



UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE INGENIERÍA DE
COMPUTACIÓN

**IMPLEMENTACIÓN DE UN ALGORITMO DE
EVITACIÓN DE OBSTÁCULOS PARA DRONES
AUTÓNOMOS**

Por:
Joao Bose Pinto Diaz

Realizado con la asesoría de:
Prof. Gerardo Fernandez López

PROYECTO DE GRADO
Presentado ante la Ilustre Universidad Simón Bolívar
como requisito parcial para optar al título de
Ingeniero de Computación

Sartenejas, Diciembre de 2023



UNIVERSIDAD SIMÓN BOLÍVAR
DECANATO DE ESTUDIOS PROFESIONALES
COORDINACIÓN DE INGENIERÍA DE COMPUTACIÓN
**IMPLEMENTACIÓN DE UN ALGORITMO DE EVITACIÓN DE
OBSTÁCULOS PARA DRONES AUTÓNOMOS**
PROYECTO DE GRADO
PRESENTADO POR:
Joao Bose Pinto Diaz, 17-10490
RESUMEN

En el ámbito de la tecnología de drones, la seguridad de las operaciones es un elemento crucial para lograr una integración exitosa en diversas aplicaciones, desde la entrega de paquetes hasta la inspección de infraestructuras críticas. Este trabajo se enfocó en la implementación de un algoritmo de evasión de obstáculos para un dron comercial de cuatro rotores, utilizando un método basado en redes neuronales convolucionales propuesto en la literatura reciente. Este método recibe como entrada un mapa de profundidad, el estado cinemático del dron y una dirección objetivo, y predice tres posibles trayectorias libres de colisión para el próximo segundo de vuelo. Para adaptar el método a las especificaciones de seguridad de la plataforma física, en particular, reducir la rapidez de ejecución, se realizó un refinamiento fino de los pesos del modelo utilizando una base de datos generada en el mismo entorno de simulación utilizado en el trabajo original, pero con una rapidez de 1 m/s en lugar de 7 m/s. La capacidad del algoritmo implementado para esquivar obstáculos se evaluó tanto en entornos de simulación como en entornos reales. Los resultados revelaron una preferencia del algoritmo por esquivar obstáculos por el flanco izquierdo, lo que produce colisiones en situaciones donde esta preferencia no es viable. Esta tendencia se atribuyó a un desbalance en la consideración de trayectorias sin colisión durante la generación de la base de datos. En este contexto, se propusieron posibles soluciones para mejorar el rendimiento y la capacidad de generalización del algoritmo en futuros trabajos.

Palabras clave: Evasión de obstáculos, Vehículos aéreos no tripulados, Redes neuronales convolucionales.

*A mis padres
Luis y Griseida
Que me hicieron la persona que soy.
Y a mi hermana Perla Marina
Siempre alegrándome con sus ocurrencias.*

Agradecimientos

A mis padres, **Griseida y Luis**, por su apoyo y amor incondicional, y por hacerme la persona que soy. Y a mi hermanita, **Perla Marina**, por las incontables veces que me alegró el día con sus ocurrencias.

A mis padrinos, **Tony y Giselle**, por siempre velar por mi, por sus consejos, y por los incontables momentos felices que compartimos como familia.

A **Gerardo Fernandez**, mi segundo padre, por creer en mi, por los consejos y enseñanzas, por todas la oportunidades y apoyo que me ha brindado, y sobretodo, por ser un gran amigo en el que puedo confiar.

A **José Ángel, Juan Manuel y Ricardo silva**, mis hermanos de otras madres, por su amistad, por su confianza, por creer en mi, y por los incontables momentos y celebraciones.

Una vez mas a **Ricardo Silva**, por ser mi compañero en Tokio, sin tu apoyo y amistad esto no hubiera sido posible.

A **Gabriel Noya y Leonardo Cabrera**, por su excelente amistad, por su confianza, por todo el crecimiento profesional y personal en el que me han apoyado, y por su dedicación y pasión por la innovación que siempre me motiva a dar todo lo que puedo dar.

A **CITE USB**, por recibirme cálidamente desde el comienzo, por ser mi hogar en la universidad, por el crecimiento profesional y las oportunidades, y por el apoyo de sus miembros, especialmente de **David Altuve y Andrés Suarez**, por todo lo que me enseñaron y por su excelente apoyo y amistad.

A **Mariana y Fernando**, por ser mis amigos incondicionales desde mucho antes que todo comenzara, por los momentos inolvidables, por ser mis confidentes y por su apoyo durante los momentos mas difíciles.

A **Metablader, Bluff, Leo, Fotiti, Juanma y Roger**, por las incontables noches de partidas en discord, y por todos los momentos que compartimos.

A **Ernesto Lugo**, por ser mi compañero y amigo desde el comienzo de la universidad, por todos los momentos que compartimos, y por todos esos maratones de videojuegos y música que tanto disfruté.

A **Nicola Bitteto y José Arguello**, por los momentos que compartimos con la música/videojuegos, por siempre estar ahí para una conversación profunda sobre cualquier tema de interés, y sobretodo, por aquellas conversaciones que dieron apoyo en los momentos difíciles. A **Eladio**, por tu apoyo cuando no tenía nada, por los consejos y las conversaciones del futuro. Y a **Guillermo**, por compartir mi torcido sentido del humor.

A los profesores **David Coronado, Ricardo Monascal, Carolina Chang, Marlene Goncalves, Guillermo Palma, Carolina Martinez, Minaya Villasana** (y muchos más) por su disposición, esfuerzo y dedicación en impartir una educación de calidad aún cuando las condiciones no son idóneas.

A **Yeimi, Kim, Eduar y Yurima**, por prestarme su apoyo durante las etapas más complicadas de este logro.

A **dios** todopoderoso y a mis santos, por colocar esta meta en mi camino. A **Maria Francia, Guaramato, Juan Herrera, Cacique de las piedras, Antonio Aguilar M., Jaime Molina** y a todos mis protectores y guías, por mantenerme en mi camino y por los favores recibidos.

A mi abuelo **Victor Julio** que en paz descance, por ser mi inspiración. Se que estarías orgulloso de este logro.

Índice general

Resumen	I
Agradecimientos	II
Índice de figuras	VI
Índice de tablas	xii
Índice de algoritmos	xiii
Lista de acrónimos	xiv
1. Introducción	1
1.1. Resumen de los antecedentes	1
1.2. Justificación y planteamiento del problema	5
1.3. Objetivos	6
1.3.1. Objetivo general	6
1.3.2. Objetivos específicos	6
1.4. Estructura del trabajo	6
2. Descripción de la empresa receptora	8
3. Marco teórico	10
3.1. Vehículos aéreos no tripulados de cuatro rotores	10
3.1.1. Componentes	11
3.1.2. Información relevante del modelo del QUAV	14
3.1.3. Localización GPS y coordenadas NED	18
3.2. Visión estereoscópica	19
3.3. Redes neuronales	23
3.3.1. Retro-propagación	26
3.3.2. Redes neuronales convolucionales	28
3.4. Comunicación entre procesos	30

3.5. Resumen	32
4. Antecedentes	33
4.1. Desarrollos que dependen de la información global del entorno	33
4.2. Desarrollos que solo dependen de la información disponible a bordo del vehículo	36
4.3. <i>Learning high-speed flight in the wild</i>	40
4.3.1. El experto privilegiado	41
4.3.2. La política estudiante	43
4.3.3. Proyección de trayectorias	46
4.4. Resumen	47
5. Implementación	49
5.1. Plataforma de implementación	49
5.1.1. Plataforma física	50
5.1.2. Entorno de simulación	51
5.2. Selección del algoritmo	51
5.3. Arquitectura de la solución	52
5.3.1. Puente del FCU	54
5.3.2. Puente de profundidad	54
5.3.3. Proceso de inferencia	55
5.3.4. Ejecutor de trayectorias	58
5.4. Ajuste fino de la política estudiante	65
5.5. Resumen	66
6. Resultados	68
6.1. Resultados del ajuste fino de la política estudiante	69
6.2. Resultados de la política de evasión de obstáculos	78
6.2.1. Vuelos en entorno simulación	78
6.2.2. Vuelos sobre la plataforma física	92
6.3. Resumen	104
7. Conclusiones	106
Bibliografía	110

Índice de figuras

1.1.	Diagrama de flujo del proceso de entrenamiento utilizado en <i>Vision Based Drone Obstacle Avoidance by Deep Reinforcement Learning</i> .	2
1.2.	Visión general del método del método utilizado en <i>Learning high-speed flight in the wild</i>	4
2.1.	Logo de ACSL (<i>Autonomous Control Systems Laboratory</i>)	8
2.2.	Algunos productos de ACSL	9
3.1.	Diagrama de comunicación general del los componentes de un QUAV.	14
3.2.	Diagrama de un QUAV.	15
3.3.	Modos de vuelo básicos de un QUAV.	17
3.4.	Modelo geodésico del mundo WGS64.	18
3.5.	Ejemplo del proceso de rectificación de un par de imágenes estéreo.	20
3.6.	Ejemplo de la representación de un mapa de disparidad como una imagen.	21
3.7.	Ejemplo del Emparejamiento de Bloques para establecer correspondencias entre un par de imágenes estéreo rectificadas.	22

3.8. Modelo de un sistema de visión estereoscópica de dos cámaras paralelas C y C'	23
3.9. Funcionamiento de un nodo de una red neuronal.	24
3.10. Ejemplo de una FFNN con dos capas ocultas.	25
3.11. Ejemplo de la arquitectura de una CNN.	29
4.1. P-CAL: Canales Alternativos Pre-calculados	36
4.2. Mecanismo de evasión de obstáculos por CNN probabilística.	37
4.3. Diagrama de flujo del proceso de entrenamiento utilizado en <i>Vision Based Drone Obstacle Avoidance by Deep Reinforcement Learning</i> .	39
4.4. Visión general del método propuesto en <i>Learning high-speed flight in the wild</i>	40
4.5. Visualización de las muestras de P generadas por el experto durante un paso de la simulación.	43
5.1. SOTEN, un QUAV diseñado por ACSL que sirve de plataforma física para la implementación de este trabajo.	50
5.2. Diagrama de comunicación de la solución.	53
5.3. Visualización de la máquina de estados que guía la ejecución del hilo principal del ejecutor de trayectorias.	61
6.1. Componente espacial de la pérdida vs. época (ajuste fino).	69
6.2. Componente de estimación de la pérdida vs. época (ajuste fino). . .	70
6.3. Componente espacial de la pérdida vs. época (ajuste fino) luego de detener el entrenamiento tempranamente.	71

6.4. Componente de estimación de la pérdida vs. época (ajuste fino) luego de detener el entrenamiento tempranamente.	71
6.5. Visualización de tres vuelos de la base de datos con $v_{des} = 7 \text{ m/s}$. . .	74
6.6. Visualización de tres vuelos de la base de datos con $v_{des} = 1 \text{ m/s}$. . .	75
6.7. Primera configuración de obstáculos dentro del entorno de AirSim. . .	78
6.8. Visualización simultánea de cuatro caminos ejecutados dentro de la primera configuración de obstáculos en entorno de simulación. . . .	79
6.9. Diagrama de caja de la desviación máxima de los caminos ejecutados en la primera configuración de obstáculos en entorno de simulación.	80
6.10. Segunda configuración de obstáculos dentro del entorno de AirSim. .	80
6.11. Visualización simultánea de 4 caminos ejecutados dentro de la segunda configuración de obstáculos en entorno de simulación. . . .	81
6.12. Diagrama de caja de la desviación máxima de los caminos ejecutados en la segunda configuración de obstáculos en entorno de simulación.	82
6.13. Visualización del funcionamiento del mecanismo que desactiva la inferencia cuando no hay obstáculos visibles. Inferencia activada, el primer obstáculo está en el campo de visión.	83
6.14. Visualización del funcionamiento del mecanismo que desactiva la inferencia cuando no hay obstáculos visibles. Inferencia desactivada, no hay obstáculo visible en campo de visión, navegando en dirección a la meta.	84
6.15. Visualización del funcionamiento del mecanismo que desactiva la inferencia cuando no hay obstáculos visibles. Inferencia activada, el segundo obstáculo está en el campo de visión.	84

6.16. Tercera configuración de obstáculos dentro del entorno de AirSim.	85
6.17. Visualización 3D de un vuelo en la tercera configuración de obstáculos (1). Vista de arriba hacia abajo.	86
6.18. Visualización 3D de un vuelo en la tercera configuración de obstáculos (2). Vista frontal.	86
6.19. Visualización 3D de un vuelo en la tercera configuración de obstáculos (3). Vista frontal. Navegando entre el espacio entre los obstáculos. .	87
6.20. Visualización 3D de un vuelo en la tercera configuración de obstáculos (4). Vista frontal. Colisión.	88
6.21. Cuarta configuración de obstáculos dentro del entorno de AirSim. . .	88
6.22. Visualización 3D de un vuelo en la cuarta configuración de obstáculos (1). Vista frontal.	89
6.23. Visualización 3D de un vuelo en la cuarta configuración de obstáculos (2). Vista de arriba hacia abajo. Búsqueda del borde del obstáculo. .	90
6.24. Visualización 3D de un vuelo en la cuarta configuración de obstáculos (3). Vista de arriba hacia abajo. Búsqueda del borde del obstáculo. .	90
6.25. Visualización 3D de un vuelo en la cuarta configuración de obstáculos (4). Vista de arriba hacia abajo. Búsqueda del borde del obstáculo. .	91
6.26. Visualización 3D de un vuelo en la cuarta configuración de obstáculos (5). Vista de arriba hacia abajo. Colisión.	91
6.27. Configuración de obstáculos en entorno real: Obstáculo simple	93
6.28. Visualización de 10 caminos ejecutados en la primera configuración de obstáculos en entorno real.	94
6.29. Diagrama de caja de la desviación máxima de los caminos ejecutados en la primera configuración de obstáculos en entorno real.	94

6.30. Capturas de una grabación de uno de los vuelos en entorno real con un obstáculo simple (1).	95
6.31. Capturas de una grabación de uno de los vuelos en entorno real con un obstáculo simple (2).	95
6.32. Capturas de una grabación de uno de los vuelos en entorno real con un obstáculo simple (3).	96
6.33. Capturas de una grabación de uno de los vuelos en entorno real con un obstáculo simple. Vista frontal (1).	96
6.34. Capturas de una grabación de uno de los vuelos en entorno real con un obstáculo simple. Vista frontal (2).	97
6.35. Capturas de una grabación de uno de los vuelos en entorno real con un obstáculo simple. Vista frontal (3).	97
6.36. Configuración de obstáculos en entorno real: Obstáculos paralelos a 3 metros de separación.	98
6.37. Capturas de la grabación del vuelo en entorno real con obstáculos paralelos a 3 metros de separación.	99
6.38. Configuración de obstáculos en entorno real: Obstáculos paralelos a 4 metros de separación.	100
6.39. Capturas de la grabación del vuelo en entorno real con obstáculos paralelos a 4 metros de separación.	100
6.40. Visualización del camino ejecutado por el QUAV para la configuración de obstáculos paralelos a 4 metros de separación en entorno real.	101
6.41. Configuración de obstáculos en entorno real: Obstáculos paralelos a 1 metro de separación.	102

6.42. Capturas de la grabación del vuelo en entorno real con obstáculos paralelos a 1 metro de separación.	102
6.43. Visualización del camino ejecutado por el QUAV para la configuración de obstáculos paralelos a 1 metro de separación en entorno real.	103
6.44. Grafico del encabezamiento del QUAV en función del tiempo para la configuración de obstáculos paralelos a 1 metro de separación en entorno real.	103

Índice de tablas

4.1. Resumen de los métodos explorados durante la revisión del estado del arte.	48
6.1. Comparación de los valores de pérdida entre modelo original y ajustado según el conjunto de datos evaluado.	72
6.2. Distribución de ejemplos en cada conjunto de datos según hacia donde se dirige la trayectoria.	76
6.3. Resumen de los resultados obtenidos en los vuelos en simulación. . .	92
6.4. Resumen de los resultados obtenidos en los vuelos en entornos de la vida real.	104

Índice de algoritmos

1.	Pseudo-código del algoritmo de transformación de coordenadas WGS64 a coordenadas NED	19
2.	Pseudo-código del algoritmo de retro-propagación usando descenso de gradiente	28
3.	Pseudo-código del algoritmo para obtener un estimado de la profun- didad al obstáculo mas cercano en un mapa de profundidad	56
4.	Pseudo-código del procedimiento ejecutado por el hilo principal del proceso de inferencia.	58
5.	Pseudo-código del procedimiento de ejecución guiada por una máqui- na de estados.	61
6.	Pseudo-código del procedimiento de ejecución del estado “Navega- ción ingenua”.	63
7.	Pseudo-código del procedimiento de ejecución del estado “Evasión de obstáculos”.	64

Lista de acrónimos

CNN	Redes Neuronales Convolucionales, del inglés: <i>Convolutional Neural Networks</i> .
ACSL	Laboratorio de Sistemas de Control Autónomos, del inglés <i>Autonomous Control Systems Laboratory</i> .
UAV	Vehículo aéreo no tripulado, del inglés <i>unmanned aerial vehicle</i> .
QUAV	UAV de cuatro rotores, del inglés <i>Quadrotor UAV</i> .
IMU	Unidad de mediciones iniciales, del inglés <i>Inertial Measurement Unit</i> .
GPS	Sistema de posicionamiento global, del inglés <i>Global Positioning System</i> .
CUDA	Arquitectura Unificada de Dispositivos de Cómputo, del inglés: <i>Compute Unified Device Architecture</i> .
WGS64	Modelo geodésico del mundo 64, del inglés: <i>World Geodetic System 64</i> .
NED	Sistema de coordenadas Norte, Este, Abajo; del inglés <i>North, East, Down</i> .
BM	Emparejamiento de Bloques, del inglés <i>Block Matching</i> .
ReLU	Unidades Lineales Rectificadas, del inglés <i>Rectified Linear Units</i> .
FFNN	Red Neuronal de Alimentación hacia Adelante, del inglés <i>Feed-forward Neural Network</i> .
FC	Red Completamente Conectada, del inglés <i>Fully Connected</i> .
IPC	Comunicación entre procesos, del inglés Inter-Process Communication.

FIFO	Proceso primero que entra, primero que sale; del inglés <i>First In, First Out</i> .
API	Interfaz de programación de aplicación, del inglés <i>Application Programming Interface</i> .
ONNX	Estándar abierto para la interoperabilidad del aprendizaje de máquinas, del inglés <i>Open Standard for Machine Learning Interoperability</i> .

Capítulo 1

Introducción

En el vertiginoso mundo de la tecnología de drones, la seguridad y la eficiencia de las operaciones son imperativos clave para su integración exitosa en diversas aplicaciones, desde la entrega de paquetes hasta la inspección de infraestructuras críticas. La navegación segura y autónoma de drones, especialmente en entornos complejos y cambiantes, representa un desafío crucial que ha impulsado la investigación y el desarrollo de algoritmos avanzados. Este trabajo se centra en un aspecto fundamental de esta problemática: la evasión de obstáculos.

1.1. Resumen de los antecedentes

En la literatura existe una amplia gama de técnicas y métodos para abordar el problema de la evasión de obstáculos para drones.

Métodos basados en aprendizaje por reforzamiento (del inglés *reinforcement learning*) tales como los empleados en [1] y en [2] han probado ser efectivos en el entrenamiento de políticas reactivas de evasión de obstáculos. Estas políticas, a partir de entradas sensoriales en forma de imágenes producidas por cámaras *RGB* o de profundidad, producen trayectorias sin colisiones definidas por un espacio de acciones, todo esto sin tener información directa acerca de la localización de los obstáculos en el espacio. Estos métodos requieren un proceso de entrenamiento complejo tal como se puede observar en la Figura 1.1. Adicionalmente, estos

métodos introducen latencia al momento de realizar inferencia debido a la complejidad de los modelos (usualmente requiriendo múltiples redes neuronales para su funcionamiento), lo cual hace que estos métodos no sean ideales para drones con capacidad de procesamiento limitada. Adicionalmente la definición del espacio de acciones es una tarea complicada y puede resultar en trayectorias difíciles de ejecutar.

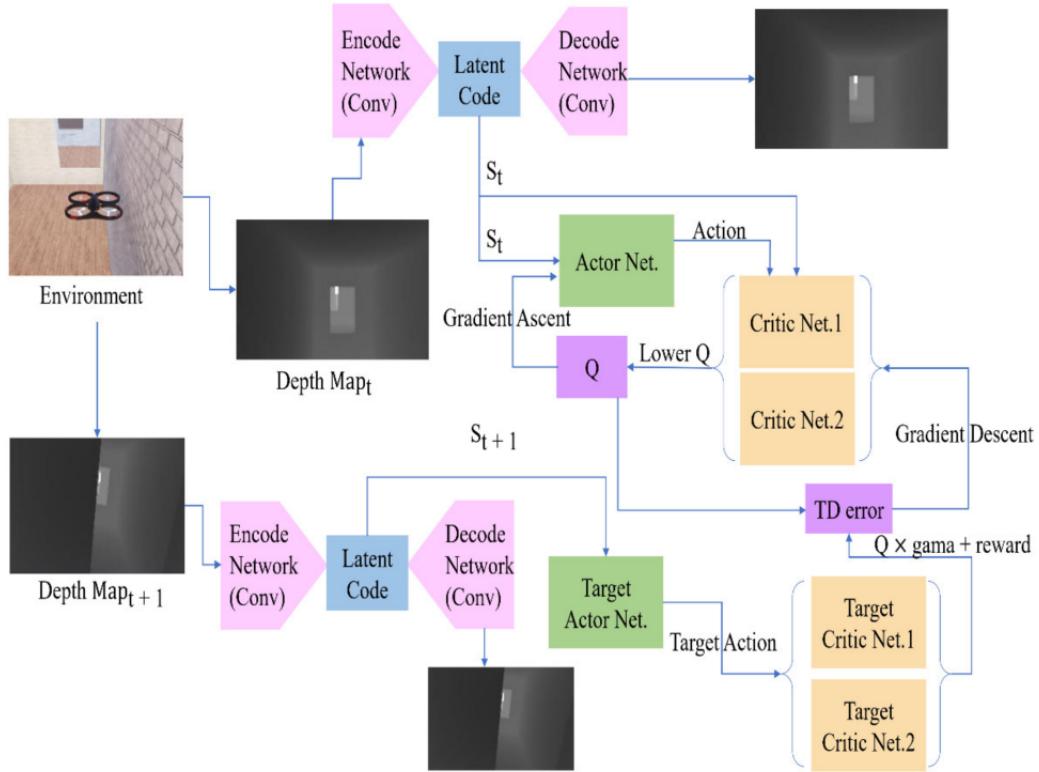


FIGURA 1.1: Diagrama de flujo del proceso de entrenamiento utilizado en [2].

Métodos con información parcial del entorno tales como el propuesto en [3] utilizan la información conocida del entorno para planear trayectorias libres de colisión antes de la ejecución del vuelo (proceso denominado *trajectory planning* o planeamiento de trayectorias), reduciendo la carga computacional del algoritmo durante la ejecución de la trayectoria, permitiendo que los recursos computacionales del dron estén disponibles para otras tareas. En particular, el método propuesto en [3], genera una colección de trayectorias libres de colisión, y alterna entre las distintas trayectorias o *lanes* al momento de ejecución, dependiendo de la información de los obstáculos observados, dándole la capacidad al dron de reaccionar a obstáculos no provistos en la información inicial del entorno. Sin embargo, el requerimiento

de poseer información previa sobre el entorno hace que sea complicado que este método opere en entornos genéricos o desconocidos.

Otros métodos utilizan redes neuronales convolucionales (*CNN* del inglés *convolutional neural networks*) para directa o indirectamente producir trayectorias libres de colisión en entornos desconocidos. En el método propuesto en [4] se utiliza una *CNN* probabilística, que a partir de una cámara monocular genera mapas de profundidad y mapas de confianza probabilísticos, que posteriormente son utilizados para navegar arbitrariamente el entorno sin generar colisiones. Sin embargo, las trayectorias generadas no van dirigidas hacia ningún objetivo en particular, lo cual hace que sea de poca utilidad en la práctica.

Los métodos que utilizan *CNN* requieren una base de datos para su proceso de entrenamiento. Esto limita la gama de tareas que puede resolver una *CNN* a las tareas para las cuales existe una base datos disponible, por ejemplo, se puede entrenar una *CNN* para estimar imágenes de profundidad a partir de imágenes *RGB* utilizando bases de datos como *KITTI* [5] y *TUM RGB-D* [6]. Las *CNN* tienen la ventaja de que son relativamente sencillas de entrenar con respecto a otras alternativas (como el aprendizaje por reforzamiento), por lo cual resultan atractivas para resolver problemáticas en donde se tiene disponible una base de datos. Es importante destacar que actualmente no hay bases de datos generales enfocadas en la tarea de la evasión de obstáculos para drones, limitando el uso que se le puede dar a las *CNN* dentro de esta problemática.

En el método propuesto por Loquercio et al. (2021) [7] se soluciona el requerimiento de poseer una base de datos para el entrenamiento de una *CNN* como política de evasión de obstáculos; empleando el denominado entrenamiento por imitación (del inglés *imitation learning*), en lugar de aplicar entrenamiento supervisado sobre una base de datos ya existente, se utiliza, dentro de un entorno de simulación, una política experta con acceso privilegiado a la información de los obstáculos del entorno (el “experto”) para generar una base de datos de trayectorias libres de colisión. Esta base de datos contiene ejemplos de trayectorias libres de colisión dado una imagen de profundidad, mediciones iniciales del estado del dron y la dirección hacia donde se encuentra el objetivo del dron. Esto permite entrenar una política “estudiante” (modelada como una *CNN*) de generación de trayectorias directa a partir de imágenes de profundidad, mediciones iniciales y

una dirección objetivo. Una visión general del método se puede observar en la Figura 1.2.

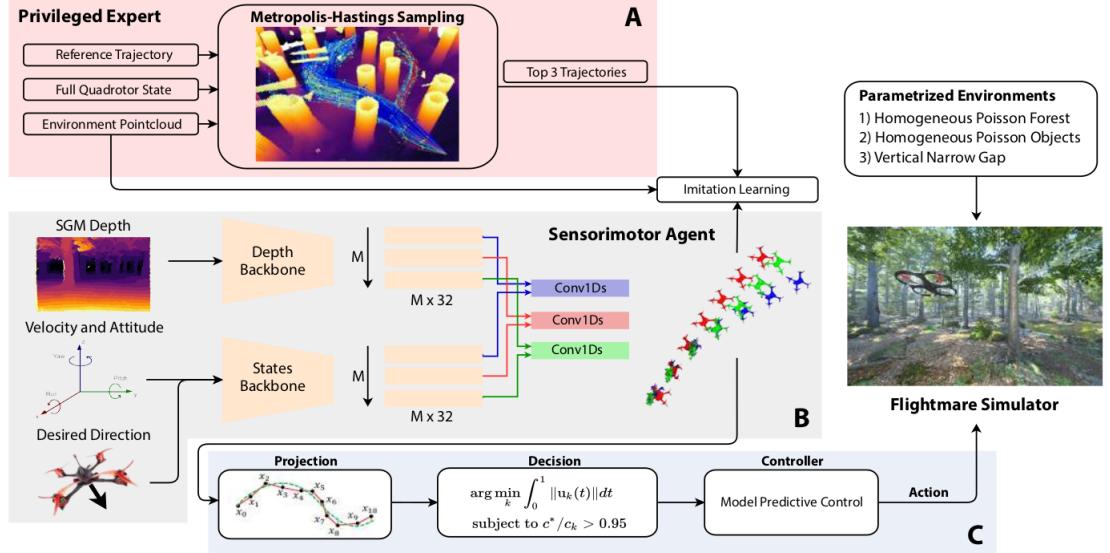


FIGURA 1.2: Visión general del método utilizado en [7]. **(A)** El “experto” genera una distribución de trayectorias libre de colisión que siguen una trayectoria de referencia. Las trayectorias generadas están condicionadas por la completa información de obstáculos del entorno. **(B)** La política “estudiante” es entrenada mediante entrenamiento por imitación para predecir las mejores 3 trayectorias a partir una estimación de la profundidad, mediciones iniciales del dron y una dirección objetivo. **(C)** Las predicciones son proyectadas en el espacio de las trayectorias polinomiales y finalmente la trayectoria con el costo de predicción mas bajo es ejecutada por el modelo de control.

La política “estudiante” puede transferirse trivialmente a un ambiente fuera de simulación, siempre y cuando al momento de generar la base datos de entrenamiento se utilicen métodos de estimación de imágenes de profundidad cuya salida posea niveles similares de ruido tanto en simulación como en entornos reales, tales como *SGM* [8] (del inglés *semi global matching*). Finalmente, el modelo propuesto en [7] resulta bastante conveniente para la evasión de obstáculos pues, es sencillo de entrenar, la política “estudiante” se modela utilizando una *CNN* ligera [9] que no introduce demasiada latencia al momento de ejecución, no necesita información previa del entorno, es fácil de transferir para funcionamiento en ambientes fuera de simulación y toma en cuenta la dirección de objetivo del dron, lo cual produce trayectorias libres de colisión que se dirigen al objetivo.

1.2. Justificación y planteamiento del problema

Los drones han experimentado un crecimiento exponencial en su adopción en diversas aplicaciones, desde la agricultura de precisión hasta la logística de entrega y la vigilancia. Sin embargo, uno de los desafíos fundamentales que enfrentan los drones es su capacidad para operar de manera segura y autónoma en entornos dinámicos y desconocidos, donde los obstáculos pueden surgir en cualquier momento y lugar. La evasión de obstáculos es esencial para garantizar la seguridad de las operaciones de los drones y su capacidad de navegación autónoma en entornos complejos.

En este contexto, el presente trabajo se centra en la implementación de un algoritmo de evasión de obstáculos para drones, específicamente diseñado para abordar la falta de información a priori sobre la ubicación de los obstáculos. Este enfoque reactivo permitirá que los drones tomen decisiones en tiempo real para evitar obstáculos de manera segura y eficiente, lo que es esencial para su aplicación en entornos urbanos, industriales y otros escenarios realistas.

La empresa ACSL (del inglés *Autonomous Control Systems Laboratory*), con sede en Japón, se ha destacado como líder en la investigación y desarrollo de soluciones de control autónomo para drones. ACSL ha identificado la necesidad crítica de mejorar la capacidad de los drones para evadir obstáculos de manera efectiva, lo que impulsará la seguridad de las operaciones y abrirá nuevas oportunidades de aplicación en una variedad de industrias.

La implementación exitosa de un algoritmo de evasión de obstáculos reactivo tiene el potencial de revolucionar la seguridad y la autonomía de los drones, lo que beneficiaría a una amplia gama de industrias, desde la entrega de paquetes hasta la inspección de infraestructuras críticas. Además, contribuirá al liderazgo continuo de ACSL en el campo de la tecnología de drones, consolidando su posición como un actor clave en la innovación y desarrollo de soluciones de vanguardia en Japón y a nivel internacional.

1.3. Objetivos

1.3.1. Objetivo general

Implementar un algoritmo de evitación de obstáculos para drones autónomos y realizar pruebas de dicha implementación tanto en entorno de simulación como en el campo.

1.3.2. Objetivos específicos

- Investigar el estado del arte de los algoritmos de evitación de obstáculos para drones autónomos.
- Implementar un algoritmo de evitación de obstáculos para drones autónomos sobre un ambiente de simulación y realizar pruebas en dicho ambiente.
- Ajustar la implementación del algoritmo seleccionado al hardware de un dron autónomo realizando las optimizaciones necesarias y realizar pruebas de campo de dicha implementación.

1.4. Estructura del trabajo

Este trabajo se encuentra dividido en seis capítulos de la siguiente manera:

El **Capítulo 2** describe la empresa receptora en donde se realizó este trabajo, el Laboratorio de Sistemas de Control Autónomo (ACSL, del inglés *Autonomous Control Systems Laboratory*), localizado en Tokio, Japón.

En el **Capítulo 3** se explican las bases teóricas de los principales algoritmos utilizados en este trabajo.

El **Capítulo 4** incluye la revisión del estado del arte sobre los métodos de evasión de obstáculos para drones autónomos.

El **Capítulo 5** muestra la implementación del algoritmo de evitación de obstáculos y la arquitectura utilizada.

En el **Capítulo 6** se muestran y se discuten los resultados obtenidos.

Finalmente en el **Capítulo 7** se presentan las conclusiones del presente trabajo, así como las recomendaciones para posibles trabajos futuros.

Capítulo 2

Descripción de la empresa receptora

La empresa donde se realizó este trabajo es el Laboratorio de Sistemas de Control Autónomos, ACSL (del inglés *Autonomous Control Systems Laboratory*), en la sede de Tokio, Japón. El logo de la organización se muestra en la Figura 2.1. ACSL fue fundada en el año 2013 y tiene como visión revolucionar la infraestructura social mediante la aplicación de tecnología robótica.



FIGURA 2.1: Logo de ACSL (*Autonomous Control Systems Laboratory*)

ACSL cuenta con distintos productos sobre los cuales se desarrollan distintas soluciones. En particular, ACSL domina el desarrollo de vehículos aéreos no tripulados, ofreciendo una gama amplia de productos todos desarrollados y ensamblados



FIGURA 2.2: Algunos productos de ACSL. **(a) PF2:** Un dron con capacidad autónoma de seis rotores utilizado para aplicaciones de entrega de paquetes (*delivery*), inspección y patrullaje en desastres naturales. **(b) AirTruck:** Un dron autónomo de seis rotores con capacidad de carga y vuelo superior utilizado para operaciones de logística. **(c) Fi4:** Un dron autónomo dedicado a la inspección de espacios confinados (tuberías, ductos de ventilación, entre otros). **(d) SOTEN:** Un dron pequeño de cuatro rotores dedicado a la fotografía aérea con capacidad autónoma utilizado para tareas de inspección, vigilancia y rescate.

en Japón. En la Figura 2.2 se muestran y describen algunos de estos productos, entre ellos SOTEN, el producto en el cual se implementó el presente trabajo.

Este trabajo se desarrolló bajo la supervisión del Equipo de Inteligencia de Máquinas (del inglés *Machine Intelligence Team*) de ACSL, contando con el Dr. Ing. Niklas Bergstroem, líder del Equipo de Inteligencia de Máquinas, como encargado de asesorar el desarrollo del trabajo. Gracias a los miembros del Equipo de Inteligencia de Máquinas de ACSL, este trabajo pudo ser desarrollado e implementado haciendo el uso de las plataformas y recursos disponibles.

Capítulo 3

Marco teórico

El presente capítulo tiene como intención exponer las bases teóricas necesarias para el desarrollo de este trabajo. Este capítulo se divide en cuatro secciones: la Sección 3.1 explica la teoría relacionada con el modelo de los vehículos aéreos no tripulados (UAV, del inglés *unmanned aerial vehicle*), en particular, los UAV de cuatro rotores; la Sección 3.2 describe el funcionamiento de los algoritmos utilizados para la obtención de mapas de profundidad a partir de un sistema de visión estereoscópica; la Sección 3.3 explica la teoría relacionada con las redes neuronales y Redes Neuronales Convolucionales (CNN, del inglés *Convolutional Neural Networks*); finalmente, la Sección 3.4 describe el funcionamiento de los protocolos de comunicación entre procesos utilizados en este trabajo.

3.1. Vehículos aéreos no tripulados de cuatro rotores

Un vehículo aéreo no tripulado de cuatro rotores, también llamado *Quadrotor* UAV (abreviado QUAV a efectos de este trabajo) es una aeronave no tripulada propulsada por cuatro rotores que es controlada por un operador en tierra. Los QUAVs pertenecen a la familia de los drones y su principal característica es que la configuración de los cuatro rotores les permite rotar y elevarse independientemente mediante control de la velocidad de rotación de los rotores [10]; el mismo

mecanismo les permite moverse omni-direccionalmente, así como también flotar en sitio (del inglés *hovering in place*). Debido a que su funcionamiento es enteramente dependiente del empuje producido los rotores, los QUAVs generalmente tienen un tiempo de vuelo máximo bastante reducido (menos de 30 minutos en promedio [10]), pues baterías de alta capacidad de carga introducen peso en el sistema que puede afectar considerablemente la capacidad de vuelo del vehículo; sobre esta misma linea, el límite de masa de carga de estos vehículos es generalmente bastante pequeño. Estas características de los QUAVs hacen que su uso sea atractivo para aplicaciones donde la agilidad es un factor importante, siempre y cuando la capacidad de carga necesaria sea relativamente baja y tiempos de vuelo inferiores a 30 minutos sean aceptables [10]. En el Capítulo 2, en la Figura 2.2 se muestra SOTEN, un QUAV diseñado por ACSL para aplicaciones de fotografía aérea (inspección, vigilancia, entre otras) y que sirve de plataforma física para el desarrollo del presente trabajo. Con la finalidad de comprender la estructura y condiciones de arquitectura de la plataforma física, a continuación se da una breve descripción de los componentes de un QUAV.

3.1.1. Componentes

La configuración de componentes general de los QUAVs, si bien bastante amplia y llena de excepciones, cuando se remiten a los QUAVs comerciales, se puede englobar en: Un conjunto de sensores, que se utilizan principalmente para localizar el vehículo en el entorno; una unidad de control de vuelo (FCU, del inglés *Flight Control Unit*), encargada de controlar la velocidad de rotación de los rotores; al menos una computadora a bordo (*Onboard PC* en inglés), que se encarga de realizar el procesamiento necesario para llevar a cabo tareas de alto nivel, como la evasión de obstáculos y el mapeo del entorno; y una estación de control en tierra (GCS, del inglés *Ground Control Station*) [10].

Sensores

Es de vital importancia que un QUAV tenga la capacidad de estimar su estado (orientación, altitud, posición y velocidad) utilizando solamente los componentes disponibles a bordo del vehículo. Para la estimación de orientación se utiliza una

unidad de mediciones inerciales [10] (IMU, del inglés *Inertial Measurement Unit*) que se compone de un acelerómetro y giroscopio, ambos midiendo sus cantidades con respecto al marco de referencia del cuerpo del vehículo. Adicionalmente, también se puede utilizar un magnetómetro (brújula), para obtener la información de la orientación del vehículo con respecto al norte magnético del planeta tierra. Utilizando la información del IMU y del magnetómetro, el FCU puede estimar la orientación del cuerpo del vehículo con un nivel de precisión aceptable [10].

Para la estimación de posición y velocidad se puede emplear localización visual utilizando algoritmos de localización y mapeo con cámaras estereoscópicas o LiDAR [10]. Estos algoritmos pueden llegar a tener limitaciones considerables cuando las condiciones de iluminación no son ideales y su rendimiento en entornos al aire libre es limitado debido a la uniformidad de las texturas del entorno [10]. Alternativamente se puede utilizar GPS (del inglés *Global Postioning System*) para la localización a partir de señales de satélite [10]; en este esquema la posición es estimada mediante triangulación entre el vehículo y un sistema de satélites, la velocidad es estimada utilizando el efecto *doppler* y la altitud se estima utilizando un barómetro (sensor de presión atmosférica) o un sistema sonar [10]. La localización por GPS es bastante efectiva para su uso en entornos al aire libre, pero no es posible utilizarlo en aplicaciones que naveguen en interiores debido al requerimiento de tener recepción de las señales de los satélites. Es común que un QUAV tenga acceso tanto a localización por visión como a localización por GPS, alternando entre ambas dependiendo de las condiciones de iluminación y recepción del satélite.

FCU

Este componente maneja la estimación del estado del QUAV a partir de las señales de los sensores, así como el cálculo de las acciones de control para mantener estabilidad durante el vuelo. El FCU ofrece distintos modos de control, desde asistencia de orientación, asistencia en posición y asistencia de velocidad; todos estos con la finalidad de que controlar el QUAV en alto nivel sea una tarea más sencilla tanto para un operador como para otro algoritmo de alto nivel (como la evasión de obstáculos, el caso de este trabajo).

Computadora a bordo

Ejecuta los algoritmos de alto nivel que requieren más poder de cómputo para su procesamiento, tales como algoritmos de localización por visión, inferencia con redes neuronales, procesamiento y *broadcasting* de imágenes, mapeo del entorno, entre otros. Algunas se equipan con unidades de procesamiento gráfico (GPU, del inglés *Graphics Processing Unit*) que contribuyen en la aceleración de las operaciones de procesamiento distribuido o de inferencia de redes neuronales, en ambos casos utilizando tecnologías como CUDA (Arquitectura Unificada de Dispositivos de Cómputo, del inglés: *Compute Unified Device Architecture*). La solución propuesta en el presente trabajo se implementa y ejecuta justamente dentro de la computadora a bordo del QUAV.

GCS

Estación de control en el suelo, permite que el operador en suelo coordine las operaciones que hace el QUAV, en otras palabras, sirve de interfaz de alto nivel entre el operador y las funciones implementadas en el QUAV. Usualmente se utiliza para planificar vuelos autónomos, monitorizar el estado de los sensores, realizar *streaming* de las imágenes de los sensores visuales del vehículo (cámaras), lanzar rutinas de control como aterrizaje o despegue automático, así como también instrucciones específicas de la aplicación (por ejemplo, entregar paquete para QUAV orientado a *delivery*).

En la Figura 3.1 se muestra el diagrama de comunicación general de los componentes de un QUAV, además de los componentes descritos anteriormente, se muestra la interacción con el radio control que es controlado por el operador, este además de encargarse de la transmisión de los comandos de operación vía señales de radio, se puede utilizar para operar el vehículo directamente o en conjunción con la GCS para orquestar operaciones complejas.

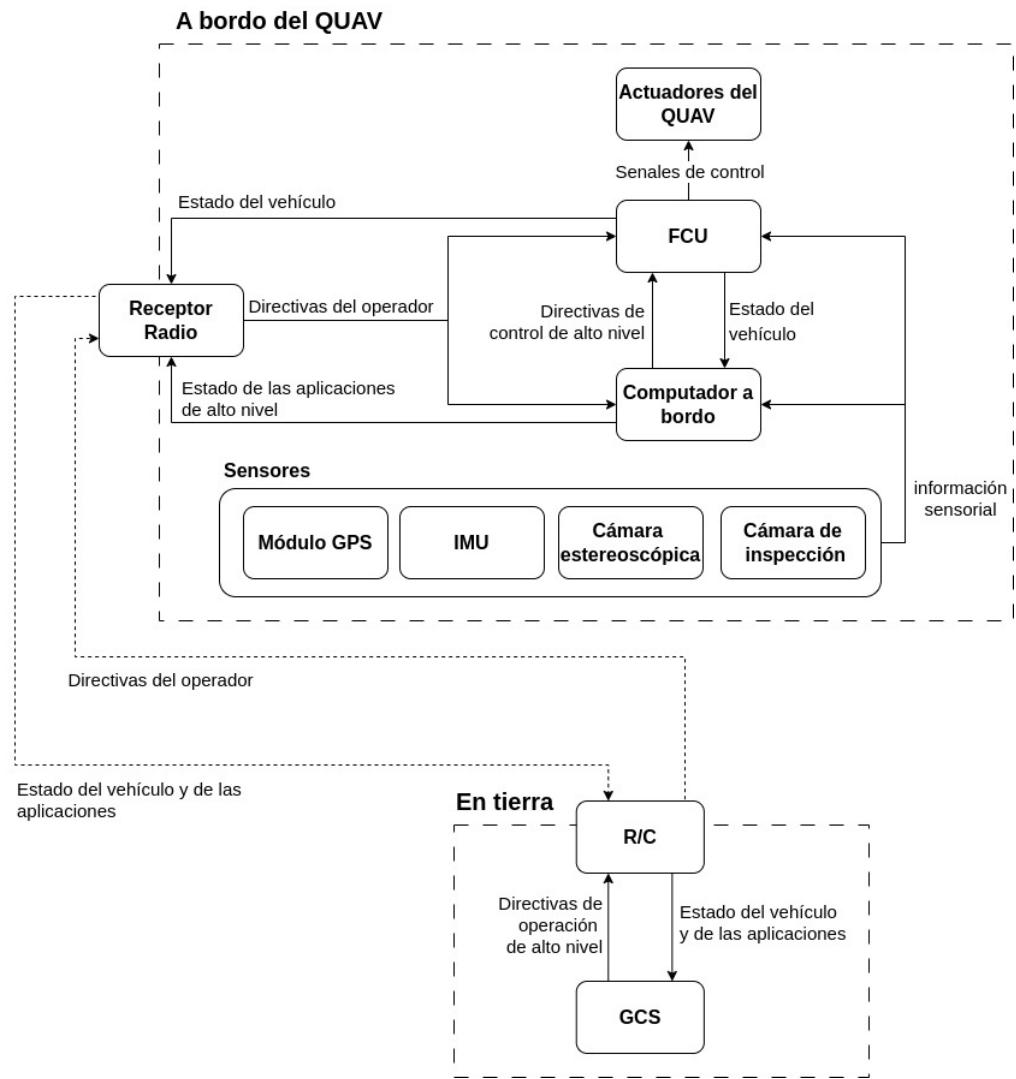


FIGURA 3.1: Diagrama de comunicación general del los componentes de un QUAV.

3.1.2. Información relevante del modelo del QUAV

En este trabajo no se hace una revisión completa del modelo cinemático y dinámico de un QUAV, sin embargo, es necesario discutir algunos conceptos que son utilizados en el desarrollo e implementación del algoritmo de evasión de obstáculos.

Sistemas de referencia

Existen dos sistemas de referencia de interés para el manejo de QUAV: el sistema de referencia inercial E , usualmente anclado a la superficie de la tierra; y el sistema de referencia del cuerpo del vehículo B , que se encuentra anclado a centro del marco del QUAV. Tal como se aprecia en la Figura 3.2, la rotación del sistema del cuerpo B , es representada por ϕ, θ, ψ [11], donde: ϕ es denominado rolido *roll* y representa la rotación con respecto al eje inercial x ; θ es denominado *pitch* y representa la rotación con respecto al eje inercial y ; y ψ es denominado *yaw* y representa la rotación con respecto al eje inercial z . Por convención, la velocidad lineal del vehículo V_B se expresa en el sistema B .

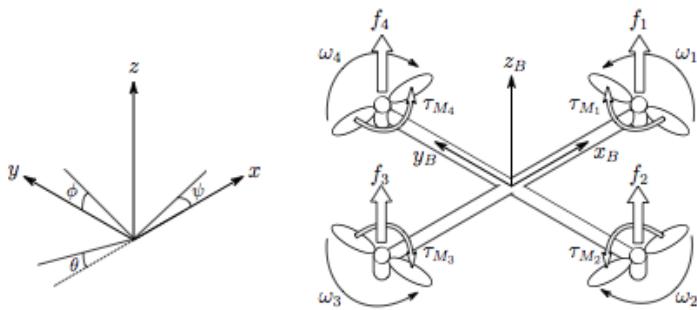


FIGURA 3.2: Diagrama de un QUAV [11].

Es necesario establecer una transformación entre B y E , para ello podemos derivar una matriz de rotación a partir de rotaciones sucesivas dadas por ϕ, θ, ψ , este tipo de transformación obtenida a partir de rotar ϕ, θ, ψ se denomina transformación por ángulos de Euler [12], se dice que ϕ, θ, ψ son los ángulos de Euler que representan la rotación de B con respecto al marco inercial E .

La matriz de transformación por ángulos de Euler se obtiene por realizar las rotaciones sucesivas en el orden *yaw*, *pitch*, *roll* [11], esto es:

- Rotar positivamente ψ alrededor del eje z .

$$R_\psi = \begin{bmatrix} \cos \psi & \sin \psi & 0 \\ -\sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (3.1)$$

- Rotar positivamente θ alrededor del eje y .

$$R_\theta = \begin{bmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{bmatrix} \quad (3.2)$$

- Rotar positivamente ϕ alrededor del eje x .

$$R_\phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & \sin \phi \\ 0 & -\sin \phi & \cos \phi \end{bmatrix} \quad (3.3)$$

Luego, la matriz de rotación R_E^B de que transforma vectores relativos a E a vectores relativos a B esta dada por [11]:

$$R_E^B = R_\phi R_\theta R_\psi \quad (3.4)$$

Esta matriz es orto-normal [12], por lo tanto la matriz inversa R_B^E que transforma vectores relativos a B a vectores relativos a E esta dada por:

$$R_B^E = (R_E^B)^{-1} = (R_E^B)^T \quad (3.5)$$

Estas matrices son bastante útiles pues permiten cambiar de sistema de referencia a voluntad, un ejemplo donde resultan particularmente útiles es para obtener la velocidad lineal del vehículo con respecto a marco de referencia inercial V_E :

$$V_E = R_B^E V_B \quad (3.6)$$

Encabezamiento

Se le llama encabezamiento (del inglés *heading*) al vector unitario que apunta en la dirección del eje y_B (Figura 3.2), este vector es importante pues determina la

dirección de movimiento preferida del QUAU ya que es hacia donde generalmente se orientan los sensores visuales (cámaras estereoscópicas, cámaras de inspección, entre otros). Es esta dirección la que se considera como “adelante” en el contexto de las aplicaciones que trabajan con QUAUs.

Modos de vuelo

Cada rotor del QUAU gira independientemente, la combinación de distintas velocidades de rotación de los rotores produce distintos patrones o modos de vuelo. La Figura 3.3 muestra los cuatro modos de vuelos básicos de un QUAU; cada uno produce una rotación sobre uno de los ángulos ϕ, θ, ψ .

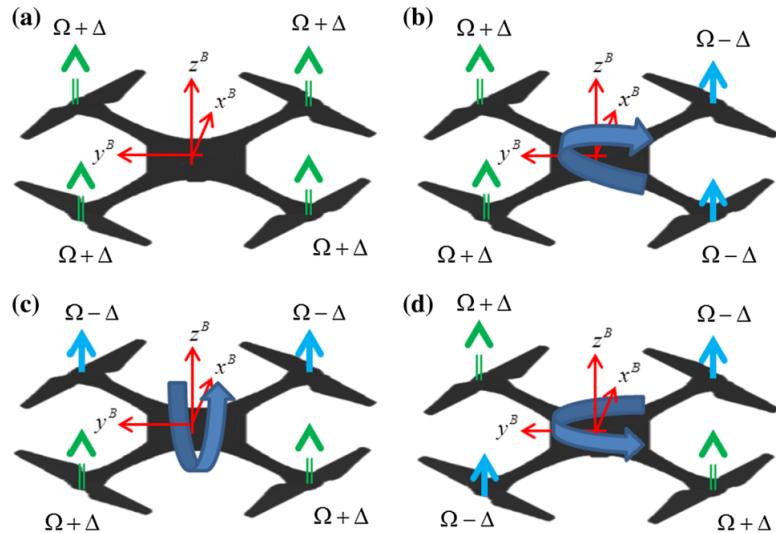


FIGURA 3.3: Modos de vuelo básicos de un QUAU [10]. (a) Ascenso vertical, (b) *Rolling*, (c) *Pitching* y (d) *Yawing*.

Rolling y *Pitching* producen rotaciones sobre los ángulos ϕ y θ respectivamente, y producen una translación asociada en la dirección de los rotores que giran a menor velocidad $\Omega - \Delta$. Efectivamente, *Rolling* y *Pitching* permiten mover al QUAU en las direcciones y_B y x_B . Por otro lado, *Yawing* produce una rotación sobre el ángulo ψ sin producir una translación asociada. *Yawing* es importante porque es el modo de movimiento que modifica directamente el encabezamiento del QUAU.

3.1.3. Localización GPS y coordenadas NED

Tal como se menciona en la Sección 3.1.1, es habitual que un QUAV tenga acceso tanto a localización por visión como a localización por GPS, sin embargo, a efectos de este trabajo, se asume que el QUAV siempre utiliza localización por GPS. Para aplicaciones de evasión de obstáculos locales, utilizar directamente las coordenadas GPS puede resultar poco práctico debido a la diferencia de escalas. En concreto, la diferencia entre dos puntos cercanos en coordenadas GPS puede ser muy pequeña, lo que puede producir errores numéricos en el cálculo de la distancia entre dos puntos. Es por esto, que es importante establecer un sistema de coordenadas equivalente que no tenga este problema.

En el modelo geodésico del mundo WGS64, el que se utiliza en este trabajo, un punto en coordenadas GPS vienen dadas por la tupla latitud, longitud y altitud (ϕ, λ, h) [13]. En la Figura 3.4, se muestra el elipsoide del modelo geodésico del mundo WGS64, donde se observa la coordenada (ϕ, λ, h) que representa el punto (x, y, z) .

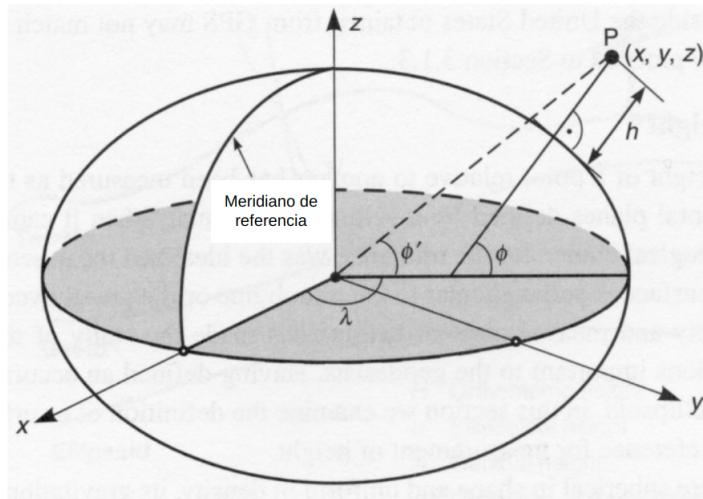


FIGURA 3.4: Modelo geodésico del mundo WGS64. Modificada de [13].

Dado S_W el conjunto de puntos en la superficie del elipsoide WGS64. Un sistema de referencia sencillo de representar y que puede modelar los puntos (ϕ, λ, h) en el modelo WGS64, es el sistema de coordenadas NED (Norte, Este y Abajo; del inglés *North, East, Down*) [14]. Este sistema es relativo a S_W y sus puntos se representan con la tupla (n, e, d) donde, dado un punto $o = (\phi_0, \lambda_0) \in S_W$:

- n es el desplazamiento en metros hacia el **norte** del elipsoide a partir de o .
- e es el desplazamiento en metros hacia el **este** del elipsoide a partir de o .
- h es profundidad del punto en la **dirección normal** del elipsoide en o .

El punto $(\phi_0, \lambda_0, 0)$ es el origen del sistema de coordenadas NED [14], y suele definirse como el punto latitud, longitud de donde despega el QUAV [10].

Dada $G : S_W \times S_W \longrightarrow \mathbb{R}^+ \times \mathbb{R}^+$ una función que recibe dos coordenadas $a = (\phi_a, \lambda_a)$ y $b = (\phi_b, \lambda_b)$ en S_W y retorna el par real (δ_n, δ_e) donde:

- δ_n es la longitud en metros del camino mas corto que va desde (ϕ_a, λ_a) hasta (ϕ_b, λ_a) sobre S_W .
- δ_e es la longitud en metros del camino mas corto que va desde (ϕ_a, λ_a) hasta (ϕ_a, λ_b) sobre S_W .

El Algoritmo 1 muestra cómo transformar un punto (ϕ, λ, h) en WGS64 a NED.

Algoritmo 1: Pseudo-código del algoritmo de transformación de coordenadas WGS64 a coordenadas NED

Datos: $G, (\phi, \lambda, h), (\phi_0, \lambda_0) \in S_W$

Resultado: (n, e, d)

- 1 $(\delta_n, \delta_e) = G((\phi_0, \lambda_0), (\phi, \lambda))$
 - 2 $n = \text{sgn}(\phi - \phi_0) \cdot \delta_n$
 - 3 $e = \text{sgn}(\lambda - \lambda_0) \cdot \delta_e$
 - 4 $d = -h;$
 - 5 **devolver** (n, e, d)
-

Donde sgn es la función signo. G usualmente es implementada utilizando la proyección de Mercator [15] y existen una amplia selección de librerías de código abierto que la implementan.

3.2. Visión estereoscópica

El objetivo de los algoritmos de visión estereoscópica es obtener información tridimensional de una escena a partir de dos imágenes con distintas observaciones.

Dado que la posición relativa entre las dos cámaras y los parámetros intrínsecos de cada una son conocidas, es posible obtener la información de profundidad de la escena utilizando relaciones geométricas entre las coordenadas tridimensionales y sus proyecciones en ambas imágenes [16]. Para que esto sea posible, es necesario que ambas imágenes estén rectificadas, esto es, que no tengan distorsión de lente y que se encuentren en el mismo plano; esto con el objetivo de que la diferencia entre una imagen y la otra sea un desplazamiento horizontal. La Figura 3.5 muestra un ejemplo del proceso de rectificación de un par de imágenes estéreo.

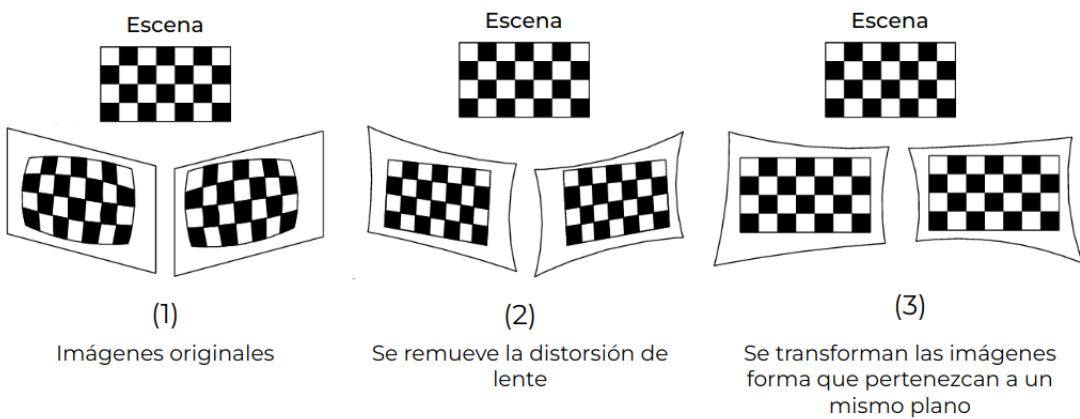


FIGURA 3.5: Ejemplo del proceso de rectificación de un par de imágenes estéreo.
Modificada de [17].

Con las imágenes rectificadas, el siguiente paso es obtener el desplazamiento horizontal entre las correspondencias píxel a píxel entre las dos imágenes, este desplazamiento se denomina disparidad. El arreglo bidimensional de la disparidad píxel a píxel entre dos imágenes estéreo se denomina mapa de disparidad, los mapas de disparidad nos permiten representar la información de la disparidad de todos los píxeles de una manera más accesible, ya que se puede interpretar como un imagen, tal como se muestra en la Figura 3.6.

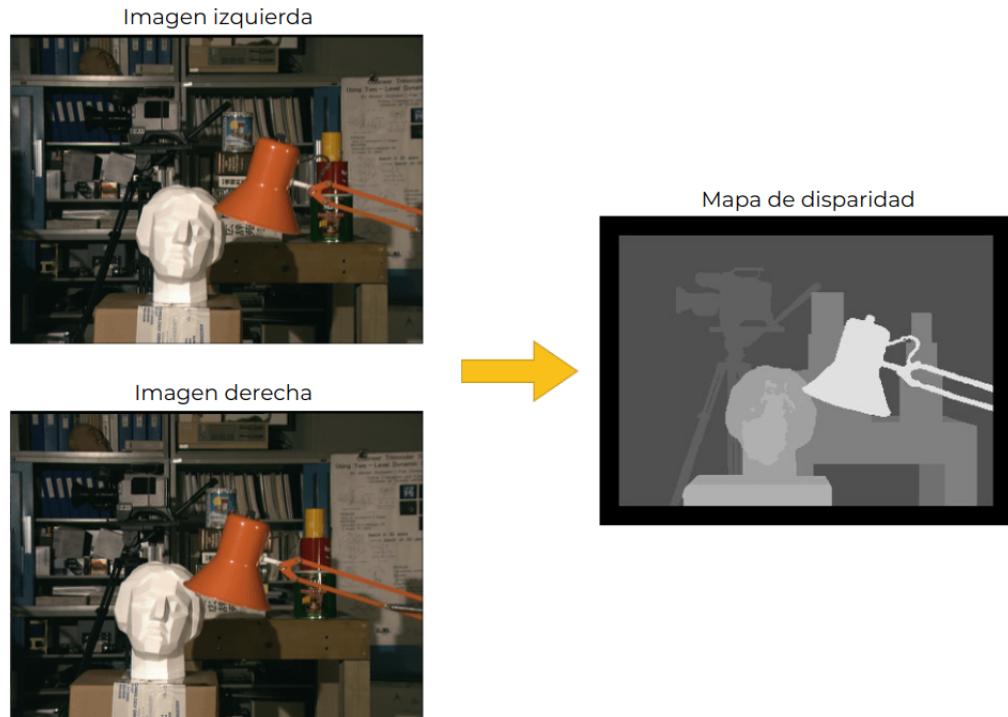


FIGURA 3.6: Ejemplo de la representación de un mapa de disparidad como una imagen¹.

Para calcular un mapa de disparidad hace falta establecer correspondencias píxel a píxel entre el par de imágenes estéreo. Un método para establecer estas correspondencias es el Emparejamiento de Bloques [18] (BM, del inglés *Block Matching*). El BM consiste en obtener correspondencias entre imágenes estéreo rectificadas, tomando bloques de píxeles en la primera imagen y buscando los bloques más similares en la segunda imagen; la búsqueda se realiza desplazando el dominio del bloque únicamente de forma horizontal a partir de la misma fila del bloque de la primera imagen; el criterio de búsqueda involucra una función de desemejanza entre bloques de píxeles, usualmente la Suma de Diferencias Absolutas (SAD, del inglés *Sum of Absolute Differences*), y se seleccionan dos bloques como correspondientes si el valor de la función de desemejanza es mínimo local a lo largo de los posibles desplazamientos a partir de la posición inicial del bloque. En la Figura 3.7 se ilustra un ejemplo de la correspondencia establecida por BM, y se observa el valor de la función de desemejanza en función de los desplazamientos

¹Modificada de: <https://www.baeldung.com/cs/disparity-map-stereo-vision>

(disparidad), el bloque seleccionado es aquel cuyo valor de la función de desempeño sea mínimo local. BM permite obtener un valor de disparidad para cada píxel; finalmente, con este resultado se puede construir un mapa de disparidad.

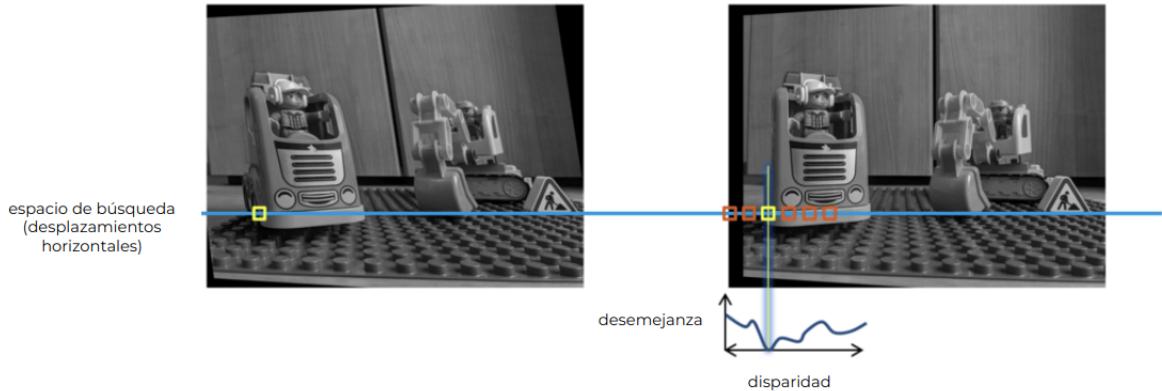


FIGURA 3.7: Ejemplo del Emparejamiento de Bloques para establecer correspondencias entre un par de imágenes estéreo rectificadas².

Una vez que se tiene el mapa de disparidad, es posible obtener la información de profundidad tridimensional de cada píxel utilizando relaciones geométricas, en particular, triangulación. En la Figura 3.8 se muestra el modelo de un sistema de visión estereoscópica de dos cámaras paralelas C y C' . En esta se observa: la distancia b , que es la distancia entre las cámaras; la distancia f , que es la distancia focal, es decir, la distancia entre el lente y el sensor; y un punto 3D en la escena, que corresponde al par de correspondencia píxel píxel u, u' con disparidad $d = (u_x - u'_x)$. Los valores de b y f son determinados por el tipo y configuración de las cámaras, y se calculan mediante un proceso de calibración. Una vez conocidos, la profundidad D del punto 3D en la escena con respecto al par estéreo se calcula utilizando la Ecuación 3.7.

²Modificada de: <https://www.andreasjakl.com/understand-and-apply-stereo-rectification-for-depth-maps-part-2/>

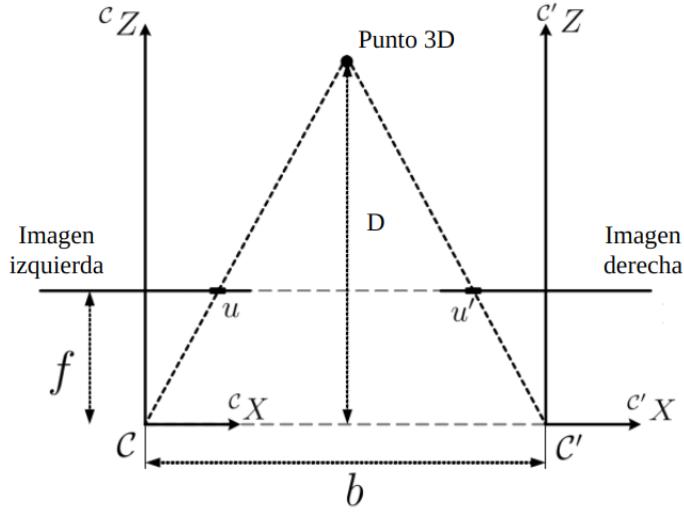


FIGURA 3.8: Modelo de un sistema de visión estereoscópica de dos cámaras paralelas C y C' . Modificada de [17].

$$D = \frac{f \cdot b}{u_x - u'_x} = \frac{f \cdot b}{d} \quad (3.7)$$

Si se calcula la profundidad D para cada correspondencia representada en el mapa de disparidad se obtiene un mapa de profundidad que codifica la información tridimensional de la escena, dicho mapa de profundidad también se puede interpretar en una imagen de manera análoga a lo que se observa en la Figura 3.6.

3.3. Redes neuronales

Una red neuronal es un modelo matemático que trata de imitar el funcionamiento del cerebro humano para resolver problemas o tareas, donde las unidades básicas son las neuronas. Cada neurona recibe datos de entrada, realiza cálculos con ellos y produce una salida. La salida de una neurona se convierte en la entrada de otras neuronas, creando así una red de procesamiento. Tal como sus equivalentes biológicos, las redes neuronales son capaces de representar aprendizaje, así como adaptarse a través de la experiencia. Durante un proceso denominado entrenamiento, estas redes ajustan las conexiones entre sus neuronas, fortaleciendo o debilitando ciertas conexiones en función de la retroalimentación proporcionada

por los datos observados. A medida que la red es expuesta a un mayor número de ejemplos y recibe retroalimentación constante, su capacidad para reconocer patrones, tomar decisiones o resolver problemas en esa tarea se fortalece.

Matemáticamente, una red neuronal es una serie de nodos interconectados, cada nodo recibe entradas de otros nodos y su respuesta se utiliza de entrada para otros nodos. Cada conexión tiene asociada un peso, que es un número real; cada nodo multiplica sus entradas con el respectivo peso de la conexión, suma los productos y le aplica una función de activación [19]; la función de activación es una función matemática simple que modela el proceso de disparo o activación de una neurona biológica. La Figura 3.9 ilustra el funcionamiento de un nodo dentro de una red neuronal.

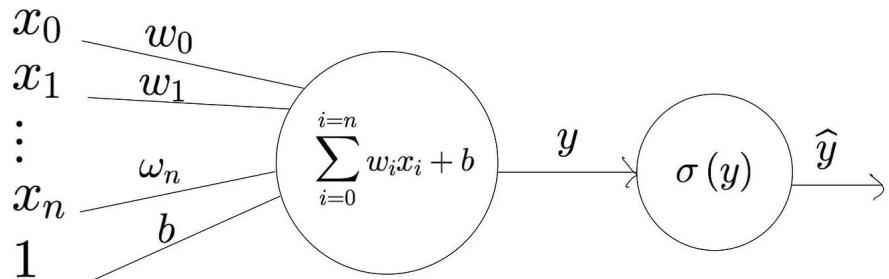


FIGURA 3.9: Funcionamiento de un nodo de una red neuronal³. n es el número entradas del nodo. x_i son las entradas del nodo y w_i son los pesos asociados a sus conexiones, con $0 \leq i \leq n$. Luego, σ es la función de activación y b es un peso asociado a una entrada constante, usualmente es llamado sesgo.

La motivación matemática de la función de activación es introducir no linealidad a la salida del nodo, para ello existen una gama de funciones de activación ampliamente utilizadas, entre estas se encuentran: la función tangente hiperbólica, la función sigmoide y las Unidades Lineales Rectificadas (ReLU, del inglés *Rectified Linear Units*), la Ecuación 3.8, la Ecuación 3.9 y la Ecuación 3.10 respectivamente muestran las definiciones de las funciones mencionadas anteriormente.

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (3.8)$$

³Obtenida de: [https://medium.com-analytics-vidhya/perceptron-learning-from-discrete-to-continuous-02-b16ddf9e5ab6](https://medium.com.analytics-vidhya/perceptron-learning-from-discrete-to-continuous-02-b16ddf9e5ab6)

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad (3.9)$$

$$\text{ReLU}(x) = \begin{cases} 0 & \text{si } x < 0 \\ x & \text{si } x \geq 0 \end{cases} \quad (3.10)$$

Las redes neuronales se pueden estructurar por capas, cada capa tiene una serie de nodos que reciben sus entradas de todos los nodos de la capa anterior y a su vez sirven de entrada todos los nodos de la capa superior; no existen conexiones entre nodos que pertenezcan a una misma capa y no existen capas que reciban como entrada a los nodos de alguna capa superior. Cuando una red neuronal se estructura por capas de la forma descrita anteriormente se le denomina Red Neuronal de Alimentación hacia adelante [20], FFNN del inglés *Feed-forward Neural Network*.

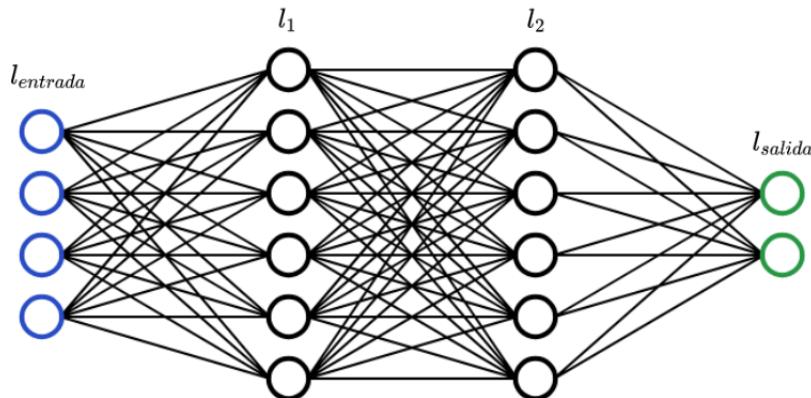


FIGURA 3.10: Ejemplo de una FFNN con dos capas ocultas⁴. l_{entrada} y l_{salida} son las capas de entrada y de salida respectivamente. l_1 y l_2 son las capas ocultas, también llamadas capas completamente conectadas, FC del inglés *Fully Connected*.

Un ejemplo de una FFNN de muestra en la Figura 3.10. Las FFNN tienen una capa de entrada y una de salida (l_{entrada} y l_{salida} en la Figura 3.10), la capa de entrada recibe el estímulo inicial de la red, mientras que la capa de salida retorna la respuesta de la red al estímulo. Además, las FFNN se caracterizan por

⁴Modificada de: <https://victorzhou.com/series/neural-networks-from-scratch/>

el número de capas FC (l_1 y l_2 en la Figura 3.10), mientras más capas ocultas, se dice que la red es más “profunda”. El conjunto de los pesos de los nodos de una red se denomina θ , mientras que un peso en particular se denomina w_j donde j es el índice de la transición asociada al peso. Al modificar pesos θ se modifica el comportamiento de la red ante un estímulo; diferentes combinaciones de valores de θ producen comportamientos distintos; así, la representación del “conocimiento” de una red neuronal se encuentra almacenado en sus pesos θ . Mediante algoritmos de aprendizaje automático se pueden modificar sistemáticamente los pesos de la red neuronal para que se adapte a una tarea en específica, este proceso de adaptación también es denominado entrenamiento [19].

3.3.1. Retro-propagación

El algoritmo de retro-propagación (del inglés *backpropagation*) es un algoritmo de aprendizaje supervisado que se utiliza para entrenar redes neuronales. Este algoritmo ajusta los pesos de las conexiones entre los nodos de la red neuronal de manera que la red pueda realizar una tarea en específico (clasificación o regresión por ejemplo). El algoritmo de retro-propagación utiliza el denominado descenso de gradiente (del inglés *Gradient Descent*), el cual es un método utilizado para minimizar una función de pérdida que cuantifica el error entre las predicciones de la red y los valores objetivos asociados a la tarea seleccionada.

Descenso de gradiente

El aprendizaje automático supervisado presenta iterativamente un conjunto de patrones de entrada (X) y de salida (Y) al modelo, en este caso una red neuronal. En cada iteración se modifican los pesos θ de acuerdo a una regla de actualización cuyo objetivo es que la salida de la red se asemeje a la salida deseada [19]. Para realizar estos cambios, la regla de actualización debe cuantificar que tan bien la red neuronal modela los datos que se le presentan; es por esto que se define una función de costo o error E que dado un dato de entrada x_k , el vector deseado y_k y los pesos θ de la red, retorna $E(x_k, y_k, \theta)$ que cuantifica que tan similar a y_k es la salida de la red (parametrizada por θ) cuando se le pasa como entrada x_k . El objetivo es que la red sea capaz de modelar la relación entre los patrones de

entrada X y salida Y , en otras palabras, se quiere minimizar la función descrita en la Ecuación 3.11; esta función se denomina función de pérdida, donde $|X| = |Y|$ y x_k, y_k son el patrón entrada salida número k con $x_k \in X \wedge y_k \in Y$. Finalmente el problema de minimización tratado en este trabajo es el descrito en la Ecuación 3.12.

$$L(X, Y, \theta) = \sum_{k=1}^{|X|} E(x_k, y_k, \theta) \quad (3.11)$$

$$\min_{\theta} L(X, Y, \theta) \quad (3.12)$$

Tal como su nombre lo indica, el descenso de gradiente en cada iteración se mueve en la dirección del gradiente, esto lo hace siguiendo la regla de actualización:

$$\theta = \theta - \alpha \nabla_{\theta} L(X, Y, \theta) \quad (3.13)$$

Donde $0 < \alpha < 1$ es un número real llamado "tasa de aprendizaje"; α se encarga de que la regla de actualización no produzca cambios drásticos que puedan dificultar la convergencia del algoritmo. Luego, la regla de actualización de cada peso individual w_j viene dada por:

$$w_j = w_j - \alpha \frac{\partial L(X, Y, \theta)}{\partial w_j} \quad (3.14)$$

Estas regla de actualización se aplica iterativamente sobre el conjunto de patrones entrada salida (también llamado conjunto de entrenamiento) hasta alcanzar un criterio de convergencia, como un valor umbral de la función de pérdida o un número de iteraciones. Justamente esta es la descripción del algoritmo del descenso de gradiente.

Dado un conjunto de datos de entrada X , un conjunto de valores deseados Y , la tasa de aprendizaje α y una función booleana $Q(\theta)$ que sirve como criterio

de convergencia. El Algoritmo 2 muestra el pseudo-código del proceso de retro-propagación usando descenso de gradiente.

Algoritmo 2: Pseudo-código del algoritmo de retro-propagación usando descenso de gradiente

Datos: X, Y, α, Q

Resultado: θ

- 1 Inicializar los pesos θ en ceros o en valores aleatorios cercanos a cero
 - 2 **mientras** $\neg Q(\theta)$ **hacer**
 - 3 **para** $w_j \in \theta$ **hacer**
 - 4 $w_j = w_j - \alpha \frac{\partial L(X, Y, \theta)}{\partial w_j}$
-

Sub-ajuste y sobre-ajuste

La arquitectura de una red neuronal (número de capas y de nodos por capa) y el criterio de convergencia del descenso de gradiente deben elegirse cuidadosamente para evitar sesgos que puedan afectar negativamente el rendimiento del modelo. Hay dos tipos de errores a los que puede conducir un sesgo inapropiado: sub-ajuste y sobre-ajuste. El sub-ajuste (del inglés *underfitting*) ocurre cuando la arquitectura de la red es demasiado simple para representar la relación entre los datos de entrada y los valores deseados del conjunto de entrenamiento, o cuando el criterio de convergencia es demasiado temprano. Por otro lado, el sobre-ajuste (del inglés *overfitting*) ocurre cuando la arquitectura de la red es muy compleja o se realizan demasiadas iteraciones del descenso de gradiente, lo que hace que el modelo se ajuste demasiado al conjunto de datos de entrenamiento. Los modelos con sub-ajuste o sobre-ajuste no generalizan correctamente y, por lo tanto, no serán buenos al momento de realizar inferencia sobre datos que estén más allá de lo que se observó durante el entrenamiento.

3.3.2. Redes neuronales convolucionales

Cuando se desea utilizar una imagen como entrada de una red neuronal, cada píxel de la imagen es una entrada de la red; considerando que incluso imágenes de resoluciones relativamente bajas contienen un número de píxeles en el orden de los miles y hasta tres canales de color, si se utilizara una red con solo capas FC,

la cantidad de pesos a entrenar sería muy elevada debido a la alta cantidad de entradas; una imagen a color de 300 píxeles de ancho por 300 de alto necesitaría $300 \cdot 300 \cdot 3 = 270000$ nodos de entrada, colocando el número de pesos de la red en el orden los cien miles con tan solo una capa, por esto el uso de redes neuronales con solo capas FC no es conveniente para trabajar con imágenes.

Las redes neuronales convolucionales (CNN, del inglés *Convolutional Neural Networks*) son un tipo de red neuronal FFNN, principalmente utilizadas para el procesamiento digital de imágenes, que debido a su estructura permiten reducir la cantidad de pesos necesarios a entrenar; habilitando un proceso de entrenamiento más sencillo y a mayor velocidad [21]. Las CNN incorporan tres tipos de capas: las capas convolucionales, las capas de submuestreo (del inglés *pooling layers*) y capas FC generalmente al final de la red.

Las CNN suelen componerse de dos partes: un denominado extractor de características, conformado por capas convolucionales y capas de submuestreo; y parte FC cuyo objetivo varia dependiendo de la tarea que se requiere de la red (regresión o clasificación por ejemplo). En la Figura 3.11 se muestra un ejemplo de la arquitectura de una CNN.

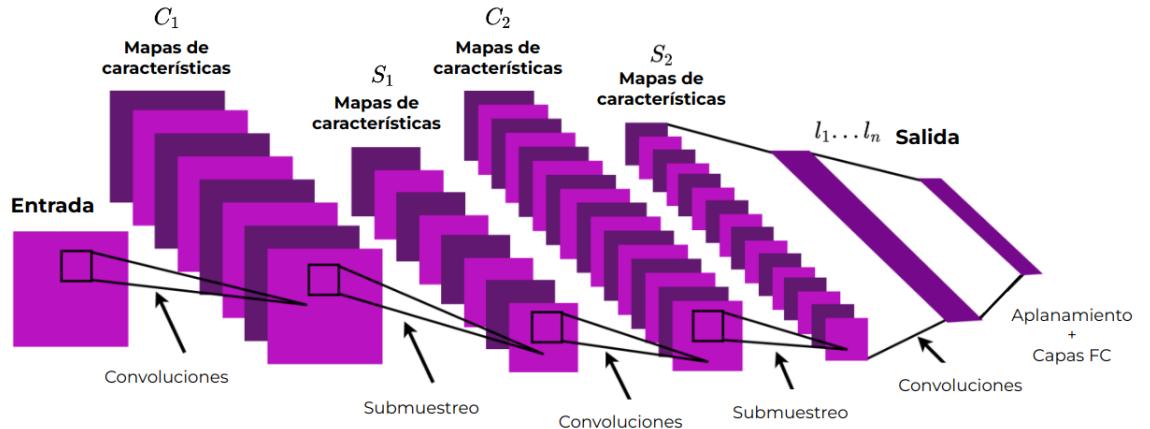


FIGURA 3.11: Ejemplo de la arquitectura de una CNN [21]. C_1 y C_2 son los resultados de capas convolucionales, S_1 y S_2 son los resultados de capas de submuestreo. $l_1 \dots l_n$ son n capas FC. Entre S_2 y l_1 se realiza un proceso de aplanamiento, que concatena los mapas de características de S_2 para producir un vector de características que sirve de entrada para l_1 .

La intuición detrás de las capas convolucionales es la de la aplicación de filtros

en espacio sobre imágenes para la extracción de mapas que codifican información de la entrada [22] [23]. Las capas convolucionales realizan operaciones de convolución discreta a la matriz (imagen) de entrada, utilizando filtros también llamados *kernels*. El resultado de una capa convolucional es una matriz en donde cada componente corresponde a la multiplicar uno de los *kernels* con una región de la imagen de entrada, sumar los resultados y opcionalmente aplicar una función de activación. Estas capas se usan para la detección de características, es por eso que a la salida de estas capas se le da el nombre de “mapa de características”. El ancho y el alto de este mapa dependen de las dimensiones del *kernel*, mientras que su número de canales es igual al número de *kernels* utilizados por la capa.

Las capas de submuestreo se tienen la finalidad de filtrar características poco relevantes de la entrada, su salida es un mapa de características donde cada componente es el resultado de una operación de reducción de dimensionalidad de una región del mapa de entrada. La operación de reducción de dimensionalidad usualmente es de uno de los siguientes tipos: promedio de la región seleccionada, en cuyo caso se le llama submuestreo por promedio (del inglés *average pooling*); o máximo de la región seleccionada, llamando entonces submuestreo por máximo (del inglés *max pooling*). El efecto de reducción de dimensionalidad ayuda a reducir el número de operaciones convolucionales de capas posteriores, haciendo el proceso de entrenamiento mas sencillo y veloz.

Entre el extractor de características y la parte FC se hace un proceso de aplanoamiento (del inglés *flattening*), cuyo resultado es la concatenación de los mapas de características para producir un vector de características que sirve de entrada para las capas FC de la red. La sección FC recibe el vector de características y genera una salida cuyas dimensiones dependen de la tarea en la cual se esta entrenando la red, estas capas FC siguen la misma estructura descrita en la Sección 3.3.

3.4. Comunicación entre procesos

La comunicación entre procesos, comúnmente abreviada como IPC (por sus siglas en inglés, Inter-Process Communication), se refiere al conjunto de técnicas y mecanismos utilizados para permitir que diferentes procesos intercambien

información entre sí de forma asíncrona. Esta comunicación es fundamental en escenarios donde varias aplicaciones deben colaborar para lograr una funcionalidad más amplia. A través de IPC, los procesos pueden compartir datos, sincronizar su ejecución, coordinar tareas y garantizar la seguridad y eficiencia en la gestión de recursos compartidos. Algunos de los métodos de IPC utilizados comúnmente son las tuberías (del inglés *pipes*), los *sockets* y la memoria compartida [24].

Tuberías

Una tubería o *pipe* es un método de comunicación unidireccional que permite la transferencia serial de información desde el proceso escritor al proceso lector. La capacidad de transferencia de datos de un tubería esta limitada por un búfer de tipo primero que entra, primero que sale (FIFO, del inglés *First In, First Out*); cuando el escritor envía datos estos se agregan al búfer, de forma análoga cuando el lector procesa datos estos se eliminan del búfer; si el escritor escribe más rápido de lo que el lector consume datos y si el búfer de la tubería está lleno, el escritor se bloqueará hasta que haya más capacidad disponible. De manera similar, si el lector lee cuando la tubería está vacía, el lector se bloqueará [24]. Con una tubería se pueden implementar esquemas de comunicación de mayor nivel de tipo productor-procesador (Pub/Sub, del inglés *Publisher/Subscriber*) siempre y cuando solo exista solo un productor y solo un procesador.

Sockets

Un *socket* es un mecanismo de comunicación bidireccional que se puede utilizar para comunicar un proceso con otro proceso, bien sea dentro de la misma máquina o en una máquina diferente [24]. Los datos enviados a través de un *socket* se dividen en fragmentos llamados paquetes. Un protocolo, como TCP/IP, especifica cómo se transmiten estos paquetes a través del *socket*. Un *socket* se identifica de manera única mediante una combinación de la dirección IP de la máquina y un número de puerto. La bidireccionalidad de los *sockets* permite construir esquemas de comunicación a alto nivel sin restricciones de cardinalidad (tales como Pub/Sub con muchos productores y muchos receptores), si bien la implementación de estos

esquemas puede ser complicado, existen múltiples librerías de implementaciones estables de estos, tales como ZeroMQ [25].

Memoria compartida

La memoria compartida permite a dos o más procesos acceder al mismo espacio de memoria, este espacio es mapeado al espacio de direcciones de memoria de cada uno de los procesos participantes [24]. Dado que esta comunicación es similar a cualquier otra referencia de memoria, no implica ninguna llamada al sistema ni latencias inducidas por protocolos. Por lo tanto, generalmente la memoria compartida ofrece latencias muy bajas. Sin embargo, el sistema no proporciona ningún mecanismo sincronización implícito para los accesos a la memoria compartida; esto significa que el uso descuidado de la memoria compartida puede producir condiciones de carrera [24]. El uso de mecanismos de primitivas como los semáforos o los *mutex* pueden habilitar el uso de la memoria compartida sin riesgo de inconsistencia en los datos, pero su implementación es responsabilidad del programador.

3.5. Resumen

En este capítulo se establecen las bases teóricas necesarias para el desarrollo del trabajo, incluyendo el modelado de vehículos aéreos no tripulados de cuatro rototores y el procedimiento para la obtención de mapas de profundidad a partir de un sistema de cámaras estereoscópicas. Adicionalmente, se explican los fundamentos de las redes neuronales, su proceso de entrenamiento y sus variantes enfocadas al procesamiento de imágenes, las CNN. Finalmente, se describieron algunos métodos de comunicación entre procesos que nos serán útiles durante el desarrollo del presente trabajo.

En el siguiente capítulo, se hace una revisión del estado del arte de los métodos de evasión de obstáculos para drones y se profundiza en *Learning high-speed flight in the wild*, un método bastante atractivo para el desarrollo de este trabajo en el contexto de la plataforma disponible para su implementación.

Capítulo 4

Antecedentes

El problema de evasión de obstáculos es un problema de planificación local de caminos. Entre las investigaciones que abordan este problema, existen dos variantes, las que dependen de la información global disponible previamente al momento de la planificación del camino, y aquellos desarrollos que presentan soluciones que dependen únicamente de la información de los sensores disponibles a bordo del vehículo, al momento de la planificación del camino. En este capítulo se hace una revisión del estado del arte de ambas variantes mencionadas anteriormente, explorando la primera variante en la Sección 4.1, la segunda variante en la Sección 4.2, y profundizando en *Learning high-speed flight in the wild* en la Sección 4.3, que es el principal antecedente del desarrollo del presente trabajo.

4.1. Desarrollos que dependen de la información global del entorno

Este tipo de investigaciones separan sus algoritmos en dos etapas, una etapa fuera de línea (del inglés *offline*) que se ejecuta antes de que el UAV comience el vuelo, y una etapa en línea (del inglés *online*) que se ejecuta durante la ejecución del vuelo. La etapa fuera de línea considera la información global del entorno, esto es, modelos del entorno, que pueden expresados por grafos, nube de puntos, entre otros; información del estado del entorno, que describen los eventos que ocurren en

el entorno en el tiempo, como listas de vuelos consecutivos o posición de obstáculos programados en función del tiempo. La etapa en línea depende de los artefactos generados por la etapa fuera de línea y generalmente permiten que la complejidad de ejecución de la etapa en línea sea ligera. Sin embargo, la dependencia de la etapa fuera de línea en la información global del entorno, hace que este tipo de soluciones no sean viables en aplicaciones donde se desconoce a priori la información global del entorno. A continuación se describen brevemente algunas investigaciones que proponen este tipo de métodos.

En el desarrollo propuesto en [26] se propone un método que genera en la etapa fuera de línea un camino libre de colisiones utilizando un algoritmo basado en A* que considera el grafo del espacio aéreo del entorno, una lista de sectores restringidos por situaciones climáticas o de regulaciones y la lista de los caminos en ejecución de otros UAV. Los caminos producidos por este algoritmo evitan obstáculos estáticos (en forma de sectores restringidos), así como también obstáculos dinámicos (en forma de caminos en ejecución). Este método tiene la ventaja de que si se mantiene correctamente la lista de caminos en ejecución, es posible orquestar la navegación de múltiples UAV en el entorno sin producir colisiones con obstáculos. Este tipo de métodos basados en A* tienen la desventaja de su etapa en línea no posee la capacidad de adaptarse a obstáculos no previstos en la etapa fuera de línea, en otras palabras, el camino se vuelve completamente estático una vez que el UAV comienza el vuelo [27]; sin mencionar que su funcionamiento depende de que exista un grafo que modele el entorno correctamente.

Otros métodos tales como los presentados por Lifen et al. (2016) [28] utilizan campos potenciales artificiales para planificar caminos libres de colisión. La planificación de caminos por campos potenciales artificiales conciben al vehículo como una partícula inmersa en un campo de potencial cuyas variaciones locales reflejan la estructura del entorno [29], donde potenciales atractivos representan posiciones objetivo y potenciales repulsivos representan obstáculos, luego la trayectoria del vehículo es determinada iterativamente siguiendo el sentido de la fuerza producida por el potencial. En [28] se utiliza este concepto para producir caminos libres de colisión para la navegación de un UAV dentro de un entorno donde la ubicación de los obstáculos es conocida antes de comenzar el vuelo. La etapa fuera de línea en [28] calcula el camino a ejecutar siguiendo el potencial establecido por la información del entorno (obstáculos y posiciones objetivo) y la etapa en línea ejecuta

este camino. Si bien el modelo propuesto en [28] no lo hace, es posible que la etapa en línea recalcule localmente el camino inicial dado que se haya detectado algún obstáculo no previsto, sin embargo, esto introduce latencia en el sistema que pueden afectar la autonomía del vuelo. Este tipo de métodos basados en campos potenciales artificiales tienen la desventaja de que pueden hacer que el vehículo alcance mínimos locales de potencial que no permiten que el vehículo alcance su objetivo. Adicionalmente, estos métodos también son propensos a producir oscilaciones en la trayectoria, lo hace que la navegación no sea óptima y se desperdicie tiempo de vuelo, que justamente es uno de los recursos más importantes para los UAV.

Por otro lado, Zhang et al. (2019) [3] introduce P-CAL (Canales Alternativos Pre-calculados, del inglés *Pre-computed Alternative Lanes*). Este método, en la etapa fuera de línea, dado un nube de puntos previamente calculada que codifica la información tridimensional del entorno, genera una colección de distintos caminos libres de colisión alternativos denominados canales (del inglés *lanes*); entre cada canal adicionalmente se genera un camino de transición que permite cambiar de canal de forma fluida. Luego, en la etapa en línea, se alterna entre los distintos canales dependiendo de la presciencia de obstáculos no previstos. En otras palabras, P-CAL calcula un conjunto de caminos alternativos y luego alterna entre estos para evitar obstáculos imprevistos en el entorno. La detección de los obstáculos no previstos se hace comparando la información tridimensional predefinida del entorno con la lectura sensorial de un sensor LiDAR. La Figura 4.1 muestra la nube de puntos del entorno previamente calculada, el resultado de la generación de los canales y caminos de transición; y el camino resultante de alternar canales para evadir obstáculos no previstos. El requerimiento de poseer una nube de puntos del entorno previo al vuelo del UAV, así como también la necesidad de utilizar un LiDAR para detectar obstáculos no previstos, hacen que este método no sea práctico para aplicaciones generales de evasión de obstáculos en donde no se tiene información previa del entorno o aquellas cuyos UAV no tengan la capacidad de utilizar un sensor LiDAR.

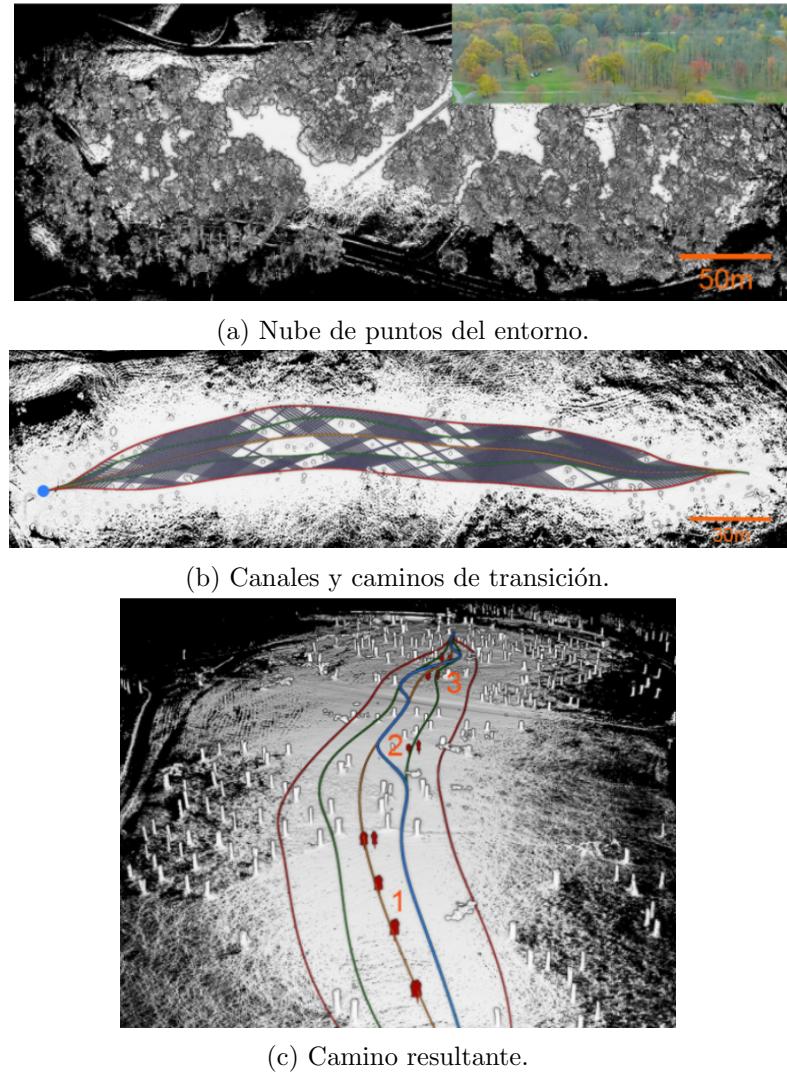


FIGURA 4.1: P-CAL, Canales Alternativos Pre-calculados [3]. **(a)** Nube de puntos del entorno previamente calculada. **(b)** Canales (amarillo, verde y rojo) y caminos de transición (azul) generados en la etapa fuera de línea. **(c)** Camino resultante (azul) de alternar entre canales para la evasión de los obstáculos no previstos 1, 2 y 3.

4.2. Desarrollos que solo dependen de la información disponible a bordo del vehículo

En la sección anterior se exploran algunos desarrollos que separan sus algoritmos en dos etapas: la etapa fuera de línea y la etapa en línea. En esta sección se van a explorar algunos desarrollos que se componen únicamente de una etapa que es análoga a la etapa en línea. Utilizando solamente la entrada sensorial de los sensores disponibles a bordo del UAV, estos métodos pueden evadir obstáculos no previstos en el entorno, así como navegar hacia su dirección objetivo. La interacción

de estos métodos con el entorno es completamente reactiva y no se requiere de información previa para su funcionamiento.

El modelo propuesto por Yang et al. (2021) [4] utiliza una CNN probabilística para generar un mapa de profundidad acompañado de un mapa de confianza a partir de una imagen RGB del entorno. El mapa de confianza expresa qué tan probable es que la información del mapa de profundidad sea correcta. Utilizando la información de ambos mapas, se calcula un mapa de profundidad efectiva tal como se expresa en la Ecuación 4.1, donde D es el mapa de profundidad, C es el mapa de confianza, v es la velocidad del QUAV, a es la aceleración del QUAV y T es la longitud del intervalo de muestreo:

$$D_{eff}(i, j) = D(i, j) - \left(vT - \frac{aT^2}{2}\right) + \ln(C(i, j)) \quad (4.1)$$

Utilizando D_{eff} , se calcula una máscara binaria de obstáculos $D_{eff} > 0$ que posteriormente se procesa agrupando los píxeles positivos (libres de obstáculos) en *clusters*, cada *cluster* representa una región libre de obstáculos. Una vez se tienen las regiones libres de obstáculos, se selecciona el centro de la región más cercana como dirección de navegación. En la Figura 4.2 se ilustra el mecanismo de evasión de obstáculos que se acaba de describir.

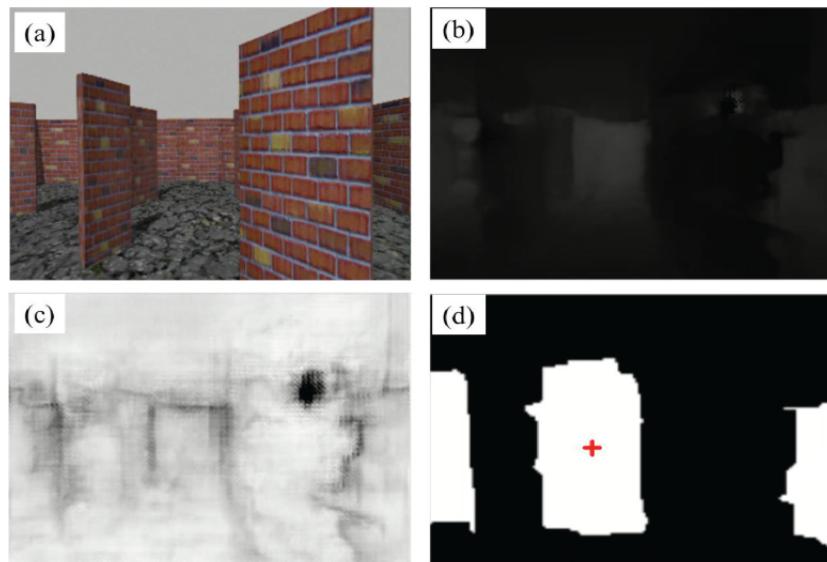


FIGURA 4.2: Mecanismo de evasión de obstáculos por CNN probabilística [4]. (a) Imagen RGB. (b) Mapa de profundidad. (c) Mapa de confianza. (d) Máscara de obstáculos y dirección de navegación seleccionada (cruz roja).

El mecanismo de evasión de obstáculos por CNN probabilística propuesto en [4] demostró ser efectivo para navegación arbitraria en interiores, sin embargo, como este método no considera el objetivo de la navegación, resulta poco útil para la mayoría de las aplicaciones.

Métodos que utilizan aprendizaje por reforzamiento abordan el problema de la evasión de obstáculos utilizando solamente la información disponible a bordo del QUAV. El método propuesto por Ting Tu et al. (2023) [1] utiliza un algoritmo basado en *Q-learning* para entrenar un agente dedicado a la evasión de obstáculos de un QUAV, donde la observación del estado del entorno viene dada por un mapa de profundidad y una imagen RGB; y donde el espacio de acciones es discreto, cada una representando una dirección de movimiento con respecto al campo de visión del QUAV. De forma similar, el método propuesto por Xue et al. (2021) [2] utiliza un algoritmo basado en el método Actor-Crílico suave (del inglés *Soft Actor-Critic*), utilizando mapas de profundidad como observaciones del entorno pero utilizando un espacio de acciones continuas, donde una acción codifica un rango continuo de movimientos en las direcciones de los ejes *y* y *z* con respecto al marco de referencia del QUAV.

El proceso de entrenamiento de estos métodos es considerablemente complicado, requiriendo múltiples redes neuronales para su funcionamiento. En la Figura 4.3 se muestra el diagrama de flujo del proceso de entrenamiento utilizado en el método propuesto por Xue et al. (2021) [2]. Se observa inmediatamente la complejidad del proceso, requiriendo el entrenamiento simultáneo de siete CNNs. Otro inconveniente de estos métodos es la sensibilidad a la definición de la función de recompensa y las restricciones producidas por la representación del espacio de acciones.

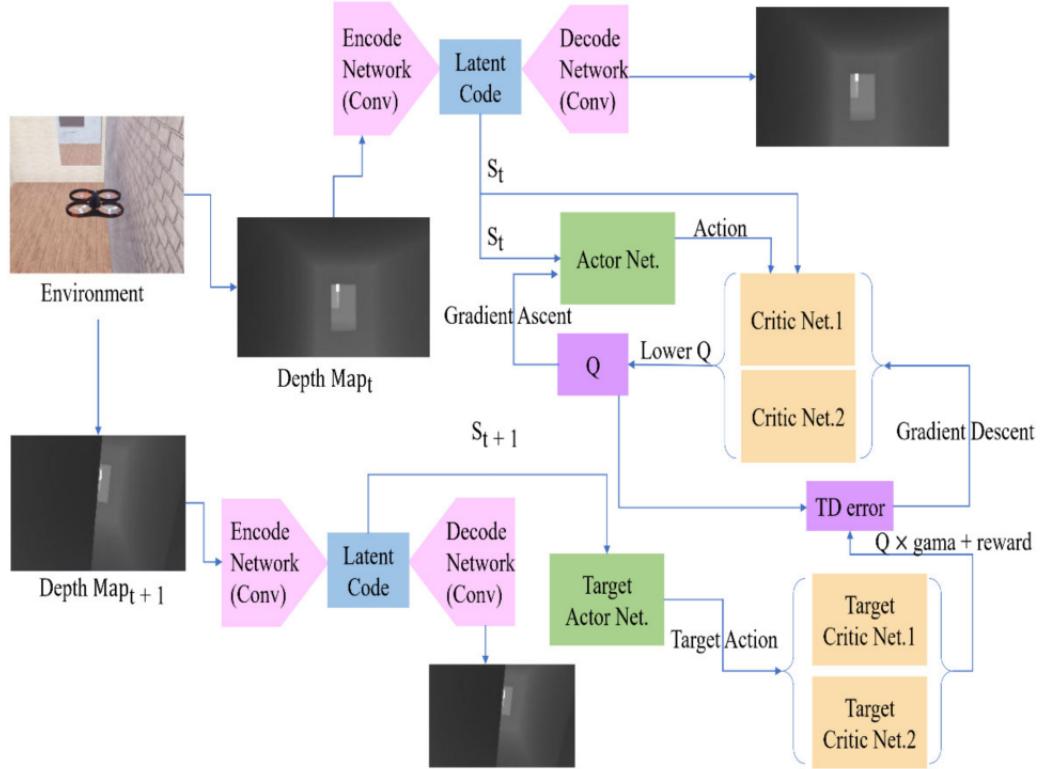


FIGURA 4.3: Diagrama de flujo del proceso de entrenamiento del método propuesto por Xue et al. (2021) [2].

Otro método que depende solamente de la información disponible a bordo del vehículo es *Learning high-speed flight in the wild*, propuesto en [7], este método utiliza una variante del entrenamiento supervisado denominado entrenamiento por imitación (*Imitation Learning* en inglés), en donde se genera una base de datos de entrenamiento utilizando un “experto” privilegiado dentro de un ambiente de simulación, para posteriormente aplicar entrenamiento supervisado sobre una política “estudiante” cuyo objetivo es aprender a generar trayectorias libres de colisión a partir de un mapa de profundidad del entorno y la mediciones iniciales del QUAV. En la siguiente sección, se profundiza en este método, que resulta ser el principal antecedente del desarrollo del presente trabajo.

4.3. Learning high-speed flight in the wild

Tal como se menciona anteriormente, *Learning high-speed flight in the wild* [7] es un método que utiliza entrenamiento por imitación para entrenar una política de generación de trayectorias libres de colisión, a partir de base de datos de ejemplos, que son generados por un experto privilegiado dentro de un entorno de simulación. La Figura 4.4 ilustra el funcionamiento de este método donde se aprecian sus tres componentes principales: El experto privilegiado, la política estudiante y la proyección de trayectorias. En las siguientes secciones se profundiza en el funcionamiento de cada uno de estos componentes.

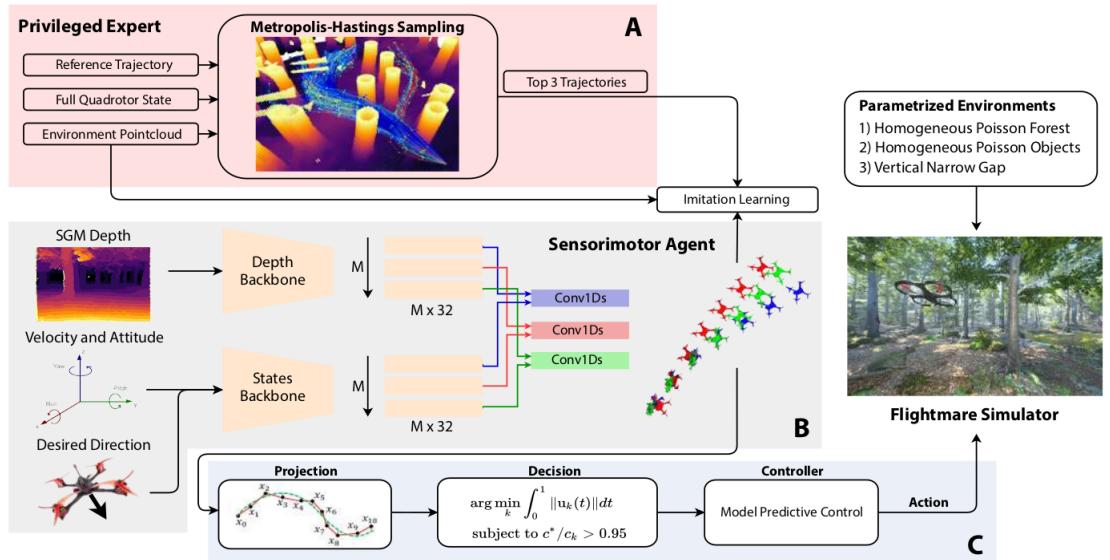


FIGURA 4.4: Visión general del método propuesto en *Learning high-speed flight in the wild* [7]. (A) El experto privilegiado genera una distribución de trayectorias libre de colisión que siguen una trayectoria de referencia. Las trayectorias generadas están condicionadas por la completa información de obstáculos del entorno. (B) La política estudiante es entrenada mediante entrenamiento supervisado para predecir las mejores tres trayectorias a partir de un mapa de profundidad, mediciones iniciales del dron y una dirección objetivo. (C) Durante la ejecución, las predicciones de la política estudiante son proyectadas en el espacio de las trayectorias polinomiales y finalmente la trayectoria con el costo de predicción más bajo es ejecutada por el modelo de control.

4.3.1. El experto privilegiado

El experto privilegiado es un algoritmo de planificación de trayectorias basado en muestreo [7]. El experto tiene conocimiento completo del estado del QUAV y de la configuración tridimensional del entorno (en forma de una nube de puntos tridimensional). El experto genera trayectorias libres de colisión τ que representan el estado deseado del QUAV $x_{des} \in \mathbb{R}^{13}$ durante el siguiente segundo, comenzando desde el estado inicial de QUAV, en otras palabras $\tau(0) = x_0$. Para realizar esta tarea, se toman muestras de una distribución P que codifica la distancia a los obstáculos y la proximidad a la trayectoria de referencia; esto es, la distribución $P(\tau|\tau_{ref}, \mathcal{C})$ es condicionada por la trayectoria de referencia τ_{ref} y la nube de puntos del entorno $\mathcal{C} \in \mathbb{R}^{n \times 3}$. De acuerdo a P , la probabilidad de una trayectoria τ es alta si esta lejos de obstáculos en \mathcal{C} y cerca de τ_{ref} . La Ecuación 4.2 muestra la definición de P [7].

$$P(\tau|\tau_{ref}, \mathcal{C}) = \frac{1}{Z} e^{-c(\tau, \tau_{ref}, \mathcal{C})} \quad (4.2)$$

Donde $Z = \int_{\tau} P(\tau|\tau_{ref}, \mathcal{C})$ es el factor de normalización y $c(\tau, \tau_{ref}, \mathcal{C}) \in \mathbb{R}^+$ es una función de costo que indica la proximidad a la trayectoria de referencia y la distancia a los obstáculos. En la Ecuación 4.3 define c de acuerdo a [7].

$$\begin{aligned} c(\tau, \tau_{ref}, \mathcal{C}) &= \int_0^1 \lambda_c C_{collision}(\tau(t), \mathcal{C}) dt \\ &+ \int_0^1 [\tau(t), -\tau_{ref}(t)]^T \mathcal{Q} [\tau(t) - \tau_{ref}(t)] dt \end{aligned} \quad (4.3)$$

Donde $\lambda_c = 1000$, \mathcal{Q} es una matriz positiva que representa los pesos de los componentes del estado del vehículo, y $C_{collision}$ una función que cuantifica la distancia del QUAV a los puntos de \mathcal{C} . Si r_q es el radio del QUAV y $d(z, \mathcal{C})$ es la distancia de $z \in \mathbb{R}^3$ al punto mas cercano en \mathcal{C} , la Ecuación 4.4 define $C_{collision}$ de acuerdo a [7].

$$C_{collision}(\tau(t), \mathcal{C}) = \begin{cases} 0 & \text{si } d(\tau(t), \mathcal{C}) > 2r_q \\ 4 - \left(\frac{d(\tau(t), \mathcal{C})}{r_q}\right)^2 & \text{si } d(\tau(t), \mathcal{C}) \leq 2r_q \end{cases} \quad (4.4)$$

Debido a que generalmente existe mas de una forma para evadir un obstáculo, la distribución P es multi-modal, haciendo que el cálculo analítico de P sea difícil de resolver. Para aproximar la densidad de P , el experto utiliza muestreo aleatorio mediante el algoritmo de *Metropolis-hastings* [30]. Para estimar P , el algoritmo de *Metropolis-hastings* necesita una función objetivo $s(\tau) \propto P(\tau|\tau_{ref}, \mathcal{C})$. Utilizando $s(\tau) = e^{-c(\tau, \tau_{ref}, \mathcal{C})}$ como función objetivo, este algoritmo estima asintóticamente la distribución P [7]. Por lo tanto, está asegurado que las muestras de trayectorias cubran asintóticamente todas las modas de P .

Para reducir la dimensionalidad del problema, el experto representa las trayectorias τ como curvas de Bezier con tres puntos de control ($\tau \in \mathbb{R}^{3 \times 3}$) tal como se establece en [31], la longitud de estas trayectorias está determinada por la rapidez promedio de ejecución deseada del QUAV. Adicionalmente, para reducir la complejidad de ejecución del experto, se discretizan las trayectorias en segmentos equitativamente espaciados de 0.1 segundos de duración y se evalúa la versión discreta de la Ecuación 4.3.

El experto luego se utiliza para generar una base de trayectorias libres de colisión mediante su ejecución en línea dentro de un ambiente de simulación. En la Figura 4.5 se visualizan las muestras de P generadas por el experto durante un paso de la simulación. Como trayectoria de referencia se utiliza una trayectoria global libre de obstáculos que se calcula utilizando el método propuesto en [32], que comienza en una posición inicial y termina en la posición objetivo dentro del entorno. Se repite este proceso múltiples veces sobre distintos entornos de simulación.

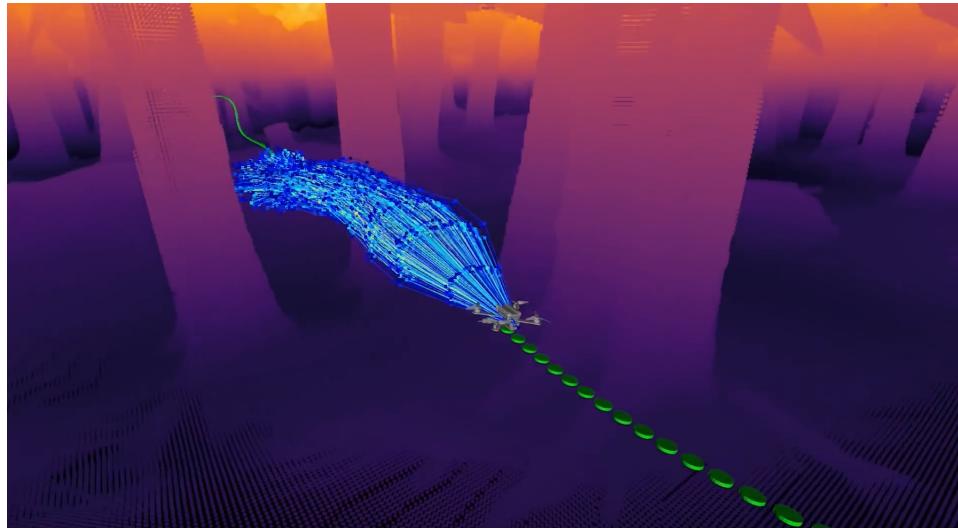


FIGURA 4.5: Visualización de las muestras de P generadas por el experto durante un paso de la simulación. Los puntos verdes representan τ_{ref} .

El conjunto de las mejores trayectorias generadas (mejores tres muestras por cada paso de la simulación) a lo largo de todas las ejecuciones se utiliza para formar una base de datos de trayectorias libres de colisión, cada ejemplo de la base de datos viene acompañado de un mapa de profundidad que se genera utilizando visión estereoscópica y del estado del QUAV (velocidad, rotación, posición y dirección objetivo). Si los entornos de simulación donde se realizan ejecuciones son los suficientemente variados (bosques, entornos urbanos, entornos de desastre, objetos aleatorios, entre otros) la base datos generada tiene la capacidad de utilizarse para entrenar políticas de evasión de obstáculos con un nivel de generalización aceptable [7]. Es importante mencionar que debido a que la longitud de las muestras de P están condicionadas por la rapidez promedio de ejecución deseada del QUAV, distintas velocidades de ejecución requieren diferentes bases de datos, en otras palabras, para cada rapidez promedio de ejecución se necesita generar una base de datos distinta, y por lo tanto, las políticas de evasión de obstáculos entrenadas usando este método están condicionadas a la rapidez promedio del QUAV utilizada durante la generación de la base de datos.

4.3.2. La política estudiante

En contraste con el experto, la política estudiante (el estudiante) produce trayectorias libres de colisión en tiempo real utilizando solamente la información de

los sensores disponible a bordo del QUAV. La información sensorial utilizada por el estudiante incluyen: un mapa de profundidad estimado utilizando visión estereoscópica, la velocidad y rotación del QUAV, y una dirección objetivo que es representada como un vector unitario que apunta hacia un punto de referencia un segundo en el futuro con respecto al último estado conocido del QUAV. Utilizando esta información y alguna base de datos generada por el experto privilegiado se puede entrenar un modelo que pueda cumplir con los requerimientos de la política estudiante.

Loquercio et al. (2021) [7] modela la política estudiante como una red neuronal. La arquitectura de esta red contiene dos ramas que codifican distintas características: una rama convolucional, que codifica la información visual del entorno a partir de un mapa de profundidad $D \in \mathbb{R}^{640 \times 480}$; y una rama que codifica el estado del QUAV a partir de la velocidad del vehículo $v \in \mathbb{R}^3$, la rotación expresada como una matriz $q \in \mathbb{R}^9$, y la dirección objetivo $r \in \mathbb{R}^3$ tal que $\|r\| = 1$.

La rama convolucional utiliza una instancia pre-entrenada de MobileNetV3 [9] para extraer eficientemente las características visuales del mapa de profundidad. Estas características son posteriormente procesadas por una capa convolucional 1D para generar tres vectores de tamaño 32, cada uno representando una moda de P .

La rama del estado del QUAV concatena v , q y r y procesa el resultado mediante una red FC de cuatro capas, cada una con $[64, 32, 32, 32]$ número de nodos y activaciones LeakyReLU; posteriormente el resultado es procesado por una capa convolucional 1D para generar tres vectores de tamaño 32, una vez mas, cada uno representando una moda de P .

Para cada moda, las codificaciones de la rama de estado y de la rama convolucional se concatenan y son procesadas por una red FC de tres capas, cada una con $[64, 128, 128]$ número de nodos y activaciones LeakyReLU. Finalmente, una última capa FC con activación LeakyReLU predice una trayectoria τ y costo de ejecución análogo al costo definido en la Ecuación 4.3.

La salida de la red representa $\mathbb{T}_n = \{(\tau_n^k, c_k) | k \in [0, 1, 2]\}$, donde $c_k \in \mathbb{R}^+$ es el costo de ejecución de la trayectoria τ_n^k (análogo al definido en la Ecuación 4.3), nótese que cada k representa una moda de P . A diferencia del experto, las

trayectorias predichas por el estudiante describen solamente el componente de la posición de la trayectoria, por esto, $\tau_n^k \in \mathbb{R}^{10 \times 3}$ y describe:

$$\tau_n^k = [p(t_i)]_{i=1}^{10}, t_i = \frac{i}{10} \quad (4.5)$$

Donde $p(t_i) \in \mathbb{R}^3$ es la posición del QUAU en el tiempo $t = t_i$ relativo al estado actual x_0 .

La red neuronal se entrena utilizando entrenamiento supervisado por descenso de gradiente. La función de pérdida para el entrenamiento tiene dos componentes, uno que representa el costo espacial entre la trayectoria predicha y la trayectoria generada por el experto; y otro que representa el costo de estimación de la función c (definida en la Ecuación 4.3).

El componente espacial de la función de pérdida es de tipo “el ganador gana todo” relajada R-WTA (del inglés *Relaxed Winner-Takes-All*) definida como:

$$\text{R-WTA}(\mathbb{T}_e, \mathbb{T}_n) = \sum_{i=0}^{|\mathbb{T}_e|} \sum_{k=0}^{|\mathbb{T}_n|} \alpha(\tau_{e,p}^i, \tau_n^k) \|\tau_{e,p}^i - \tau_n^k\|^2 \quad (4.6)$$

Donde \mathbb{T}_e y \mathbb{T}_n son el conjunto de trayectorias producidas por el experto y por el estudiante respectivamente, $\tau_{e,p}$ es el componente de la posición de τ_e y $\alpha(\cdot)$ se define como:

$$\alpha(\tau_{e,p}^i, \tau_n^k) = \begin{cases} 1 - \epsilon & \text{si } \|\tau_{e,p}^i - \tau_n^k\|^2 \leq \|\tau_{e,p}^j - \tau_n^k\|^2 \forall i \neq j \\ \frac{\epsilon}{|\mathbb{T}_n|-1} & \text{en caso contrario} \end{cases} \quad (4.7)$$

Con $\epsilon = 0,05$. Esta definición hace que intuitivamente se asocie una trayectoria $\tau_e^i \in \mathbb{T}_e$ con la trayectoria más cercana predicha por el estudiante $\tau_n^k \in \mathbb{T}_n$, asignándole un peso de $1 - \epsilon = 0,95$; dejando un peso de $\epsilon/(|\mathbb{T}_n| - 1) = 0,025$ para el resto de las predicciones, de ahí el nombre “el ganador gana todo”. Esta formulación permite que la red pueda aprender el comportamiento multi-modal de las trayectorias generadas por el experto.

El componente de estimación de la función c está dado por la suma de diferencias cuadradas entre el valor a estimar y la predicción de la red. El valor a estimar se obtiene para cada predicción utilizando la asociación generada por el componente R-WTA, donde $h(k)$ retorna el índice de la trayectoria del experto asociada con la predicción número k . Finalmente, la función de pérdida utilizada para el entrenamiento de la red neuronal que modela al estudiante es [7]:

$$L(\mathbb{T}_e, \mathbb{T}_n) = 10 \cdot \text{R-WTA}(\mathbb{T}_e, \mathbb{T}_n) + \frac{1}{10} \cdot \sum_{k=0}^{|\mathbb{T}_e|} [c_k - c(\tau_n^k, \tau_{e,p}^{h(k)}, \mathcal{C})]^2 \quad (4.8)$$

Esta función de pérdida es promediada sobre un *batch* de 8 muestras y se minimiza utilizando descenso de gradiente [7] con una tasa de aprendizaje de 1×10^{-3} .

4.3.3. Proyección de trayectorias

Las trayectorias predichas por la política estudiante son proyectadas al espacio de los polinomios de grado N con $N \geq 3$ para cada eje independientemente. Esta representación polinómica asegura continuidad en posición, velocidad, aceleración y facilita la viabilidad dinámica de la trayectoria [31]. Por ejemplo, para el eje x, se define la proyección polinómica $\mu_x(t) = a_x^T \cdot T(t)$, donde $a_x^T = [a_0, a_1, \dots, a_N]$ y $T(t) = [1, t, \dots, t^{N-1}, t^N]$. El término de proyección a_x se obtiene resolviendo el siguiente problema de optimización:

$$\min_{a_x} \sum_{i=1}^{10} (\tau_{n,x}^{k,i} - a_x^T \cdot T\left(\frac{i}{10}\right))^2$$

sujeto a:

$$p_x(0) - a_x^T \cdot T(0) = 0 \quad (4.9)$$

$$v_x(0) - a_x^T \cdot \frac{dT}{dt}(0) = 0$$

$$a_x(0) - a_x^T \cdot \frac{d^2T}{dt^2}(0) = 0$$

Donde $\tau_{n,x}^{k,i}$ es el componente en el eje x del elemento número i de τ_n^k , y $p_x(0), v_x(0), a_x(0)$ son el componente x de la posición, velocidad y aceleración del QUAV obtenidas del último estado x_0 . Adicionalmente, para evitar cambios abruptos en la velocidad durante el vuelo, se limita la rapidez promedio del polinomio a un valor definido v_{des} que coincide con la rapidez promedio de ejecución de la base de datos con la que se entrenó la política estudiante. Para ello, se escala el tiempo t del polinomio $\mu(t)$ por un factor $\beta = v_{des}/v_\mu$ donde $v_\mu = \|\mu(0) - \mu(1)\|$.

Una vez que todas las trayectorias han sido proyectadas, se escoge una para ejecutarse. Para esto, se seleccionan las trayectorias que cumplen con $c^*/c_k \geq 0,95$ ($c^* = \min_k c_k$) y se selecciona aquella de que tenga el menor valor de entrada acuedro al criterio establecido por [31]. Esto refuerza continuidad temporal de las trayectorias. Por ejemplo, si se está esquivando un obstáculo por la derecha no es ideal navegar hacia la izquierda en la siguiente iteración, a no ser que sea estrictamente necesario por la apariencia del obstáculo. Finalmente, se obtienen valores de posición, velocidad y aceleración de la trayectoria seleccionada y se envían al FCU para su ejecución.

4.4. Resumen

Este capítulo abordó el problema de la evasión de obstáculos, centrándose en dos variantes de planificación local de caminos: aquellas que requerían información global previa y las que se basaban únicamente en la información de los sensores a bordo del vehículo. La Sección 4.1 exploró desarrollos relacionados con la primera variante, mientras que la Sección 4.2 se centró en la segunda. Además, se profundizó en *Learning high-speed flight in the wild* en la Sección 4.3, destacando su papel como un antecedente crucial para el desarrollo del presente trabajo. Este capítulo proporcionó una visión del estado actual en el campo, resaltando las perspectivas relevantes para la resolución del problema de evasión de obstáculos para QUAVs.

La Tabla 4.1 resume los métodos explorados en este capítulo, destacando sus características más relevantes.

Método	Dominio ¹	Consideraciones
<i>Basado en A*</i> [26]	Global	<ul style="list-style-type: none"> - Genera trayectorias estáticas. - Requiere modelado del entorno.
<i>Campos de potencial</i> [28]	Global/Local	<ul style="list-style-type: none"> - Puede atascarse en mínimos locales. - Produce trayectorias no óptimas.
<i>P-CAL</i> [3]	Global/Local	<ul style="list-style-type: none"> - Requiere nube de puntos global del entorno. - Requiere sensor LiDAR.
<i>P-CNN</i> [4]	Local	<ul style="list-style-type: none"> - Entrenamiento supervisado. - Requiere base de datos de entrenamiento. - Navegación arbitraria.
<i>Q-learning</i> [1]	Local	<ul style="list-style-type: none"> - Entrenamiento por reforzamiento. - Limitado a espacios de acciones discretas. - Sensible a la elección de función de recompensa.
<i>Soft Actor-Critic</i> [2]	Local	<ul style="list-style-type: none"> - Entrenamiento por reforzamiento. - Sensible a la elección de función de recompensa.
<i>Learning high-speed flight in the wild</i> [7]	Local	<ul style="list-style-type: none"> - Entrenamiento por imitación dentro de un entorno de simulación. - Altamente flexible. - Sensible a la velocidad de ejecución del QUAV. - Requiere ajuste-fino.

TABLA 4.1: Resumen de los métodos explorados durante la revisión del estado del arte.

En el siguiente capítulo se describe la implementación de la solución propuesta por este trabajo, teniendo en cuenta el contexto de la plataforma disponible, así como también fundamentando la selección del algoritmo de evasión de obstáculos y detallando sus componentes de implementación.

¹Global implica que el método requiere información global previa, local implica que el método opera con la información disponible a bordo del vehículo.

Capítulo 5

Implementación

En el capítulo anterior se realiza una revisión del estado del arte de los algoritmos de evasión de obstáculos en el contexto de los QUAVs. En este capítulo se hace la descripción de la implementación de la solución propuesta, para el problema de evasión de obstáculos sobre el contexto de la plataforma de implementación de los productos de ACSL. En la Sección 5.1 se describe brevemente la metodología y plataforma de implementación; en la Sección 5.2 se fundamenta la selección del algoritmo de evasión de obstáculos, inspirado en *Learning high-speed flight in the wild* [7]; en la Sección 5.3 se describe la arquitectura de la solución propuesta en este trabajo, así como también se describe el funcionamiento de cada uno de sus componentes; y finalmente en la Sección 5.4 se describe el proceso de refinamiento fino de la red neuronal encargada de la inferencia de trayectorias libre de colisión, es decir, el ajuste fino de la política estudiante.

5.1. Plataforma de implementación

El componente principal de la metodología de implementación de soluciones sobre los productos de ACSL, es una librería propietaria de ACSL llamada *ACSL Vision Core*. Esta librería, escrita en C++, proporciona un marco de trabajo para la implementación de aplicaciones sobre los UAV de ACSL, adicionalmente admitiendo el uso de vehículos simulados por el entorno de simulación AirSim [33]. Bajo este marco de trabajo, una solución es implementada como una colección

de procesos de linux independientes que se comunican utilizando memoria compartida y *sockets* de la librería ZeroMQ [25]. La implementación del algoritmo a seleccionar debe estar condicionada a este marco de trabajo, para que la solución propuesta pueda ejecutarse en concordancia y con capacidad de interacción con otras aplicaciones de los vehículos de ACSL. La librería *ACSL Vision Core* expone un API (Interfaz de programación de aplicación, del inglés *Aplication Programming Interface*) llamada *ACSL Flight API* que permite interactuar con el FCU¹ del QUAV, exponiendo métodos para leer la información de la estimación del estado del QUAV (posición, velocidad, rotación y estado de los actuadores) y para enviar comandos de control de alto nivel desde un proceso ejecutándose en el computador a bordo del vehículo.

5.1.1. Plataforma física

El vehículo sobre el cual se implementó la solución propuesta en el presente trabajo es SOTEN, un QUAV de tamaño ligero diseñado por ACSL para aplicaciones de fotografía aérea (Inspección, vigilancia, rescate, entre otros). En la Figura 5.1 se muestra una imagen del vehículo en cuestión.



FIGURA 5.1: SOTEN, un QUAV diseñado por ACSL que sirve de plataforma física para la implementación de este trabajo.

¹Unidad de control de vuelo, tal como se define en la Sección 3.1.1

SOTEN viene equipado con un chip NVIDIA Jetson Xavier NX, que funciona como computadora a bordo del QUAV, que posee capacidades de aceleración gráfica y es precisamente donde se implementó la solución propuesta en este trabajo. Con respecto a los sensores, SOTEN viene equipado con un conjunto de cámaras estereoscópicas, de las cuales en este trabajo solo se utilizó el par frontal; así como también se equipa de una cámara de inspección estabilizada por un *gimbal* que si bien no es utilizada en este trabajo, representa una de las funciones principales de SOTEN, su capacidad de inspección. Como mecanismo de localización tiene acceso a locación por GPS y localización por visión, en este trabajo solo se utilizó la localización por GPS.

5.1.2. Entorno de simulación

El entorno de simulación que es soportado por *ACSL Vision Core* es AirSim [33], donde se pueden simular una variedad de vehículos equivalentes a los productos de ACSL en entornos de simulación ejecutados por *UnrealEngine*. En este trabajo se utiliza AirSim para simular un QUAV equivalente a SOTEN, que permite que dentro de las simulaciones se tengan las mismas propiedades dinámicas y dimensiones de la plataforma física. La implementación de la solución propuesta en este trabajo es agnóstica de la utilización de AirSim, el *ACSL Flight API* se encarga de traducir los comandos de control al equivalente en el marco de trabajo del simulador.

5.2. Selección del algoritmo

El algoritmo seleccionado sigue al propuesto en *Learning high-speed flight in the wild* [7]. Las características del método propuesto en [7] resultan atractivas para los productos de ACSL, en concreto: solo requiere de la información disponible a bordo del vehículo; no requiere sensores que alguno de los productos de ACSL no tenga disponible; tiene la capacidad de adaptarse a aplicaciones específicas si se escogen adecuadamente los entornos de simulación para la generación de la base datos; como la información visual se representa en forma de un mapa de profundidad, es sencillo trasladar el algoritmo entre plataformas; y finalmente,

los autores de *Learning high-speed flight in the wild* dan acceso público al código fuente para la generación de las base de datos y del entrenamiento de la política estudiante, así como también los pesos entrenados de la red neuronal utilizada, reduciendo el trabajo necesario para implementar el algoritmo sobre la plataforma de ACSL.

Sin embargo, es necesario realizar ciertas modificaciones al algoritmo. Primero, por motivos de seguridad es necesario que la rapidez de ejecución esté limitada a 1 m/s, esto es, $v_{des} = 1$; los pesos de la política estudiante publicados en [7] fueron entrenados con $v_{des} = 7$, por lo tanto es necesario hacer un proceso de ajuste fino de los pesos. Segundo, debido a limitaciones del *ACSL Flight API*, solo se puede enviar instrucciones para modificar la referencia de la velocidad y de la velocidad rotacional; a diferencia de [7] en donde se envían instrucciones con referencia de posición, rotación, velocidad y aceleración simultáneamente, por lo tanto, la ejecución de las trayectorias para este trabajo solamente envía la información de la velocidad lineal, junto con un estimado de la velocidad rotacional necesaria para que el encabezado del QUAV apunte hacia la dirección tangencial de la trayectoria en ejecución. Tercero, por motivos de validación del concepto por parte de ACSL, la ejecución de la evasión de obstáculos va a estar limitada al plano x, y del marco de referencia del cuerpo del QUAV. En resumen, el algoritmo que se implementa en este trabajo es el mecanismo de inferencia de la política estudiante de [7] luego de un proceso de refinamiento fino; junto con otros componentes implementados dentro del marco de trabajo de ACSL que se encargan de la ejecución de las trayectorias y de la adquisición de la información necesaria para la inferencia.

5.3. Arquitectura de la solución

Tal como se mencionó en la sección anterior, la solución propuesta por este trabajo es la combinación del mecanismo de inferencia de la política estudiante propuesto por Loquercio et al. [7] y una serie de componentes bajo el marco de trabajo de ACSL encargados de la ejecución de las trayectorias y de la adquisición de la información necesaria para la inferencia. La Figura 5.2 muestra el diagrama de comunicación de la solución propuesta; se compone de cuatro componentes: Un puente del FCU, que se encarga de obtener continuamente la última información del estado del QUAV; un puente de profundidad, que procesa los pares de imágenes

estéreo producidas por las cámaras frontales para calcular mapas de profundidad; un proceso de inferencia que utiliza la información producida por los puentes de FCU y de profundidad para predecir trayectorias libre de colisión utilizando la red neuronal de [7]; y un ejecutor de trayectorias que ejecuta las trayectorias inferidas por el proceso de inferencia. Los componentes se ejecutan como procesos independientes y se comunican utilizando *sockets* de la librería ZeroMQ [25] y memoria compartida.

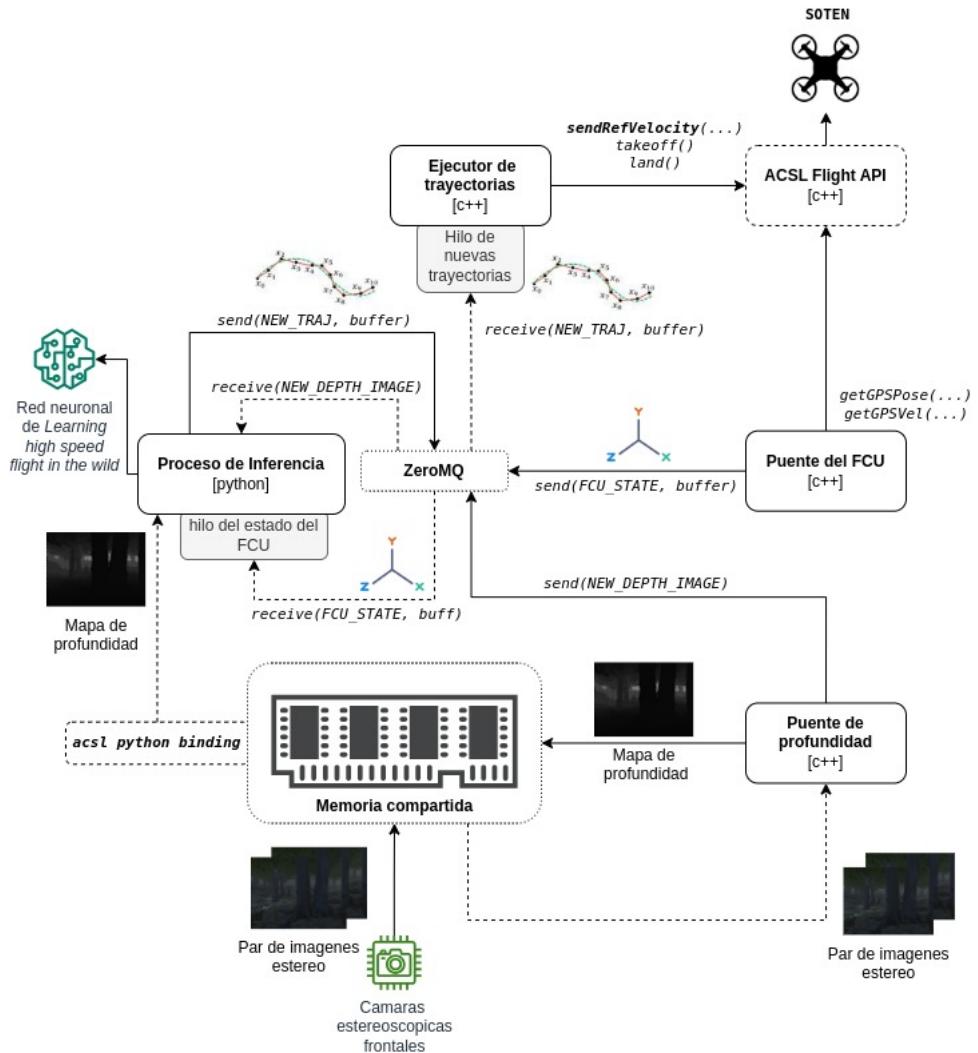


FIGURA 5.2: Diagrama de comunicación de la solución. En alto nivel, el flujo de información comienza cuando se produce un nuevo par de imágenes estéreo; el puente de profundidad procesa el par para producir un mapa de profundidad; luego el proceso de inferencia recibe el mensaje NEW_DEPTH_IMAGE, obtiene la información mas reciente del puente FCU, invoca a la red neuronal y envía el resultado en el mensaje NEW_TRAJ. Finalmente, el ejecutor de trayectorias, al recibir el mensaje NEW_TRAJ, proyecta la trayectoria de acuerdo al procedimiento descrito en la Sección 4.3.3 y procede a ejecutar la trayectoria utilizando el *ACSL Flight API*.

A continuación se profundiza en cada uno de los componentes.

5.3.1. Puente del FCU

Este componente tiene como función obtener la información mas reciente del estado del QUAV y transmitir dicho estado mediante un *socket* de ZeroMQ para su utilización por el proceso de inferencia. Debido a que el proceso de inferencia se ejecuta en python y el *ACSL Flight API* solo está disponible para entornos de C++, es necesario que un proceso externo obtenga la información del API y se la comunique al proceso de inferencia, esta es justamente la función del puente del FCU.

Si B es el marco de referencia del cuerpo del QUAV, a una frecuencia de 120 Hz, este componente invoca las funciones `getGPSPose` y `getGPSSVel` del *ACSL Flight API* para obtener los estimados mas recientes de: latitud, longitud y altitud (GPS); los ángulos de Euler *roll*, *pitch* y *yaw* de B ; los componentes $v_{B_x}, v_{B_y}, v_{B_z}$ de la velocidad lineal con respecto a B ; y la velocidad angular del eje B_z , ω_z . Adicionalmente, este componente también convierte las coordenadas latitud, longitud, altitud en coordenadas NED utilizando el Algoritmo 1, tomando como origen al punto de donde despega el QUAV. Finalmente, estas mediciones son empaquetadas en un *buffer* binario y son enviadas como cuerpo del mensaje `FCU_STATE` para su posterior uso por el proceso de inferencia.

5.3.2. Puente de profundidad

Este componente es el encargado de generar los mapas de profundidad necesarios para el funcionamiento del proceso de inferencia. Para realizar esta tarea, obtiene el par más reciente de imágenes estéreo de la cámara frontal, calcula un mapa de disparidad utilizando la funcionalidad de emparejamiento de bloques de la librería `opencv` [34], aplica el método explicado en la Sección 3.2 para obtener un mapa de profundidad a partir del mapa de disparidad y constantes de las cámaras; una vez disponible el mapa de profundidad, se escribe en memoria compartida y envía el mensaje `NEW_DEPTH_IMAGE`, que sirve para notificar al proceso de inferencia que hay un nuevo mapa de profundidad disponible. Este proceso se

repite con una frecuencia máxima de 30 Hz, que coincide con la tasa de muestreo del sistema de cámaras estéreo frontales de SOTEN.

5.3.3. Proceso de inferencia

Este componente es el encargado de predecir trayectorias libres de colisión a partir de los mapas de profundidad generados por el puente de profundidad y del estado del QUAV obtenido por el puente del FCU. El procesamiento de este componente se divide en dos hilos: el hilo principal, que realiza inferencia en la red neuronal de *Learning High Speed Flight in the Wild* [7] cada vez que llega el mensaje NEW_DEPTH_IMAGE; y el hilo del estado del FCU, que se encarga de recibir los mensajes FCU_STATE para mantener una copia global del último estado del QUAV.

Hilo del estado del FCU

El procesamiento realizado por este hilo es relativamente simple; cada vez que llega un mensaje FCU_STATE se obtiene el *buffer* binario del cuerpo del mensaje, se convierte la representación binaria en una representación estructurada y se le asigna a la variable global fcu_state. La variable fcu_state, al ser global, es accesible por ambos hilos. Para mantener la integridad de fcu_state y evitar condiciones de carrera, el acceso de escritura y lectura esta controlado por un semáforo de exclusión mutua (*mutex*).

Hilo principal

El procesamiento del hilo principal puede resumirse en cinco etapas: primero, esperar por un nuevo mensaje de NEW_DEPTH_IMAGE; segundo, obtener el último mapa de profundidad de la memoria compartida; tercero, detectar la presencia de obstáculos en el mapa de profundidad, si no existe obstáculo cercano, repetir desde la primera etapa; cuarto, preparar las entradas de la red y realizar inferencia; finalmente, seleccionar las trayectorias que cumplan $c^*/c_k \geq 0,95$ ($c^* = \min_k c_k$),

codificarlas en un *buffer* binario y enviar un mensaje de NEW_TRAJ utilizando el *buffer* como cuerpo del mensaje.

La etapa tercera es el resultado de una optimización necesaria para aliviar la carga de procesamiento sobre la computadora a bordo del QUAV. El proceso de realizar inferencia utiliza una porción importante de los recursos de procesamiento del QUAV, si se limita la inferencia exclusivamente a los escenarios donde existe un obstáculo a esquivar, el QUAV puede utilizar los recursos de procesamiento en otras aplicaciones. Para realizar la tarea de detectar obstáculos en un mapa de profundidad, se utiliza un algoritmo que a partir un mapa de profundidad retorna un aproximado de la profundidad al obstáculo mas cercano, este algoritmo asume que los obstáculos mas cercanos aparecen mas grandes en el mapa de profundidad y utiliza la función `findContours` de la librería `opencv` [34] para obtener los contornos de una máscara binaria. Si ℓ es la distancia máxima en milímetros donde un obstáculo se considera “cercano” y D es un mapa de profundidad en milímetros, el pseudo-código del algoritmo se muestra en el Algoritmo 3.

Algoritmo 3: Pseudo-código del algoritmo para obtener un estimado de la profundidad al obstáculo mas cercano en un mapa de profundidad

Datos: D, ℓ

Resultado: d , estimado de la profundidad en metros del obstáculo mas cercano.

```

1  $D_{mask} = D > 10 \wedge D \leq \ell$ 
2 si  $\sum_{(i,j) \in D_{mask}} D_{mask}(i,j) > \frac{1}{10} |\{(i,j) \in D\}|$  entonces
3    $contornos = opencv2.findContours(D_{mask})$ 
4    $candidato =$  el contorno en contornos con mayor área.
5    $a_{cand} = candidato.area$ 
6   si  $a_{cand} > \frac{1}{10} |\{(i,j) \in D\}|$  entonces
7      $I_{cont} = \{(i,j) | (i,j) \in D \wedge (i,j) \in candidato\}$ 
8      $d = \frac{1}{1000 \cdot |I_{cont}|} \sum_{(i,j) \in I_{cont}} D(i,j)$ 
9     devolver  $d$ 
10 devolver  $\infty$ 

```

Dicho esto, a continuación se describe el pseudo-código del procedimiento ejecutado por el hilo principal del proceso de inferencia. Dadas las definiciones:

- $\Phi : \mathbb{R}^{640 \times 480} \times \mathbb{R}^{15} \longrightarrow \{\mathbb{T}_n | \mathbb{T}_n \text{ es tal como se describe en la Sección 4.3.2}\}$ es una función que representa la red neuronal de *Learning High Speed Flight in the Wild* [7].

- `fcu_state` es la variable global que contiene el estado más reciente del QUAV.
- `fcu_mutex` es el semáforo de exclusión mutua para acceder a `fcu_state`.
- `obs_distance` : $\mathbb{R}^{640 \times 480} \rightarrow \mathbb{R}^+$ es una implementación del Algoritmo 3 con $\ell = 7500$ mm.
- `stop` : `void` $\rightarrow \mathbb{B}$ es un predicado que indica si el proceso debe terminar.
- `aplanar` : $\mathbb{R}^{3 \times 3} \rightarrow \mathbb{R}^9$ es una función que transforma una matriz 3x3 en el vector resultante de concatenar sus filas en un vector con nueve coordenadas.
- B representa el marco de referencia en el cuerpo del QUAV.

El pseudo-código del procedimiento ejecutado por el hilo principal del proceso de inferencia se muestra en el Algoritmo 4.

Es importante mencionar, que para que la red neuronal de *Learning High Speed Flight in the Wild* [7] pudiera utilizarse sobre la plataforma de SOTEN, fue necesario realizar un proceso de conversión a un formato ligero orientado a la inferencia. Para esto se utilizó el estándar abierto para la interoperabilidad del aprendizaje de máquinas, ONNX [35] (del inglés *Open Standard for Machine Learning Interoperability*). De no haberse realizado esta conversión, el proceso de inferencia **no** podría ejecutarse sobre SOTEN debido a restricciones del *Firmware* del chip NVIDIA Jetson Xavier NX.

Algoritmo 4: Pseudo-código del procedimiento ejecutado por el hilo principal del proceso de inferencia.

Datos: Φ , `fcu_state`, `fcu_mutex`, `obs_distance`, `stop`

- 1 **mientras** $\neg \text{stop}()$ **hacer**
- 2 Esperar a la llegada de un mensaje de `NEW_DEPTH_IMAGE`.
- 3 Obtener D , el mapa de profundidad mas reciente en memoria compartida.
- 4 $d = \text{obs_distance}(D)$
- 5 **si** $d \leq 7,5$ **entonces**
- 6 **continuar**
- 7 `fcu_mutex.incrementar()`
- 8 Utilizar `fcu_state` para obtener $V_B \in \mathbb{R}^3$, la velocidad del QUAV con respecto a B .
- 9 Utilizar `fcu_state` para obtener $q \in \mathbb{R}^9$, la rotación de B en forma de matriz, esto es, $q = \text{aplanar}(R_\phi R_\theta R_\psi)$ donde ϕ, θ, ψ son los ángulos de Euler de B .
- 10 Utilizar `fcu_state` para obtener $\mu \in \mathbb{R}^3$, posición del QUAV en coordenadas NED.
- 11 Utilizar `fcu_state` para obtener $\mu_{goal} \in \mathbb{R}^3$, la posición del objetivo del vuelo en coordenadas NED.
- 12 Definir la dirección objetivo como $r = \frac{1}{\|\mu_{goal} - \mu\|}(\mu_{goal} - \mu)$.
- 13 `fcu_mutex.decrementar()`
- 14 $X = \text{concatenar}(V_B, q, r)$
- 15 Realizar inferencia sobre la red neuronal $T = \Phi(D, X)$.
- 16 $c^* = \min \{c \mid (\tau, c) \in T\}$
- 17 $T_f = \{(\tau, c) \mid (\tau, c) \in T \wedge c^*/c \geq 0,95\}$
- 18 Codificar T_f en un buffer binario `traj_buffer`.
- 19 Enviar mensaje `NEW_TRAJ` con `traj_buffer` como cuerpo del mensaje.

5.3.4. Ejecutor de trayectorias

Este componente es el encargado de ejecutar las trayectorias predichas por el proceso de inferencia, así como también manejar el estado de la ejecución del vuelo. El procesamiento de este componente se divide en dos hilos: el hilo de nuevas trayectorias, que recibe los mensajes de `NEW_TRAJ` y proyecta las trayectorias en el espacio de los polinomios de grado $N = 3$ de acuerdo a lo especificado en la Sección 4.3.3; y el hilo principal, que ejecuta la trayectoria mas reciente y gestiona el estado del vuelo.

Hilo de nuevas trayectorias

Este hilo se encarga de procesar los mensajes de NEW_TRAJ. Cuando llega un mensaje de NEW_TRAJ, se decodifica $\mathbb{T}_n = \{(\tau_n^k, c_k) | 0 \leq k < \beta\}$ y t_{ref} , con β siendo el número de trayectorias que pasaron la selección $c^*/c \geq 0,95$ y t_{ref} el tiempo en el que se construyó el mensaje, el tiempo t_{ref} sirve como referencia al momento de evaluar la trayectoria; luego, por cada $(\tau_n^k, c_k) \in \mathbb{T}_n$, se proyecta τ_n^k en el espacio de los polinomios de grado $N = 3$ resolviendo el problema de optimización de mínimos cuadrados descrito en la Sección 4.3.3; posteriormente, a cada proyección μ_k se le escala el tiempo t de acuerdo a lo descrito en la Sección 4.3.3 con $v_{des} = 1$ m/s; finalmente, se selecciona la proyección con menor costo c_k y se le asigna a la variable global `current_traj`. La variable `current_traj`, al ser global, es accesible por ambos hilos. Para mantener la integridad de `current_traj` y evitar condiciones de carrera, el acceso de escritura y lectura está controlado por un semáforo de exclusión mutua (`mutex`).

Hilo principal y máquina de estados

Este hilo se encarga de la ejecución de la trayectoria actual `current_traj` y del manejo del estado del vuelo. Para gestionar el estado del vuelo, utiliza una máquina de estados, la máquina de estados es una estructura que se compone de: un conjunto de estados; una referencia al estado actual; y un espacio de variables compartidas entre estados, tal como el estado del QUAV. Cada estado tiene un identificador, un predicado que indica si se debe entrar a el estado y un procedimiento que se ejecuta en cada iteración que el estado se encuentra como estado actual.

Antes de describir el conjunto de estados de la máquina manejada por el hilo principal, se procede a describir la ejecución guiada por una máquina de estados. En cada iteración: primero, se actualizan las variables compartidas entre estados; segundo, se revisa cuales predicados de transición se cumplen y se modifica el estado actual al estado cuyo predicado de transición se haya cumplido primero; y finalmente, si el QUAV es controlable, se ejecuta el procedimiento del estado actual.

El hecho de que el QUAV sea controlable depende del operador del vehículo. En el radio control o en la GCS, existe una opción de control que marca al QUAV como no controlable. Cuando un QUAV no es controlable, las instrucciones de alto nivel que se reciben en el FCU desde la computadora a bordo son ignoradas. Esto es simplemente una medida de seguridad que permite que el operador del QUAV pueda manejar manualmente el vehículo en caso de que la ejecución autónoma produzca algún riesgo de colisión o accidente.

Volviendo a la ejecución guiada por una máquina de estados, dadas las definiciones:

- `stop : void —> B` es un predicado que indica si el proceso debe terminar.
- `controlable : void —> B` es un predicado que indica si el QUAV es controlable.
- `M` es una máquina de estados tal como se describió anteriormente, con `M.shared` el espacio de las variables compartidas entre estados, `M.states` el conjunto de estados y `M.current_state` una referencia al estado actual.
- Si `s` es un estado tal como se describió anteriormente. `s.pred` es el predicado de transición del estado, que recibe el identificador del estado actual y una referencia al espacio de variables compartidas; `s.proc` es el procedimiento de ejecución, que recibe una referencia al espacio de variables compartidas; y `s.id` es el identificador del estado.

El pseudo-código del procedimiento de ejecución guiada por una máquina de estados se muestra en el Algoritmo 5.

Ahora que el concepto de ejecución guiada por máquina de estado está claro, solo basta con conocer el conjunto de estados, sus transiciones y sus procedimientos de ejecución para comprender el funcionamiento del hilo principal. La Figura 5.3 muestra una visualización de la máquina de estados que guía la ejecución del hilo principal, se puede observar sus identificadores y sus predicados de transición. A continuación se describe cada uno de los estados de la Figura 5.3.

Algoritmo 5: Pseudo-código del procedimiento de ejecución guiada por una máquina de estados.

Datos: stop, controlable, M

```

1 mientas  $\neg stop()$  hacer
2   Actualizar las variables de M.shared. En este caso, se utilizan las
      funciones getGPSPose y getGPSVel del ACSL Flight API para
      actualizar el estado del QUAV.
3   para  $s \in M.states$  hacer
4     si  $s.pred(M.current\_state.id, M.shared)$  entonces
5       M.current_state = s
6       romper línea 3
7   si  $controlable()$  entonces
8     M.current_state.proc(M.shared)

```

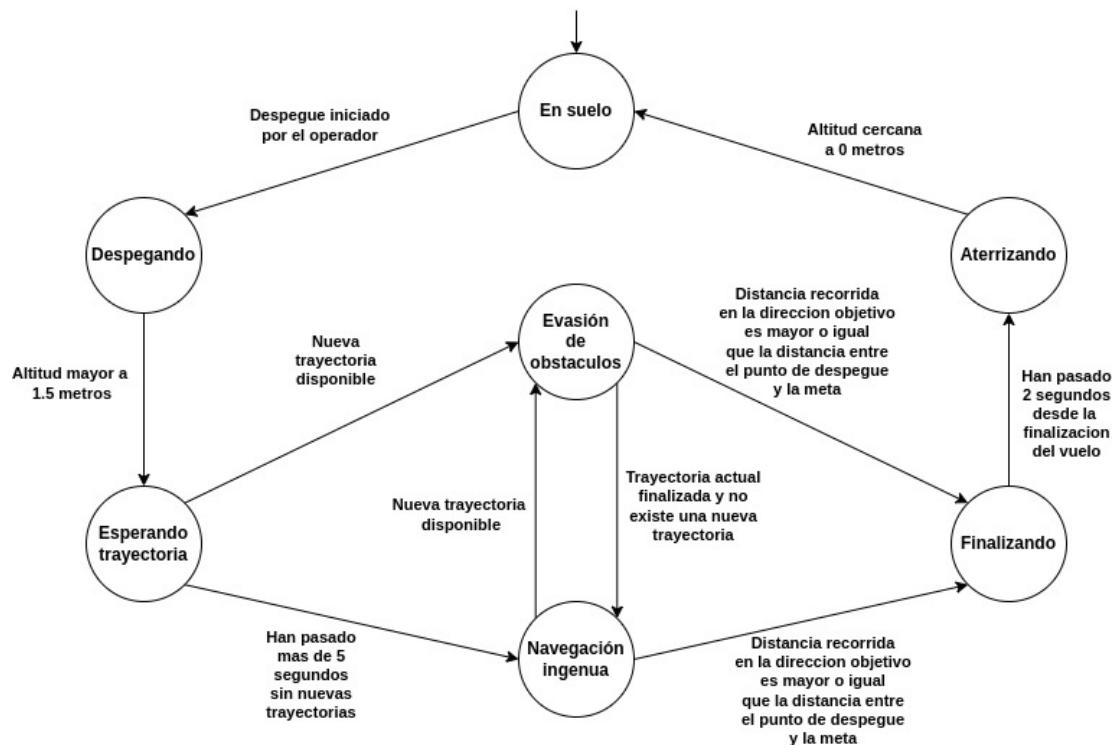


FIGURA 5.3: Visualización de la máquina de estados que guía la ejecución del hilo principal del ejecutor de trayectorias. Los nodos del grafo dirigido representan los estados, sus lados las transiciones y el texto asociado a cada lado representa el predicado de transición.

El estado “En suelo” es el estado inicial de la máquina de estados, representa el momento en el que QUAV está en suelo, bien sea porque se está preparando para despegar o porque acaba de completar un aterrizaje. El estado “Despegando”

está activo durante el proceso despegue, el proceso de despegue comienza cuando el operador del QUAV inicia la secuencia de despegue, bien sea manualmente o de forma autónoma. Una vez que el vehículo alcanza una altitud de 1.5 metros o mayor, el estado “Esperando trayectoria” se activa; este estado espera a que se genere una trayectoria, esto es, que el proceso de inferencia envíe el mensaje `NEW_TRAJ` y que dicho mensaje se procese por el hilo de nuevas trayectorias, en otras palabras, este estado espera a que `current_traj` esté definido. Luego, si pasados cinco segundos aún no se genera una trayectoria, el estado “Navegación ingenua” se activa; en caso contrario, el estado “Evasión de obstáculos” se activa.

Los estados “Evasión de obstáculos” y “Navegación ingenua” son los estados más importantes para la ejecución del vuelo evadiendo obstáculos. En resumen, el estado “Evasión de obstáculos” ejecuta la trayectoria más reciente generada por el proceso de inferencia y el estado “Navegación ingenua” navega directamente en dirección a μ_{goal} , la meta. Recordando que el proceso de inferencia envía nuevas trayectorias siempre y cuando hayan obstáculos a esquivar; el estado “Evasión de obstáculos” se activa cuando es necesario evadir un obstáculo y el estado “Navegación ingenua” se activa cuando no hay obstáculos cercanos en la dirección objetivo.

El procedimiento de ejecución del estado “Navegación ingenua” es bastante simple, se obtiene la dirección objetivo, se le multiplica $v_{des} = 1 \text{ m/s}$ la rapidez de ejecución deseada, y se invoca la función `sendRefVelocity` del *ACSL Flight API* para que el QUAV navegue directamente en esa dirección. Recordando que B es el sistema de referencia del cuerpo del QUAV y E es el sistema de referencia inercial; la función `sendRefVelocity` permite especificar ω_{B_z} la velocidad rotacional alrededor del eje z de B . Este estado, además de especificar la velocidad, estima la cantidad ω_{B_z} necesaria para que el encabezamiento del QUAV y la dirección objetivo estén en la misma linea. Dicho esto, El pseudo-código del procedimiento de ejecución del estado “Navegación ingenua” se muestra en el Algoritmo 6.

Algoritmo 6: Pseudo-código del procedimiento de ejecución del estado “Navegación ingenua”.

Datos: `acs1FlightApi`, `M.shared`, v_{des}

- 1 Obtener `fcu_state` de `M.shared`.
 - 2 Utilizar `fcu_state` para obtener $\mu \in \mathbb{R}^3$, posición del QUAV en coordenadas NED.
 - 3 Utilizar `fcu_state` para obtener $\mu_{goal} \in \mathbb{R}^3$, la posición del objetivo del vuelo en coordenadas NED.
 - 4 Definir la dirección objetivo como $r_E = \frac{1}{\|\mu_{goal} - \mu\|}(\mu_{goal} - \mu)$.
 - 5 Estimar la velocidad angular instantánea alrededor del eje B_z para alinear el encabezamiento del QUAV con r_E , esto es $\omega_{B_z} = \arctan_2(r_{E_y}, r_{E_x}) - \psi$ donde ψ es el ángulo *yaw* del vehículo.
 - 6 Utilizar `fcu_state` para obtener la matriz de transformación entre E y B , esto es $R_E^B = R_\phi R_\theta R_\psi$ donde ϕ, θ, ψ son los ángulos de Euler de B .
 - 7 Obtener la dirección objetivo con respecto a B , $r_B = R_E^B r_E$.
 - 8 Definir la nueva velocidad, $V_B = v_{des} \cdot r_B$
 - 9 `acs1FlightApi.sendRefVelocity(V_B, \omega_{B_z})`
-

Por otro lado, el procedimiento de ejecución del estado “Evasión de obstáculos” evalúa la proyección polinómica de la trayectoria actual `current_traj` para obtener la velocidad instantánea, posteriormente invoca `sendRefVelocity` para enviar la referencia de la velocidad y de ω_{B_z} al FCU. Si `traj_mutex` es el semáforo de exclusión mutua para el acceso a `current_traj`, el pseudo-código del procedimiento de ejecución del estado “Evasión de obstáculos” se muestra en el Algoritmo 7.

Tanto el estado “Navegación ingenua” como el estado “Evasión de obstáculos” tienen una transición al estado “Finalizando”, esta transición se cumple cuando cuando la distancia recorrida en la dirección objetivo es mayor o igual que la distancia entre el punto de despegue y la meta; esta condición se utilizó para preferir que el QUAV aterrice si la evasión de obstáculos lo desvió considerablemente de camino original, así como también para que se aterrice una vez que se alcanzó la meta. Dependiendo de la configuración de los obstáculos, es posible que no sea posible avanzar hacia la meta sin desviarse considerablemente, esta transición se asegura de que en ese caso se aterrice el vehículo y se termine el vuelo. En términos formales, trabajando en el sistema de coordenadas NED, si μ_0 es posición del QUAV después del despegue y antes de comenzar la navegación; μ es la posición actual del QUAV; y μ_{goal} es la posición de la meta. Esta transición se cumple si la expresión de la Ecuación 5.1 se evalúa a verdadero.

Algoritmo 7: Pseudo-código del procedimiento de ejecución del estado “Evaluación de obstáculos”.

Datos: `acs1FlightApi`, `M.shared`, `current_traj`, `traj_mutex`

- 1 Obtener el tiempo actual del sistema t_{now} en segundos.
- 2 `traj_mutex.incrementar()`
- 3 Obtener t_{ref} de `current_traj` en segundos.
- 4 $dt = t_{now} - t_{ref}$
- 5 si $dt > 1$ entonces
 - 6 Trayectoria completada, `traj_mutex.decrementar()` y devolver
- 7 Obtener a_x, a_y los coeficientes de la proyección polinómica de `current_traj` para los ejes x, y respectivamente.
- 8 $T' = [0, 1, 2dt, 3dt^2]$
- 9 $v_x = a_x^T \cdot T'$
- 10 $v_y = a_y^T \cdot T'$
- 11 `traj_mutex.decrementar()`
- 12 Por motivos de validación de concepto, se limita la ejecución de las trayectorias inferidas al plano B_xB_y , esto es $v_z = 0$.
- 13 $V_B = [v_x, v_y, v_z]^T$
- 14 Obtener `fcu_state` de `M.shared`.
- 15 Utilizar `fcu_state` para obtener la matriz de transformación entre B y E , esto es $R_B^E = (R_E^B)^T = (R_\phi R_\theta R_\psi)^T$ donde ϕ, θ, ψ son los ángulos de Euler de B .
- 16 $V_E = R_B^E V_B$
- 17 Estimar la velocidad angular instantánea alrededor del eje B_z para que el encabezamiento del QUAV apunte hacia la dirección tangencial de la trayectoria, esto es $\omega_{B_z} = \arctan_2(V_{E_y}, V_{E_x}) - \psi$ donde ψ es el ángulo *yaw* del vehículo.
- 18 `acs1FlightApi.sendRefVelocity(V_B, \omega_{B_z})`

$$(\mu - \mu_0)^T \cdot \left(\frac{\mu_{goal} - \mu_0}{\|\mu_{goal} - \mu_0\|} \right) \geq \|\mu_{goal} - \mu_0\| \quad (5.1)$$

El estado “Aterrizando” se activa una vez que han pasado dos segundos desde que activó el estado “Finalizando”, este estado invoca al método `land` del *ACSL Flight API* para ordenar al FCU que aterrice el QUAV de forma autónoma. Una vez que la altitud es cercana a cero metros, se activa una vez más el estado “En suelo” y la ejecución finaliza.

Es importante mencionar que esta máquina de estados fue diseñada para el caso donde se hace un solo vuelo, en donde el objetivo se encuentra a distancia predefinida en la dirección del encabezado del QUAV al momento del despegue. Para aplicaciones donde exista una secuencia de puntos objetivos sera necesario

utilizar una variante de esta máquina de estados en donde se tome en cuenta la secuencia de vuelo.

5.4. Ajuste fino de la política estudiante

Para que las trayectorias generadas por el proceso de inferencia cumplan con la restricción de poder ejecutarse a una rapidez promedio de $v_{des} = 1 \text{ m/s}$; es necesario realizar un ajuste de los pesos de la red neuronal publicada por [7]. Los pesos que se descargaron del repositorio de *Learning High Speed Flight in the Wild* fueron entrenados con una base de datos de trayectorias con $v_{des} = 7 \text{ m/s}$. Por lo tanto, siguiendo las recomendaciones de los autores de [7], para obtener una política estudiante que sea capaz de evadir obstáculos a una velocidad considerablemente inferior, es necesario realizar un proceso de ajuste fino de los pesos provistos. Según las recomendaciones de uso del método propuesto en [7], entrenar una política estudiante requiere de una base de datos extensa de trayectorias generadas por el experto privilegiado, esto acompañado de un tiempo de entrenamiento considerable; es por esto que en su lugar, recomiendan ajustar los pesos de la política estudiante que ejecuta con $v_{des} = 7 \text{ m/s}$ en lugar de entrenar desde cero.

Para realizar este ajuste fino, se utilizó la plataforma de generación de base de datos del experto informado publicada en el repositorio de [7]. Luego de ajustar la velocidad de ejecución del experto a $v_{des} = 1 \text{ m/s}$, se procedió a generar una base de datos de trayectorias libres de colisión dentro del entorno de “bosque denso”, que justamente es el entorno principal utilizado en el trabajo original [7] y que consiste en un bosque con árboles cuya posición se determina por una distribución aleatoria de *Poisson* [7]. Se realizaron 193 simulaciones, cada una generando entre 300 y 350 ejemplos. De esas 193 simulaciones, se tomaron 168 para formar una base de datos de entrenamiento y 25 para una base datos de validación; en otras palabras, se partió la base de datos en 88 % conjunto de entrenamiento y 12 % conjunto de validación.

Se realizó entrenamiento supervisado con la base de datos generada por un total de 33 épocas; comenzando desde la época en el que se encontraba el punto de control de los pesos publicados por [7], la época número 188; hasta la época

número 221. Por limitaciones de memoria del computador de entrenamiento, el valor del tamaño del *batch* se redujo de 8 a 1. El valor del resto de los hiperparámetros del entrenamiento fueron los mismos utilizados por [7] para la época 188.

5.5. Resumen

En este capítulo se detalla la implementación de la solución propuesta para abordar el problema de evasión de obstáculos en el contexto de los QUAVs. Comienza presentando la metodología y la plataforma de implementación en la Sección 5.1, destacando la infraestructura utilizada para desarrollar y evaluar la solución. En la Sección 5.2, se justifica la elección del algoritmo de evasión de obstáculos, basándose en el enfoque presentado en *Learning high-speed flight in the wild* [7].

La Sección 5.3 se dedica a la descripción detallada de la arquitectura de la solución propuesta, incluyendo el funcionamiento de cada uno de sus componentes. Allí se profundiza en los aspectos cruciales de la ejecución de la solución, en donde destacan el funcionamiento del proceso de inferencia y del ejecutor de trayectorias. Finalmente, en la Sección 5.4, se explora el proceso de refinamiento fino de la red neuronal encargada de la inferencia de trayectorias libre de colisión, es decir, el perfeccionamiento de la política estudiante.

El trabajo desarrollado en este capítulo representa el núcleo del proyecto. Especialmente hay que destacar el logro de implementar una solución que permite a un QUAV evadir obstáculos de forma autónoma y sin información previa del entorno, que además puede ejecutarse en tiempo real sobre un hardware ligero y con recursos limitados. Sin mencionar que la totalidad de la solución trabaja en concordancia y sobre el marco de trabajo de ACSL, permitiendo una integración automática con el ecosistema de aplicaciones de los vehículos de ACSL.

En general, este capítulo sienta las bases para la evaluación del desempeño de la solución propuesta, que se abordará en el próximo capítulo. El análisis detallado de la implementación proporciona una comprensión completa de la estructura y

funcionamiento de la solución, preparando el terreno para la discusión de resultados y conclusiones.

Capítulo 6

Resultados

En el capítulo anterior, se detalló la implementación de la solución propuesta para abordar el problema de evasión de obstáculos en QUAVs, centrándose en la metodología, la plataforma de implementación, la justificación del algoritmo seleccionado y la descripción del funcionamiento de los componentes principales. Este capítulo se adentra ahora en la evaluación y presentación de los resultados obtenidos, proporcionando una perspectiva crítica sobre el rendimiento de la solución propuesta.

En la Sección 6.1 se examinan los resultados del ajuste fino de la política estudiante, destacando los desafíos asociados al sobre-ajuste de los datos de entrenamiento y a la generación de la base de datos. Además, se exploran las consecuencias derivadas de estos problemas al evaluar la política de evasión de obstáculos.

La Sección 6.2 examina los resultados de la política de evasión de obstáculos, dividida en dos subsecciones clave: la Sección 6.2.1, que presenta los resultados de vuelos en simulación, y la Sección 6.2.2, centrada en vuelos sobre la plataforma física de implementación, SOTEN. Estos resultados ofrecen una perspectiva del rendimiento de la solución en configuraciones simples de obstáculos, cuya política de evasión se muestra estable. Además, se evalúan otras configuraciones de obstáculos en donde la política no logra completar su función de manera satisfactoria, y se identifica la relación entre este resultado y los desafíos asociados al ajuste fino de la política estudiante.

El análisis detallado de estos resultados proporciona una evaluación crítica de la efectividad de la solución propuesta, sirviendo como base para la discusión en el siguiente capítulo. Aquí, se extraerán conclusiones significativas y se identificarán posibles direcciones para mejorar el rendimiento en futuras investigaciones en el campo de evasión de obstáculos en QUAVs.

6.1. Resultados del ajuste fino de la política estudiante

Como se menciona en el capítulo anterior, el objetivo del ajuste fino de la política estudiante, es ajustar los pesos provistos por los autores del trabajo original [7] para que la red neuronal genere trayectorias con $v_{des} = 1$ m/s; idealmente, manteniendo un nivel de generalización comparable a la red publicada en [7]. Recordando que la función de pérdida de la red neuronal de la política estudiante tiene dos componentes: un componente espacial que cuantifica qué tan cerca se encuentran las trayectorias predichas a las generadas por el experto, y un componente de estimación que cuantifica qué tan correctamente se hace la estimación del costo de ejecución c_k [7]; la Figura 6.1 y la Figura 6.2 muestran el valor del componente espacial y de estimación en función de las épocas, tanto para el conjunto de entrenamiento como para el de validación.

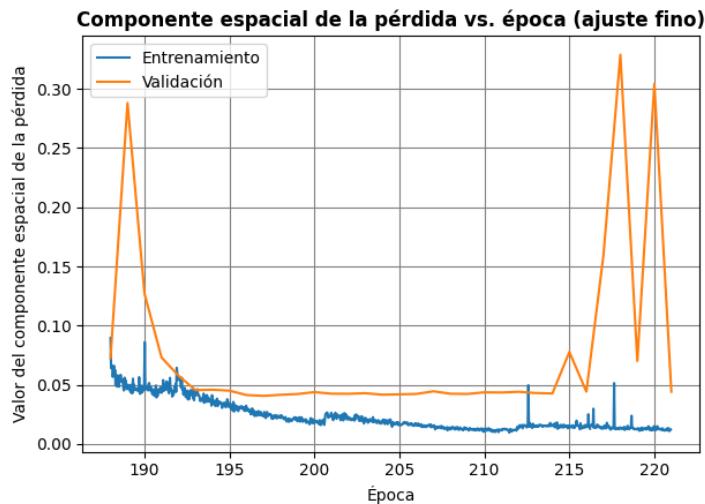


FIGURA 6.1: Componente espacial de la pérdida vs. época (ajuste fino). Se observa claramente la divergencia de la pérdida sobre el conjunto de validación a partir de la época 215.

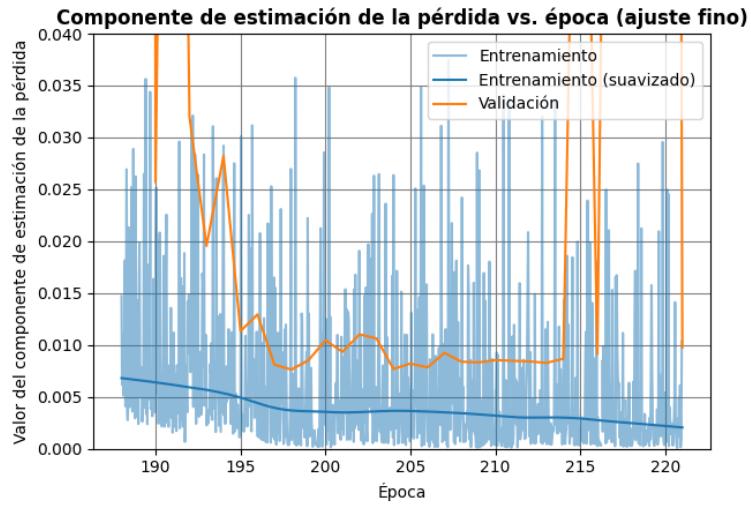


FIGURA 6.2: Componente de estimación de la pérdida vs. época (ajuste fino). Se observa claramente la divergencia de la pérdida sobre el conjunto de validación a partir de la época 215.

La Figura 6.1 y la Figura 6.2 reflejan la divergencia del valor de la pérdida sobre el conjunto de validación a partir de la época 215. Para contrarrestar este efecto, se decidió utilizar un punto de control temprano antes de la divergencia, efectivamente deteniendo el entrenamiento de forma temprana. El punto de control más cercano anterior a la época 215 fue el punto de control de la época 213. Al regresar a dicho punto de control se eliminó la divergencia del valor de pérdida de validación. La Figura 6.3 y la Figura 6.4 muestran el valor del componente espacial y de estimación en función de las épocas luego de detener tempranamente el entrenamiento, tanto para el conjunto de entrenamiento como para el de validación.

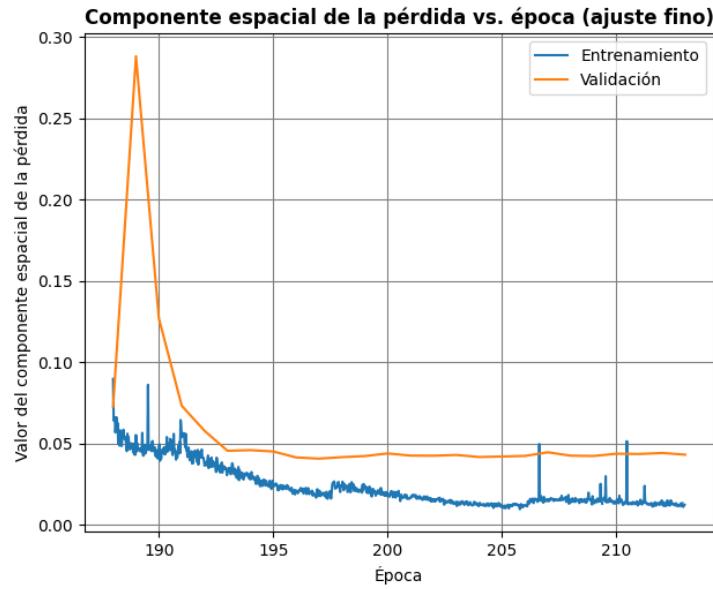


FIGURA 6.3: Componente espacial de la pérdida vs. época (ajuste fino) luego de detener el entrenamiento tempranamente. No se observa divergencia en el valor de la pérdida sobre el conjunto de validación.

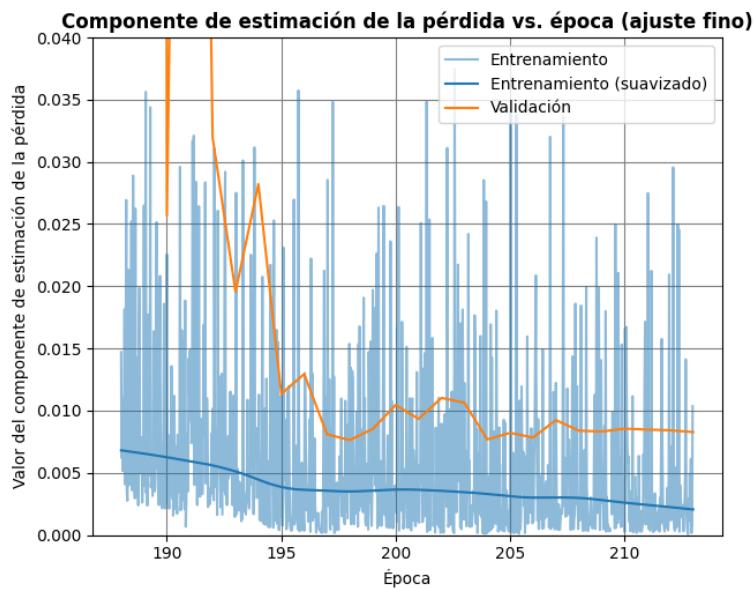


FIGURA 6.4: Componente de estimación de la pérdida vs. época (ajuste fino) luego de detener el entrenamiento tempranamente. No se observa divergencia en el valor de la pérdida sobre el conjunto de validación.

Luego de eliminar la divergencia de la pérdida de validación, se puede proceder a evaluar este modelo contra el modelo original. Como las métricas de utilizadas

en la evaluación del método propuesto en [7] evalúan el sistema completo, la única métrica que se puede utilizar para comparar modelos es, exclusivamente, el valor de la función de pérdida a lo largo de un conjunto de datos. Para que esta comparación sea efectiva, lo importante es seleccionar conjuntos de datos representativos de la comparación deseada. Precisamente esto es lo que se utilizó en este trabajo como herramienta de comparación del desempeño de los modelos de interés. El procedimiento y resultados de la evaluación se describe a continuación.

Los modelos a comparar son el modelo original, que destaca en la capacidad de predecir trayectorias sin colisión con $v_{des} = 7 \text{ m/s}$; y el modelo ajustado en este trabajo para predecir trayectorias sin colisión con $v_{des} = 1 \text{ m/s}$. Para ello, se seleccionaron dos conjuntos de datos, el conjunto de datos de validación con trayectorias con $v_{des} = 7 \text{ m/s}$ (utilizado en el entrenamiento del modelo original) y el conjunto de datos de validación con las trayectorias con $v_{des} = 1 \text{ m/s}$ (el generado en el presente trabajo). La diferencia en los valores de pérdida entre esos modelos y conjuntos de datos, dan una idea del rendimiento relativo de cada uno de los modelos dentro de los escenarios presenciados en el conjunto de datos. La Tabla 6.1 compara los valores de pérdida de cada modelo sobre cada conjunto de datos de interés.

Modelo	Conjunto de datos	L_{esp}	L_{est}
Original	Original a 7 m/s	1.407	0.1069
Original	Generado a 1 m/s	41.7108	0.0842
Ajustado	Generado a 1 m/s	0.0441	0.0008

TABLA 6.1: Comparación de los valores de pérdida entre modelo original y ajustado según el conjunto de datos evaluado. L_{esp} es el valor del componente espacial de la pérdida y L_{est} es el valor del componente de estimación de la pérdida.

De esta comparación podemos realizar algunas observaciones relevantes. Primero, el modelo original tiene dificultades en generar trayectorias con las características espaciales de las trayectorias con $v_{des} = 1 \text{ m/s}$, esto tiene sentido pues la diferencia espacial entre moverse a 7 m/s y 1 m/s es bastante alta; esto simplemente confirma la dependencia entre v_{des} del experto y el desempeño de la política estudiante a distintas velocidades, tal como se afirma en [7]. Por otro lado, el modelo original logra generalizar la tarea de estimar el costo de ejecución de una trayectoria, los valores del componente de estimación de la pérdida están en el mismo orden tanto para $v_{des} = 7 \text{ m/s}$ como para $v_{des} = 1 \text{ m/s}$.

Segundo, y más relevante para este trabajo, los valores de pérdida del modelo ajustado sobre el conjunto con $v_{des} = 1$ m/s, están dos órdenes de magnitud por debajo de los valores del modelo original sobre el conjunto con $v_{des} = 7$ m/s; esto puede indicar la presencia de sobre-ajuste (pérdida de capacidad de generalización), sin embargo, este sobre-ajuste no existe al nivel de conjuntos de entrenamiento y de validación, sino que existe al nivel de los conjuntos de datos generados por el experto con distintos valores de v_{des} . Para confirmar esta línea de pensamiento, se procede a visualizar una muestra de las trayectorias generadas con $v_{des} = 1$ m/s y compararlas con una muestra de las trayectorias generadas con $v_{des} = 7$ m/s. Si ambos conjuntos de datos tienen niveles equivalentes de generalización, se debería observar en cada muestra una representación amplia de distintas trayectorias. Si alguna de las dos muestras tiene preferencia por un caso particular de trayectorias con respecto a la otra muestra, entonces esa base de datos tiene menos capacidad de generalización relativamente. De cada base de datos se seleccionaron tres vuelos aleatoriamente, la Figura 6.5 y la Figura 6.6 visualizan las muestras seleccionadas. Cada muestra se visualiza como una vista “de arriba hacia abajo” de la nube de puntos del entorno junto con las trayectorias generadas durante el vuelo; el color de las trayectorias interpola entre azul y rojo, mientras más rojo, más costo de ejecución tiene la trayectoria. Este experimento se repitió 10 veces con resultados similares. Para facilitar la visualización de los resultados, solo se muestra un solo experimento.

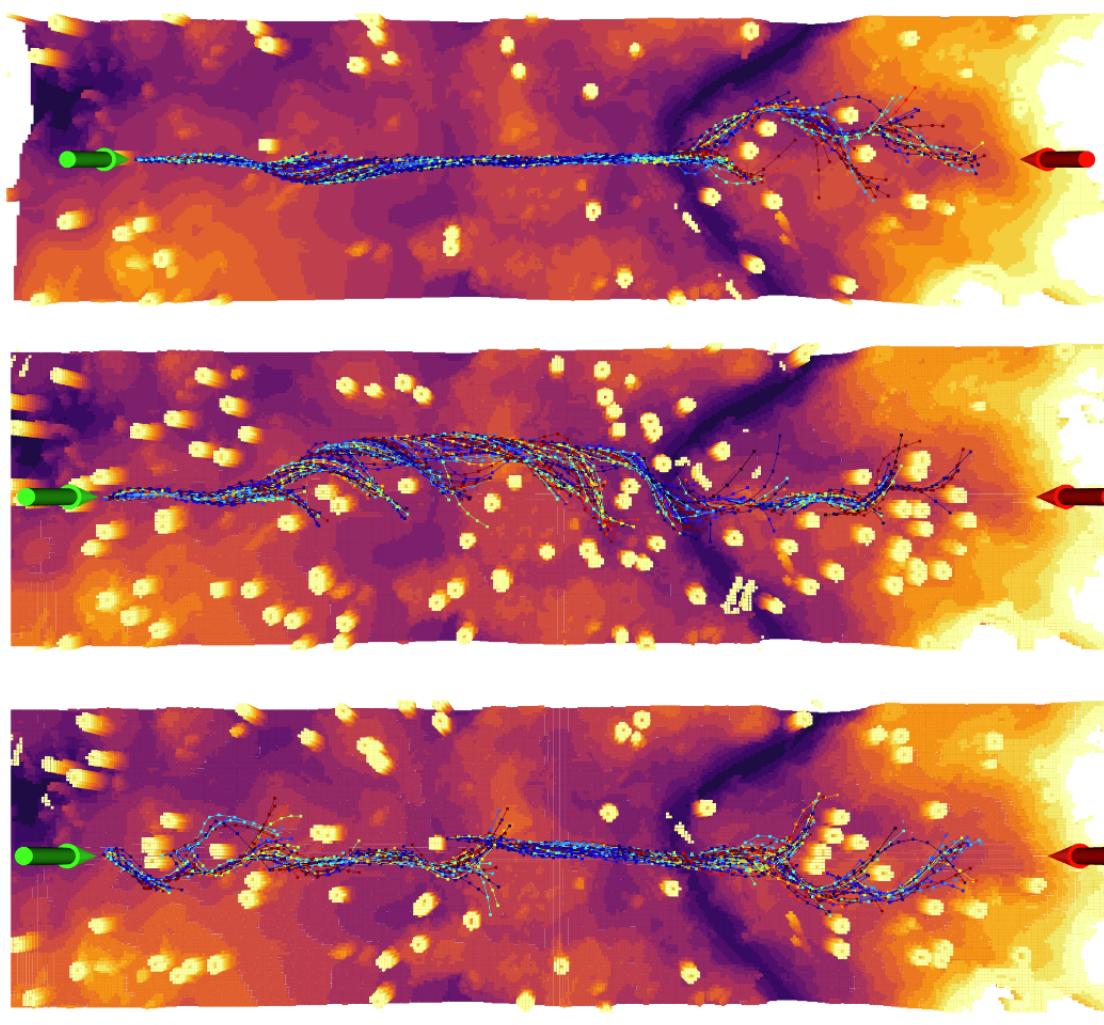


FIGURA 6.5: Visualización de tres vuelos de la base de datos con $v_{des} = 7 \text{ m/s}$.

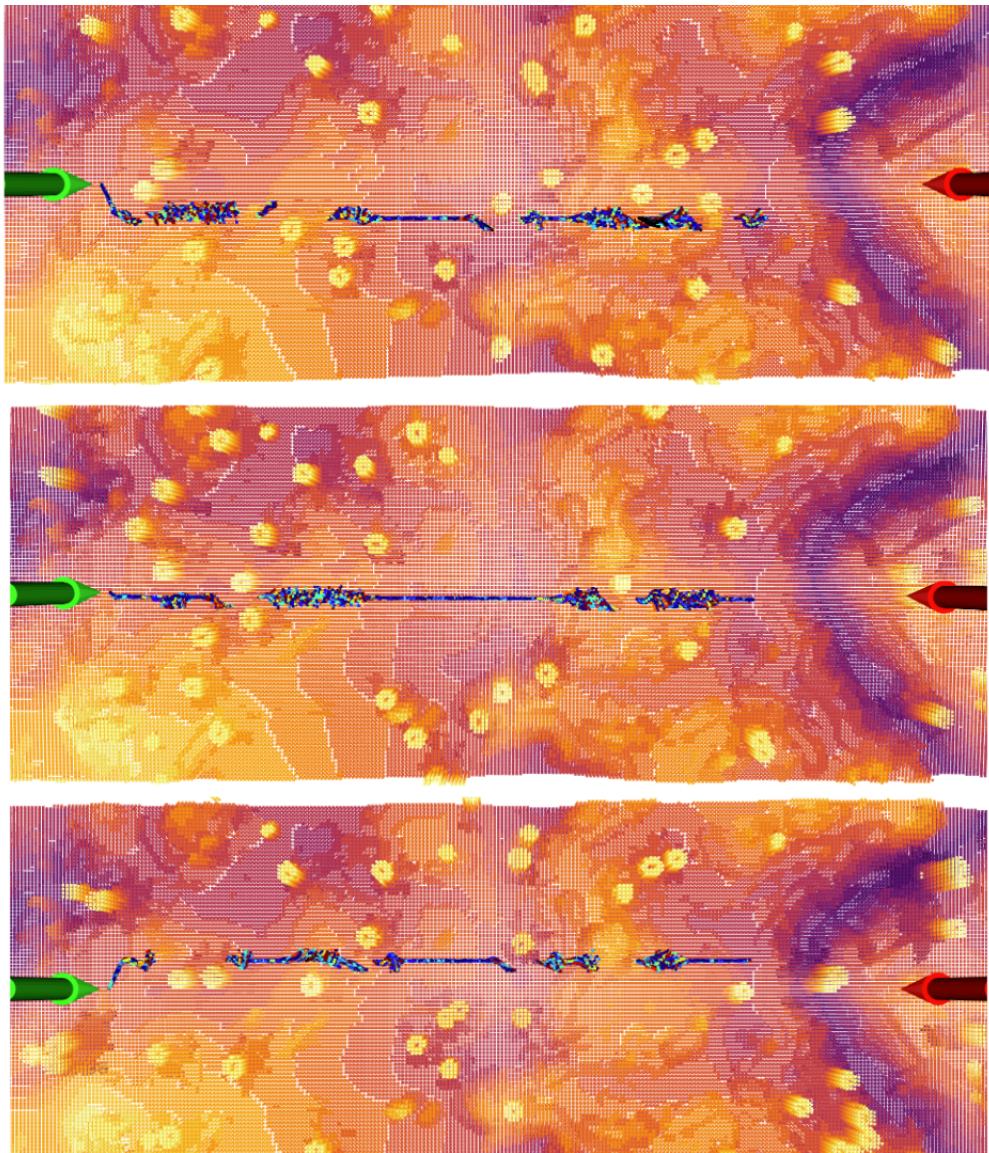


FIGURA 6.6: Visualización de tres vuelos de la base de datos con $v_{des} = 1 \text{ m/s}$.

A simple vista se observa una clara diferencia, en la base de datos con $v_{des} = 1 \text{ m/s}$, los obstáculos solo se esquivan por uno solo de los flancos del obstáculo, mientras en la base de datos con $v_{des} = 7 \text{ m/s}$ se exploran ambos flancos como posibilidades de esquivar el obstáculo. Para cuantificar esta observación, se procedió a contar para cada conjunto de datos, el porcentaje de ejemplos en donde la trayectoria se dirige hacia la izquierda, hacia la derecha y hacia al frente. Para determinar si un ejemplo de la base de datos va hacia a la izquierda, derecha o hacia el frente; se calculan D_r y D_l los desplazamientos máximos hacia la derecha y hacia la izquierda respectivamente, con respecto al marco de referencia del cuerpo del QUAV. Luego si $|D_r - D_l| < 0,1$ se considera que la trayectoria va

hacia el frente, si $D_r > D_l$ se considera que la trayectoria va hacia la derecha y si $D_r < D_l$ se considera que la trayectoria va hacia la izquierda. Solo se consideran los ejemplos en donde existe un obstáculo visible a 5m de distancia. La Tabla 6.2 muestra los resultados de este conteo.

Conjunto de datos	P_{left}	P_{right}	P_{str}	$N_{samples}$
$v_{des} = 7 \text{ m/s}$	51.3 %	44.1 %	4.5 %	5517
$v_{des} = 1 \text{ m/s}$	40.6 %	35.1 %	24.3 %	18244

TABLA 6.2: Distribución de ejemplos en cada conjunto de datos según hacia donde se dirige la trayectoria. P_{left} es el porcentaje de ejemplos en donde la trayectoria se dirige hacia la izquierda, P_{right} es el porcentaje de ejemplos en donde la trayectoria se dirige hacia la derecha, P_{str} es el porcentaje de ejemplos en donde la trayectoria se dirige hacia el frente y $N_{samples}$ es el número de ejemplos en el conjunto de datos.

De los resultados presentados en la Tabla 6.2 podemos realizar varias observaciones. Primero ambos conjuntos de datos tienen un desbalance entre P_{left} y P_{right} , con P_{left} dominando por 5 %. Segundo, el conjunto de datos con $v_{des} = 1 \text{ m/s}$ contiene un número mucho mayor de trayectorias que se dirigen hacia el frente, con un 24.3 % de los ejemplos, mientras que el conjunto de datos con $v_{des} = 7 \text{ m/s}$ solo tiene un 4.5 % de los ejemplos. Tercero, el conjunto de datos con $v_{des} = 1 \text{ m/s}$ tiene un 230 % más ejemplos que el conjunto de datos con $v_{des} = 7 \text{ m/s}$, esto probablemente como consecuencia de recorrer una distancia menor en promedio por trayectoria (por la menor rapidez de desplazamiento).

Como la decisión de esquivar un obstáculo en el plano xy se reduce a esquivar por el flanco izquierdo o por el flanco derecho del obstáculo (pues seguir hacia el frente implica colisión), el desbalance entre P_{left} y P_{right} es el factor mas relevante en la diferencia de capacidad de generalización entre los conjuntos de datos. Este factor es además acentuado en el conjunto con $v_{des} = 1 \text{ m/s}$ por el mayor número de ejemplos que se dirigen hacia el frente, y por el mayor número de ejemplos en general; haciendo que el 5 % de desbalance entre P_{left} y P_{right} sea mucho más relevante. Es por esto que el modelo ajustado va a tener una preferencia a esquivar obstáculos por el flanco izquierdo. En otras palabras, existe un desbalance de las modas de la distribución P^1 en la base de datos con $v_{des} = 1 \text{ m/s}$ que no existe significativamente en la base de datos con $v_{des} = 7 \text{ m/s}$. Este desbalance hace que el proceso de refinamiento fino produzca un efecto de sobre-ajuste con respecto

¹La distribución P se describe en la Ecuación 4.2

al rendimiento de la política publicada en [7]. El modelo se ajusta demasiado a los ejemplos de la base datos con $v_{des} = 1$ m/s, en donde existe una preferencia significativa a esquivar obstáculos por el flanco izquierdo, potencialmente reduciendo su capacidad multi-modal al momento de generar trayectorias.

Un posible origen de este problema, es que la poca longitud de las trayectorias generadas para $v_{des} = 1$ m/s no permite que el experto genere trayectorias que consideren todas las modas de la distribución P con un valor de costo bajo. Cuando el experto selecciona las mejores tres trayectorias, algunas de las trayectorias que consideran las modas de P se descartan, produciendo el desbalance. Otra posible razón puede ser que el planificador de trayectorias globales que genera τ_{ref} no permita la exploración de dichas trayectorias, principalmente por planificar trayectorias triviales en donde se navegue mayoritariamente sin encontrar obstáculos.

Una posible solución para este problema es modificar el entorno de simulación para que la configuración de los obstáculos permita que el experto considere las modas de P con alto valor de costo, por ejemplo, incrementando la densidad de los obstáculos y reduciendo sus dimensiones. Otra posible solución es modificar los parámetros del planificador de trayectorias globales de forma que la ejecución de τ_{ref} fomente la exploración de obstáculos en donde se generen muestras que representen el carácter multi-modal de P , por ejemplo, intencionalmente dirigiéndose hacia la cercanía de grupos de obstáculos. Alternativamente se pueden ejecutar las predicciones de la red en lugar de seguir una τ_{ref} planificada globalmente, y en su lugar utilizar una τ_{ref} que simplemente sea el camino en línea recta desde el punto de despegue hasta la meta; esto potencialmente va a incrementar la exploración de obstáculos en donde se generen muestras que representen múltiples modas de P .

Dicho esto, por limitaciones del tiempo de desarrollo, y recomendaciones por parte del equipo de ACSL, en este trabajo no se soluciona el problema descrito anteriormente. Esto significa que la política estudiante utilizada en este trabajo sufre las consecuencias del sobre-ajuste ocasionado por el desbalance de modas en la base de datos con $v_{des} = 1$ m/s, esto puede ocasionar que la política de evasión de obstáculos tenga una preferencia fuerte en esquivar obstáculos de una forma en particular. En las siguientes secciones se observa que efectivamente, la política de evasión de obstáculos tiene una fuerte preferencia a esquivar obstáculos por el

flanco izquierdo, así como también, que este comportamiento hace que la ejecución en entornos no triviales termine en potenciales colisiones.

6.2. Resultados de la política de evasión de obstáculos

6.2.1. Vuelos en entorno simulación

En esta sección se describen las configuraciones de los vuelos realizados en el entorno de simulación AirSim [33], así como también sus resultados y comportamientos observados. Es importante recordar que por motivos de validación del concepto por parte de ACSL, la ejecución de la evasión de obstáculos está limitada al plano xy del marco de referencia del cuerpo del QUAUAV.

La primera configuración de obstáculos utilizada es una configuración simple con un solo obstáculo en la línea directa de visión del QUAUAV, un diagrama de la configuración se muestra en la Figura 6.7. En el diagrama se muestran dos vistas, la frontal (plano yz) a partir desde el punto de despegue y la vista de arriba hacia abajo (plano xy).

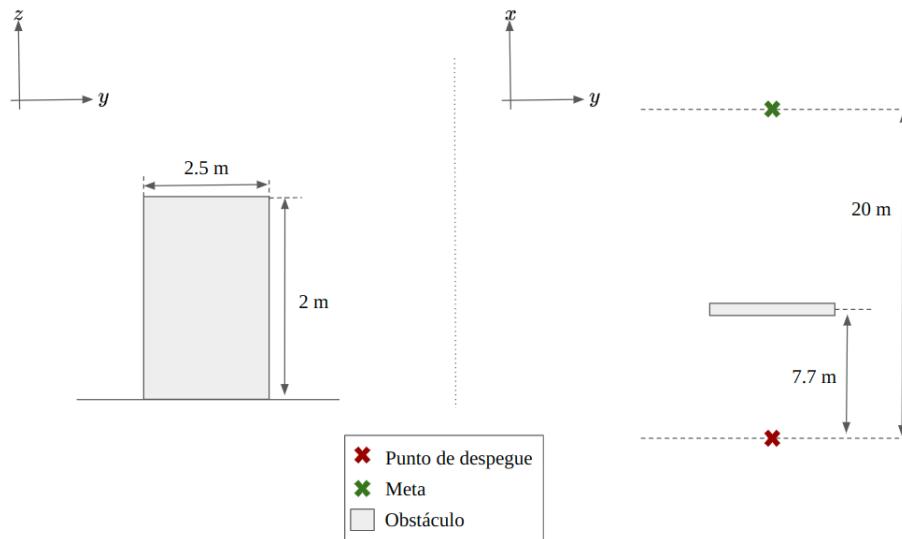


FIGURA 6.7: Primera configuración de obstáculos dentro del entorno de AirSim.
Obstáculo simple.

El comportamiento del algoritmo se mostró capaz de evadir el obstáculo de forma robusta, en concreto, de 10 vuelos ejecutados, 10 resultaron en vuelos sin colisión. La Figura 6.8 muestra una visualización desde arriba hacia abajo de los caminos resultantes de seguir las trayectorias generadas por el algoritmo para cuatro vuelos. Todos los vuelos realizados en esta configuración muestran un comportamiento similar a lo que se observa en la Figura 6.8, esto es, esquivar el obstáculo hacia la dirección $-y_B$ ¹ y dirigirse hacia la meta una vez no es visible el obstáculo.

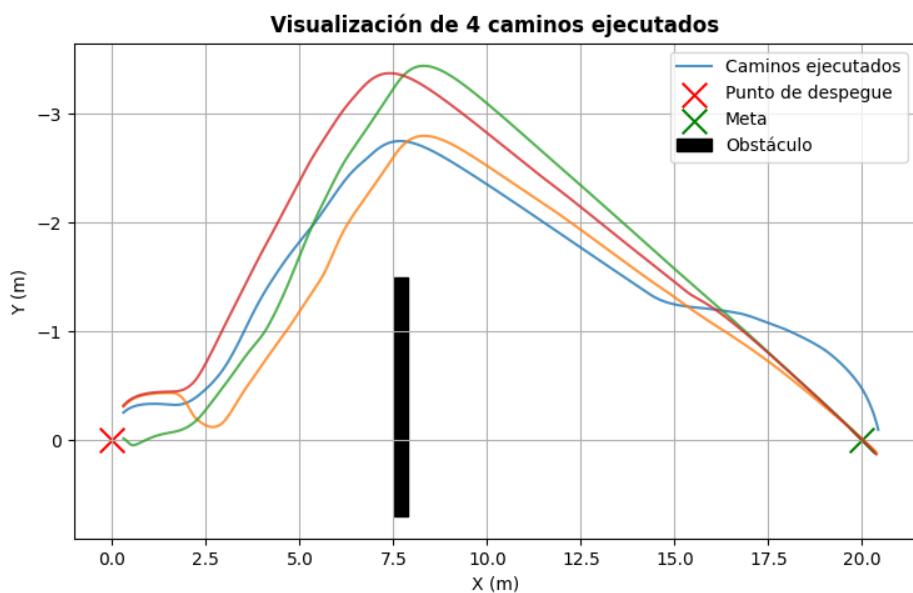


FIGURA 6.8: Visualización simultánea de cuatro caminos ejecutados dentro de la primera configuración de obstáculos en entorno de simulación. De 10 vuelos ejecutados dentro de esta configuración, 10 resultaron sin colisión.

La diferencia más marcada entre vuelos en esta configuración es la desviación máxima del camino ejecutado con respecto al camino directo entre el punto de despegue y la meta, la Figura 6.9 muestra un diagrama de caja de la desviación máxima de los caminos ejecutados en esta configuración. La desviación máxima promedio de los caminos ejecutados en esta configuración es cerca de 3.1 metros, con una desviación máxima de cerca de 3.45 metros y una desviación mínima de cerca de 2.75 metros.

¹Donde B es el marco de referencia del cuerpo del QUAV.

Diagrama de caja de las desviaciones máximas para:
Simulación, obstáculo simple

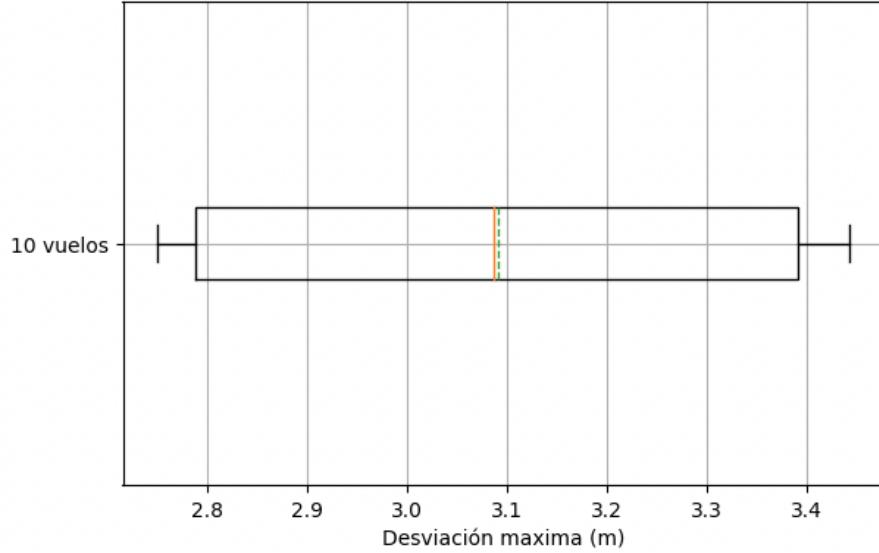


FIGURA 6.9: Diagrama de caja de la desviación máxima de los caminos ejecutados en la primera configuración de obstáculos en entorno de simulación.

La siguiente configuración utilizada es una configuración similar a la anterior pero con un obstáculo adicional también en la misma línea de visión del QUAV. La Figura 6.10 muestra el diagrama de la segunda configuración.

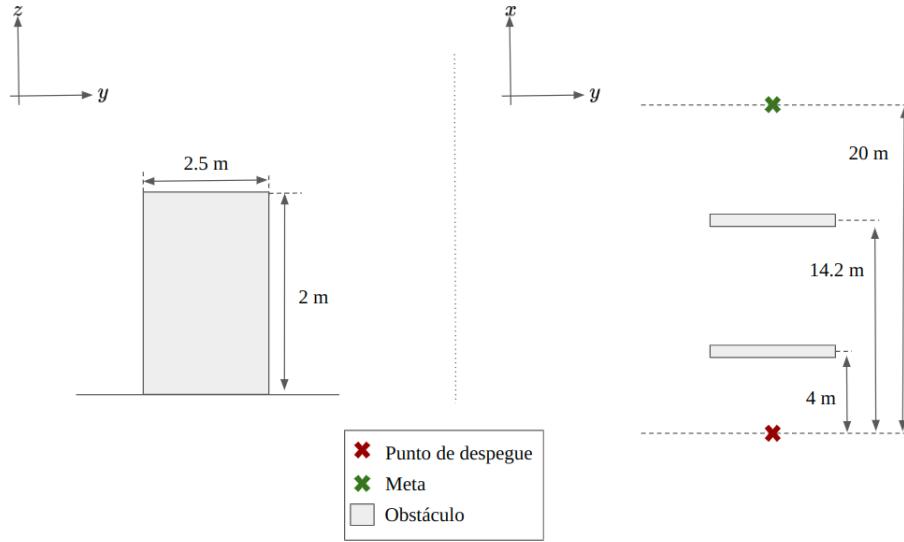


FIGURA 6.10: Segunda configuración de obstáculos dentro del entorno de AirSim. Dos obstáculos consecutivos en la línea de visión del QUAV.

Tal como en la configuración anterior, el comportamiento de algoritmo fue estable, de 10 vuelos ejecutados, 10 resultaron en vuelos sin colisión. La Figura

6.11 muestra una visualización desde arriba hacia abajo de caminos resultantes de seguir las trayectorias generadas por el algoritmo para cuatro vuelos. Todos los vuelos realizados en esta configuración muestran un comportamiento similar a lo que se observa en la Figura 6.11.

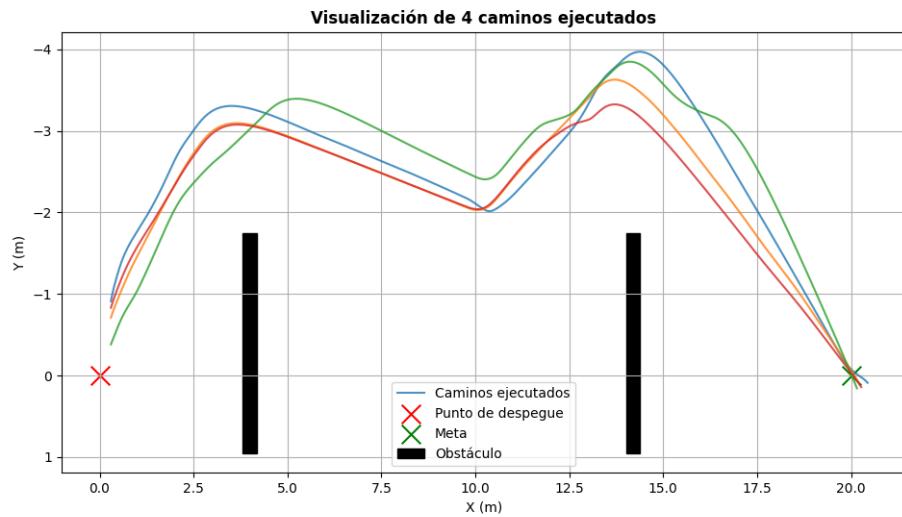


FIGURA 6.11: Visualización simultánea de 4 caminos ejecutados dentro de la segunda configuración de obstáculos en entorno de simulación. De 10 vuelos ejecutados dentro de esta configuración, 10 resultaron sin colisión.

En lo que respecta a la desviación máxima de los caminos ejecutados en esta configuración, la Figura 6.12 muestra un diagrama de caja de la desviación máxima de los caminos ejecutados. La desviación máxima promedio de los caminos ejecutados en esta configuración es cerca de 3.7 metros, con una desviación máxima de cerca de 4 metros y una desviación mínima de cerca de 3.3 metros. Estos resultados muestran como la maniobra para esquivar el segundo obstáculo incrementa la desviación máxima de los caminos ejecutados.

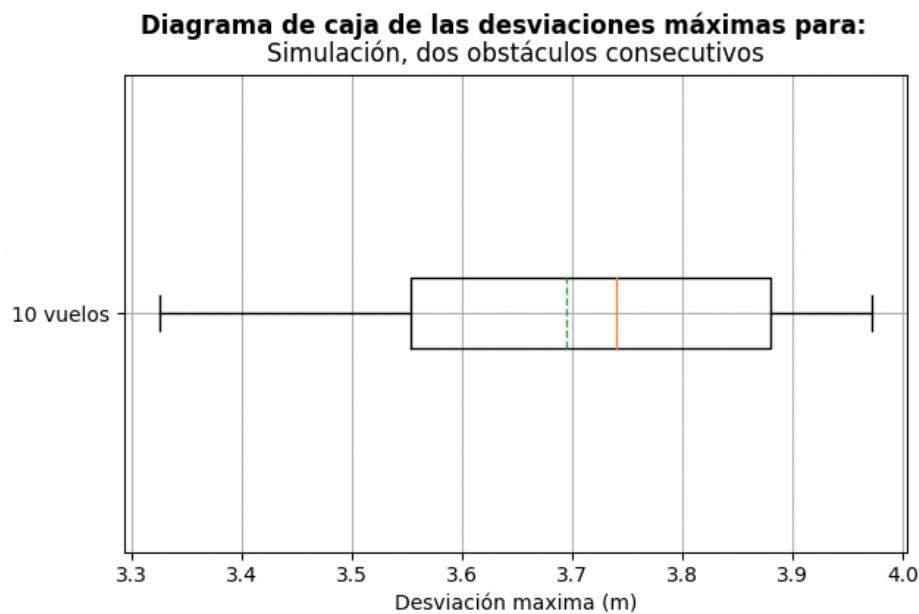


FIGURA 6.12: Diagrama de caja de la desviación máxima de los caminos ejecutados en la segunda configuración de obstáculos en entorno de simulación.

Es interesante resaltar, que en los vuelos dentro de la segunda configuración, se confirma el funcionamiento del mecanismo que desactiva la inferencia cuando no hay obstáculos visibles. Para visualizar esto, se desarrolló una visualización 3D que permite visualizar la ejecución del algoritmo incluyendo la información sensorial de profundidad y las trayectorias generadas. En esta visualización se muestra:

- La nube de puntos del entorno percibido por las cámaras estereoscópicas del QUAV, el color de los puntos indica la profundidad con respecto al sistema de cámaras, mientras más rojo más cerca y mientras más azul más lejos.
- El camino ejecutado por el QUAV, que se muestra como una curva de color blanco.
- La trayectoria más reciente generada por la política estudiante, que se muestra como una curva de color verde que comienza en la posición en donde se realizó la inferencia.
- El radio de colisión del QUAV, que se muestra como una circunferencia de color azul alrededor de la posición del QUAV. Si un obstáculo entra dentro de esta circunferencia se considera una colisión.

- La dirección del encabezamiento del QUAV, que se muestra como una línea roja que parte de la posición del QUAV.
- Las posiciones de despegue y de meta, que se muestran como un círculo rojo y verde respectivamente.
- El camino directo entre el punto de despegue y la meta.
- La orientación tridimensional del entorno, que se muestra como tres flechas; una azul que representa la dirección del eje z , una verde que representa la dirección del eje y y una roja que representa la dirección del eje x . Es importante mencionar que debido a limitaciones del programa de visualización, la dirección del eje y se encuentra invertida con respecto al resto de visualizaciones en este capítulo.

La Figura 6.13, la Figura 6.14 y la Figura 6.15 muestran capturas consecutivas de la visualización 3D de un vuelo dentro de la segunda configuración, desde una perspectiva de arriba hacia abajo. En estas figuras podemos observar como solo se generan trayectorias cuando hay un obstáculo cerca en el campo de visión del QUAV.

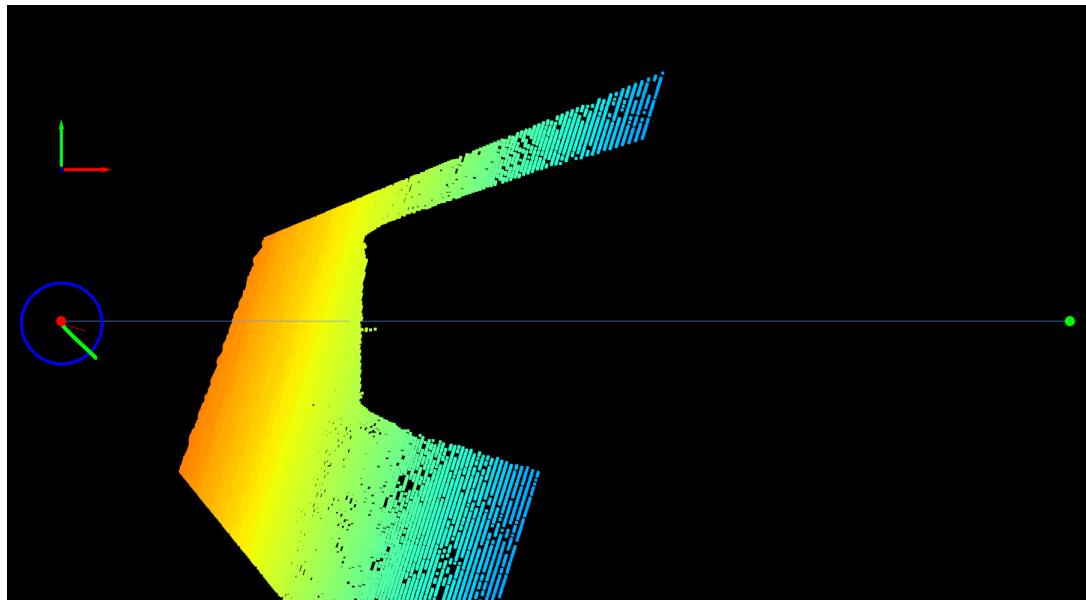


FIGURA 6.13: Visualización del funcionamiento del mecanismo que desactiva la inferencia cuando no hay obstáculos visibles. Inferencia activada, el primer obstáculo está en el campo de visión.

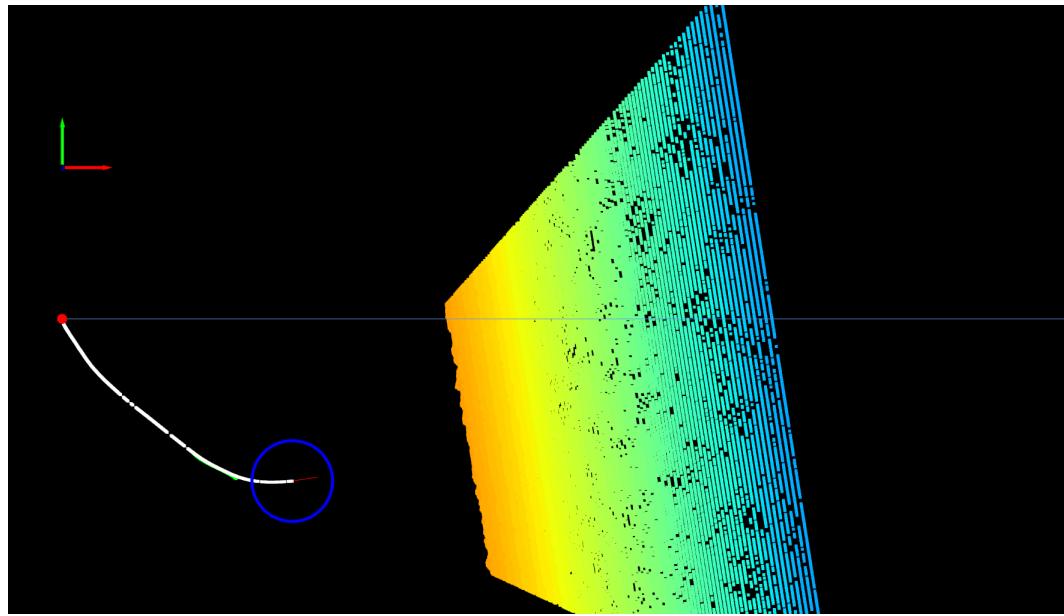


FIGURA 6.14: Visualización del funcionamiento del mecanismo que desactiva la inferencia cuando no hay obstáculos visibles. Inferencia desactivada, no hay obstáculo visible en campo de visión, navegando en dirección a la meta.

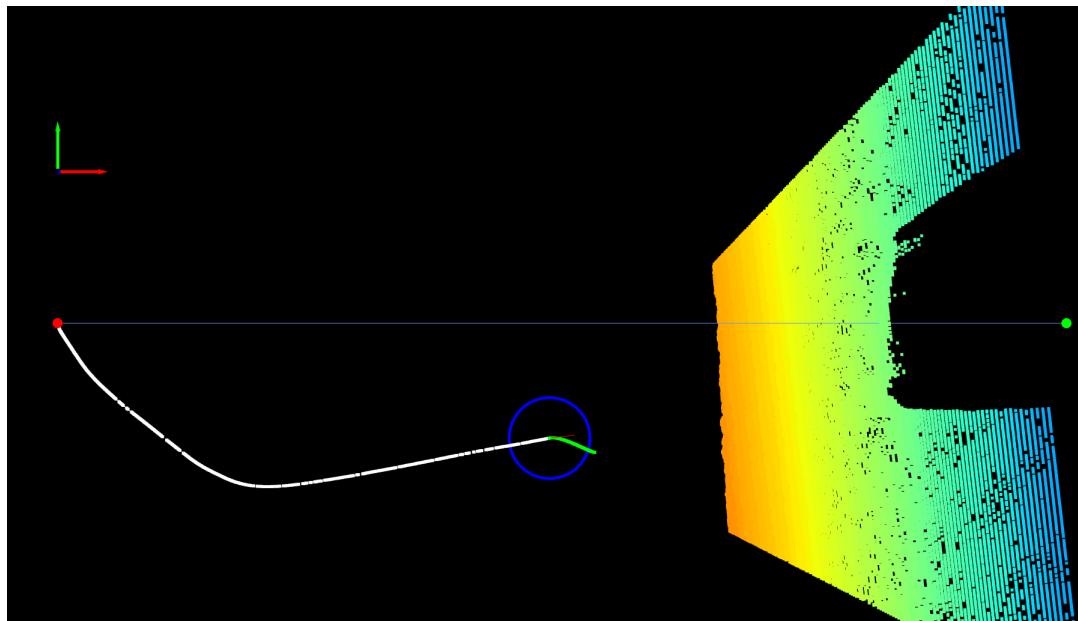


FIGURA 6.15: Visualización del funcionamiento del mecanismo que desactiva la inferencia cuando no hay obstáculos visibles. Inferencia activada, el segundo obstáculo está en el campo de visión.

Durante todas las pruebas descritas anteriormente, se observó que la política de evasión de obstáculos tiene una preferencia certera a esquivar obstáculos en la dirección $-y_B$, para evaluar este comportamiento el siguiente conjunto de pruebas

utiliza una configuración que coloca un obstáculo adicional hacia esa dirección. La Figura 6.16 muestra la tercera configuración utilizada, que consiste de dos obstáculos simples paralelos entre si con una separación de 2 metros.

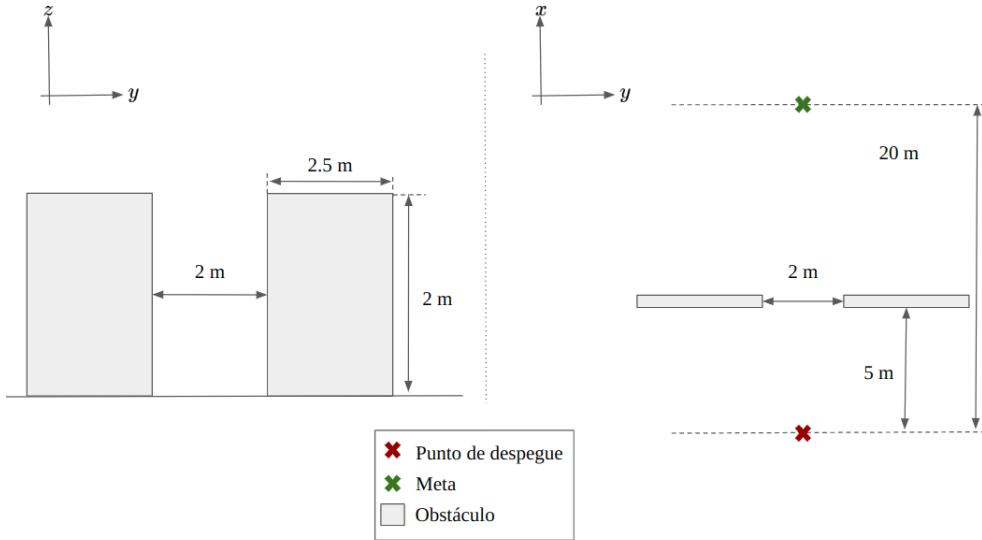


FIGURA 6.16: Tercera configuración de obstáculos dentro del entorno de AirSim.
Dos obstáculos paralelos con una separación de 2 metros.

Incluso con la presencia de un obstáculo adicional en la dirección $-y_B$, la política de evasión de obstáculos en todas las pruebas realizadas generó trayectorias dirigidas hacia $-y_B$, lo cual produjo colisiones frecuentes, de 10 vuelos ejecutados, 2 fueron sin colisión. La Figura 6.17, la Figura 6.18, la Figura 6.19 y la Figura 6.20 muestran una secuencia consecutiva de capturas de la visualización 3D de un vuelo que resultó en colisión. Tal como se aprecia en las figuras, mientras ambos obstáculos son visibles en el campo de visión del QUAU, la trayectoria se dirige al espacio entre ambos obstáculos, con un comportamiento que parece guiar al QUAU a una ejecución sin colisiones; sin embargo, en el momento que solo un de los dos obstáculos es visible, en lugar de continuar dirigiéndose hacia el espacio entre ambos obstáculos, la política intenta corregir la trayectoria para esquivar por el flanco en dirección $-y_B$ relativo al obstáculo, lo cual produce una colisión.

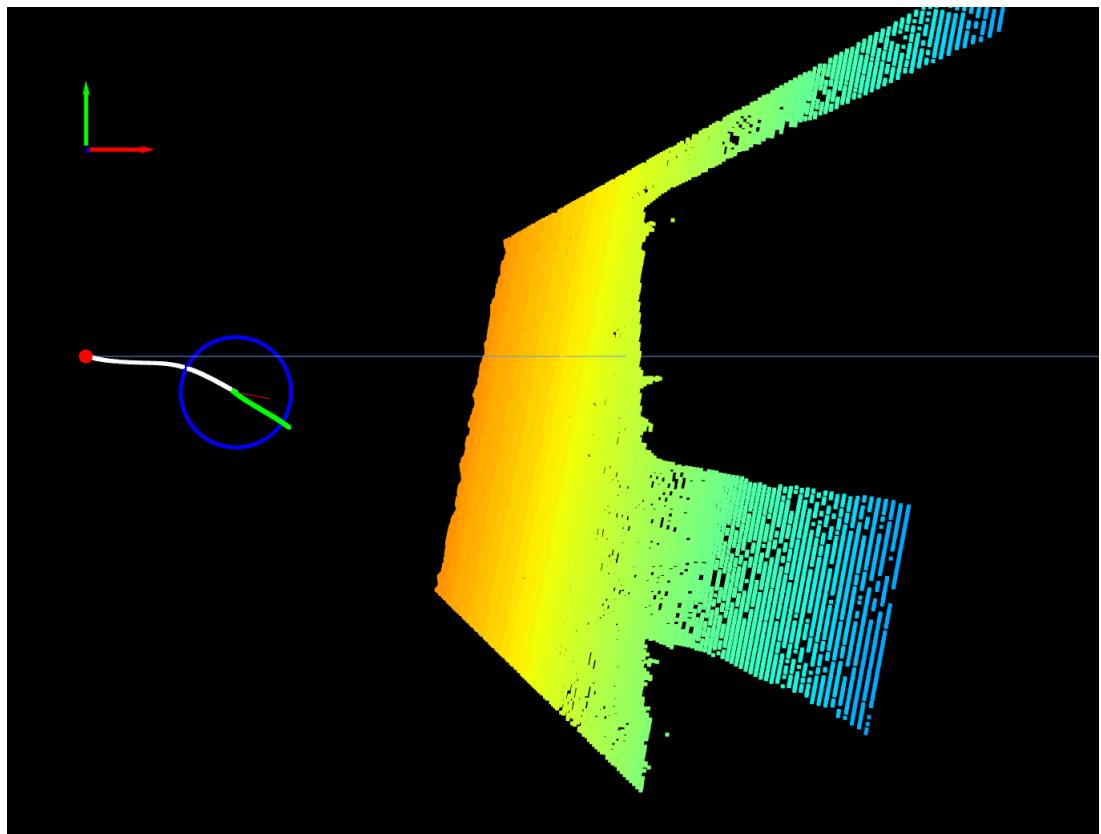


FIGURA 6.17: Visualización 3D de un vuelo en la tercera configuración de obstáculos (1). Vista de arriba hacia abajo. Ambos obstáculos están dentro del campo de visión, las trayectorias generadas se dirigen hacia el espacio entre los obstáculos.

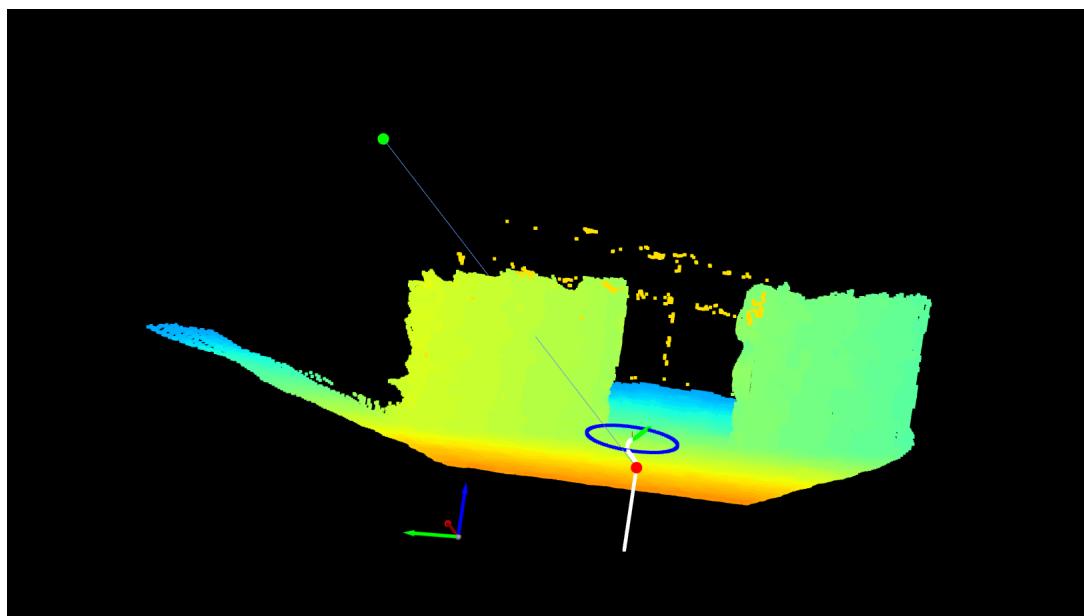


FIGURA 6.18: Visualización 3D de un vuelo en la tercera configuración de obstáculos (2). Vista frontal. Ambos obstáculos están dentro del campo de visión, las trayectorias generadas se dirigen hacia el espacio entre los obstáculos.

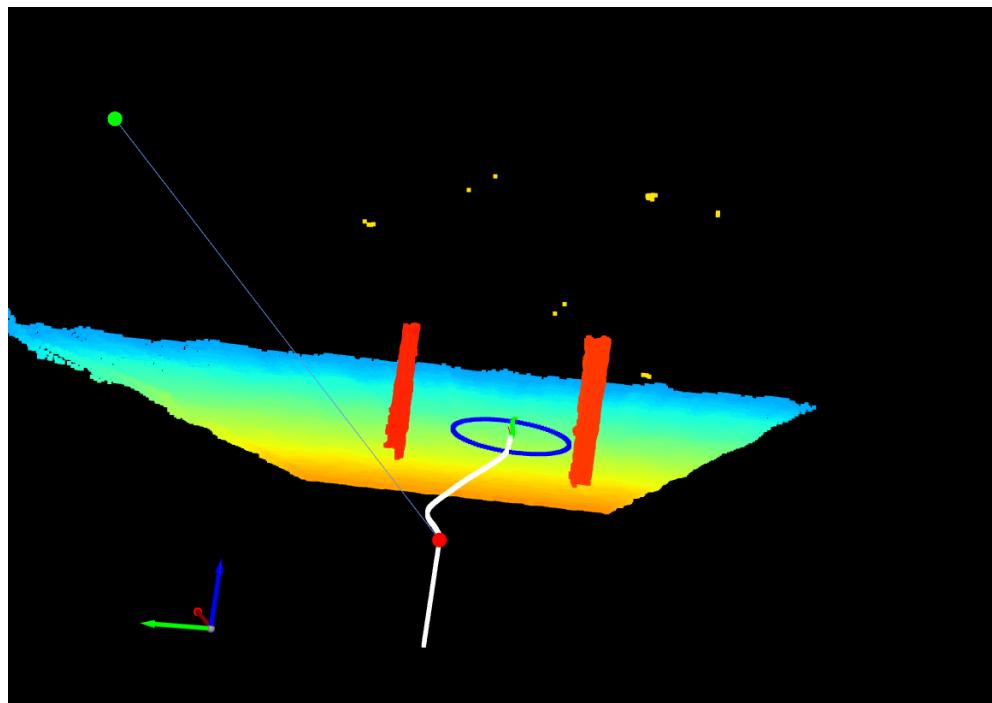


FIGURA 6.19: Visualización 3D de un vuelo en la tercera configuración de obstáculos (3). Vista frontal. Navegando entre el espacio entre los obstáculos. Ambos obstáculos están dentro del campo de visión, las trayectorias generadas se dirigen hacia el espacio entre los obstáculos.

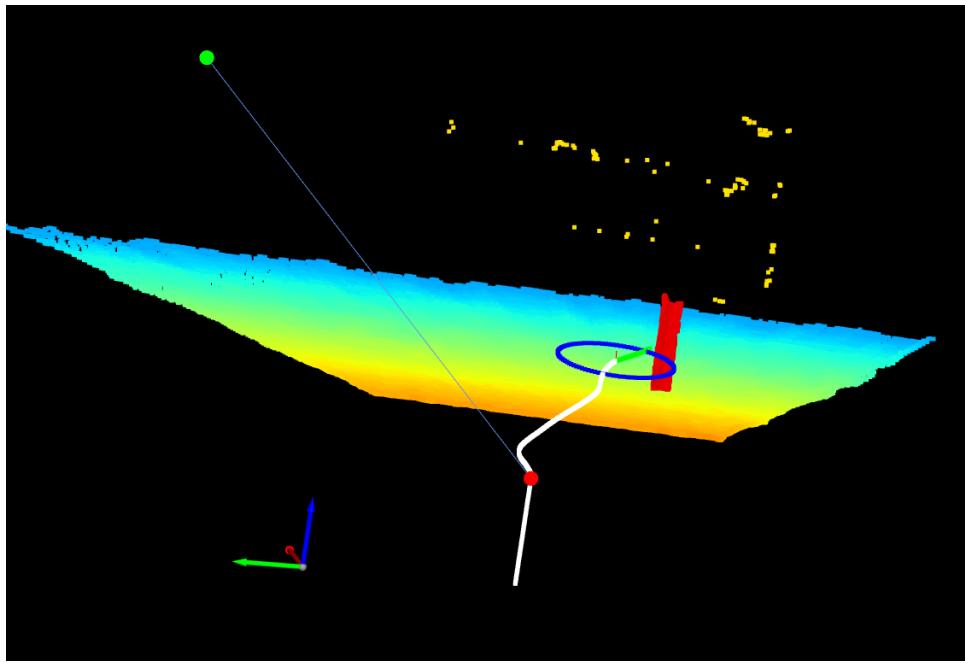


FIGURA 6.20: Visualización 3D de un vuelo en la tercera configuración de obstáculos (4). Vista frontal. En el momento que uno de los dos obstáculos desaparece del campo de visión, en lugar de continuar dirigiéndose hacia el espacio entre ambos obstáculos, la política intenta corregir la trayectoria para esquivar por el flanco en dirección $-y_B$ relativo al obstáculo, lo cual produce una colisión.

Para evaluar esta preferencia fuerte a esquivar por el flanco en la dirección $-y_B$ con respecto al obstáculo, la siguiente configuración a evaluar es de tal forma que no existe una forma de esquivar por el flanco en la dirección $-y_B$ sin regresar hacia atrás, la Figura 6.21 muestra un diagrama de la configuración en cuestión.

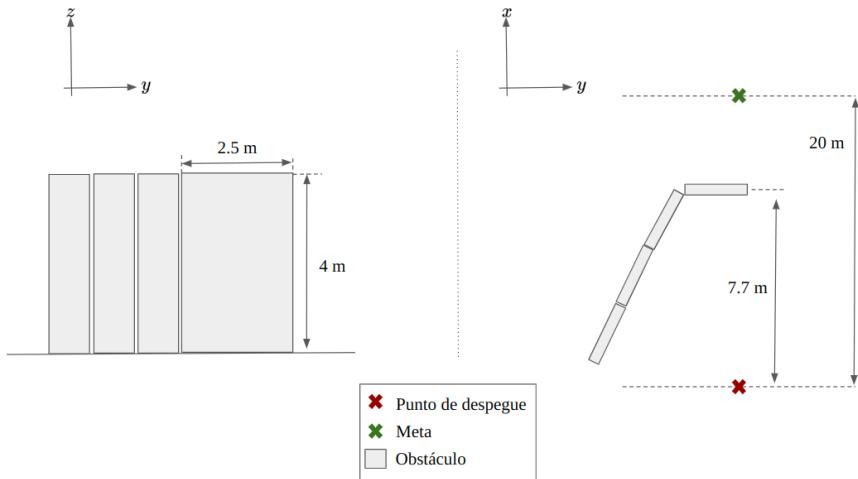


FIGURA 6.21: Cuarta configuración de obstáculos dentro del entorno de AirSim. Obstáculo simple con pared del lado izquierdo.

Todos los vuelos realizados en esta configuración terminaron en una colisión. La Figura 6.22, la Figura 6.23, la Figura 6.24, la Figura 6.25 y la Figura 6.26 muestran una secuencia consecutiva de capturas de la visualización 3D de un vuelo en esta configuración que resultó en colisión. En las figuras se observa como la política de evasión de obstáculos intenta generar trayectorias que se dirigen hacia la región en la dirección $-y_B$ en donde no hay información de profundidad, probablemente esperando encontrar el borde del obstáculo en esa dirección, pero como en esta configuración de obstáculos ese borde se encuentra hacia atrás, se produce una colisión.

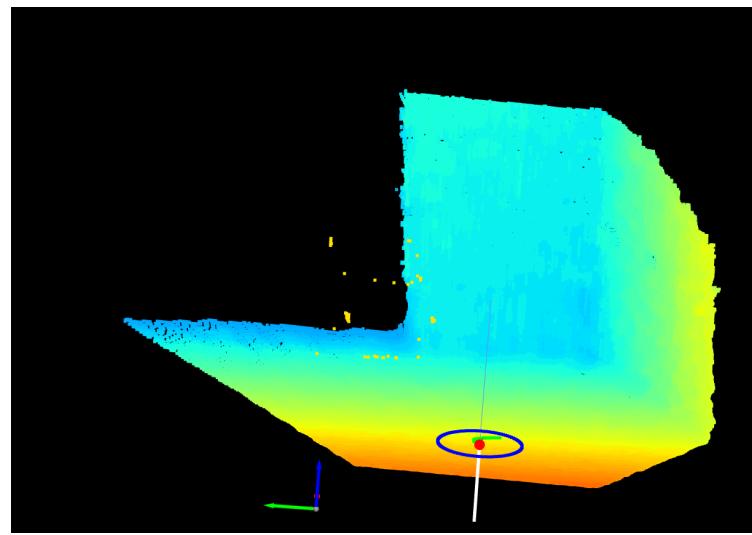


FIGURA 6.22: Visualización 3D de un vuelo en la cuarta configuración de obstáculos (1). Vista frontal.

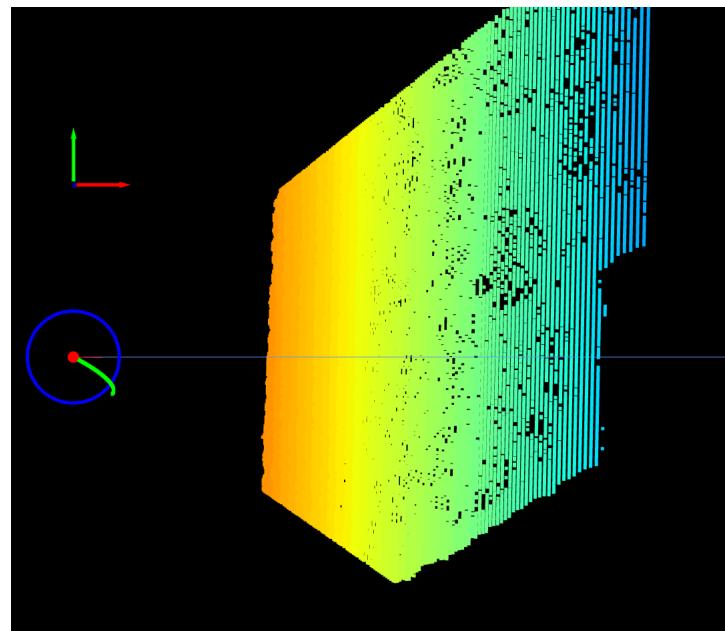


FIGURA 6.23: Visualización 3D de un vuelo en la cuarta configuración de obstáculos (2). Vista de arriba hacia abajo. Las trayectorias se dirigen hacia la región en dirección $-y_B$ sin información de profundidad, probablemente buscando el borde del obstáculo.

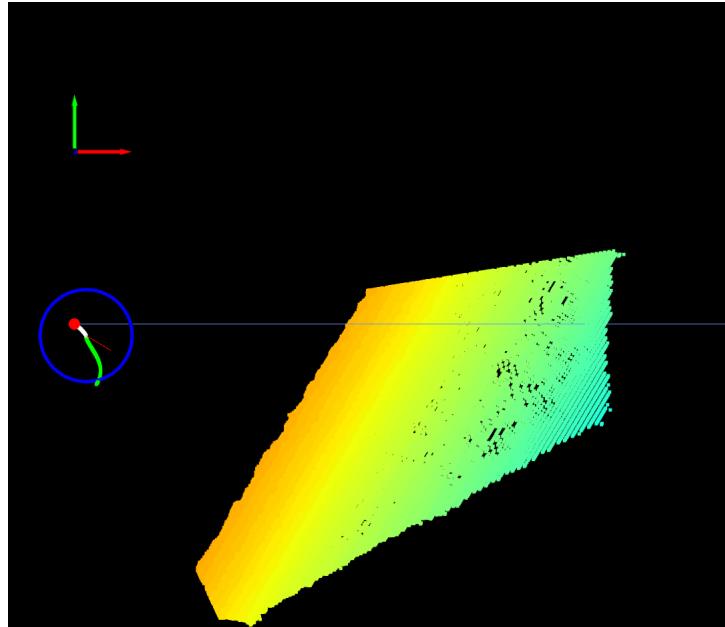


FIGURA 6.24: Visualización 3D de un vuelo en la cuarta configuración de obstáculos (3). Vista de arriba hacia abajo. Las trayectorias continúan dirigiéndose hacia la región en dirección $-y_B$ sin información de profundidad, probablemente buscando el borde del obstáculo.

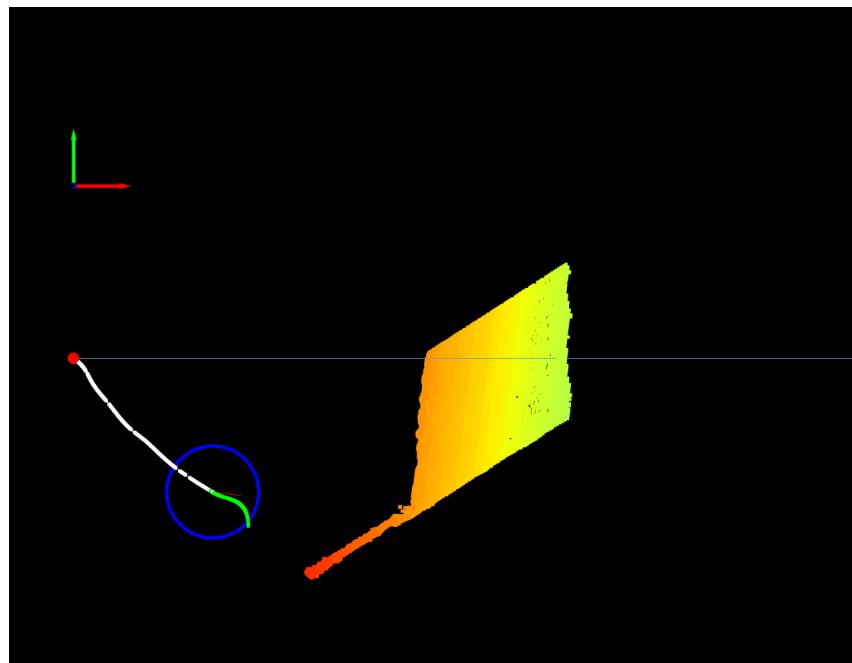


FIGURA 6.25: Visualización 3D de un vuelo en la cuarta configuración de obstáculos (4). Vista de arriba hacia abajo. Las trayectorias continúan dirigiéndose hacia la región en dirección $-y_B$ sin información de profundidad, probablemente buscando el borde del obstáculo.

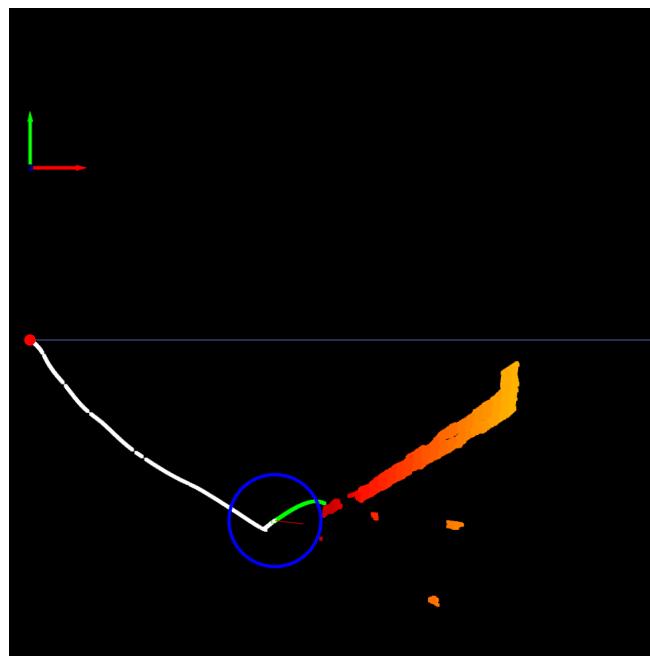


FIGURA 6.26: Visualización 3D de un vuelo en la cuarta configuración de obstáculos. Vista de arriba hacia abajo. Al no conseguir el borde del obstáculo a tiempo, se produce una colisión.

De estos resultados se confirma que los efectos consecuencia de las dificultades encontradas durante el refinamiento fino de la política estudiante son la preferencia certera a esquivar obstáculos hacia el flanco izquierdo (en dirección $-y_B$), y que estos efectos producen colisiones en configuraciones de obstáculos donde no es posible esquivar hacia ese flanco.

En la Tabla 6.3 se muestra un resumen de los resultados obtenidos en los vuelos en simulación. En esta tabla se muestra que la política es estable para las configuraciones de obstáculos donde es trivial esquivar por el flanco izquierdo. De igual forma, se muestra que la política es inestable y propensa a producir colisiones para las configuraciones de obstáculos donde es no trivial esquivar por el flanco izquierdo.

Configuración	N	N_e	P	\bar{D}	$\text{máx}(D)$	$\text{mín}(D)$
Obstáculo simple	10	10	100 %	3.1	3.4	2.7
Dos obstáculos consecutivos	10	10	100 %	3.7	4	3.3
Dos obstáculos paralelos	10	2	20 %	N/A	N/A	N/A
Obstáculo con pared izquierda	10	0	0 %	N/A	N/A	N/A

TABLA 6.3: Resumen de los resultados obtenidos en los vuelos en simulación. N y N_e son el número de vuelos totales y sin colisión respectivamente. **P** es el porcentaje de vuelos sin colisión. \bar{D} , $\text{máx}(D)$ y $\text{mín}(D)$ son la desviación máxima promedio, máxima y mínima respectivamente (en metros).

6.2.2. Vuelos sobre la plataforma física

En esta sección se describen las configuraciones y resultados de los vuelos realizados sobre la plataforma física de implementación (QUAV SOTEN¹) en un entorno de la vida real. Estos vuelos fueron ejecutados en un campo de vuelo para drones ubicado en la prefectura de Chiba, Japón; las condiciones climáticas del día fueron: soleado con un ligero viento en dirección hacia el oeste. Durante los vuelos, un piloto experto estuvo preparado para controlar manualmente el QUAV en caso de que una colisión fuera inminente. En esta sección se utiliza el término “colisión” para referirse al evento de interrumpir el vuelo autónomo para evitar una colisión inminente. Al igual que durante los vuelos en simulación, la ejecución de la evasión de obstáculos está limitada al plano xy del marco de referencia del cuerpo del QUAV.

¹En la Figura 5.1 se muestra al QUAV SOTEN.

La primera configuración de obstáculos utilizada fue una configuración simple con un solo obstáculo en la línea directa de visión del QUAV, la Figura 6.27 muestra una fotografía ilustrada que describe la configuración.

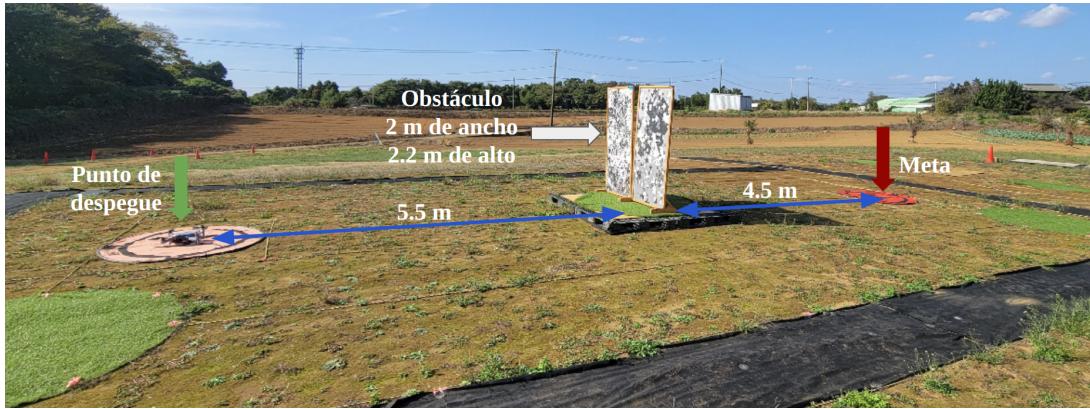


FIGURA 6.27: Configuración de obstáculos en entorno real: Obstáculo simple.

Se realizaron 11 vuelos en esta configuración, y los resultados fueron análogos a los obtenidos en simulación, en concreto, de 11 vuelos ejecutados, 10 resultaron sin colisión y uno fue abortado por el piloto debido a la influencia de una fuerte ráfaga de viento. Los caminos resultantes de los vuelos también fueron análogos a los obtenidos en simulación, sin embargo, debido a variaciones en el viento y en las condiciones de iluminación, algunas ejecuciones se desviaron considerablemente del camino directo. La Figura 6.28 muestra una visualización de arriba hacia abajo de 10 de los caminos ejecutados en esta configuración, incluyendo el vuelo que fue abortado por el piloto. Se puede observar que la desviación máxima de los caminos ejecutados con respecto al camino directo entre el punto de despegue y la meta varía bastante entre vuelos (probablemente debido a las variaciones en el viento), la Figura 6.29 muestra un diagrama de caja de la desviación máxima de los caminos ejecutados en esta configuración. La desviación máxima promedio de los caminos ejecutados en esta configuración es cerca de 3.2 metros, con una desviación máxima de cerca de 4.5 metros y una desviación mínima de cerca de 2 metros.

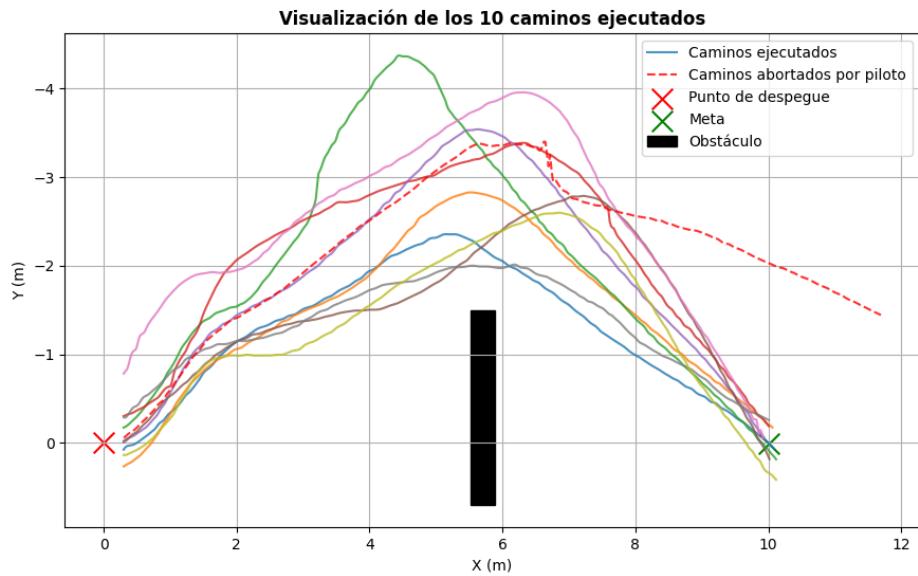


FIGURA 6.28: Visualización de 10 caminos ejecutados en la primera configuración de obstáculos en entorno real. Incluyendo el vuelo que fue abortado por el piloto.

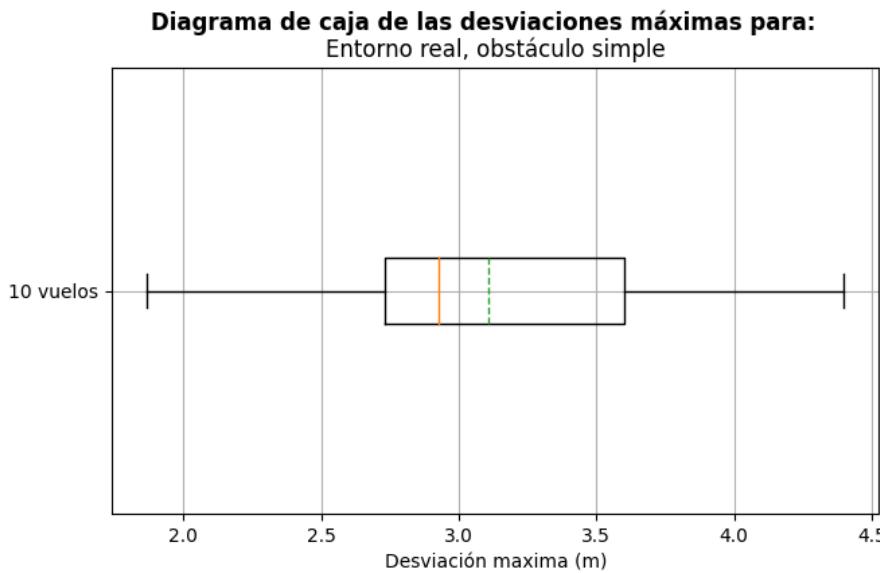


FIGURA 6.29: Diagrama de caja de la desviación máxima de los caminos ejecutados en la primera configuración de obstáculos en entorno real.

En todos los vuelos ejecutados en esta configuración, el algoritmo esquivó el obstáculo por el flanco izquierdo y continuo en dirección a la meta. La Figura 6.30, la Figura 6.31 y la Figura 6.32 muestran capturas de una grabación de uno de los vuelos sobre esta configuración. La Figura 6.33, la Figura 6.34 y la

Figura 6.35 también muestran capturas de una grabación de uno de los vuelos sobre esta configuración pero en una perspectiva diferente. En estas figuras se aprecia claramente el comportamiento del algoritmo, en particular: comenzar el vuelo, esquivar el obstáculo por el flanco izquierdo, continuar avanzando hasta que se termine la trayectoria actual y finalmente, como no se observan obstáculos, navegar en dirección a la meta.

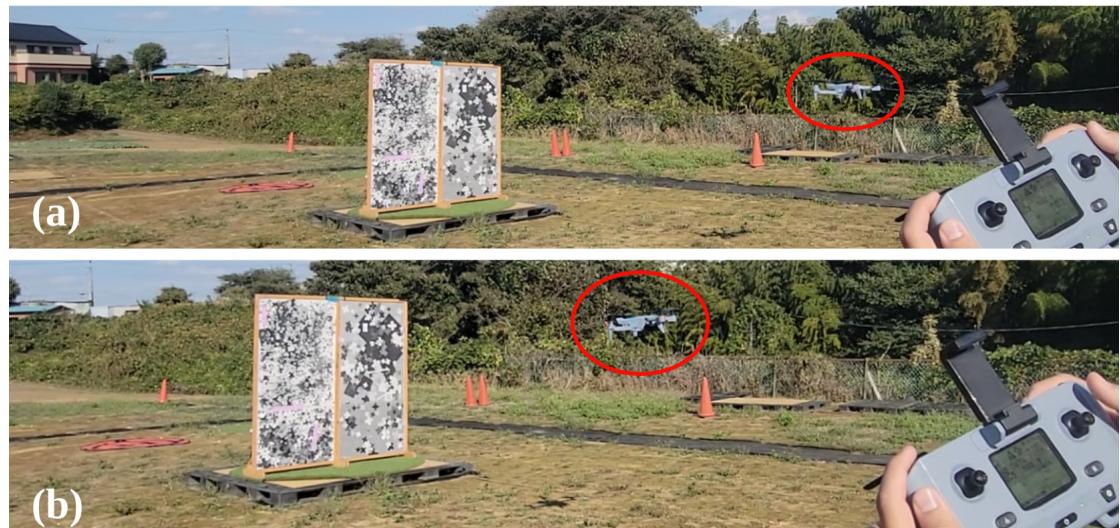


FIGURA 6.30: Capturas de una grabación de uno de los vuelos en entorno real con un obstáculo simple (1). **(a)** Comienza el vuelo. **(b)** Se comienza a esquivar el obstáculo.

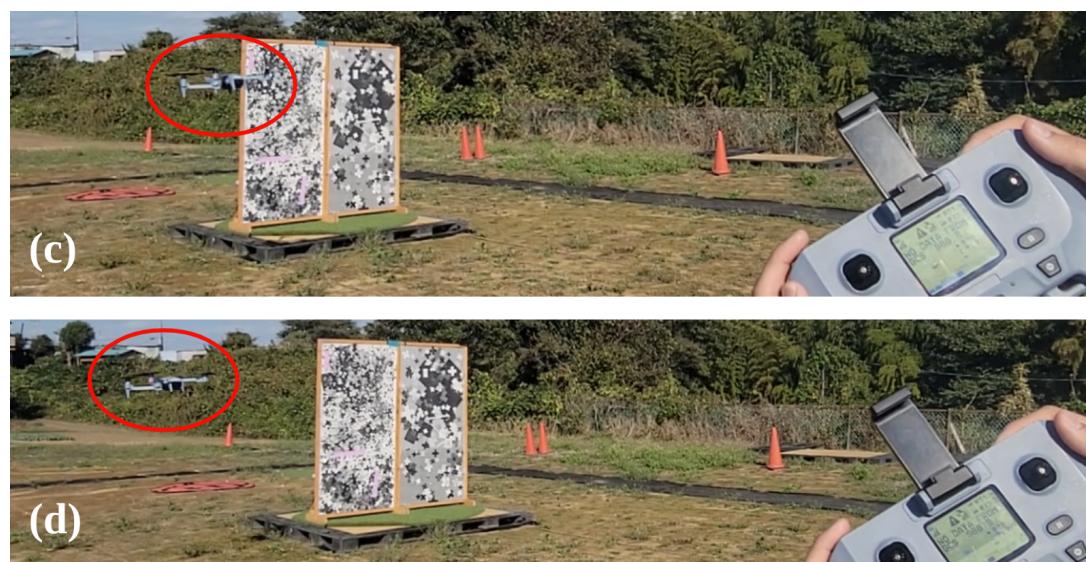


FIGURA 6.31: Capturas de una grabación de uno de los vuelos en entorno real con un obstáculo simple (2). En **(c)** y **(d)**, se esquia el obstáculo por el flanco izquierdo.

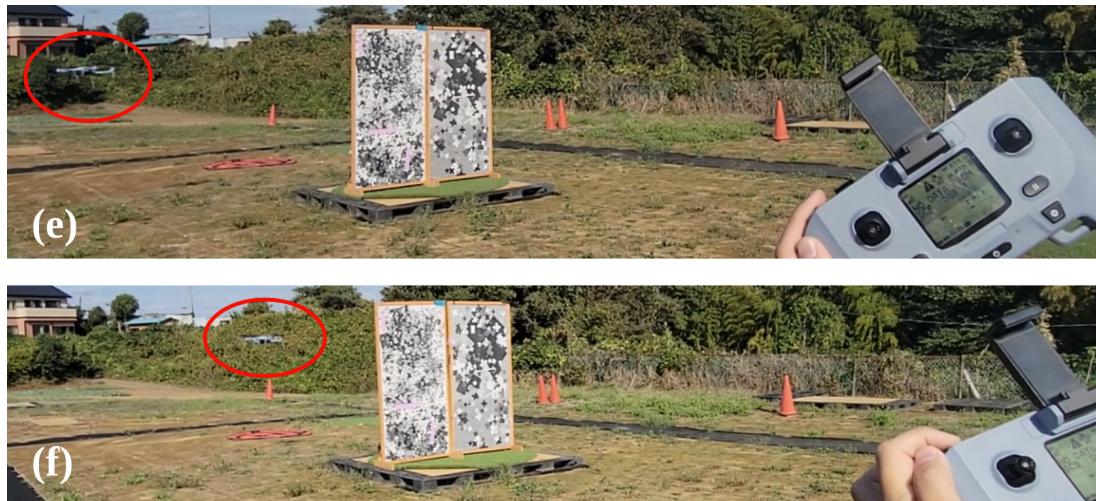


FIGURA 6.32: Capturas de una grabación de uno de los vuelos en entorno real con un obstáculo simple (3). En (e) y (f), como no se observan obstáculos, se navega en dirección a la meta.

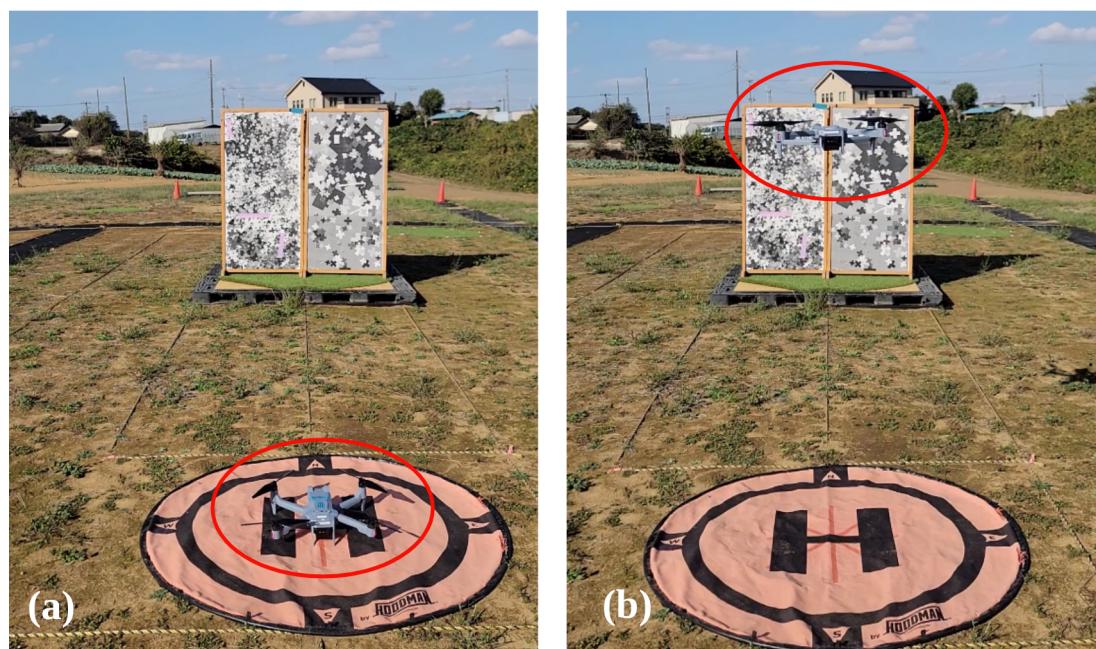


FIGURA 6.33: Capturas de una grabación de uno de los vuelos en entorno real con un obstáculo simple. Vista frontal (1). (a) Antes del despegue. (b) Comienza el vuelo.

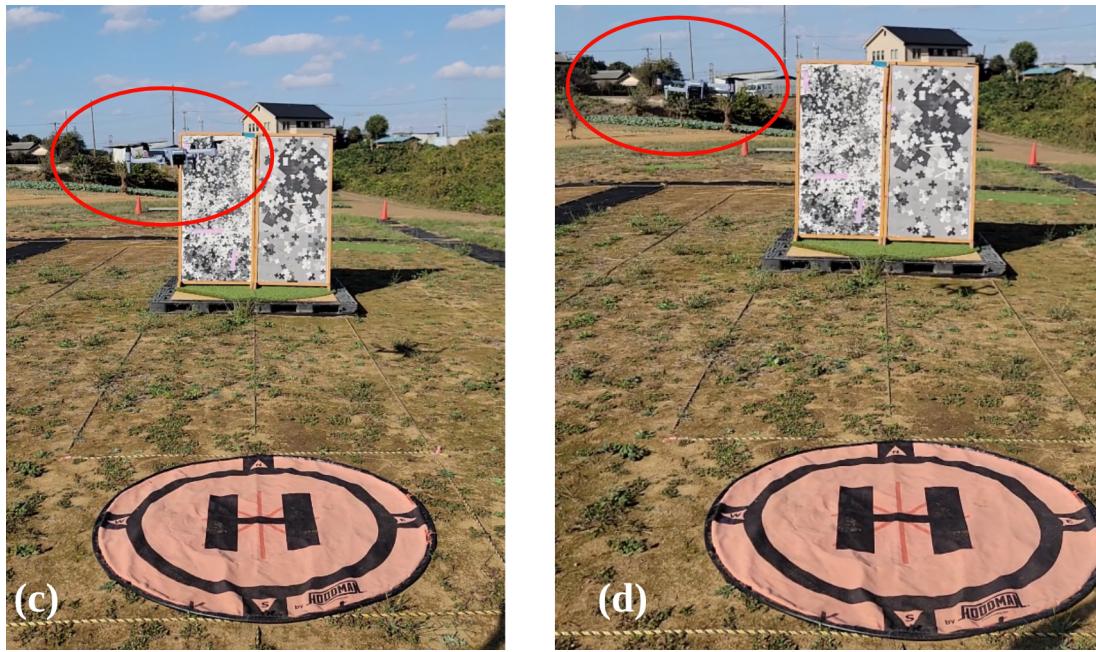


FIGURA 6.34: Capturas de una grabación de uno de los vuelos en entorno real con un obstáculo simple. Vista frontal (2). En (c) y (d), se esquiva el obstáculo por el flanco izquierdo.

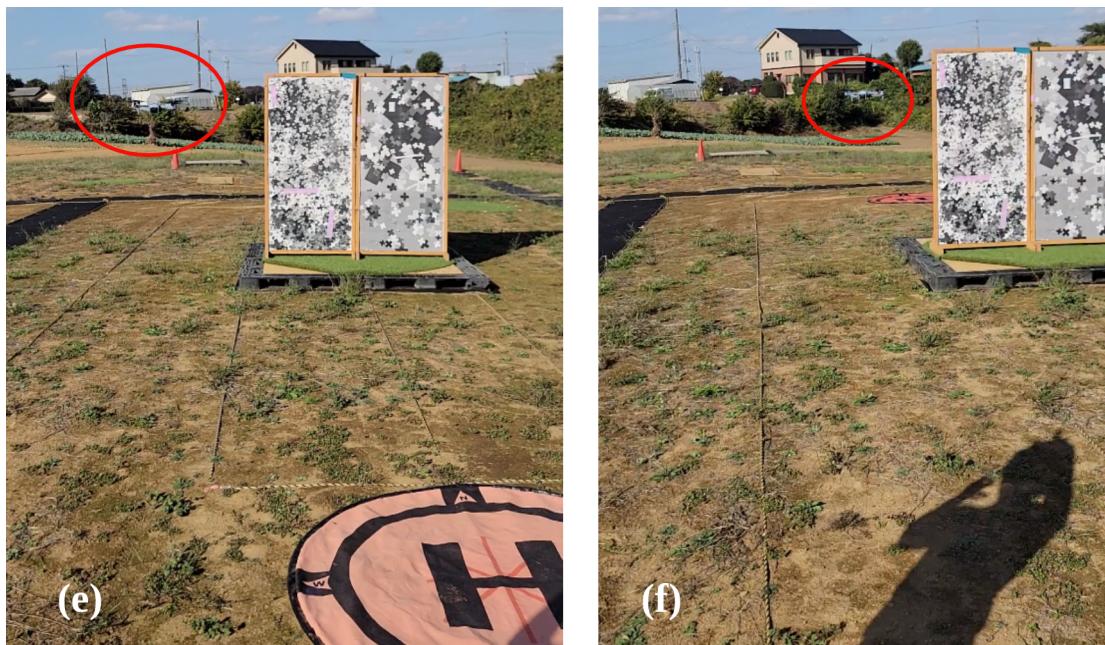


FIGURA 6.35: Capturas de una grabación de uno de los vuelos en entorno real con un obstáculo simple. Vista frontal (3). En (e) y (f), como no se observan obstáculos, se navega en dirección a la meta.

La segunda configuración de obstáculos utilizada fue una configuración con dos obstáculos paralelos con una separación entre si, esta separación fue distinta para

cada vuelo. A diferencia de la configuración anterior, en esta configuración solo se hicieron 3 vuelos. El objetivo de estos vuelos era observar el comportamiento del algoritmo en este escenario, en particular, confirmar la presencia de la tendencia certera de esquivar obstáculos por el flanco izquierdo que se observó durante vuelos análogos en entornos de simulación; así como también observar si esta tendencia produce colisiones inminentes.

En el primer vuelo en esta configuración se utilizó una separación de 3 metros, la Figura 6.36 muestra una fotografía que ilustra la configuración utilizada. Este vuelo fue abortado por colisión inminente, la Figura 6.37 muestra capturas de la grabación de este vuelo en donde se observa el comportamiento del algoritmo. Primero (Figura 6.37(a)), la ejecución comienza; segundo (Figura 6.37(b)), el QUAV se dirige hacia el espacio entre ambos obstáculos en lugar de esquivar los obstáculos por el flanco derecho; y tercero (Figura 6.37(c)), cuando el obstáculo derecho sale del campo de visión, el algoritmo intenta esquivar el obstáculo que aun es visible por el flanco izquierdo, lo cual produce una colisión inminente, haciendo que el piloto tenga que abortar el vuelo. Estos resultados coinciden con el comportamiento observado en simulación¹, y confirman la presencia de la tendencia certera a esquivar obstáculos por el flanco izquierdo.



FIGURA 6.36: Configuración de obstáculos en entorno real: Obstáculos paralelos a 3 metros de separación.

¹Vea la Figura 6.20



FIGURA 6.37: Capturas de la grabación del vuelo en entorno real con obstáculos paralelos a 3 metros de separación. (a) La ejecución comienza. (b) El QUAV se dirige hacia el espacio entre ambos obstáculos en lugar de esquivar los obstáculos por el flanco derecho. (c) Cuando el obstáculo derecho sale del campo de visión, el algoritmo intenta esquivar el obstáculo que aun es visible por el flanco izquierdo, lo cual produce una colisión inminente.

El segundo vuelo en la configuración de obstáculos paralelos utilizó una separación de 4 metros, la Figura 6.38 muestra una fotografía que ilustra la configuración utilizada. Este vuelo resultó exitoso, la Figura 6.39 muestra capturas de la grabación de este vuelo en donde se observa el comportamiento del algoritmo. Primero (Figura 6.39(a)), la ejecución comienza y el QUAV navega hacia el espacio entre los dos obstáculos; segundo (Figura 6.39(b)), esta vez, el obstáculo izquierdo sale del campo de visión, y el QUAV continúa hacia el espacio entre los dos obstáculos; y tercero (Figura 6.39(c)), se superan los obstáculos y se navega en dirección a la meta. De este vuelo confirmamos que si la configuración de obstáculos paralelos es tal que el primer obstáculo que desaparece el campo de visión es el obstáculo izquierdo, entonces es posible que el algoritmo sea capaz de completar el vuelo sin producir colisiones. La Figura 6.40 muestra una visualización de arriba hacia abajo del camino ejecutado por el QUAV en este vuelo.

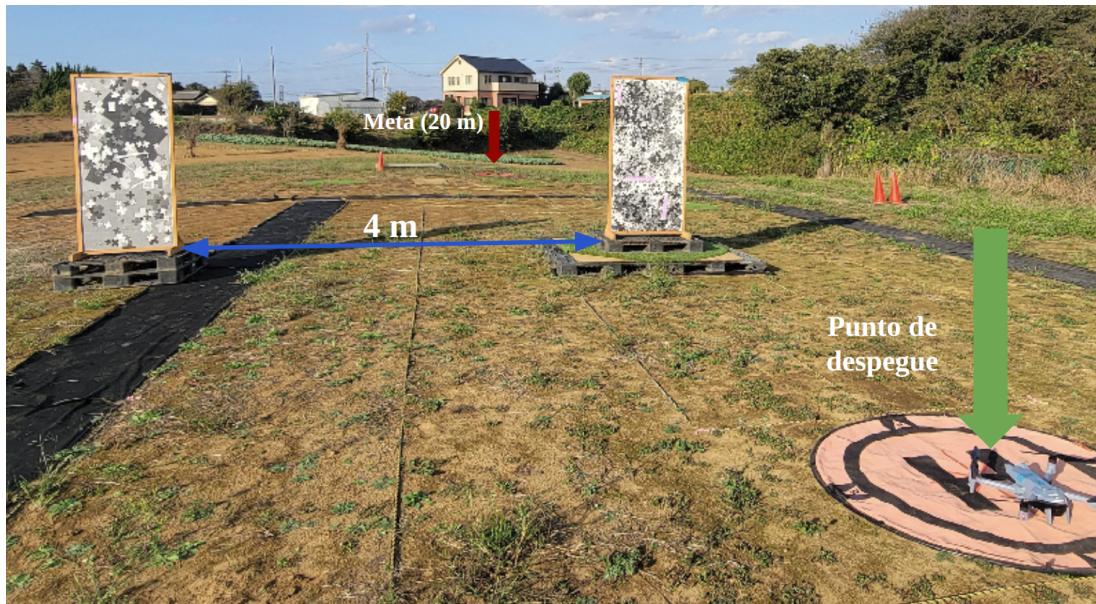


FIGURA 6.38: Configuración de obstáculos en entorno real: Obstáculos paralelos a 4 metros de separación.



FIGURA 6.39: Capturas de la grabación del vuelo en entorno real con obstáculos paralelos a 4 metros de separación. (a) La ejecución comienza y el QUAV navega hacia el espacio entre los dos obstáculos. (b) El obstáculo izquierdo sale del campo de visión, y el QUAV continúa hacia el espacio entre los dos obstáculos.

(c) Se superan los obstáculos y se navega en dirección a la meta.

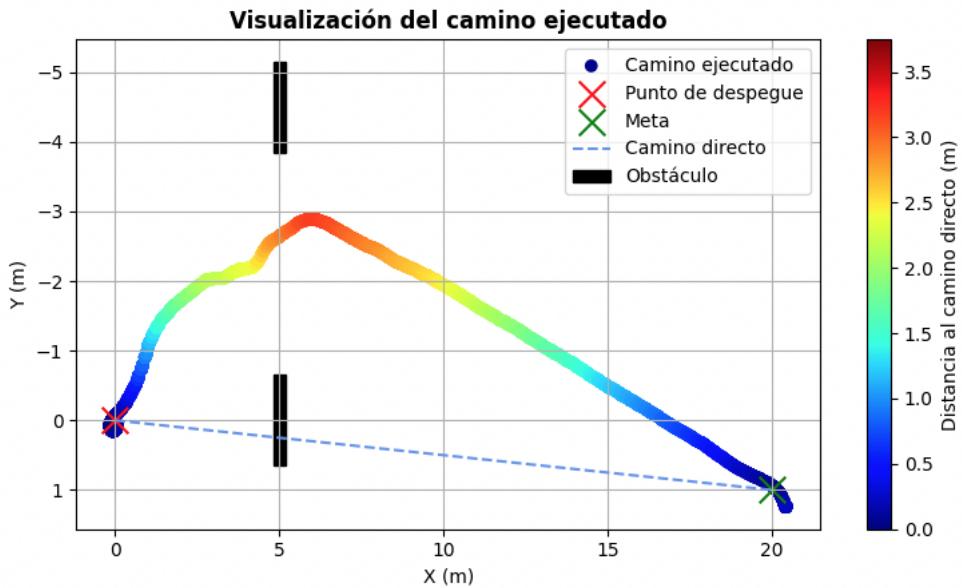


FIGURA 6.40: Visualización del camino ejecutado por el QUAV para la configuración de obstáculos paralelos a 4 metros de separación en entorno real.

El tercer y último vuelo en la configuración de obstáculos paralelos utilizó una separación de un metro, la Figura 6.41 muestra una fotografía que ilustra la configuración utilizada. Este vuelo resultó exitoso y la razón por la cual fue exitoso provee información adicional sobre la capacidad de evasión de obstáculos del algoritmo. La Figura 6.42 muestra capturas de la grabación de este vuelo en donde se observa el comportamiento del algoritmo. Primero (Figura 6.42(a)), la ejecución comienza y el QUAV navega hacia el espacio entre los dos obstáculos; segundo (Figura 6.42(b)), en el momento que el obstáculo izquierdo se vuelve el obstáculo mas grande en el campo de visión, el algoritmo intenta fuertemente corregir la trayectoria para esquivar el conjunto de obstáculos por el flanco izquierdo del obstáculo izquierdo, esto ocurre tempranamente en el vuelo pues la separación entre los obstáculos es relativamente pequeña; y tercero (Figura 6.42(c)), como la decisión de corregir la trayectoria hacia la izquierda ocurre de forma temprana, el algoritmo tiene tiempo de completar la trayectoria sin producir colisiones, resultando en una ejecución exitosa. La Figura 6.43 muestra una visualización de arriba hacia abajo del camino ejecutado por el QUAV en esta configuración, en donde se evidencia el momento en el que el algoritmo intenta corregir fuertemente; adicionalmente, para respladar la magnitud de la corrección en la Figura 6.44 se muestra el grafico del encabezamiento del QUAV en función del tiempo, en donde se observan oscilaciones de relativamente alta magnitud. El resultado de este vuelo

expone que si existe el espacio y el tiempo para esquivar un conjunto de obstáculos por el flanco mas hacia la izquierda, es posible que la política de evasión de obstáculos sea capaz de completar el vuelo exitosamente.

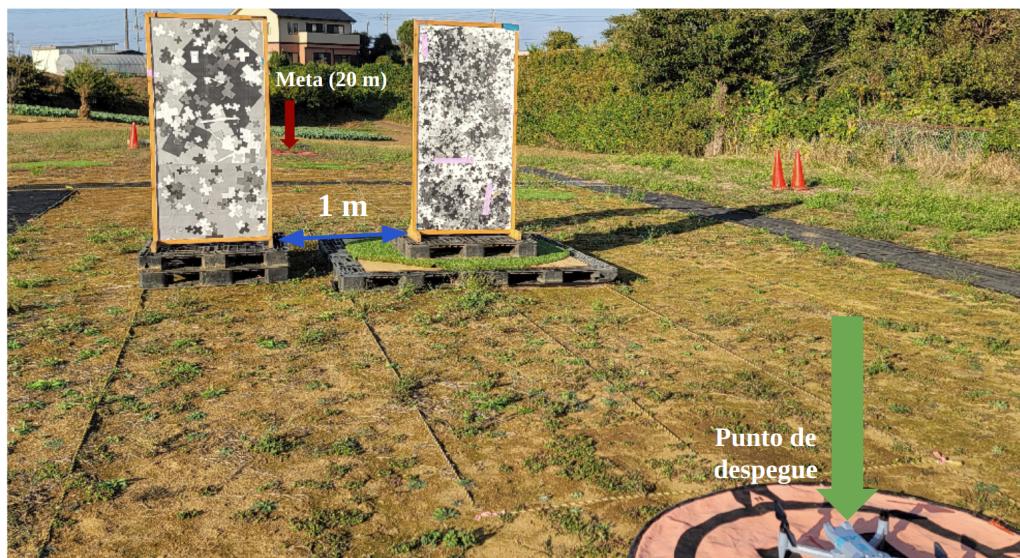


FIGURA 6.41: Configuración de obstáculos en entorno real: Obstáculos paralelos a 1 metro de separación.



FIGURA 6.42: Capturas de la grabación del vuelo en entorno real con obstáculos paralelos a 1 metro de separación. (a) La ejecución comienza y el QUAV navega hacia el espacio entre los dos obstáculos. (b) El obstáculo izquierdo se vuelve el obstáculo mas grande en el campo de visión y el algoritmo intenta fuertemente corregir la trayectoria para esquivar el conjunto de obstáculos por el flanco izquierdo del obstáculo izquierdo. (c) Como la decisión de corregir la trayectoria hacia la izquierda ocurre de forma temprana, el algoritmo tiene tiempo de completar la trayectoria sin producir colisiones, resultando en una ejecución exitosa.

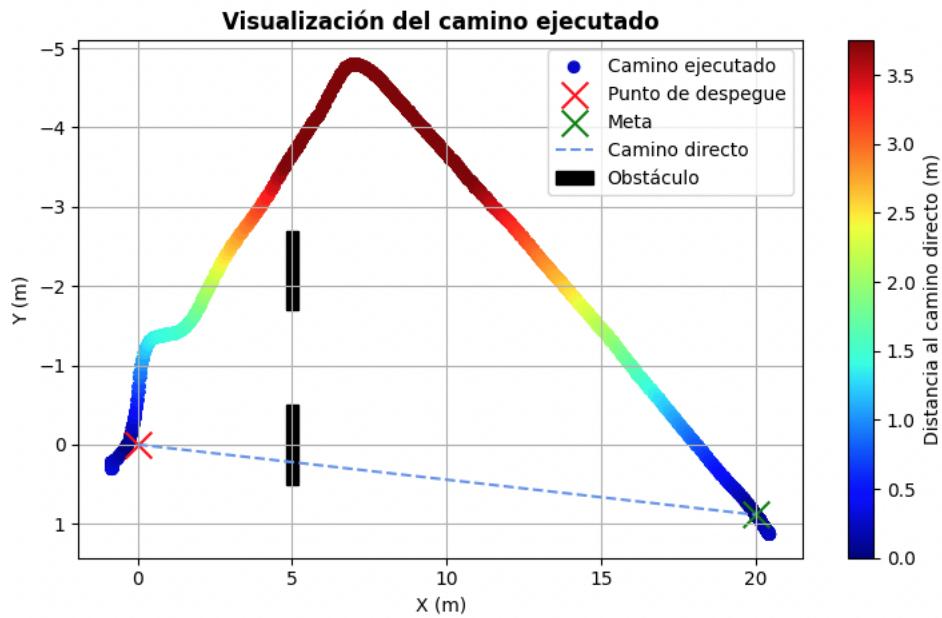


FIGURA 6.43: Visualización del camino ejecutado por el QUAV para la configuración de obstáculos paralelos a 1 metro de separación en entorno real. Se puede observar el momento en el que el algoritmo intenta corregir fuertemente la trayectoria hacia la izquierda.

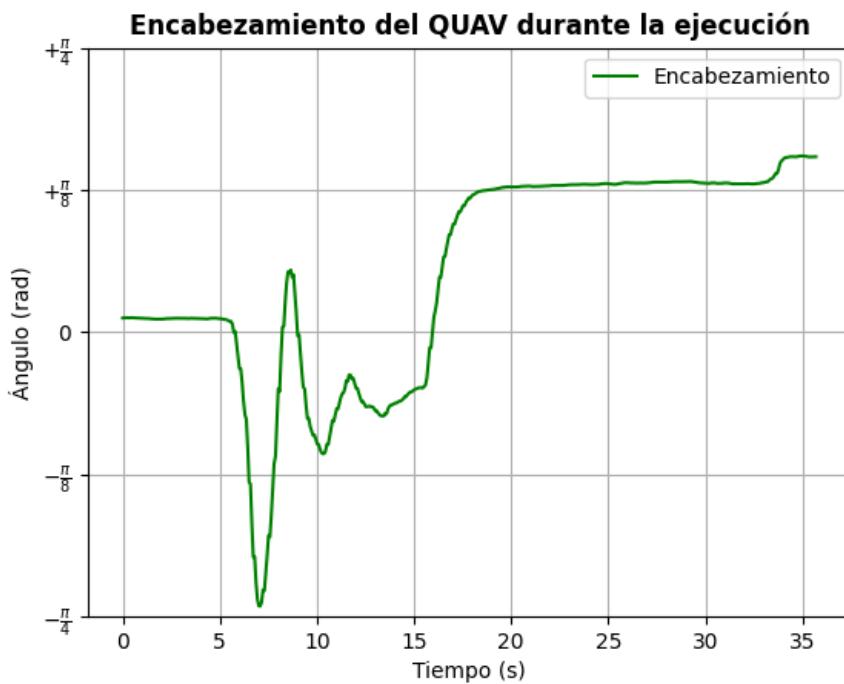


FIGURA 6.44: Grafico del encabezamiento del QUAV en función del tiempo para la configuración de obstáculos paralelos a 1 metro de separación en entorno real. Se observan oscilaciones de relativamente alta magnitud que corresponden al momento en el que el algoritmo intenta corregir fuertemente la trayectoria hacia la izquierda.

Los vuelos realizados permiten evaluar la estabilidad del algoritmo de evasión de obstáculos en entornos de la vida real, resaltando su consistencia, especialmente en la configuración de un obstáculo simple. No obstante, los resultados también revelan los efectos derivados de las dificultades encontradas durante el refinamiento fino de la política estudiante, manifestándose principalmente en la preferencia absoluta por esquivar obstáculos hacia el flanco izquierdo. Como consecuencia directa de estos vuelos, se exploran escenarios que ilustran el impacto de esta preferencia en situaciones de la vida real que involucran configuraciones de dos obstáculos paralelos. Dicho esto, finalmente, en la Tabla 6.4 se muestra un resumen de los resultados obtenidos en los vuelos en entornos de la vida real.

Configuración	N	N_e	P	\bar{D}	$\max(D)$	$\min(D)$
Obstáculo simple	11	10	90 %	3.2	4.5	2
Dos obstáculos paralelos	3	2	66 %	4	4.9	3.1

TABLA 6.4: Resumen de los resultados obtenidos en los vuelos en entornos de la vida real. N y N_e son el número de vuelos totales y sin colisión inminente respectivamente. **P** es el porcentaje de vuelos sin colisión inminente. \bar{D} , $\max(D)$ y $\min(D)$ son la desviación máxima promedio, máxima y mínima respectivamente (en metros).

6.3. Resumen

Este capítulo abordó la evaluación y presentación de los resultados obtenidos, proporcionando una perspectiva crítica sobre el rendimiento de la solución propuesta. En la Sección 6.1, se exploraron los resultados del ajuste fino de la política estudiante, destacando los desafíos asociados al sobre-ajuste de los datos de entrenamiento y a la generación de la base de datos. Además, se aclaró que, debido a limitaciones de tiempo de desarrollo y recomendaciones del equipo de ACSL, no se abordó completamente el problema del sobre-ajuste, lo que produjo efectos evidentes al evaluar la política de evasión de obstáculos. En particular, se observó cómo esta decisión ha llevado a una fuerte preferencia de la política por esquivar obstáculos por el flanco izquierdo.

La Sección 6.2 detalló los resultados de la política de evasión de obstáculos, presentando los vuelos en simulación en la Sección 6.2.1 y los vuelos sobre la plataforma física de implementación, SOTEN, en la Sección 6.2.2. Estos resultados

ofrecieron una perspectiva del rendimiento de la solución en configuraciones simples de obstáculos, donde la política de evasión se mostró estable. También se exploraron configuraciones donde la fuerte preferencia de la política por esquivar obstáculos por el flanco izquierdo produjo colisiones, y se observaron distintos escenarios y consecuencias de ello.

En resumen, este capítulo proporcionó una evaluación que sirve de base para la discusión del siguiente capítulo, el de conclusiones. El contexto introducido en este capítulo también permite considerar direcciones para trabajos futuros que se construyan sobre la implementación presentada en este trabajo.

Capítulo 7

Conclusiones

En el capítulo anterior, se abordan los desafíos del ajuste fino de la política estudiante y se proporciona en detalle los resultados de la política de evasión de obstáculos, tanto en entornos de simulación como en entornos de la vida real. En el presente capítulo, se confirma el logro de los objetivos propuestos para este trabajo, se destacan lecciones aprendidas y se abordan las limitaciones identificadas. Así mismo, se presentan sugerencias para futuras investigaciones que busquen aprovechar y mejorar la implementación de algoritmos para la evasión autónoma de obstáculos para QUAVs.

Con respecto al cumplimiento de los objetivos propuestos para este trabajo, se puede observar que:

- Se llevó a cabo una revisión del estado del arte de los algoritmos de evitación de obstáculos para drones autónomos, como se detalla en el Capítulo 4. La investigación se organizó clasificando los estudios relevantes según su dependencia y uso de información global, generando así una visión general del estado actual del campo, para poder evaluar e implementar un algoritmo adecuado a las necesidades de ACSL. Se profundizó específicamente en el trabajo titulado *Learning high-speed flight in the wild* [7] debido a sus características atractivas para el caso de implementación por parte de ACSL.
- Se implementó un algoritmo de evitación de obstáculos para drones autónomos en un entorno de simulación. El algoritmo seleccionado se basó en la

metodología propuesta por Loquercio et al. (2021) [7], y los detalles de su implementación se presentan en el Capítulo 5. La evaluación de la implementación se llevó a cabo en el entorno de simulación AirSim [33], y los resultados de los vuelos realizados funcionan como base sólida para el desarrollo de nuevos proyectos e investigaciones, ya que se obtuvieron resultados positivos donde la política de evasión se mostró estable con configuraciones simples, además, se hicieron pruebas con distintos escenarios para evaluar las consecuencias que conllevan las variaciones, dichos resultados se describen de forma detallada en la Sección 6.2.1.

- Se trasladó la implementación realizada al hardware de un dron autónomo realizando las optimizaciones necesarias, y se realizaron de pruebas de campo de dicha implementación. Se empleó el dron SOTEN, un QUAV ligero diseñado y producido por ACSL para aplicaciones de vigilancia e inspección. Con el objetivo de reducir la carga computacional de la implementación, se introdujo un mecanismo que desactiva la inferencia de trayectorias cuando no hay obstáculos en el campo de visión del vehículo, según se detalla en el Algoritmo 3 y en el Algoritmo 4. Además, para posibilitar la ejecución en el hardware físico de SOTEN, se tradujo la representación del modelo de inferencia al estándar abierto para la interoperabilidad del aprendizaje de máquinas (ONNX). Finalmente, se llevaron a cabo pruebas en un entorno de la vida real que mostraron resultados positivos para configuraciones de obstáculos simples; teniendo en cuenta las limitaciones de tiempo y las restricciones de la implementación, tal como se describe en la Sección 6.2.2.

La consecución de cada uno de estos objetivos específicos valida y respalda el logro del objetivo general: implementar un algoritmo de evitación de obstáculos para drones autónomos y realizar pruebas tanto en un entorno de simulación como en el campo. Por lo tanto, se concluye que el presente trabajo cumplió satisfactoriamente con todos los objetivos propuestos.

Por otro lado, en relación a los resultados observados del algoritmo implementado, se destaca principalmente la marcada preferencia del algoritmo por esquivar obstáculos por el flanco izquierdo. Esta preferencia se atribuye a las dificultades encontradas durante el proceso de refinamiento fino de la política estudiante, específicamente al problema de sobre-ajuste generado por el desbalance en la generación de trayectorias con una velocidad de ejecución de 1 m/s para la base

de datos de refinamiento fino. Debido a limitaciones de tiempo en el desarrollo del trabajo, y de acuerdo con las recomendaciones del personal de ACSL, la evaluación del comportamiento del algoritmo se llevó a cabo bajo la presencia de la preferencia descrita anteriormente.

En general, los principales comportamientos defectuosos observados en los resultados de la evaluación de la solución fueron causados por las consecuencias del sobre-ajuste del modelo durante el refinamiento fino de la política estudiante. Se concluyó que la generación adecuada de la base de datos de trayectorias para el refinamiento fino es crucial para que el rendimiento y la capacidad de generalización del algoritmo sean aceptables.

Se logró implementar una solución que permite a un QUAV evadir obstáculos de forma independiente sin proporcionar anteriormente información sobre su entorno, y que además se puede ejecutar a tiempo real sobre un hardware ligero y con recursos limitados. Sin mencionar que la totalidad de la solución trabaja en concordancia y sobre el marco de trabajo de ACSL, permitiendo una integración automática con el ecosistema de aplicaciones de los vehículos de ACSL.

Es importante resaltar los beneficios obtenidos para ACSL dentro del marco de la presente investigación; destaca lo complejo que resulta ejecutar aplicaciones con redes neuronales en un sistema embebido, como es el hardware de dron ligero. Se logró ejecutar satisfactoriamente la navegación del dron, apesar de lo sensible que son los algoritmos que utilizan redes neuronales a la calidad de los datos de entrenamiento, habiendo verificado que el método original no genera conjuntos de datos buenos para velocidades relativamente lentas.

La implementación del algoritmo en SOTEN es de importancia vital para ACSL como base para trabajos futuros. Se demostró que realizando un refinamiento fino adecuado de la política estudiante, se puede tener ya en funcionamiento un algoritmo de evasión de obstáculos que revolucione la autonomía y seguridad de los drones, manteniendo así a ACSL como uno de los líderes mundiales en la innovación y desarrollo de la tecnología de drones.

En este sentido, y para finalizar, se recomienda para futuras investigaciones, asegurarse de que el entorno de simulación utilizado para generar las trayectorias

sea adecuado a la rapidez promedio de ejecución v_{des} . Es importante que las dimensiones de los obstáculos permitan que en una observación se puedan considerar distintas alternativas para esquivar un mismo obstáculo. Si las alternativas consideradas tienden a alinearse hacia un comportamiento en particular, la política estudiante tendrá una fuerte tendencia a utilizar ese comportamiento incluso en casos cuando resulte riesgoso, tal como sucedió en este trabajo con la preferencia por esquivar hacia el flanco izquierdo. Este comportamiento del dron podría mejorarse en futuras investigaciones si se modifica la configuración de obstáculos, si se incrementa la longitud de las trayectorias generadas o si se utiliza un algoritmo de planificación global diferente, adaptado a velocidades v_{des} menores a 3 m/s. En síntesis, lo importante es generar una base de datos de refinamiento fino con nivel de generalización similar al utilizado para el entrenamiento del modelo original.

Se recomienda mejorar la instrumentación del dron para poder tener acceso a las medidas de aceleración y jerk, y de esta manera planificar de mejor forma las trayectorias globales en todos los rangos de velocidades, desde velocidades bajas hasta velocidades altas.

Bibliografía

- [1] Tu, Guan Ting y Jih Gau Juang: *UAV Path Planning and Obstacle Avoidance Based on Reinforcement Learning in 3D Environments*. Actuators, 12, Febrero 2023, ISSN 20760825.
- [2] Xue, Zhihan y Tad Gonsalves: *Vision Based Drone Obstacle Avoidance by Deep Reinforcement Learning*. AI (Switzerland), 2:366–380, Septiembre 2021, ISSN 26732688.
- [3] Zhang, Ji, Vivek Velivela, Sanjiv Singh y Gupta Chadha: *P-CAL: Pre-computed Alternative Lanes for Aggressive Aerial Collision Avoidance*. 2019. <https://www.researchgate.net/publication/335401931>.
- [4] Yang, Xin, Jingyu Chen, Yuanjie Dang, Hongcheng Luo, Yuesheng Tang, Chunyuan Liao, Peng Chen y Kwang Ting Cheng: *Fast Depth Prediction and Obstacle Avoidance on a Monocular Drone Using Probabilistic Convolutional Neural Network*. IEEE Transactions on Intelligent Transportation Systems, 22:156–167, Enero 2021, ISSN 15580016.
- [5] Geiger, Andreas, Philip Lenz, Christoph Stiller y Raquel Urtasun: *Vision meets Robotics: The KITTI Dataset*. The International Journal of Robotics Research, 32:1231–1237, 2013. <http://www.cvlibs.net/datasets/kitti>.
- [6] Sturm, Jrgen, Nikolas Engelhard, Felix Endres, Wolfram Burgard y Daniel Cremers: *A benchmark for the evaluation of RGB-D SLAM systems*. páginas 573–580, 2012, ISBN 9781467317375.
- [7] Loquercio, Antonio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun y Davide Scaramuzza: *Learning high-speed flight in the wild*. Science Robotics, 6, Octubre 2021, ISSN 24709476.

- [8] Hirschmüller, Heiko: *Stereo Processing by Semi-Global Matching and Mutual Information*. IEEE Transactions on pattern analysis and machine intelligence, 30:328–341, 2007. www.middlebury.edu/stereo.
- [9] Howard, Andrew, Mark Sandler, Grace Chu, Liang Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le y Hartwig Adam: *Searching for MobileNetV3*. páginas 1314–1324, Mayo 2019. <http://arxiv.org/abs/1905.02244>.
- [10] Yang, Hyunsoo, Yongseok Lee, Sang Yun Jeon y Dongjun Lee: *Multi-rotor drone tutorial: systems, mechanics, control and state estimation*. Intelligent Service Robotics, 10:79–93, 2017.
- [11] Shirsat, A: *Modeling and Control of a Quadrotor Aircraft UAV*. Ph. D. Thesis, 2015.
- [12] Diebel, James y cols.: *Representing attitude: Euler angles, unit quaternions, and rotation vectors*. Matrix, 58(15-16):1–35, 2006.
- [13] Drake, Samuel Picton: *Converting GPS coordinates [phi, lambda, h] to navigation coordinates (ENU)*. 2002.
- [14] Cai, Guowei, Ben M Chen, Tong Heng Lee, Guowei Cai, Ben M Chen y Tong Heng Lee: *Coordinate systems and transformations*. Unmanned rotorcraft systems, páginas 23–34, 2011.
- [15] Vis, MJP: *History of the Mercator projection*. B.S. thesis, 2018.
- [16] D, Poggio T Marr: *A computational theory of human stereo vision*. Proc. R. Soc. Lond. B, 204:301–328, 1979.
- [17] Alberto, Sergio y Rodriguez Florez: *Contributions by Vision Systems to Multi-sensor Object Localization and Tracking for Intelligent Vehicles*. 2010. <https://www.researchgate.net/publication/265126161>.
- [18] Baudes, A, Coll B, Morel, J.M. y Rouge B: *Procedimiento de establecimiento de correspondencia entre una primera imagen digital y una segunda imagen digital de una misma escena para la obtencion de disparidades*, 2009.
- [19] Gurney, Kevin: *An Introduction to Neural Networks*. CRC Press, 1997.
- [20] Huang, Kaizhu, Amir Hussain, Qiu Feng Wang y Rui Zhang: *Deep learning: fundamentals, theory and applications*, volumen 2. Springer, 2019.

- [21] Lecun, Yann, Koray Kavukcuoglu y Clément Farabet: *Convolutional Networks and Applications in Vision*. páginas 253–256, 2010. <http://www.cs.nyu.edu/>.
- [22] Gupta, Gaurav y Ruchika Chandel: *Image Filtering Algorithms and Techniques: A Review*, 2013. <https://www.researchgate.net/publication/325681876>.
- [23] Coady, James, Andrew O’Riordan, Gerard Dooley, Thomas Newe y Daniel Toal: *An Overview of Popular Digital Image Processing Filtering Operations*. En *2019 13th International Conference on Sensing Technology (ICST)*, páginas 1–5, Dec 2019.
- [24] Venkataraman, Aditya y Kishore Kumar Jagadeesha: *Evaluation of inter-process communication mechanisms*. Architecture, 86:64, 2015.
- [25] Sústrik, Martin y cols.: *ZeroMQ*. Introduction Amy Brown and Greg Wilson, página 16, 2015.
- [26] Shi, Ziji y Wee Keong Ng: *A collision-free path planning algorithm for unmanned aerial vehicle delivery*. En *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, páginas 358–362. IEEE, 2018.
- [27] Park, Je Kwan y Tai Myoung Chung: *Boundary-RRT* algorithm for drone collision avoidance and interleaved path re-planning*. Journal of Information Processing Systems, 16(6):1324–1342, 2020.
- [28] Lifen, Liu, Shi Ruoxin, Li Shuandao y Wu Jiang: *Path planning for UAVS based on improved artificial potential field method through changing the repulsive potential function*. En *2016 IEEE Chinese Guidance, Navigation and Control Conference (CGNCC)*, páginas 2011–2015. IEEE, 2016.
- [29] Bermudez, Giovanni, Luis Alejandro Rojas Castellar, Holman Montiel, Marco Ceballos y cols.: *Aplicación del método de campos de potencial artificial para un robot móvil autónomo*. Tecnura, 7(14):86–96, 2004.
- [30] Hastings, W Keith: *Monte Carlo sampling methods using Markov chains and their applications*. 1970.
- [31] Mellinger, Daniel y Vijay Kumar: *Minimum snap trajectory generation and control for quadrotors*. En *2011 IEEE international conference on robotics and automation*, páginas 2520–2525. IEEE, 2011.

- [32] Liu, Sikang, Kartik Mohta, Nikolay Atanasov y Vijay Kumar: *Search-based motion planning for aggressive flight in se (3)*. IEEE Robotics and Automation Letters, 3(3):2439–2446, 2018.
- [33] Shah, Shital, Debadeepa Dey, Chris Lovett y Ashish Kapoor: *Airsim: High-fidelity visual and physical simulation for autonomous vehicles*. En *Field and Service Robotics: Results of the 11th International Conference*, páginas 621–635. Springer, 2018.
- [34] Bradski, Gary: *The openCV library*. Dr. Dobb's Journal: Software Tools for the Professional Programmer, 25(11):120–123, 2000.
- [35] onnx: *Open standard for machine learning interoperability*. <https://github.com/onnx/onnx>.