inteligencia del drone, se utilizaron principalmente los softwares ROS y Gazebo. ROS[24], del inglés Robotic Operating System, se trata de una plataforma open source enfocada en el desarrollo de software para robótica. Corre en sistema Ubuntu y contiene varios servicios incluyendo abstracción de hardware, control de dispositivos en bajo nivel, comunicación interna entre distintos procesos y manejo de paquetes. También tiene varias herramientas y librerías creadas y mantenidas por la comunidad (Figura 13.1).

Entre sus principales características se destacan:

- Independencia de lenguaje: El sistema es capaz de trabajar con Python y C++.
- Fácil de testear: ROS tiene integrado una unidad de testing `rostest`.
- Liviano: Fue diseñado para ser lo más liviano posible.

La base del funcionamiento del sistema son los denominados nodos, es decir, procesos que realizan cálculos e interactúan con el hardware. Se utiliza un nodo para cada elemento que formará parte del sistema de control del robot, de forma de independizarlos con el resto a nivel de lógica.

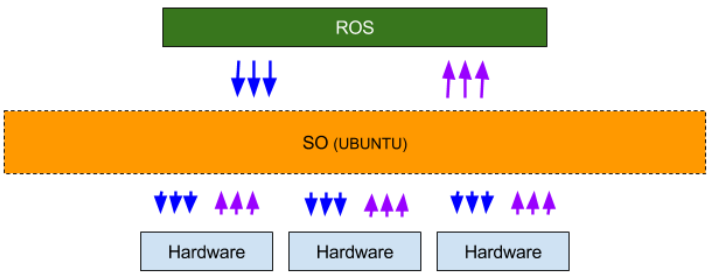


Figura 13.1: Diagrama de comunicación ROS.

Para comunicar nodos entre sí ROS define un protocolo de publicación y suscripción a una cola de mensajes con un formato definido e identificada con un tópico. Uno o varios nodos pueden agregar a la cola nuevos mensajes (publicación), y otros nodos pueden adquirir los nuevos mensajes agregados a la cola de forma independiente (suscripción). La Figura 13.2 muestra en el funcionamiento general de ROS.

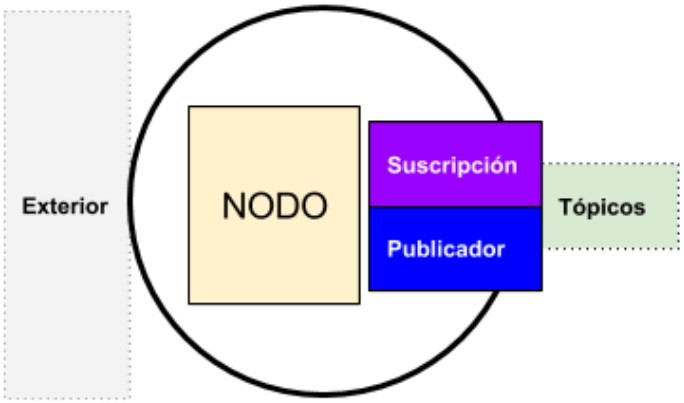


Figura 13.2: Diagrama de funcionamiento de ROS.

Se utilizó ROS y Gazebo para modelar el drone, validar los entrenamientos de la red neuronal que será la inteligencia del mismo, así como también el sistema completo.

Capítulo 14

Sistema Simulado

14.1. Simulador Gazebo

Para realizar las pruebas de las redes neuronales y los distintos algoritmos implementados en ROS, se optó por utilizar un simulador capaz de simular la física del drone y como interacciona con los objetos que podría encontrarse en un escenario real. Debido la emergencia sanitaria mencionada en el Capítulo 2, la simulación del sistema y el entorno constituye una parte importante del proyecto, utilizándola para pre-validar la mayoría de los algoritmos en lugar del sistema físico, antes de hacer los ensayos de Laboratorio.

Se utilizó Gazebo [25], plataforma 3D desarrollada para la simulación de robots en mundos virtuales compatible con ROS. El mismo integra el motor físico ODE, permitiendo simular comportamientos físicos y colisiones. Además, junto al sistema de renderizado OGRE se puede lograr ecosistemas con luces, sombras y texturas de alta calidad. Este se utiliza principalmente para testear robots en entornos controlados y cuenta con librerías de sensores para generar un modelo muy similar al modelo real.

El soporte con ROS que posee permite crear modelos y sensores capaces de publicar y suscribirse a nodos de ROS, de forma tal que el programa diseñado trabaje de forma indiferente e independiente de la simulación, a través de plugins escritos en C++ utilizando la API de Gazebo, los cuales se anexan a los modelos.

14.2. Modelo del drone

Con el objetivo de obtener una simulación que cumpla con los requisitos de esta tesis, se desarrolló un modelo personalizado del drone para Gazebo. Este modelo cuenta con tres partes principales: un plugin de dinámica, un modelo 3D y sensores solidarios al modelo.

El plugin de dinámica crea un nodo de ROS el cual se suscribe a los valores de PWM correspondiente a cada motor, de los cuales utilizando las fórmulas del modelo y la caracterización de los motores descripta en los Capítulos 4 y 6 respectivamente, se obtiene la fuerza y torque que ejercen los motores sobre el drone.

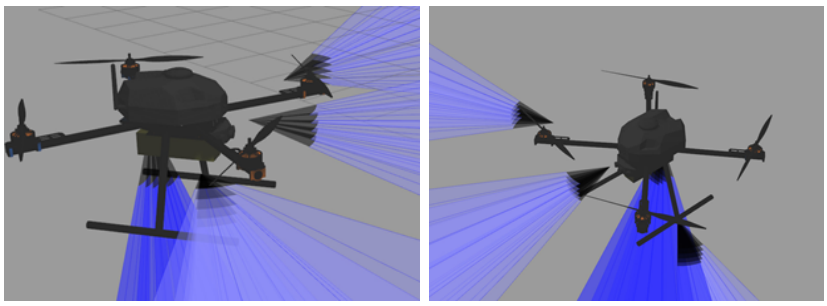


Figura 14.1: Modelo 3D del drone en Gazebo.

Finalmente se le aplica estas fuerzas al modelo y Gazebo se encarga de simular el comportamiento provocado por estas.

El modelo 3D se utiliza para las colisiones del drone con el entorno. En nuestro caso es de importancia para simular el aterrizaje y despegue. Para esto se modeló el drone con programas de modelado 3D, obteniendo el modelo que se observa en la Figura 14.1.

El modelo se completa agregándole los sensores utilizados en el sistema físico al modelo virtual para poder obtener nodos de publicación de las medidas. En este caso se utilizaron plugins proporcionados por Gazebo y el paquete hector_quadrotor [26].

14.3. Estructura de nodos en simulación

La Figura 14.2 muestra el diagrama con los nodos del sistema, sustituyendo los nodos de comunicación con dispositivos del sistema físico como sensores y motores, por nodos auxiliares que interactúan con sus equivalentes en la simulación.

Se separó el sistema en cinco grandes bloques siguiendo la línea de cómo se fue implementando la programación.

- Control de vuelo: está formado por dos nodos, el nodo position_controller que se encarga de controlar la posición del drone. Este nodo implementa los controladores proporcionales y le publica al controlador de actitud los setpoints de velocidades angulares y velocidad en z.

  El nodo controller es el controlador de actitud que realiza el control de las posiciones angulares y la velocidad en z y genera los valores de PWM para las ESCs. Es donde se ejecuta la red neuronal.

- Autonomía de vuelo: el nodo path_planning recibe de la interfaz de usuario un listado con los waypoints que contiene una misión. Allí se procesan y se calculan las distintas subtrayectorias que formarán la trayectoria global de la misión. Estas trayectorias se envían a la máquina de estados en un vector que indica el tipo de curva a seguir como se vio en el Capítulo 12, las posiciones en el plano iniciales y finales para cada trayectoria así como también los ángulos de yaw inicial y final y los radios de curvatura.

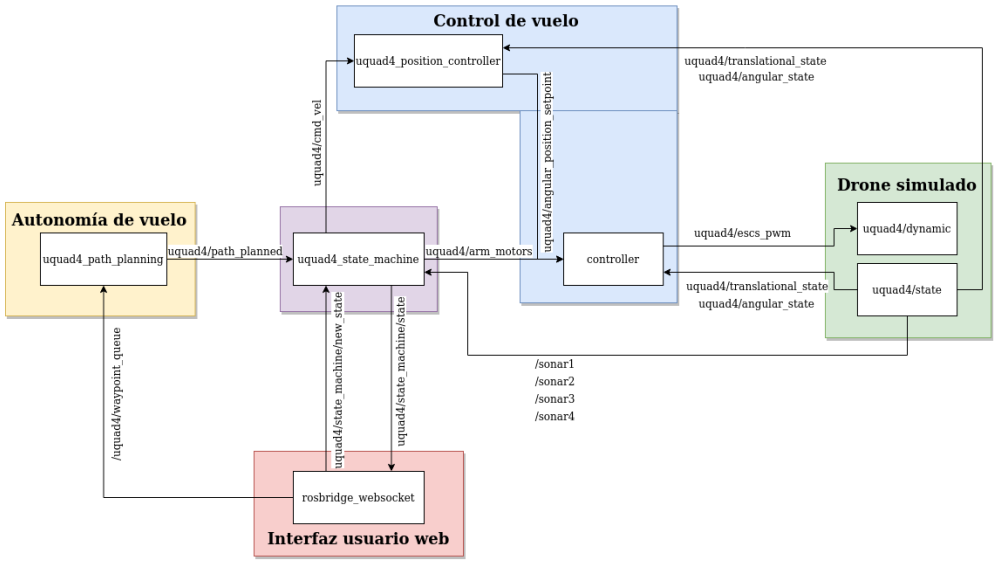


Figura 14.2: Diagrama de nodos de ROS, ejecutado en el sistema simulado.

- Máquina de estados: es la encargada de ejecutar los distintos nodos correspondientes a un vuelo del drone. El estado del drone puede elegirse mediante la interfaz de usuario (se detallan los funcionamientos a nivel de usuario de cada uno de los estados en la Sección 14.6).

  El estado inicial se encarga de armar los motores y se prepara para pasar al estado "landed_state". En el segundo estado, el drone se mantiene en el suelo con los motores armados y parados esperando para recibir la orden de despegue.

  En el estado "taking_off" el drone tiene como objetivo despegar a una altura de dos metros sobre el nivel del piso. Para eso se publica velocidad lineal en z, hasta llegar a la altura deseada. Una vez logrado, se pasa automáticamente al estado de hovering, donde la función principal es mantenerse en el aire de forma estable. Aquí se espera por una de dos opciones: que se genere una misión o que se decida aterrizar.

  En caso de generarse una misión, se ejecuta "mission_state", el estado más importante. Allí se ejecuta el nodo de seguimiento de la trayectoria planeada y, en el caso de detectar un obstáculo, pasar a la ejecución de la evasión.

  Una vez terminada la misión se puede realizar otra, o pasar al estado de aterrizaje en el cual el drone desciende al piso desde la altura que se encuentra al momento de recibir la orden de aterrizaje. En este estado se utiliza el sensor de ultrasonido para poder detectar el suelo e ir disminuyendo la velocidad para no producir un choque.

- Drone simulado: se tienen dos nodos, el de estado que obtiene los estados angular y traslacional del drone, así como también los valores del los sonares en el entorno simulado. El nodo de la dinámica es el que recibe los valores de

PWM a enviar a las ESCs y tiene los modelos de los motores caracterizados con los valores obtenidos a partir de la caracterización de propulsores vista en el Capítulo 6.

- Interfaz usuario web: el nodo rosbridge_websocket entabla un puerto de comunicación utilizando websockets entre el drone y una computadora externa vía web local. Se usa la red WiFi cuando se está cerca. Accediendo desde un navegador se accede a la interfaz web que muestra estos datos y gestiona la comunicación.

La estructura de los paquetes implementados se puede ver más detallada en el Anexo D.

14.4. Control de estabilización en Gazebo

Con el objetivo de probar el comportamiento del drone ante una acción externa, se realizó un experimento en Gazebo que consiste en llevar el drone a un estado de hovering, esperar unos segundos para que se estabilice la posición y una vez estable, se fuerza que la orientación en roll sea 1 radian, dejando correr la simulación hasta estabilizarse nuevamente.

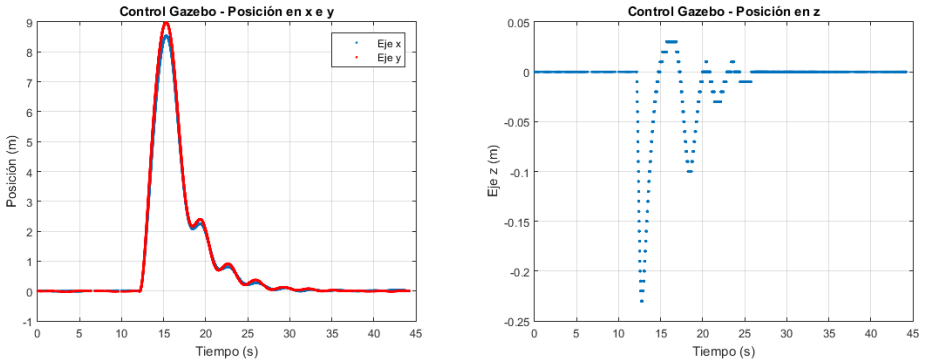
En la Figura 14.3(a) se muestra la posición del drone en función del tiempo en los ejes x e y comenzando desde el origen. Al cambiar su orientación en $t = 12$ s, el drone rápidamente se aleja del origen ya que el controlador de altura compensa la inclinación aumentando la propulsión. Por este motivo, el drone tan solo cae 0,23 metros, como se aprecia en la Figura 14.3(b), ya que por diseño se prioriza el control de altura antes que el de posición.

Una vez llegado al alejamiento máximo en los ejes x e y, se converge al origen con un comportamiento sobre amortiguado, pero deteniéndose en algunos puntos intermedios al intentar reducir la velocidad en estos ejes para llegar al origen con velocidad nula. Este comportamiento no es bueno ya que el drone debería ser capaz de realizar esta maniobra de forma fluida para así lograr una convergencia más rápida y con menor consumo. Modificando las constantes de los proporcionales se debería mejorar el comportamiento, pero se optó por dejarlos de esta manera y corregir estos comportamientos en los experimentos físicos, evitando así sobre ajustar los controladores al modelo simulado y que no funcione en la realidad.

Respecto al control de orientación, a partir de la Figura 14.4 se concluye que se logra volver a la orientación objetivo de forma satisfactoria, observando un comportamiento sobre amortiguado en roll provocado por el controlador de posición. El pitch y yaw permanecen constantes a lo largo del experimento como es esperado, ya que solo se modifica la orientación en roll.

14.5. Simulación del sistema completo

Para presentar los resultados y evaluar el desempeño de los distintos algoritmos trabajando en conjunto, se realizó una misión con y sin obstáculos. Para esto, se



(a) Posición en x e y. (b) Eje z.

Figura 14.3: Control estabilización en Gazebo, comportamiento posición.

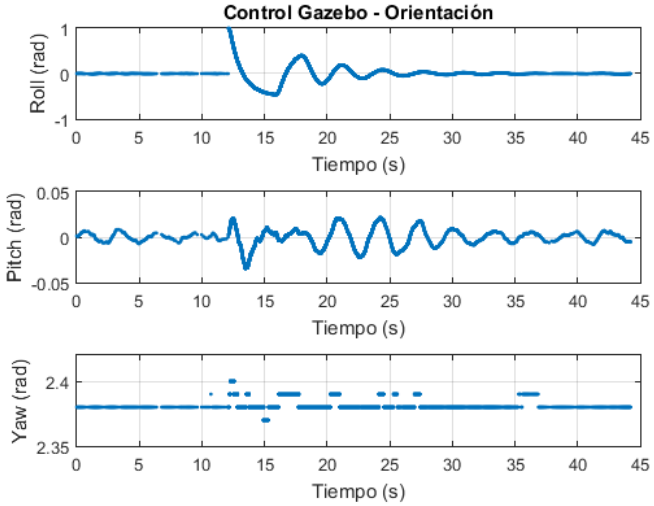


Figura 14.4: Control estabilización en Gazebo, comportamiento orientación.

creó un modelo simplificado y a escala de los edificios de la Facultad de Ingeniería (Figura 14.5). La misión consta de 5 waypoints que se detallan en la Tabla 14.1.

La Figura 14.6 muestra la trayectoria planeada junto con la trayectoria recorrida por el drone realmente. Se observa que en la mayor parte son idénticas, generándose una diferencia al momento de cambiar de orientación para dirigirse al siguiente waypoint. Esto se debe a que la trayectoria planeada genera un "loop" al cambiar de waypoint debiendo pasar por un mismo lugar dos veces. El drone evita ese lazo, pasando una única vez, para luego continuar con la trayectoria.

En la Figura 14.7, la trayectoria realizada por el drone comienza de la misma forma que en el caso sin obstáculos. Una vez que llega al edificio, los sonares detectan que hay un obstáculo frente al drone, dejando de seguir la trayectoria planeada para pasar a evadir el obstáculo con el algoritmo de evasión explicado en la Sección 12.3.2. Una vez que termina de evadir, se retoma la trayectoria planeada

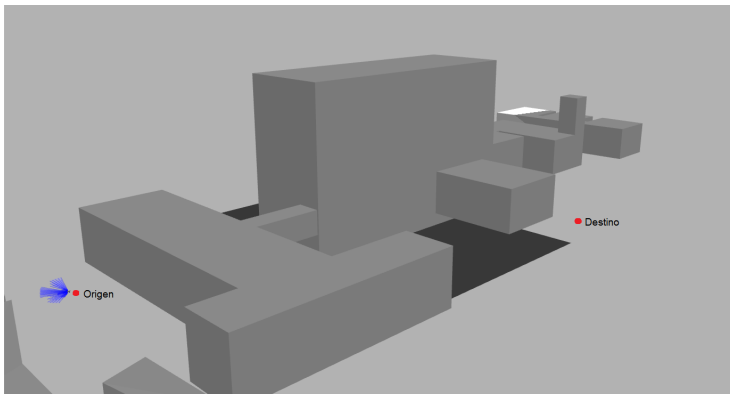


Figura 14.5: Entorno 3D con de la Facultad de Ingeniería en Gazebo, utilizado para la simulación de misión con obstáculos.

Tabla 14.1: Waypoints de misión realizada en la simulación.

Waypoint	x (m)	y (m)	z (m)	yaw (rad)
1	0,0	0,0	2,0	$-2{,}35$
2	$-49{,}0$	22,0	2,0	$-1{,}50$
3	$-29{,}0$	92,0	2,0	3,00
4	58,0	77,00	2,0	0,00
5	71,0	122,0	2,0	0,00

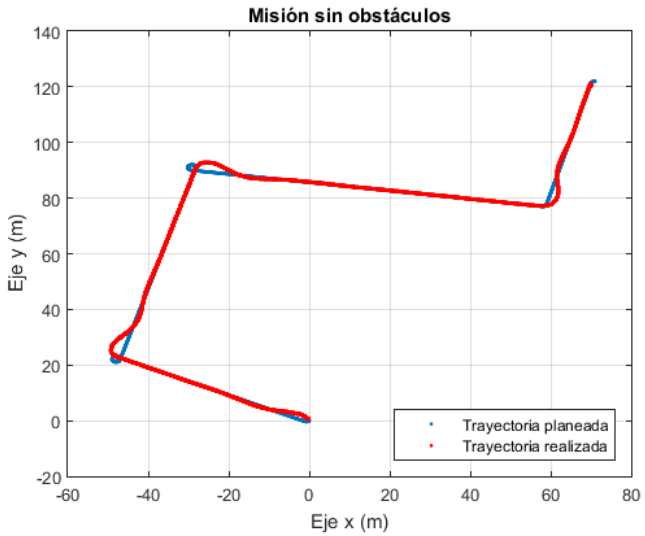


Figura 14.6: Posición del drone, en misión simulada sin obstáculos.

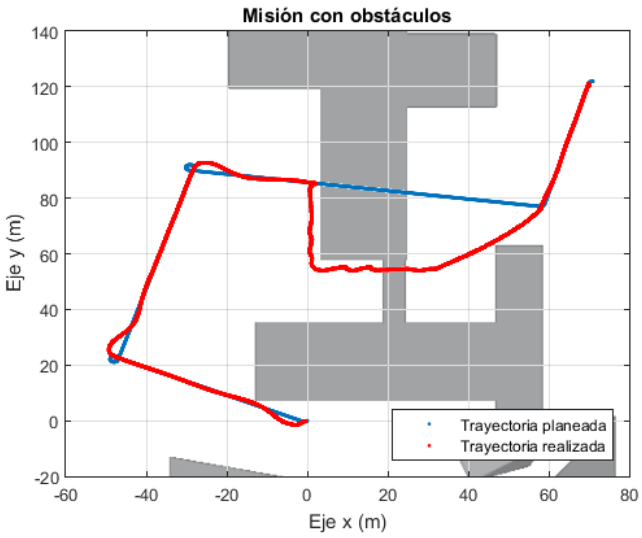


Figura 14.7: Posición del drone, en misión simulada con obstáculos.

para llegar a destino.

14.6. Interfaz de Usuario Web

Se generó una interfaz de usuario para poder controlar el drone y enviarle las misiones. Esta realiza la conexión con ROS y cuenta con las distintas fases de una misión, desde el despegue hasta el aterrizaje. La Figura 14.8 muestra la página principal de la interfaz creada.

Las distintas etapas de comunicación con el drone son:

1. Init: Estado inicial del drone y de la conexión con el mismo.

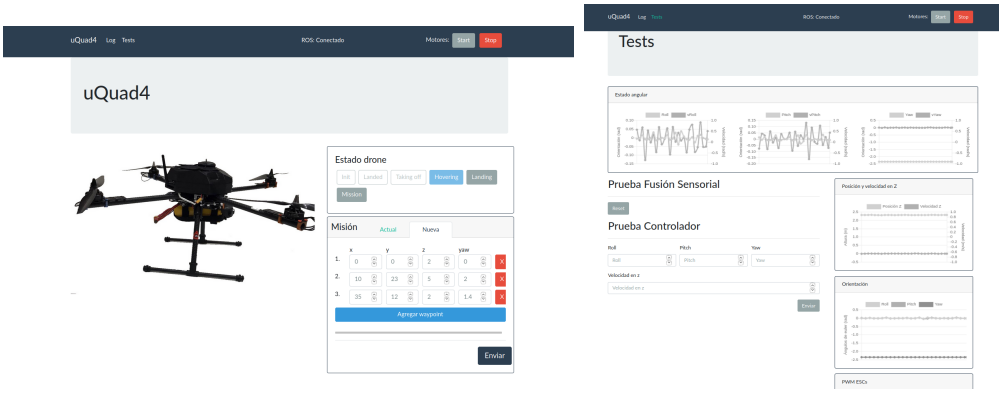
2. Landed: El drone está aterrizado y se arman los motores para luego poder despegar y hacer una misión.

3. Taking off: El drone despega y se posiciona a una altura de 2 metros sobre el nivel del piso.

4. Hovering: Luego del despegue, el drone pasa a estar mantenerse volando en la misma posición gracias al controlador programado.

5. Mission: En esta fase se le puede mandar una misión al drone ingresando los distintos waypoints por los que se quiere que pase. Se ingresa la ubicación en el espacio del waypoint (x, y, z) así como también el ángulo de yaw que se quiere que tome el drone al llegar al objetivo.

6. Landing: Se le da la orden de aterrizaje al drone ya sea luego de una misión o de estar en el estado de hovering.



(a) Página principal (b) Tests

Figura 14.8: Interfaz de usuario web.

Además del envío de la misión, se pueden observar gráficas con los valores de PWM de cada una de las ESC, así como la posición, la velocidad en z y la orientación del drone en tiempo real.

Desde la propia interfaz web, se pueden descargar logs con los datos de interés para luego poder hacer análisis y gráficas según las pruebas realizadas.

Capítulo 15

Sistema Real

15.1. Estructura de nodos en el sistema físico

En el caso del drone real, el diagrama de nodos de la Figura 15.1 se puede agrupar en dos partes, lo que se ejecuta en el Teensy y lo que se corre en el NVidia.

El nodo de la dinámica del drone, está representado por el Teensy que es el que maneja las ESCs. Los valores de PWM son recibidos del NVidia mediante comunicación serial.

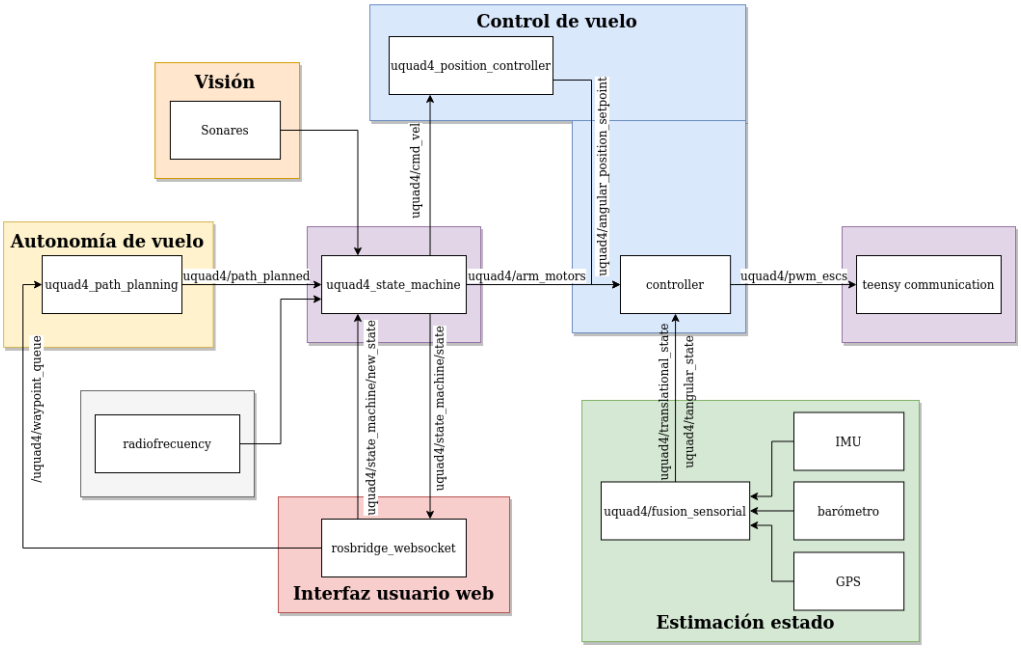


Figura 15.1: Diagrama de nodos de ROS, ejecutado en el sistema real.

El estado traslacional y angular del drone se obtiene por la lectura de los sensores y la posterior fusión sensorial, todo realizado en el NVidia. Luego, el

Figura 15.2: Drone montado en el dispositivo de pruebas del control de actitud.

control y autonomía de vuelo así como la máquina de estados también se ejecutan en el NVidia.

El drone se comunica con el usuario a través de la interfaz al igual que con el modelo simulado.

15.2. Ensayo de control de actitud

Para el testeo del sistema se utiliza un dispositivo que inmoviliza el drone en un punto fijo del espacio, dejando libre el movimiento de pitch y yaw y bloqueando el de roll (pudiéndose también intercambiar el bloqueo de roll por el de pitch). El dispositivo permite entonces probar el controlador de actitud en un entorno controlado.

El procedimiento consiste en fijar el drone al dispositivo por medio de tornillos y tuercas (Figura 15.2). Luego se energiza el drone y se inicializa el sistema. Por último, se envían los setpoints deseados por medio de la interfaz web.

La primera prueba consistió en mantener un estado estático, partiendo con valores en la orientación $(\phi, \theta, \psi) = (0, 0, 0)$. Se tuvo un control relativo del drone pero se presentó una oscilación en pitch constante, sin importar que valores de proporcionales se usarán, que en un principio era a 5 Hz.

Se logró reducir la oscilación agregando un proporcional ($K_{\dot{\Omega}} < 1$) en la realimentación de la velocidad angular (Figura 15.4), de esta forma se redujo la oscilación del drone a 2,78 Hz, mejorando también el error asintótico y la amplitud de oscilaciones en la orientación.

Entre posibilidades se estudió la frecuencia de oscilación del eje del dispositivo de prueba. Para ello, se inmovilizó el drone y se agitó el dispositivo para medir la oscilación. En la Figura 15.3 se muestra la FFT resultante de la velocidad en pitch medida dejando oscilar el dispositivo y la del drone durante una prueba de control.

Se puede observar que el dispositivo de prueba oscila a una frecuencia de 1,84 Hz mientras que el drone oscila a 2,78 Hz, descartando que sea la causa principal de las oscilaciones.

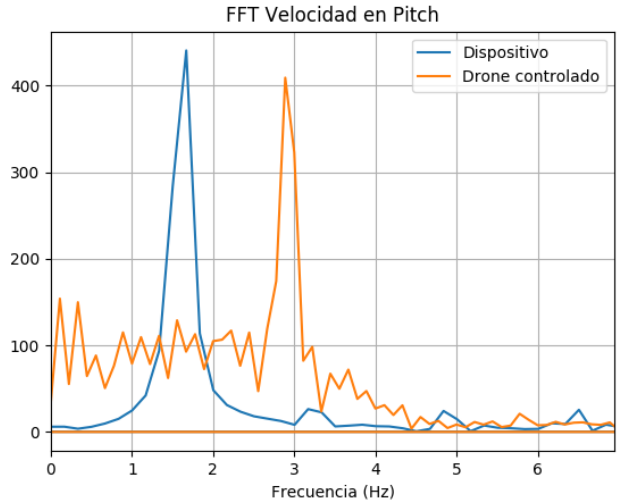


Figura 15.3: FFT velocidad de dispositivo de pruebas y drone durante pureba de control, utilizando la velocidad de roll medida por la IMU.

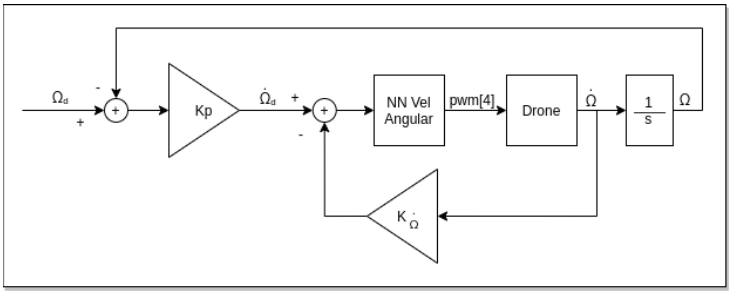


Figura 15.4: Diagrama de bloques del controlador.

Los siguientes valores fueron los que se utilizaron para los proporcionales:

$$\begin{cases} K_P = \{0{,}5; 0{,}5; 1{,}5\} \\ K_{\dot{\Omega}} = \{0{,}2; 0{,}2; 1\} \end{cases} . \tag{15.1}$$

Con estos valores, los resultados obtenidos sobre el control fueron los más óptimos. Luego, se realizó una prueba con diferentes setpoints en pitch y yaw que se pueden observar en la Tabla 15.1.

Los resultados del control se muestran en la Figura 15.5. Allí se observa como el movimiento en yaw es acorde a lo deseado, mientras que en pitch se tiene un offset y la oscilación antes mencionada.

Pitch (rad)	0	0,2	-0,2	0,2	0,2	0,2	-0,2	0
Yaw (rad)	0	0	0	1	2	3	3	0

Tabla 15.1: Setpoints enviados durante la prueba.

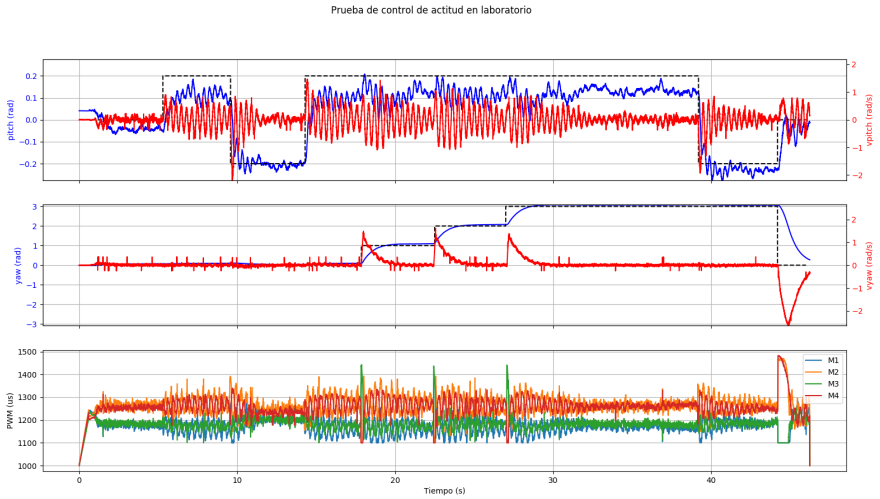


Figura 15.5: Resultados prueba de control de actitud en laboratorio.

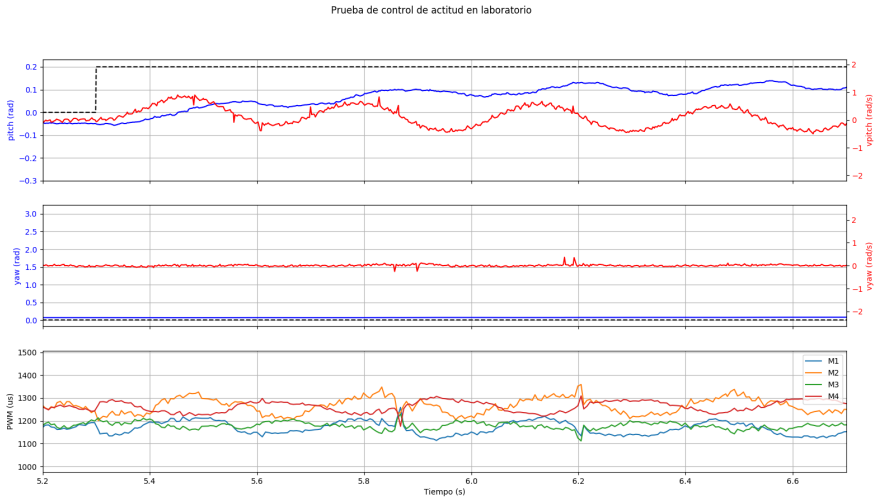


Figura 15.6: Resultados prueba de control de actitud en laboratorio.

Recortando el experimento de forma que se aprecien mejor los resultados se tiene lo que se muestra en la Figura 15.6, Figura 15.7 y Figura 15.8.

Por un lado, se ve un offset en pitch para los valores positivos de 0.1 rad (Figura 15.6), en cambio para valores negativos de pitch (Figura 15.7) se alcanza el setpoint deseado considerando el valor medio de las oscilaciones. Esta asimetría en el control es debido a que el eje de giro del dispositivo no coincide exactamente con el centro de masa del drone causando que el torque provocado por la gravedad no sea despreciable. En una prueba de vuelo real este offset debería ser despreciable en ambos casos.

Respecto de las oscilaciones, se aprecia claramente como se tiene una oscilación

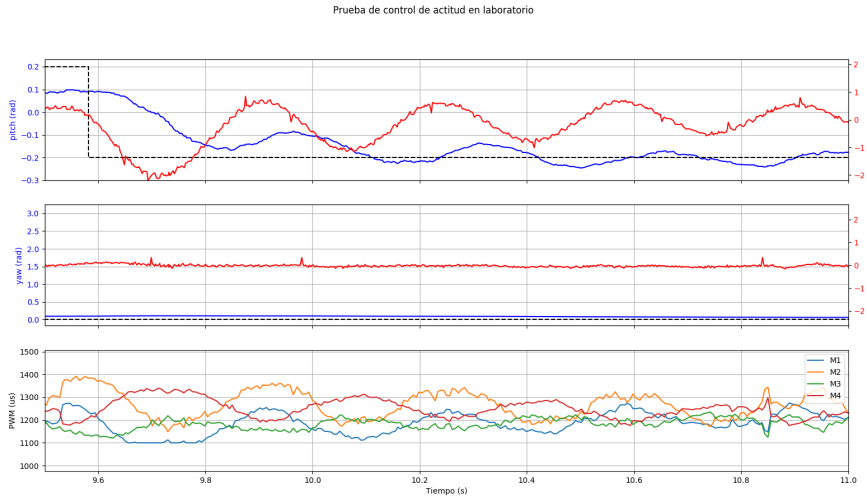


Figura 15.7: Resultados prueba de control de actitud en laboratorio.

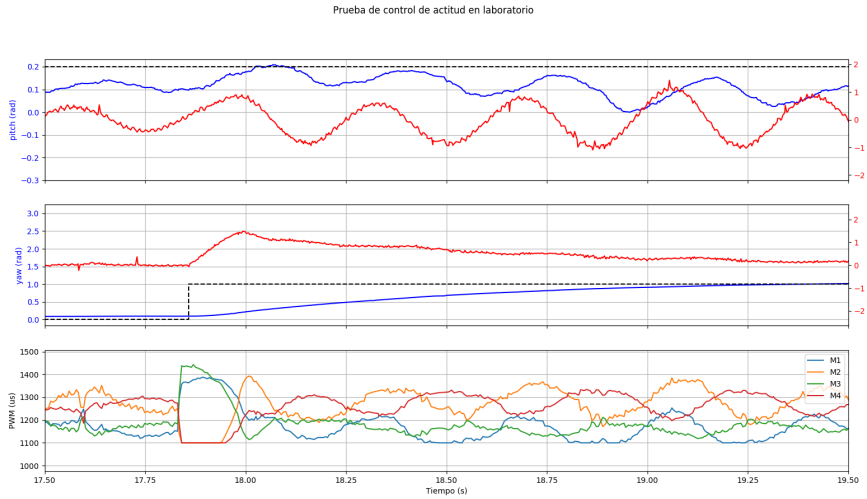


Figura 15.8: Resultados prueba de control de actitud en laboratorio.

en pitch y en consecuencia, también en los motores. Si bien se logró controlar bastante (al comienzo estaba en 5 Hz), aún persiste y esto puede deberse a efectos no modelados.

Analizando los resultados en yaw (Figura 15.8), se ve como el valor deseado es alcanzado de forma correcta sin presentar offset en la medida ni oscilaciones.

15.3. Estudio analítico del sistema

El sistema de control se puede representar como el diagrama de lazo de la Figura 15.9. Se toma como hipótesis que el sistema es lineal y se considera que los

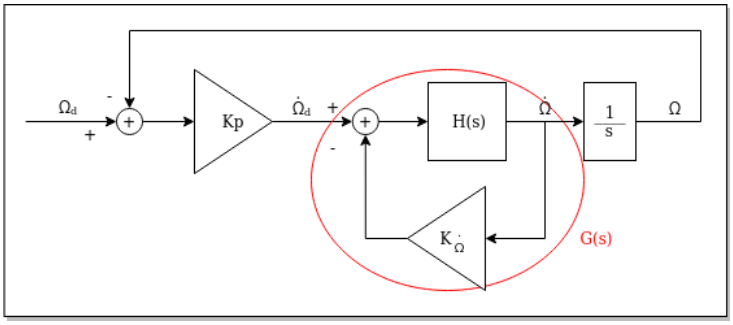


Figura 15.9: Diagrama de bloques del sistema.

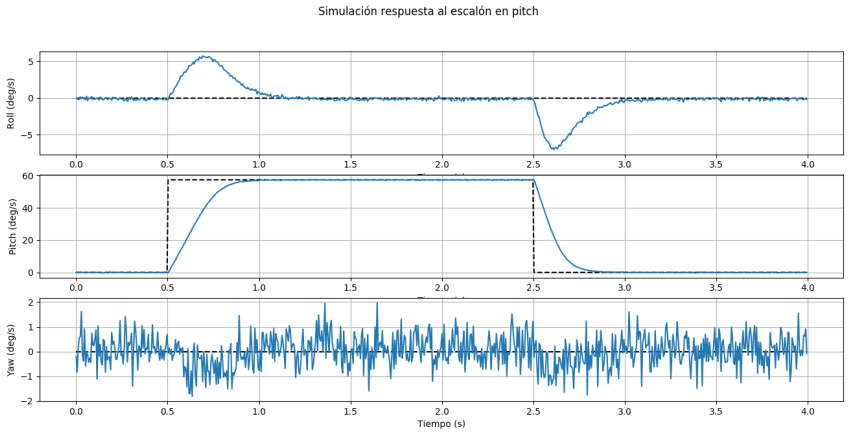


Figura 15.10: Simulación respuesta al escalón con $K_{\dot{\Omega}} = \{0{,}2; 0{,}2; 1\}$.

ejes se comportan de forma independiente.

La red neuronal junto con del drone se puede representar con la función de transferencia H(s). Por otro lado, la función de transferencia del lazo interno es:

$$G(s) = \frac{H(s)}{1 + K_{\dot{\Omega}} H(s)} = \frac{a}{s + a}. \tag{15.2}$$

Simulando la respuesta al escalón del lazo interno (Figura 15.10) para pitch, se tiene que los tiempos de asentamiento para distintos setpoints son los que se muestran en la Tabla 15.2.

Sabiendo que

$$t_s = 3\tau \tag{15.3}$$

y

$$a = \frac{1}{\tau}, \tag{15.4}$$

se tiene que $a_{pitch} = 2{,}128$.

Realizando el lugar geométrico de las raíces del sistema general, se tienen dos polos como se muestra en la Figura 15.11, donde el polo correspondiente al lazo

Setpoint en pitch (rad)	t_s (s)
0,5	1,39
0,4	1,37
0,3	1,18
0,2	1,73
0,1	1,39

Tabla 15.2: Tiempos de asentamiento para setpoints en pitch.

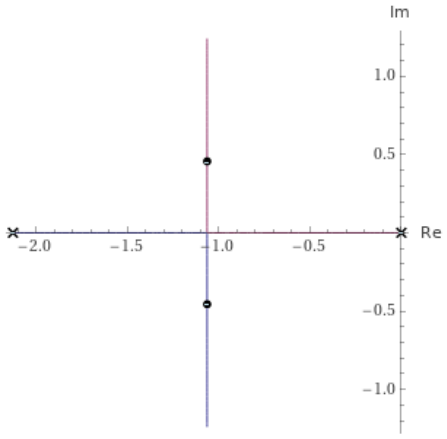


Figura 15.11: Lugar geométrico de las raíces para pitch con ganancia $K_P \in [0{,}1{,}25]$.

interno se puede variar utilizando K_P. De esta forma podemos ver que, para que la estabilidad sea óptima, el polo debe situarse en:

$$p = \frac{-a}{2}. \tag{15.5}$$

Por lo tanto

$$K_P = \frac{p^2}{a} = \frac{a}{4}. \tag{15.6}$$

Entonces para pitch $K_{P_{pitch}} = 0{,}532$. Tomando que roll es simétrico a pitch, los valores de los proporcionales coinciden.

Para el caso de yaw, se realizó el mismo procedimiento que en pitch. Los tiempos de asentamiento se muestran en la Tabla 15.3. Por lo que $a_{yaw} = 2{,}293$ y $K_{P_{yaw}} = 0{,}573$.

Resumiendo, el vector K_P toma valores

$$K_P = \{0,532; 0,532; 0,573\}. \tag{15.7}$$

Comparando con los elegidos durante las pruebas experimentales se tiene que en el caso de pitch y roll el proporcional se encuentra cercano al valor teórico, pero experimentalmente se presentan oscilaciones confirmando que es un problema del modelado

Setpoint en yaw (rad)	t_s (s)
0,5	1,48
0,4	1,36
0,3	1,31
0,2	1,32
0,1	1,25

Tabla 15.3: Tiempos de asentamiento para setpoints en yaw.

Para yaw, se aprecia una diferencia notable. Esta diferencia se debe a que, para las pruebas, se ajustó el proporcional de yaw en función de las oscilaciones que se presentaron en pitch.

Parte VII

Conclusiones

Capítulo 16

Conclusiones

16.1. Conclusiones

Se concluye que se logró construir un drone a partir de la búsqueda y selección de componentes adecuados que cumplan con los requisitos planteados de consumo y condiciones de vuelo. Para eso se hizo un estudio comparativo del comportamiento de los motores frente a dos tipos de hélices.

Otro aspecto importante es que se preparó una plataforma funcional de simulación de drones que sirve, entre otras cosas, como environment para el desarrollo y testing de nuevos programas basados en algoritmos de inteligencia artificial.

Dentro del entorno simulado, se modeló el drone con los parámetros del drone físico construido de forma que los resultados obtenidos en la simulación sean lo más parecidos a lo que se puede obtener físicamente.

Además, en este entorno, se logró implementar el control de vuelo a través de técnicas de control modernas basadas en redes neuronales, en particular Reinforcement Learning, área que antes de comenzar el proyecto se desconocía y por el que tuvimos que realizar una investigación para luego poder llevarlo a cabo.

Se generó una autonomía de vuelo que permite la planeación de trayectorias y la evasión de obstáculos de forma satisfactoria.

Se logró un código modular que permite una fácil adopción para futuros proyectos que deseen utilizar ROS. Además se deja una buena documentación del código desarrollado.

En cuanto a pruebas realizadas, se obtuvo un control del drone aceptable en un entorno controlado como es un dispositivo mecánico de pruebas. Sin embargo, a causa de la replanificación por la emergencia sanitaria declarada en el país en la etapa final del proyecto, no se pudieron realizar pruebas exhaustivas de vuelo real para comprobar las especificaciones iniciales del proyecto. Se realizaron pruebas controladas de control de actitud en su lugar.

16.2. Mejoras a futuro

Se plantea como continuación del proyecto, poder llevar a cabo pruebas de vuelo en espacios exteriores para poder terminar de ajustar los parámetros del controlador a las condiciones externas, además de comprobar el funcionamiento de los algoritmos de planeación de trayectorias y evasión de obstáculos y poder ajustarlos para que el vuelo sea óptimo.

En cuanto a la implementación de los algoritmos, se podría realizar la autonomía de vuelo mediante el uso de redes neuronales como se hizo con el control de vuelo. Esta idea estuvo al comienzo del proyecto, pero se decidió por mejorar los ya implementados en los proyectos anteriores de uQuad.

La combinación de la cámara integrada más la GPU del NVidia brindan una plataforma capaz de abordar problemas de visión por computadora utilizando inteligencia artificial, entre esto destacamos su uso para la evasión de obstáculos y detección de objetos en tiempo real.

Parte VIII

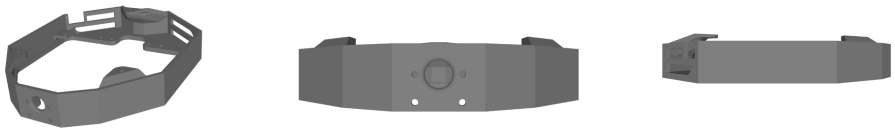
Apéndices

Apéndice A

Piezas adicionales de hardware

Se presentan las piezas de hardware diseñadas que se utilizaron como soportes para poder conectar los distintos componentes del drone.

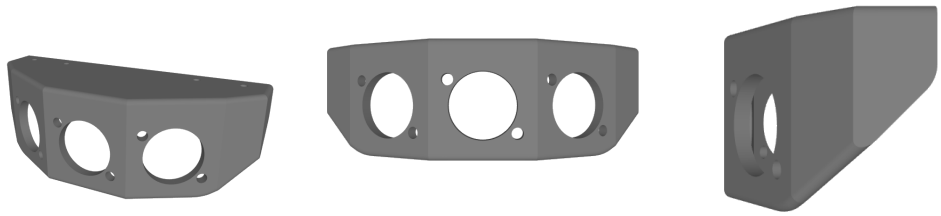
A.1. Cúpula



(a) Vista en perspectiva. (b) Vista frontal. (c) Vista lateral.

Figura A.1: Diseño de la cúpula.

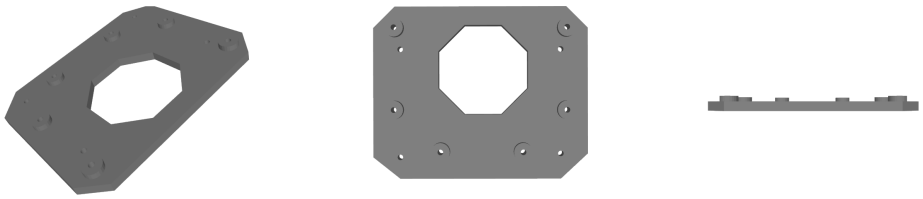
A.2. Soportes sensores ultrasonido



(a) Vista en perspectiva. (b) Vista frontal. (c) Vista lateral.

Figura A.2: Diseño del soporte del sensor de ultrasonido.

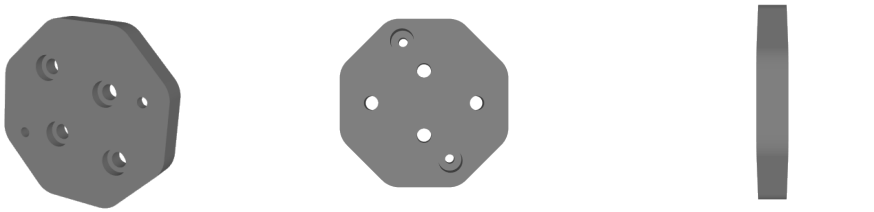
A.3. Soporte NVidia



(a) Vista en perspectiva. (b) Vista frontal. (c) Vista lateral.

Figura A.3: Diseño del soporte del Nvidia.

A.4. Soporte IMU



(a) Vista en perspectiva. (b) Vista frontal. (c) Vista lateral.

Figura A.4: Diseño del soporte de la IMU.

Esta página ha sido intencionalmente dejada en blanco.

Apéndice B

Cuaterniones

B.1. Introducción

Los cuaterniones [27], [28] fueron inventados por Sir William Rowan Hamilton (1809 *- 1865 †) en octubre de 1843. Con la finalidad de describir rotaciones en el espacio de la misma forma que un número complejo describe las rotaciones en el plano, Hamilton llegó a la conclusión de que para describir una rotación seguida de un cambio de escala se necesitan cuatro dimensiones.

Uno de los números describiría el tamaño del cambio de escala, otro los grados de rotación, y los últimos dos el plano en el que el vector debería ser rotado. Luego descubrió la multiplicación cerrada para números complejos de cuatro dimensiones de la forma $\mathbf{i}x + \mathbf{j}y + \mathbf{k}z$ donde $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$.

Los cuaterniones usualmente se escriben como $[s, \mathbf{v}]$, con $s \in \mathbb{R}$ y $\mathbf{v} \in \mathbb{R}^3$, donde s es la parte escalar, y $\mathbf{v} = \{x, y, z\}$ la parte vectorial.

B.2. Álgebra de Cuaterniones

Un cuaternión q es definido como la suma del escalar q_0 y el vector $\mathbf{q} = (q_1, q_2, q_3)$:

$$q = q_0 + \mathbf{q} = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}. \tag{B.1}$$

B.2.1. Suma y Multiplicación

La suma de dos cuaterniones sigue la convención de la suma vectorial. Sea el cuaternión p

$$p = p_0 + p_1\mathbf{i} + p_2\mathbf{j} + p_3\mathbf{k},$$

la suma se define como:

$$p + q = (p_0 + q_0) + (p_1 + q_1)\,\mathbf{i} + (p_2 + q_2)\,\mathbf{j} + (p_3 + q_3)\,\mathbf{k}. \tag{B.2}$$

Cada cuaternión q tiene un negativo $-q$ con sus componentes $-q_i, i = 0, 1, 2, 3$. El producto de dos cuaterniones satisface las reglas fundamentales introducidas por

Hamilton:

$$\begin{cases} \mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1 \\ \mathbf{ij} = \mathbf{k} = -\mathbf{ji} \\ \mathbf{jk} = \mathbf{i} = -\mathbf{kj} \\ \mathbf{ki} = \mathbf{j} = -\mathbf{ik} \end{cases} . \tag{B.3}$$

Dicho esto se puede definir el producto entre dos cuaterniones q y p como:

$$\begin{aligned} pq &= (p_0 + p_1\mathbf{i} + p_2\mathbf{j} + p_3\mathbf{k})(q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}) \\ &= p_0q_0 - (p_1q_1 + p_2q_2 + p_3q_3) + p_0(q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}) + q_0(p_1\mathbf{i} + p_2\mathbf{j} + p_3\mathbf{k}) \\ &+ (p_2q_3 - p_3q_2)\mathbf{i} + (p_3q_1 - p_1q_3)\mathbf{j} + (p_1q_2 - p_2q_1)\mathbf{k}. \end{aligned} \tag{B.4}$$

B.2.2. Conjugado, norma e inverso de un cuaternión

Sea el cuaternión $q = q_0 + \mathbf{q} = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k}$. El conjugado complejo de q se denota como q^* y se define:

$$q^* = q_0 - q_1\mathbf{i} - q_2\mathbf{j} - q_3\mathbf{k}. \tag{B.5}$$

De la definición se desprenden estas propiedades:

1. $(q^*)^* = q$
2. $q + q^* = 2q_0$
3. $q^*q = q_0^2 + q_1^2 + q_2^2 + q_3^2$
4. $(pq)^* = q^*p^*$

La norma de un cuaternión q es el escalar $|q| = \sqrt{q^*q}$. Se cumple la propiedad de que la norma del producto de dos cuaterniones p y q es el producto de las normas individuales.

Por último, inverso de un cuaternión q es definido como

$$q^{-1} = \frac{q^*}{|q|^2}. \tag{B.6}$$

Además, se puede verificar que $q^{-1}q = 1$.

B.2.3. El cuaternión como operador de rotaciones

Consideremos el cuaternión de norma unitaria $q = q_0 + \mathbf{q}$. La igualdad $q_0^2 + \|\mathbf{q}\| = 1$ implica que existe al menos un ángulo $\theta \in [0, \pi]$ que cumple:

$$\begin{cases} \cos^2\theta = q_0^2 \\ \sin^2\theta = \|\mathbf{q}\| \end{cases} .$$

Ahora, el cuaternión unitario se puede expresar en términos de θ y el vector unitario $\mathbf{u} = \frac{\mathbf{q}}{\|\mathbf{q}\|}$:

$$q = \cos\theta + \mathbf{u}\sin\theta. \tag{B.7}$$

Usando el cuaternión unitario y recordando que un vector $\mathbf{v} \in \mathbb{R}^3$ es un cuaternión cuya parte real es cero, se define el operador sobre vectores $\mathbf{v} \in \mathbb{R}^3$:

$$L_q(\mathbf{v}) = q\mathbf{v}q^* = \left(q_0^2 - \|\mathbf{q}\|^2\right)\mathbf{v} + 2(\mathbf{q}\cdot\mathbf{v})\mathbf{q} + 2q_0(\mathbf{q}\times\mathbf{v}) \tag{B.8}$$

De aquí se pueden hacer dos importantes observaciones:

1. El operador (B.8) no cambia el largo del vector $\mathbf{v}$:

$$\begin{aligned}\|L_q(\mathbf{v})\| &= \|q\mathbf{v}q^*\| \\ &= |q|\cdot\|\mathbf{v}\|\cdot|q^*| \\ &= \|\mathbf{v}\|.\end{aligned}$$

2. Si la dirección del vector $\mathbf{v}$ es la misma que el cuaternión $\mathbf{q}$, al aplicarse el operador, se mantiene la dirección original. Para verificarlo, se opera sobre el vector $\mathbf{v} = k\mathbf{q}$:

$$\begin{aligned}q\mathbf{v}q^* &= q(k\mathbf{q})q^* \\ &= \left(q_0^2 - \|\mathbf{q}\|^2\right)(k\mathbf{q}) + 2(\mathbf{q}\cdot k\mathbf{q})\mathbf{q} + 2q_0(\mathbf{q}\times k\mathbf{q}) \\ &= k\left(q_0^2 + \|\mathbf{q}\|^2\right)\mathbf{q} \\ &= k\mathbf{q}.\end{aligned}$$

Ambas observaciones sugieren que el operador L_q funciona como una rotación del vector $\mathbf{v}$ respecto de $\mathbf{q}$, y esto queda plasmado en los dos siguientes teoremas.

Teorema B.2.1 *Sea q un cuaternión unitario cualquiera, se cumple que*

$$q = q_0 + \boldsymbol{q} = \cos\frac{\theta}{2} + \boldsymbol{u}\sin\frac{\theta}{2} \tag{B.9}$$

y para cualquier vector $\boldsymbol{v} \in \mathbb{R}^3$ el operador L_q sobre $\boldsymbol{v}$

$$L_q(\boldsymbol{v}) = q\boldsymbol{v}q^*$$

es equivalente a una rotación de ángulo θ con $\boldsymbol{u}$ como eje de rotación.

Teorema B.2.2 *Sea q un cuaternión unitario cualquiera, se cumple que*

$$q = q_0 + \boldsymbol{q} = \cos\frac{\theta}{2} + \boldsymbol{u}\sin\frac{\theta}{2} \tag{B.10}$$

y para cualquier vector $\boldsymbol{v} \in \mathbb{R}^3$ el operador L_q sobre $\boldsymbol{v}$

$$L_q^*(\boldsymbol{v}) = q * \boldsymbol{v}(q^*)^* = q^*\boldsymbol{v}q$$

es una rotación del sistemas de coordenadas sobre el eje $\boldsymbol{u}$ sobre un ángulo θ, mientras que $\boldsymbol{v}$ no es rotado.

Dicho de otra forma: el operador L_q^* rota el vector $\boldsymbol{v}$ con respecto al sistema de coordenadas un ángulo $-\theta$ sobre el eje $\boldsymbol{q}$.

En resumen, el operador $L_q(\mathbf{v})$ se puede interpretar como una rotación vectorial respecto de su sistema fijo de coordenadas. Mientras que el operador $L_q^*(\mathbf{v})$ se puede interpretar como una rotación del sistema de coordenadas.

Esta página ha sido intencionalmente dejada en blanco.

Apéndice C

Calibración ESCs

Para poder utilizar las ESCs, primero se debieron configurar y luego calibrar.

C.1. Configuración

Para la configuración se utiliza el programa "BLHeliSuite32", donde es posible configurar los parámetros de funcionamiento de la ESC como la frecuencia del PWM, el throttle mínimo y máximo y el sonido de encendido. Para el proyecto se utilizó la siguiente configuración:

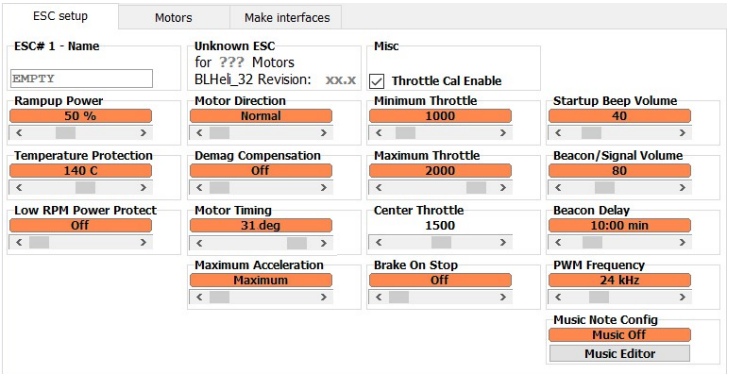


Figura C.1: Configuración de las ESCs.

Esta configuración permite que motores de bajo Kv, como los utilizados funcionen correctamente.

C.2. Calibración

Para la correcta calibración se deben seguir los siguientes pasos:

1. Estando la ESC desconectada, enviar una señal de máxima velocidad. En este caso se utilizó un pulso de 2000 μs.

2. Conectar la ESC y esperar hasta que se escuche el sonido de encendido.
3. Enviar a la ESC un pulso de mínima duración (se utilizó 1000 μs). Volverá a sonar la ESC quedando calibrada.

Apéndice D

Software

El código desarrollado se puede encontrar en el repositorio GitLab: `https://gitlab.fing.edu.uy/araceli.rodriguez/tesis-uquad4`

D.1. Paquetes ROS

A continuación se detallan los paquetes de ROS desarrollados.

D.1.1. uquad4_communication

Paquete con mensajes utilizados en la comunicación entre nodos.

Mensajes

uquad4_communication/escs - PWMs de escs.

uquad4_communication/setpoint - Vector de setpoints deseados.

D.1.2. uquad4_controller

El paquete de uquad4_controller se encarga de implementar la red de control de actitud entrenada a un nodo de ROS.

También implementa la funcionalidad de generar el grafo de la red a partir de las redes entrenadas en el paquete de uquad4_neural_networks de control de velocidad angular y velocidad en z, así como los controladores proporcionales y throttle mixing.

Nodos

uquad4_controller_node - Nodo principal el cual ejecuta la red neuronal de control de actitud.

Tópicos publicados:

- uquad4/escs_pwms (uquad4_communication/escs): PWMs en microsegundos a aplicar a las ESCs.

Tópicos suscritos:

- uquad4/translational_state (uquad4_sensors/drone_state): Estado traslacional del drone.
- uquad4/angular_state (uquad4_sensors/drone_state): Estado angular del drone.
- uquad4/z_velocity_setpoint (uquad4_communication/Setpoint): Velocidad en z deseada.
- uquad4/angular_position_setpoint (uquad4_communication/Setpoint): Velocidad angular deseada.
- uquad4/emergency/stop_motors (std_msgs/Empty): Parada de motores en caso de emergencia.

Ya que la red debe ejecutarse a una frecuencia alta, el nodo se implementó en C++ utilizando la api de Tensorflow en dicho lenguaje. Para esto, previamente se debió exportar la red entrenada en Python obteniendo el archivo "./src/model/optimized_saved_model.pb" el cual contiene la arquitectura y pesos de la misma. Las instrucciones para realizar dicho proceso se encuentran en el archivo README.md de este paquete.

D.1.3. uquad4_description

Paquete con modelo y plugins de Gazebo para simular el drone en Gazebo.

Plugins

drone_dynamic - Simulación dinámica de fuerzas de motores.

Tópicos suscritos:

- uquad4/escs_pwms (uquad4_communication/escs): PWMs en microsegundos a aplicar a las escs.

state_pluging - Publicador de estado del drone sin la necesidad de pasa por la fusión sensorial.

Tópicos publicados:

- uquad4/translational_state (uquad4_sensors/drone_state): Estado traslacional del drone.
- uquad4/angular_state (uquad4_sensors/drone_state): Estado angular del drone.

uquad4_gazebo_ros_imu - Simulador de IMU para Gazebo.

Tópicos publicados:

- imu (sensor_msgs/Imu): Datos de la IMU.

Modelos

drone Modelo SDF de drone con sensores configurados.

D.1.4. uquad4_logs

Paquete con nodo que guarda los datos publicados en los tópicos a archivos, para su posterior análisis.

Nodos

uquad4_logs_node - Nodo de logging, el cual se puede configurar los tópicos a logear.

D.1.5. uquad4_neural_network

Paquete auxiliar para entrenar las redes neuronales. La estructura de carpetas es la siguiente:

uquad4_neural_network
– src
– – enviromets
– – stable_baselines
– – training_results

En la carpeta src se encuentra el código del paquete que contiene lo siguiente:

- En **stable_baselines**, se encuentra la versión 2.8.0 del repositorio de github[1], con las modificaciones mencionadas en el Capítulo 10, el cual contiene los algoritmos de entrenamiento.
- La carpeta **environments** contiene los environments programados para el entrenamiento de la red de control de velocidad angular y de control de velocidad en z utilizando las ecuaciones del modelo físico.
- En **training_results** se encuentran los modelos entrenados y logs de tensorboards de estos entrenamientos.

En src también se encuentran los archivos .py de entrenamiento y testeo de ambas redes.

D.1.6. uquad4_path_planning

Paquete con nodo de planeación de trayectoria y módulo de python para seguimiento de las mismas.

[1] https://github.com/hill-a/stable-baselines

Mensajes

uquad4_path_planning/Trajectory - Curvas de Dubins entre 2 waypoints.

uquad4_path_planning/path_planned - Trayectoria planeada y a seguir, la cual contiene una lista del mensaje uquad4_path_planning/Trajectory.

Nodos

path_planning_uQuad4 - Nodo principal que ejecuta la red neuronal de control de actitud.

Tópicos publicados:

- /uquad4/path_planned (uquad4_path_planning/path_planned): Trayectoria de misión planeada.

Tópicos suscritos:

- /uquad4/waypoint_queue (geometry_msgs/PoseArray): Lista de waypoints por los que debe pasar el drone.

D.1.7. uquad4_position_controller

Este paquete implementa el controlador de posición, a través del nodo uquad4_position_controller_node.

Nodos

uquad4_position_controller_node - Nodo con controlador de misión implementado.

Tópicos publicados:

- uquad4/z_velocity_setpoint (uquad4_communication/Setpoint): Setpoint de velocidad en z necesario para cumplir el comando.
- uquad4/angular_position_setpoint (uquad4_communication/Setpoint): Setpoint de velocidad angular neceario para cumplir el comando.

Tópicos suscritos:

- uquad4/translational_state (uquad4_sensors/drone_state): Estado traslacional del drone.
- uquad4/angular_state (uquad4_sensors/drone_state): Estado angular del drone.
- uquad4/cmd_vel (geometry_msgs/Twist): Comando de control de velocidad. Sólo se tienen en cuenta los campos lineal.x, lineal.z y angular.z

D.1.8. uquad4_sensor_fusion

Paquete con nodo que realiza la fusión sensorial y publica el estado del drone.

Nodos

uquad4_sensor_fusion_node - Nodo de fusión sensorial escrito en C++, donde se ejecuta el filtro de Kalman extendido y se fusionan los distintos mensajes de los sensores en el estado del drone.

Tópicos publicados:

- /uquad4/translational_state (uquad4_sensors/drone_state): Estado traslacional del drone.
- /uquad4/angular_state (uquad4_sensors/drone_state): Estado angular del drone.

Tópicos suscritos:

- /imu (sensor_msg/Imu): Datos de la IMU.
- /fix (sensor_msg/NavSatFix): Datos del GPS.
- /barometer (geometry_msgs/PoseWithCovarianceStamped): Altura proporcionada por el barómetro.

D.1.9. uquad4_obstacle_avoidance

Paquete con módulo de Python para la evasión de obstáculos.

Nodos

obstacle_avoidance - Nodo principal que ejecuta la evasión de obstáculos.

Tópicos publicados:

- /uquad4/arm_motors (std_msgs/Empty): Armado de motores.
- /uquad4/cmd_vel (geometry_msgs/Twist): Comando de control de velocidad. Sólo se tienen en cuenta los campos lineal.x y angular.z.

Tópicos suscritos:

- /uquad4/translational_state (uquad4_sensors/drone_state): Estado traslacional del drone.
- /uquad4/angular_state (uquad4_sensors/drone_state): Estado angular del drone.
- /sonar1 (sensors_msgs/Range): Sensor de ultrasonido 1.
- /sonar2 (sensors_msgs/Range): Sensor de ultrasonido 2.
- /sonar3 (sensors_msgs/Range): Sensor de ultrasonido 3.

D.1.10. uquad4_state_machine

Paquete que implementa la máquina de estados del drone.

Nodos

uquad4_state_machine - Máquina de estados.

Tópicos publicados:

- /uquad4/cmd_vel (geometry_msgs/Twist): Comando de control de velocidad.
- /uquad4/state_machine/state (std_msgs/String): Estado actual.

Tópicos suscritos:

- /uquad4/translational_state (uquad4_sensors/drone_state): Estado traslacional del drone.
- /uquad4/angular_state (uquad4_sensors/drone_state): Estado angular del drone.
- /uquad4/state_machine/new_state (std_msgs/String): Nuevo estado deseado.
- /sonar4 (std_msgs/Range): Sensor de ultrasonido para el aterrizaje.
- /uquad4/path_planned (uquad4_path_planning/path_planned): Trayectoria de misión planeada.

D.1.11. uquad4_sensors

Paquete con mensajes utilizados en la comunicación entre nodos.

Mensajes

uquad4_sensor/drone_state - Valores de estado de drone de posición y velocidad.

Apéndice E

Manual de Usuario

E.1. Instalación del Software

El software desarrollado en esta tesis fue probado en computadoras con Ubuntu 18.04 LTS y en el Nvidia Jetson Nano con Ubuntu 18.04.

Se debe instalar la versión Desktop-Full de ROS Melodic para Ubuntu el cual incluye la instalación de Gazebo, en caso de utilizarlo en una computadora con arquitectura x86 para trabajar con el sistema simulado. Para el Nvidia Jetson Nano se debe instalar la versión Desktop de ROS Melodic. En ambos casos también se debe configurar ROS para utilizar Python 3.

Además, será necesario instalar ciertos librerías y paquetes que están indicados para cada paquete de ROS que se desee utilizar.

E.2. Uso del sistema

E.2.1. Inicialización

En caso de desear trabajar con el sistema simulado se debe compilar el proyecto utilizando el comando catkin_make, en este punto es posible lanzar el simulador con el comando "roslaunch uquad4_gazebo_demos demo_5.launch", donde se podrá probar el sistema en un entorno seguro.

Para el sistema real, solo se deben compilar los paquetes que no requieran Gazebo utilizando el comando indicado en el archivo "Info
Compilacion"dentro del repositorio del proyecto. Los nodos se lanzan con el comando "roslaunch uquad4_start uquad4_start.launch".

E.2.2. Uso del Web Server

Para comunicarse con el drone se debe ingresar desde un navegador al puerto 5000 de la IP del drone, accediendo a una página web.

Figura E.1: Servidor Web - Tests.

Para ensayos de laboratorio

Una vez inicializado el sistema, se debe acceder a la pestaña "Tests" del servidor web (Figura E.1). Allí se pueden mandar setpoints de orientación para probar el controlador de actitud y visualizar en las gráficas los valores que toma el drone en tiempo real.

Lo que se debe hacer es configurar todos los valores que se muestran en la pantalla y luego hacer click en "Enviar". Se pueden enviar comandos nuevos cambiando los valores durante la prueba sin tener que comenzar desde cero con la inicialización.

Para realizar misiones

Para poder simular misiones de vuelo en Gazebo, se utiliza la página principal del Web Server (Figura E.2). En este caso, la secuencia de uso arranca en el estado "Init" pasando a "Landed" automáticamente.

Luego, se debe seleccionar la opción "Taking off" para poder ejecutar el despegue del drone. Una vez completado pasará al estado de "Hovering" esperando una de las siguientes opciones:

- Mission: Se elige realizar una misión, debiendo completar la tabla que se encuentra debajo con los datos de cada waypoint. Una vez completa, se selecciona "Enviar" y el drone comienza a realizar la trayectoria enviada.

  Al finalizar la misión, se pasa automáticamente al estado Hovering, pudiendo seleccionar una nueva misión o pasar al aterrizaje.

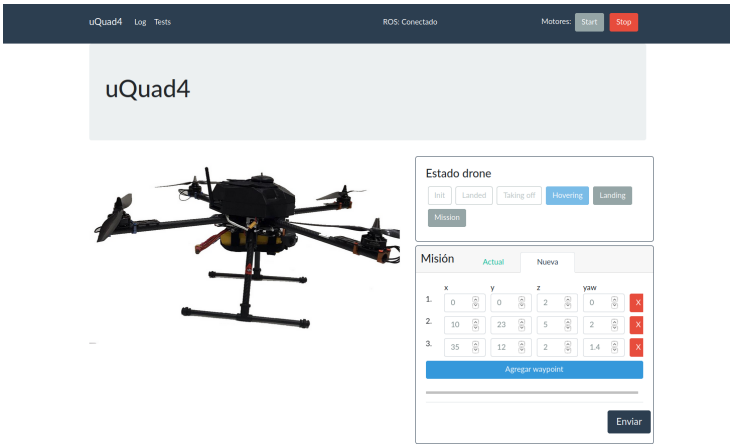


Figura E.2: Servidor Web - Misiones de vuelo.

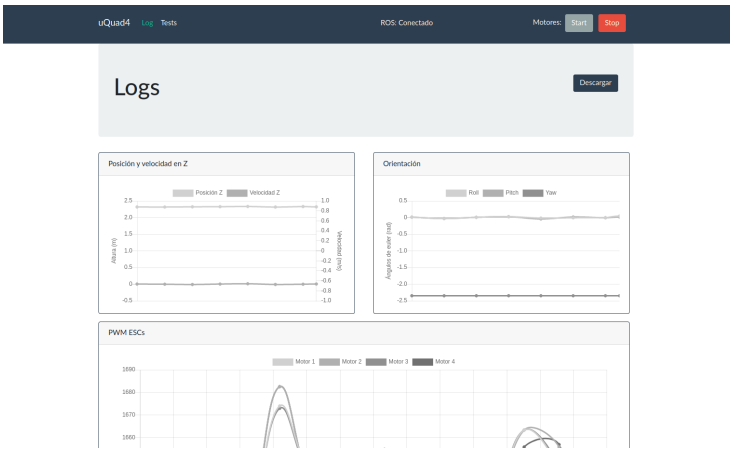


Figura E.3: Servidor Web - Logs.

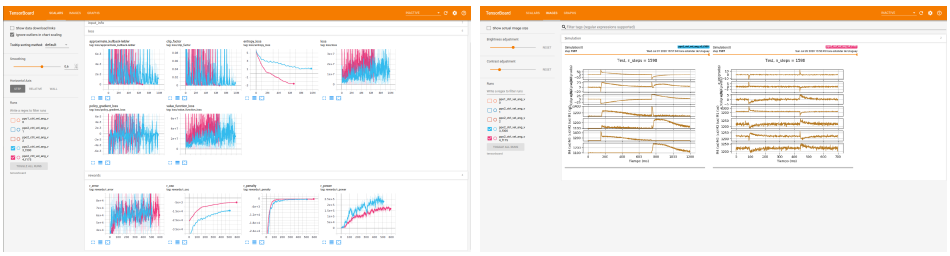
- Landing: el drone ejecuta la secuencia de aterrizaje. Para poder comenzar nuevamente, se debe seleccionar la opción “Init”.

E.2.3. Configuración de Logs

Para poder grabar Logs en pruebas de laboratorio, se debe descomentar la línea “ <node pkg=“uquad4_logs” type=“uquad4_logs_node” name=“uquad4_logs_node” ></node>” del archivo de launch del sistema ubicado en:

“src/uquad4_start/launch/uquad4_start.launch”. Luego, los logs podrán ser descargados desde el servidor web en la pestaña “Log” (Figura E.3).

Para dejar de generar los logs se debe comentar la línea mencionada anteriormente en el launch del sistema.



(a) Funciones de costos, recompensas y otros datos.

(b) Simulaciones.

Figura E.4: Logs de entrenamientos en Tensorboard.

E.3. Entrenamiento redes

En primer lugar, antes de comenzar el entrenamiento, se deben modificar los siguientes archivos, dependiendo la red que se desea entrenar.

- "src/uquad4_neural_networks/src/training_ctrl_vel_ang.py", contiene la arquitectura de la red y los hyper-parámentros de entrenamiento del controlador de velocidad angular.
- "src/uquad4_neural_networks/src/enviroments/control_vel_ang.py", contiene el environment y recompensas de entrenamiento del controlador de velocidad angular.
- "src/uquad4_neural_networks/src/training_ctrl_z.py", contiene la arquitectura de la red y los hyper-parámetros de entrenamiento del controlador de velocidad en z.
- "src/uquad4_neural_networks/src/enviroments/control_z.py", contiene el environment y recompensas de entrenamiento del controlador de velocidad en z.

Una vez ajustados los parámetros, el entrenamiento se comienza ejecutando los archivos "training_ctrl_vel_ang.py" o "training_ctrl_z.py" utilizando Python 3. Una vez terminado el entrenamiento, los datos de la red entrenada se guardarán automáticamente en un archivo .zip en la carpetas:

- "src/uquad4_neural_networks/src/training_results/control_vel_ang/temp"
- "src/uquad4_neural_networks/src/training_results/control_vel_z/temp"

Durante el entrenamiento dentro la carpeta "temp" donde se guarda el resultado, se guardan datos del entrenamiento como valores de recompensas y resultados de simulaciones realizadas en la carpeta "tensorboard". Es posible visualizar estos datos con Tensorboard, el cual se instala automáticamente al instalar Tensorflow. La Figura E.4 muestra como se visualizan los datos.

Una vez finalizado el entrenamiento, debe generarse un archivo protobuf de las redes entrenadas junto a los controladores con Tensorflow, de forma de tener un grafo el cual contiene el controlador de actitud.

Para realizarlo se debe ejecutar con Python 3 el archivo "optimiza_controller.py", ubicado en"src/uquad4_controller/src/uquad4_controller_src", configurando previamente la ruta de los modelos entrenados.

Realizado esto último paso, es posible probar la redes tanto en el sistema físico como en el real.

Esta página ha sido intencionalmente dejada en blanco.

Bibliografía

[1] S. Paternain, R. Rosa y M. Tailanián, *Implementación de un UAV con arquitectura de cuadricóptero*. Universidad de la República, jul. de 2012.

[2] J. Berruti, L. Falkestein y F. Favaro, *Vuelo autónomo de un cuadricóptero*. Universidad de la República, jul. de 2015.

[3] F. Cayafa, S. Torterolo y J. M. Torterolo, *Diseño y automatización de un UAV*. Universidad de la República, ago. de 2016.

[4] G. Hu, Z. Zhang, A. Armaou y Z. Yan, "Robust extended Kalman filter based state estimation for nonlinear dynamic processes with measurements corrupted by gross errors", *Journal of the Taiwan Institute of Chemical Engineers*, vol. 106, págs. 20-33, 2020.

[5] J. Redolfi, D. Gaydou y H. Agustin, "Filtro complementario para estimación de actitud aplicado al controlador embebido de un cuatrirrotor", en *Congreso Argentino de Sistemas Embebidos*, 2011, págs. 120-125.

[6] E. L. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents", *American Journal of Mathematics*, vol. 79, n.º 3, págs. 497-516, 1957.

[7] A. M. Shkel y V. Lumelsky, "Classification of the Dubins set", *Robotics and Autonomous Systems*, vol. 34, n.º 4, págs. 179-202, 2001.

[8] P. B. Sujit, S. Saripalli y J. B. Sousa, "An evaluation of UAV path following algorithms", en *2013 European Control Conference (ECC)*, 2013, págs. 3332-3337.

[9] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems", *Journal of Basic Engineering*, vol. 82, n.º 1, págs. 35-45, 1960.

[10] R. Schneider y C. Georgakis, "How To NOT Make the Extended Kalman Filter Fail", *Industrial & Engineering Chemistry Research*, vol. 52, n.º 9, págs. 3354-3362, 2013.

[11] T. Moore y D. Stouch, "A Generalized Extended Kalman Filter Implementation for the Robot Operating System", en *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*, Springer, 2014.

[12] Ozzmaker, *Berry IMU v2*, `https://ozzmaker.com/product/berryimu-accelerometer-gyroscope-magnetometer-barometricaltitude-sensor/`.

[13] Ublox, *NEO-M8 series*, `https://www.u-blox.com/en/product/neo-m8-series`.

[14] OpenAI, *OpenAI Spinning Up*, `https://spinningup.openai.com`.

[15] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. M. O. Heess, T. Erez, Y. Tassa, D. Silver y D. Wierstra, "Continuous Control with Deep Reinforcement Learning.", *CoRR*, 2016.

[16] J. Schulman, F. Wolski, P. Dhariwal, A. Radford y O. Klimov, *Proximal Policy Optimization Algorithms*, 2017. arXiv: `1707.06347 [cs.LG]`.

[17] OpenAI, *Baselines*, `https://github.com/openai/baselines/`, 2017.

[18] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang y W. Zaremba, *OpenAI Gym*, 2016. arXiv: `1606.01540 [cs.LG]`.

[19] W. Koch, *Flight Controller Synthesis Via Deep Reinforcement Learning*, 2019. arXiv: `1909.06493 [cs.LG]`.

[20] OpenAI, *PPO2*, `https://github.com/openai/baselines/tree/master/baselines/ppo2/`, 2017.

[21] I. Kamon y E. Rivlin, "Sensory-Based Motion Planning with Global Proofs", *Transactions on Robotics and Automation*, vol. 13, n.º 6, págs. 814-822, 1997.

[22] A. Yufka y O. Parlaktuna, "Performance Comparison of the BUG's Algorithms for Mobile Robots.", *International Symposium on Innovations in Intelligent SysTems and Applications - Trabazon, Turkey*, 2009.

[23] A. Y. Quiñonez, F. B. Pincheira, I. Burgueno y J. Bekios-Calfa, "Simulation and path planning for quadcopter obstacle avoidance in indoor environments using ROS framework", *International Conference on Software Process Improvement*, 2018.

[24] Open Robotics, *ROS (Robot Operating System)*, `https://www.ros.org/`.

[25] Open Robotics, *Gazebo*, `http://gazebosim.org/`.

[26] Technische Universität Darmstadt, *Hector Quadrotor*, `https://github.com/tu-darmstadt-ros-pkg/hector_quadrotor`.

[27] E. B. Dam, M. Koch y M. Lillholm, "Quaternions, interpolation and animation", Institute of computer science University of Copenhagen, inf. téc., 1998.

[28] Y. Jia, "Quaternions and Rotations", Com S 477/577 Notes, Stanford Computer Graphics Laboratory, inf. téc., 2013.

Esta página ha sido intencionalmente dejada en blanco.

Índice de tablas

3.1. Listado de componentes utilizados. 19

5.1. Datos para el período en cada uno de los ejes. 31

6.1. Datos relevados para hélice 13x40. 35
6.2. Datos relevados para hélice 13x40. 37

8.1. Especificaciones de la IMU. 49
8.2. Especificaciones del GPS. 51

10.1. Hiper-parámetros entrenamiento controlador velocidad angular, ρ es una variable que corresponde al progreso del entrenamiento (de 1 a 0). 63
10.2. Hiper-parámetros entrenamiento controlador velocidad en eje z. . . 67

14.1. Waypoints de misión realizada en la simulación. 92

15.1. Setpoints enviados durante la prueba. 97
15.2. Tiempos de asentamiento para setpoints en pitch. 101
15.3. Tiempos de asentamiento para setpoints en yaw. 102

Esta página ha sido intencionalmente dejada en blanco.

Índice de figuras

2.1. Diagrama físico del drone. [3] 7
2.2. Diagrama simplificado de control del sistema. Donde Ω y Ω^* es la orientación del drone y la orientación deseada respectivamente, z y z^* la altura y altura deseada y $\dot{\Omega}_{1.,4}$ las velocidades angulares de los propulsores. 8
2.3. Diagrama de la solución funcional del sistema. 8
2.4. Diagrama simplificado de control de actitud. 10
2.5. Modelo del drone simulado. 12

3.1. Especificaciones motores EMAX MT3506. 14
3.2. Diagrama de conexionado - NVidia Jeston Nano. 19
3.3. Diagrama de conexionado - Teensy 3.5. 20
3.4. Fotografías del drone armado. Vista Frontal. 20
3.5. Fotografías del drone armado. 20

4.1. Diagrama físico del drone. [3] 23
4.2. Sistemas de referencia.[3] 24
4.3. Cambio de coordenadas mediante rotaciones en los ángulos de Euler. 25
4.4. Diagrama de cuerpo libre. [3] 27

5.1. Esquema de conexión.[3] 30
5.2. Técnica de intersección de verticales.[3] 31

6.1. Diagrama de armado mecánico.[3] 34
6.2. Diagrama de armado eléctrico.[3] 34
6.3. Empuje vs. rpm hélice 13x40. 35
6.4. Diagrama de armado mecánico.[3] 36
6.5. Torque vs. rpm para la hélice 13x40. 37
6.7. Torque vs. rpm hélices 12x55 y 13x40. 38
6.6. Empuje vs. rpm hélices 12x55 y 13x40. 38
6.8. Empuje vs. potencia hélices 12x55 y 13x40. 39
6.9. Eficiencia de la hélice 13x40. 39

7.1. Loop de cálculo del algoritmo EKF.[4] 45
7.2. Diagrama de bloques del filtro complementario.[5] 47

Índice de figuras

8.1. Campo magnético de la IMU sin calibrar. 50
8.2. Campo magnético una vez calibrada la IMU. 50
8.3. Experimento estático GPS. 52

9.1. Diagrama de Reinforcement Learning. 55

10.1. Arquitectura red neuronal controlador velocidad angular. 61
10.2. Recompensas obtenidas en simulaciones realizadas durante el entrenamiento de red neuronal, capaz de controlar la velocidad angular. 63
10.3. Simulación controlador velocidad angular entrenado en entorno de entrenamiento. Las primeras tres gráficas muestran la velocidad angular del drone por curvas continuas azules, el setpoint deseado por las curvas ralladas negras. La última gráfica muestra los valores de PWM aplicado a las ESCs. 64
10.4. Diagrama de bloques control de orientación con controlador proporcional, junto a red neuronal de control de velocidad angular. 65
10.5. Arquitectura red neuronal controlador velocidad en eje z. 66
10.6. Simulación controlador velocidad en eje z. 68
10.7. Simulación controlador velocidad en eje z. 69

11.1. Diagrama de bloques controlador de posición. 72
11.2. Diagrama de bloques controlador de velocidades en x e y. 72

12.1. Tabla de decisión sobre curva mínima.[7] 77
12.2. Carrot chasing.[8] 78
12.3. Puntos notables del Distbug. [21] 80
12.4. Ejemplo de uso del algoritmo. [21] 81

13.1. Diagrama de comunicación ROS. 85
13.2. Diagrama de funcionamiento de ROS. 86

14.1. Modelo 3D del drone en Gazebo. 88
14.2. Diagrama de nodos de ROS, ejecutado en el sistema simulado. . . 89
14.3. Control estabilización en Gazebo, comportamiento posición. 91
14.4. Control estabilización en Gazebo, comportamiento orientación. . . 91
14.5. Entorno 3D con de la Facultad de Ingeniería en Gazebo, utilizado para la simulación de misión con obstáculos. 92
14.6. Posición del drone, en misión simulada sin obstáculos. 92
14.7. Posición del drone, en misión simulada con obstáculos. 93
14.8. Interfaz de usuario web. 94

15.1. Diagrama de nodos de ROS, ejecutado en el sistema real. 95
15.2. Drone montado en el dispositivo de pruebas del control de actitud. 96
15.3. FFT velocidad de dispositivo de pruebas y drone durante pureba de control, utilizando la velocidad de roll medida por la IMU. 97
15.4. Diagrama de bloques del controlador. 97
15.5. Resultados prueba de control de actitud en laboratorio. 98

15.6. Resultados prueba de control de actitud en laboratorio. 98
15.7. Resultados prueba de control de actitud en laboratorio. 99
15.8. Resultados prueba de control de actitud en laboratorio. 99
15.9. Diagrama de bloques del sistema. 100
15.10Simulación respuesta al escalón con $K_{\dot{\Omega}} = \{0{,}2; 0{,}2; 1\}$. 100
15.11Lugar geométrico de las raíces para pitch con ganancia $K_P \in [0{,}1{,}25]$. 101

A.1. Diseño de la cúpula. 109
A.2. Diseño del soporte del sensor de ultrasonido. 110
A.3. Diseño del soporte del Nvidia. 110
A.4. Diseño del soporte de la IMU. 111

C.1. Configuración de las ESCs. 117

E.1. Servidor Web - Tests. 126
E.2. Servidor Web - Misiones de vuelo. 127
E.3. Servidor Web - Logs. 127
E.4. Logs de entrenamientos en Tensorboard. 128