UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

Facultad de Ingeniería, Arquitectura y Diseño

Programa de Maestría y Doctorado en Ciencias e Ingeniería

EVALUACIÓN DE MÉTODOS DE CONTROL CONVENCIONALES Y NO CONVENCIONALES PARA PLANIFICACIÓN DE RUTAS DE CUADRICÓPTERO

T E S I S

Que para obtener el grado de

Maestría en Ingeniería

Presenta:

Susana Borrego Domínguez

Director de Tesis: **Dr. Everardo Inzunza González**

Co-directora de Tesis: **Dra. Laura Jiménez Beristain**

Ensenada, Baja California, México

Agosto, 2025

UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

Facultad de Ingeniería, Arquitectura y Diseño

Evaluación de métodos convencionales y no convencionales para planificación de rutas de cuadricóptero

TESIS

que para obtener el grado de MAESTRO en INGENIERÍA presenta:

Susana Borrego Domínguez

Y aprobada por el siguiente comité:

Dr. Everardo Inzunza González

Director del Comité

Dra. Laura Jiménez Beristain

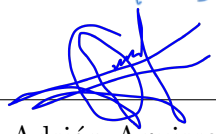
Co-directora del Comité

Dr. Enrique Efren García Guerrero

Miembro del Comité

Dr. José Jaime Esqueda Elizondo

Miembro del Comité



Dr. Oscar Adrián Aguirre Castro

Miembro del Comité

Dr. Oscar Roberto López Bonilla

Miembro del Comité

Agosto, 2025

RESUMEN de la tesis de **Susana Borrego Domínguez**, presentada como requisito parcial para obtener el grado de MAESTRO EN INGENIERÍA, del programa de Maestría y Doctorado en Ciencias e Ingeniería de la UABC. Ensenada, B.C. México, Agosto, 2025.

Evaluación de métodos convencionales y no convencionales para planificación de rutas de cuadricóptero

Resumen aprobado por:

Dr. Everardo Inzunza González
Director del Comité

Dra. Laura Jiménez Beristain
Co-directora del Comité

El presente trabajo de tesis establece una comparación entre técnicas de control no convencional (Aprendizaje por Refuerzo, RL) y técnicas de control convencional (PID) en el contexto de la planificación de trayectorias de cuadricópteros. La exigencia de trayectorias estables y eficaces en contextos dinámicos estimula la investigación con el objetivo de destacar las ventajas de las estrategias adaptativas en comparación con los controladores tradicionales. El entorno ROS 2 y el simulador Ignition Gazebo fueron utilizados para la modelización del comportamiento del cuadricóptero. Se utilizaron tres trayectorias de referencia - lemniscata, circular y aleatoria - aplicando diferentes ganancias del PID. Paralelamente, se entrenó al cuadricóptero con un modelo SAC (Soft Actor-Critic), para realizar tareas de despegue y vuelo estacionario (hover). Se realizaron valoraciones del error medio (EM), el sobreimpulso, la variabilidad, el error absoluto integral (EAI) y el error cuadrático medio (ECM). Los hallazgos indicaron que el controlador PID proporciona robustez y precisión en trayectorias predecibles, aunque demanda una sintonización meticulosa. El algoritmo SAC demostró habilidad para gestionar tareas de alta complejidad, aunque exhibió inestabilidad en etapas avanzadas del proceso de entrenamiento. Se deduce que el control convencional continúa demostrando su eficacia bajo condiciones claramente definidas, mientras que el RL posee potencial en contextos dinámicos, con futuras oportunidades de optimización a través de controladores híbridos e implementación de técnicas inteligentes para la modificación de parámetros en tiempo real. Esta investigación establece los cimientos para investigaciones experimentales de mayor complejidad y validación práctica en sistemas reales.

Palabras clave: quadcopter, fuzzy logic, PID control, Intelligent Control, Smart PID, Artificial intelligence, Reinforcement Learning, Machine Learning, Deep Learning, ROS, Robotic Operating System.

ABSTRACT of the thesis of **Susana Borrego Domínguez**, presented as a partial requirement to obtain the degree of MASTER in ENGINEERING, of the program of MSc and PhD in Sciences and Engineering of UABC. Ensenada, B.C. México, August, 2025.

Evaluation of conventional and non-conventional methods for quadcopter path planning

Summary approved by:

Dr. Everardo Inzunza González
Director del Comité

Dra. Laura Jiménez Beristain
Co-directora del Comité

The present thesis establishes a comparison between unconventional control techniques (Reinforcement Learning, RL) and conventional control techniques (PID) in the context of quadcopter trajectory planning. The demand for stable and efficient trajectories in dynamic contexts stimulates research with the aim of highlighting the advantages of adaptive strategies compared to traditional controllers. The ROS 2 environment and the Ignition Gazebo simulator were used for modeling the quadcopter's behavior. Three reference trajectories were used - lemniscate, circular, and random - applying different PID gains. Simultaneously, the quadcopter was trained with a SAC (Soft Actor-Critic) model to perform takeoff and hovering tasks. Evaluations were conducted on the mean error (ME), overshoot, variability, integral absolute error (IAE), and mean squared error (MSE). The findings indicated that the PID controller provides robustness and precision in predictable trajectories, although it requires meticulous tuning. The SAC algorithm demonstrated the ability to manage high-complexity tasks, although it exhibited instability in the advanced stages of the training process. It is deduced that conventional control continues to demonstrate its effectiveness under clearly defined conditions, while RL has potential in dynamic contexts, with future opportunities for optimization through hybrid controllers and the implementation of intelligent techniques for real-time parameter modification. This research lays the groundwork for more complex experimental investigations and practical validation in real systems.

Keywords: quadcopter, fuzzy logic, PID control, Intelligent Control, Smart PID, Artificial intelligence, Reinforcement Learning, Machine Learning, Deep Learning, ROS, Robotic Operating System.

A mi familia

Agradecimientos

Agradezco profundamente al ***Dr. Everardo Inzunza González****, mi director de tesis, por su invaluable guía durante el desarrollo de esta investigación. Su acompañamiento constante, tanto como tutor académico, docente y ser humano, fue un faro que me dio dirección y ánimo.*

Agradezco profundamente a la ***Dra. Laura Jimenez Beristain****, mi co-directora de tesis, por compartir generosamente su conocimiento y enriquecer mi formación con cada consejo y enseñanza. Su dedicación y entrega dejan huella en mi proceso de aprendizaje.*

Agradezco al ***Dr. Enrique Efrén García Guerrero*** *y al* ***Dr. Oscar Adrián Aguirre Castro****, por su respaldo durante el posgrado, por su tiempo, su tutoría y el conocimiento que aportaron para fortalecer esta investigación.*

A ***Rubén González Orozco****, mi compañero de vida, gracias por estar presente con palabras de aliento y abrazos sinceros. Tu amor, apoyo incondicional y motivación constante fueron impulso vital para no rendirme y seguir adelante en los días difíciles.*

*A mis abuelos,****Francisca López Álvarez*** *y* ***José Domínguez Armas****, porque su amor ha sido base firme sobre la que he crecido. Este logro también es de ustedes, una pequeña muestra del fruto de los valores y enseñanzas que sembraron en mí desde siempre.*

A mi mamá, hermana y familia, por ser ese sostén invisible, pero poderoso, que siempre me ha levantado. Cada paso que doy lleva un pedacito de ustedes.

Un agradecimiento al ***Consejo Nacional de Humanidades, Ciencias y Tecnologías (SECIHTI)****, por el apoyo económico brindado durante el desarrollo de la investigación, identificado por el registro (CVU) 1293519.*

Un agradecimiento a la ***Universidad Autónoma de Baja California*** *la* ***FIAD*** *y la*

FCQI *por abrirme las puertas y brindarme no solo el conocimiento necesario para mi formación profesional, sino también por ofrecer espacios, laboratorios e instalaciones que fueron clave para el desarrollo de este proyecto.*

A todos los docentes por su compromiso con la excelencia académica.

A todos mis amigos, por su apoyo y amistad incondicional.

Ensenada, B.C. México
Junio, 2024

Susana Borrego Domínguez

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

En el ámbito de la ingeniería de control, los sistemas que no tienen una relación lineal o proporcional entre sus entradas y salidas se denominan sistemas no lineales [1, 2]. A diferencia de los sistemas lineales, en los que un cambio en la entrada produce un cambio proporcional en la salida, los sistemas no lineales muestran un comportamiento más complejo [3] como oscilaciones múltiples, bifurcaciones o incluso caos, y no pueden describirse completamente mediante ecuaciones lineales. La resolución de sus ecuaciones diferenciales generalmente no admite soluciones analíticas, por lo que es necesario aplicar métodos de linealización o aproximaciones numéricas [4].

Para determinar la estructura y parámetros de un sistema, estos se pueden encontrar estudiando los procesos dinámicos en sistemas automáticos utilizando la teoría de control automático (TAC, theory of automatic control). En los últimos años, se han logrado resultados impresionantes en la TAC, principalmente asociados con la creación de sistemas de control automático (ACS, automatic control systems) para sistemas complejos de cohetes, sistemas robóticos, plantas de energía nuclear, así como vehículos aéreos no tripulados (UAV) que operan en modo automático la mayor parte del tiempo [4]. El tipo más común de vehículo aéreo no tripulado (UAV) es el cuadricóptero, también conocido como dron [5], cuya forma se muestra en la Figura 1.1, el cual es usado en [6].

Figura 1.1: *Modelo de cuadricóptero Parrot Mambo.*

El concepto de cuadricóptero surgió en 1923, cuando De Bothezat propuso un helicóptero multirotor con palas de paso fijo, pero la tecnología de la época no estaba lo suficientemente madura como para crear una máquina práctica [7]. Si bien, tuvo sus raíces en la aviación experimental del siglo XX, su auge se debe a la microelectrónica, los sensores inerciales y el GPS, que han hecho posible construir vehículos autónomos pequeños y baratos [8].

Hoy en día se utilizan en una variedad de contextos civiles y militares, incluyendo patrullas aéreas, exploración de recursos, seguimiento de víctimas de desastres, cartografía, monitoreo, identificar personas y brindar asistencia en operaciones de búsqueda y rescate [9].

En el contexto de la pandemia de COVID-19, los cuadricópteros fueron utilizados para tareas significativas. En algunos países, los cuadricópteros se han utilizado en la pulverización aérea y desinfección, transporte de muestras, verificación remota de temperatura en personas y supervisión del proceso de toque de queda, entre otras aplicaciones [10]. Por lo que este tipo de vehículos es una opción para tareas que requieren movimientos extremadamente ágiles en situaciones desafiantes. Para llevar a cabo operaciones con fiabilidad, se necesitan estrategias de control que resulten en un funcionamiento robusto y de alto rendimiento.

1.1. Motivación

La planificación de rutas en cuadricópteros ha adquirido gran relevancia debido a su amplia gama de aplicaciones en vigilancia, entrega de paquetes, agricultura de precisión y misiones de búsqueda y rescate [9]. Sin embargo, su control representa un reto complejo debido a su comportamiento altamente no lineal, multivariable y acoplado. Existen múltiples enfoques de control, desde métodos clásicos como control Proporcional-Integral-Derivativo (PID) o Control Lineal Cuadrático (LQR), hasta estrategias no convencionales como control difuso, modelos deslizantes o aprendizaje por refuerzo [11, 12].

Esta investigación se motiva por la necesidad de explorar y analizar comparativamente estos dos enfoques de control en la planificación de rutas de cuadricópteros. Se busca aportar evidencia comparativa desde un entorno que permita evaluar el desempeño de dos metodologías de control aplicadas a cuadricópteros.

1.2. Planteamiento del problema

La planificación de rutas para cuadricópteros es un desafío crítico en aplicaciones como entrega de mercancías, monitoreo ambiental y rescates [13]. Los métodos de control convencionales, como los algoritmos basados en reglas predefinidas, presentan limitaciones en entornos dinámicos y desconocidos. Por otro lado, los enfoques no convencionales, como el aprendizaje por refuerzo o las redes neuronales, ofrecen soluciones adaptativas, pero su implementación requiere un análisis exhaustivo de eficiencia y viabilidad [14]. Esto plantea la necesidad de evaluar comparativamente ambos enfoques para determinar cuál es más adecuado en términos de precisión, estabilidad y capacidad de respuesta en distintos escenarios operativos.

1.2.1. Descripción del problema

El problema se centra en la dificultad para seleccionar el método de control más adecuado para garantizar una trayectoria estable y eficiente de un cuadricóptero de manera autónoma. Se pretende evaluar al menos un controlador clásico y un controlador no convencional bajo un mismo marco experimental, realizando simulaciones en el entorno ROS 2 (Robot Operating

System 2), Gazebo y RViz (ROS Visualization), apoyándose en scripts de Python y C++, aplicados a un modelo dinámico de cuadricóptero.

1.2.2. Preguntas de investigación

- ¿Cuáles son las diferencias en desempeño entre los métodos de control convencionales y no convencionales en la planificación de rutas de cuadricópteros?
- ¿Cuál de los métodos estudiados ofrece mayor precisión y estabilidad en el seguimiento de trayectorias en un entorno simulado con ROS?
- ¿Cómo influyen los parámetros del controlador en el comportamiento dinámico del cuadricóptero en diferentes tipos de trayectorias?
- ¿Qué ventajas ofrece el uso de estrategias inteligentes como control difuso o aprendizaje por refuerzo frente a los métodos clásicos en escenarios simulados?

1.2.3. Delimitación del problema

Esta investigación se limita exclusivamente a simulaciones de planificación de trayectorias de un cuadricóptero dentro del entorno ROS 2, haciendo uso de Python y C++ para el desarrollo de scripts de control. No se incluyen pruebas experimentales con hardware real ni se abordan aspectos de comunicación, percepción o navegación autónoma completa. El análisis se centra únicamente en el desempeño de los métodos de control sobre el comportamiento del dron en simulación, considerando condiciones ideales y perturbaciones controladas.

1.3. Objetivos

1.3.1. Objetivo general

Evaluar y comparar el desempeño de distintos métodos de control convencionales y no convencionales aplicados a la planificación y seguimiento de trayectorias de un cuadricóptero, utilizando el entorno de simulación de ROS.

1.3.2. Objetivos específicos

- Implementar un modelo dinámico de cuadricóptero en un entorno simulado utilizando ROS 2.
- Diseñar e integrar un controlador convencional y no convencional para el seguimiento de trayectoria de un cuadricóptero.
- Establecer métricas de desempeño que permitan comparar precisión, estabilidad y robustez del método aplicado.
- Analizar los resultados obtenidos para identificar las fortalezas y limitaciones de cada enfoque de control en escenarios simulados.

1.4. Organización de la tesis

La presente tesis se estructura en seis capítulos que permiten abordar de manera progresiva y sistemática la problemática de estudio. En el **Capítulo 1** se presenta el planteamiento del problema, la motivación de la investigación y el contexto general en el que se sitúa el estudio, destacando la relevancia de los vehículos aéreos no tripulados, particularmente los cuadricópteros. El **Capítulo 2** está dedicado al marco teórico, donde se describen los fundamentos esenciales sobre sistemas no lineales, teoría de control automático, características dinámicas de los cuadricópteros, así como los distintos enfoques de control convencionales y no convencionales aplicables a este tipo de sistemas. En el **Capítulo 3** se realiza una revisión de la literatura especializada, explorando los aportes más relevantes de diversos autores en torno al control de trayectorias de cuadricópteros. El **Capítulo 4** expone la metodología empleada en la investigación, detallando el diseño del sistema de simulación, el hardware y software seleccionado, especialmente el uso del entorno ROS. El **Capítulo 5** presenta los resultados obtenidos tras la experimentación, acompañados de un análisis comparativo entre los diferentes métodos de control aplicados. Finalmente, en el **Capítulo 6** se ofrecen las conclusiones generales del estudio, destacando los hallazgos más significativos, así como posibles líneas futuras de investigación que permitan profundizar o ampliar el trabajo desarrollado.

Capítulo 2

Marco Teórico

2.1. Sistemas no lineales

Los sistemas no lineales representan una desviación significativa del mundo simplificado de la teoría de sistemas lineales [15], introduciendo complejidades que requieren técnicas analíticas y computacionales avanzadas para su comprensión y control. A diferencia de los sistemas lineales, en los que rige el principio de superposición, los sistemas no lineales presentan una sensiblidad considerable a las condiciones iniciales y a los parámetros de un sistema [16], lo que puede generar respuestas notablemente diferentes ante variaciones pequeñas.

En los sistemas reales, la relación entre la entrada y la salida es usualmente no lineal [4], lo que implica que no existe una proporcionalidad directa entre ambas variables. Esta condición, complica su análisis y diseño, ya que pueden emerger comportamientos complejos y sensibles a las condiciones iniciales. El control por realimentación, en particular la realimentación de salida, desempeña un papel crucial en la gestión de sistemas no lineales, en los que la salida del sistema se utiliza para realizar correcciones y mantener la estabilidad. Esta retroalimentación puede ser positiva o negativa, dependiendo de si los valores de salida suman o se restan de un valor de referencia [1].

El estudio de los sistemas no lineales suele implicar el examen de la estabilidad, la controlabilidad y la observabilidad, pero estos conceptos resultan mucho más intrincados que sus homólogos lineales. Suelen utilizarce herramientas matemáticas como la teoría de estabilidad

de Lyapunov, el análisis de funciones descriptivas y la teoría de bifurcación para analizar el comportamiento de los sistemas no lineales.

2.2. Vehículos aéreos no tripulados: cuadricóptero

Los vehículos aéreos no tripulados (UAV) son plataformas voladoras que operan sin un piloto humano a bordo. Estas aeronaves pueden ser controladas de forma remota o funcionar de manera autónoma, siguiendo trayectorias predefinidas o haciendo uso de algoritmos inteligentes. Los UAV han sido ampliamente utilizados en sectores como la vigilancia, la agricultura, la entrega de paquetes, la exploración científica y la defensa, esto gracias a su capacidad para acceder a entornos difíciles o peligrosos para los humanos [17].

Figura 2.1: *Cuadricóptero modelo S500 utilizado en [11].*

Dentro de esta categoría, el cuadricóptero es una de las configuraciones más populares. Se trata de un tipo de dron con cuatro rotores ubicados en las esquinas de una estructura en forma de cruz, así como se muestra en Figura 2.1. Para que un cuadricóptero se mantenga en vuelo estable, dos de sus hélices giran en sentido horario y las otras dos en sentido antihorario. Esta disposición permite cancelar el par neto y facilita la estabilidad del vuelo sin necesidad de un rotor de cola, a diferencia de los helicópteros tradicionales [18].

El cuadricóptero es un sistema subactuado, esto porque cuenta con seis grados de libertad, pero solo dispone de cuatro entradas de control, correspondientes a la velocidad de rotación

de cada uno de los rotores [5]. Esta particularidad implica que se deben coordinar múltiples acciones para lograr un control completo del dron.

Desde una perspectiva funcional, el modelo dinámico del cuadricóptero puede dividirse en dos subsistemas principales:

- Subsistema traslacional: se encarga del movimiento en el espacio a partir del desplazamiento en los ejes x, y, z. Los movimientos esperados son adelante/atrás, izquierda/-drecha y subir/bajar.
- Subsistema rotacional: este es responsable de las rotaciones del dron, como alabeo (roll), cabeceo (pitch) y guiñada (yaw), las cuales afectan directamente la orientación y dirección de vuelo [5]. Los movimientos a observar son el giro lateral del cuadricóptero, su inclinación hacia delante o atrás y su giro horizontal sobre su eje vertical (como gira una brújula).

El control de estos subsistemas es interdependiente, significando que lo que pasa en un subsistema influye directamente en el comportamiento del otro. Las rotaciones inducidas, por ejemplo, la inclinación hacia adelante, permiten generar desplazamientos horizontales mediante el redireccionamiento del empuje vertical producido por los rotores [18].

Esta relación entre rotación y desplazamiento se refleja directamente en los movimientos del cuadricóptero. El cuadricóptero puede realizar ocho tipos básicos: despegue, aterrizaje, desplazamiento hacia adelante y atrás, hacia la derecha e izquierda, así como rotación en sentido horario y antihorario. Estos movimientos se logran mediante el ajuste diferencial de las velocidades de los motores [5].

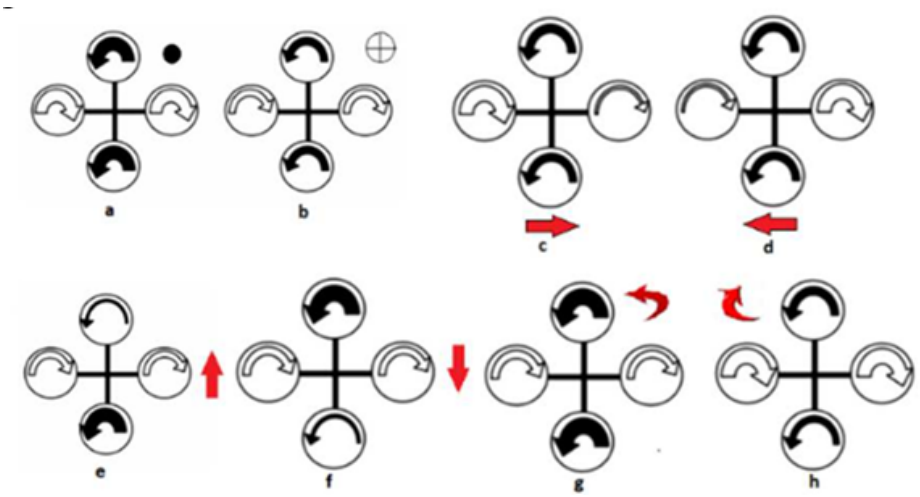


Figura 2.2: *a) Despegue, b) aterrizaje, c) derecha, d) izquierda, e) adelante, f) atrás, g) en el sentido de las agujas del reloj y h) en sentido contrario a las agujas del reloj [10].*

Para llevar a cabo estos movimientos de manera eficaz, es necesario tener en cuenta la configuración estructural del cuadricóptero, dado que esta tiene un impacto directo en su comportamiento dinámico y en las estrategias de control que se deben implementar. El cuadricóptero tiene dos configuraciones principales: la configuración + y la configuración **X** [5], como se muestran en la Figura 2.3, donde se documentan ambas configuraciones respecto al sistema de coordenadas [19].

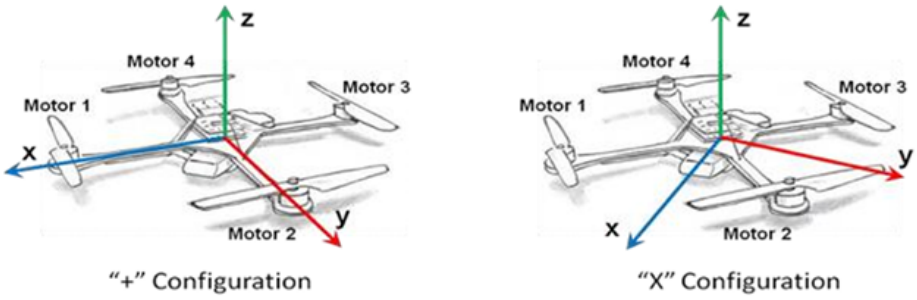


Figura 2.3: *Ejemplo de la distribución del sistema de coordenadas, en configuración + y* ***X*** *[19].*

La elección de la configuración del cuadricóptero depende en gran medida de la aplicación o el enfoque que se usará en este. En la Tabla 2.1 se explica cuál es la orientación de los brazos, cuál es la dirección frontal, cómo es el control y maniobrabilidad, y el campo de visión del

cuadricóptero según su configuración [18, 20].

Tabla 2.1: Comparación entre configuraciones + y **X** del cuadricóptero.

Descripción	**Configuración +**	**Configuración X**
Orientación de los brazos	Los cuatro brazos del cuadricóptero están alineados con los ejes principales.	Los brazos están orientados en ángulos de 45° respecto a los ejes principales.
Dirección frontal	Uno de los brazos apunta directamente hacia adelante.	El frente del cuadricóptero está situado entre dos brazos.
Control y maniobrabilidad	En esta configuración, los movimientos de cabeceo y alabeo están alineados con los motores.	Esta configuración suele ofrecer una mejor respuesta en maniobras complejas.
Campo de visión	Puede ser menos ideal para montar cámaras frontales, debido a la obstrucción por los brazos.	Es más adecuada para aplicaciones que requieren una vista clara hacia adelante.

Comúnmente se hace uso de la configuración **X**, ya que ofrece una mejor respuesta aerodinámica y estabilidad durante maniobras rápidas o vuelos de alta velocidad [21]. También permite agregar aditamentos al cuadricóptero, como sensores y cámaras, sin mayor inconveniente. Pero esto no significa que no se use la configuración +. Por ejemplo, en Beneder et al. [22] se hace uso de la configuración +, donde se modela y controla el cuadricóptero. En la Figura 2.4 se observa que los autores añadieron una barra fija, situada entre las hélices azules, para montar sensores ambientales y una cámara orientada hacia adelante. Si bien mencionan que obtuvieron resultados satisfactorios, el agregar dicha barra afectó el rendimiento de la maniobrabilidad del cuadricóptero.

Figura 2.4: *Cuadricóptero con sensores y cámara montada en barra fija, en configuración + [22].*

2.3. Modelado del cuadricóptero

Un cuadricóptero es un sistema dinámico de seis grados de libertad. Sin embargo, sólo tiene cuatro entradas independientes, lo que lo convierte en un sistema subactuado. La dinámica del cuadricóptero puede dividirse en dos subsistemas:

- Subsistema traslacional: determina la posición de altitud (ejes x, y, z).
- Subsistema rotacional: genera la posición de actitud (alabeo, cabeceo y guiñada).

Hay diferentes maneras de modelar un cuadricóptero, las más comunes son: la Newton-Euler, la Euler-Lagrange y la de Newton. La ecuación Euler-Lagrange permite describir la dinámica del cuadricóptero [5]. La ecuación de Newton - Euler un método de modelado matemático que se basa en las leyes de Newton del movimiento y las ecuaciones de Euler para describir el movimiento de cuerpos rígidos [18, 23]. Este enfoque considera fuerzas y momentos que actúan sobre el cuadricóptero y establece ecuaciones diferenciales que describen su movimiento traslacional y rotacional. El modelo de Newton es una simplificación del modelo de Newton-Euler, en el cual se desprecian algunos términos no lineales y se linealizan las ecuaciones de movimiento.

Para obtener la posición y orientación del cuadricóptero en el espacio, se considera un marco de coordenadas inercial

$$G = \{X_G\ , Y_G\ , Z_G\} \tag{2.1}$$

y un marco de coordenadas adjunto al cuadricóptero,

$$B = \{X_B\ , Y_B\ , Z_B\} \tag{2.2}$$

como se muestra en laFigura 2.5. La orientación en el espacio se representa por medio de los tres ángulos de Euler: alabeo(ϕ), cabeceo (θ), guiñada (ψ), con respecto al marco B. El ángulo de alabeo (ϕ) representa la rotación del cuadricóptero respecto al eje X_B, el ángulo de cabeceo (θ) representa la rotación respecto al eje Y_B y el ángulo de guiñada (ψ) representa la rotación respecto al eje Z_B. En la Figura 2.5 se muestra el modelo dinámico del cuadricóptero usado en [24].

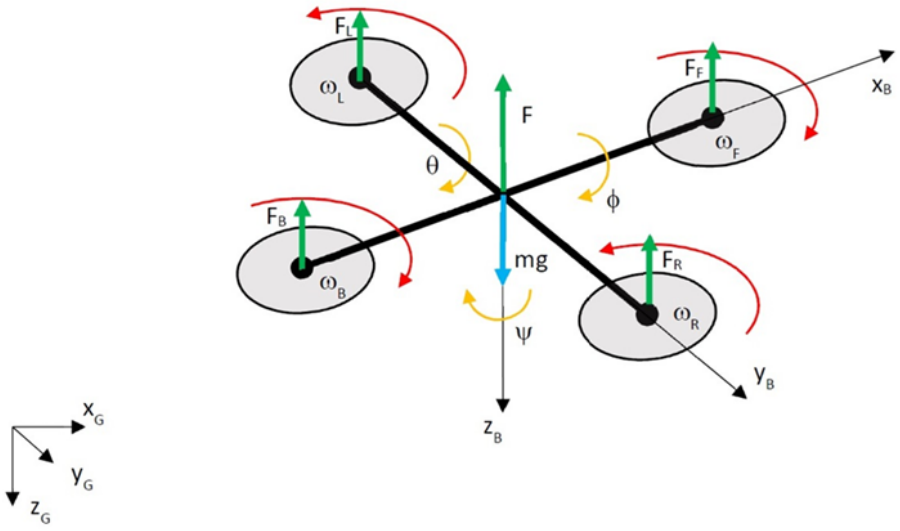


Figura 2.5: *Modelo dinámico del cuadricóptero [24].*

Para lograr el vuelo estacionario (hover) en el cuadricóptero, es necesario que la suma de las fuerzas de empuje de los 4 rotores sea igual al peso del vehículo. Por otro lado, un rotor al girar genera un momento reactivo debido a las fuerzas aplicadas sobre la hélice, este momento reactivo es de sentido contrario al giro de la hélice, debido a este fenómeno, dos hélices giran en sentido horario y las otras dos giran en sentido antihorario, con el fin de contrarrestar los momentos reactivos que el cuerpo experimenta al hacer girar los rotores, así cada par de motores opuestos giran en el mismo sentido. Al variar las velocidades angulares de sus rotores se pueden modificar los ángulos de Euler, lo cual produce movimiento traslacional en el cuadricóptero.

Para obtener el modelo matemático del cuadricóptero se hacen las siguientes suposiciones [24]:

1. El cuadricóptero tiene una estructura indeformable.
2. Los efectos aerodinámicos en la estructura del vehículo son despreciables.
3. El cuadricóptero es perfectamente simétrico respecto a sus ejes.
4. El cuadricóptero no realiza maniobras agresivas, es decir, $(\phi, \theta) \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$ y $\psi \in [-\pi, \pi]$.
5. La dinámica de los rotores es instantánea, es decir, la fuerza de empuje generada por los rotores es alcanzada instantáneamente.

6. La dinámica rotacional, en lazo cerrado, es mucho más rápida que su dinámica traslacional.

2.3.1. Modelo dinámico de un cuadricóptero

Las coordenadas generalizadas están dadas por el cuadricóptero puede ser descrito por los vectores $\zeta = \begin{bmatrix} x, & y, & z \end{bmatrix}^T$ y $\eta = \begin{bmatrix} \phi, & \theta, & \psi \end{bmatrix}^T$, donde ξ denota la posición en el espacio del centro de gravedad del cuadricóptero con respecto al marco G y η denota los ángulos de Euler [25]. Usando el formalismo de Newton-Euler, se considera al cuadricóptero como un cuerpo rígido bajo fuerzas externas aplicadas a su centro de masa, entonces el modelo dinámico del cuadricóptero está dado por

$$\begin{aligned} m\ddot{\xi} &= R^B F - mg^G D, \\ J\dot{\Omega} &= -\Omega \times J\Omega + \tau, \end{aligned} \tag{2.3}$$

donde m es la masa total del cuadricóptero. ${}^G D = \begin{bmatrix} 0, & 0, & 1 \end{bmatrix}^T$ es un vector unitario en dirección Z_G, g es la constante de gravedad, Ω representa la velocidad angular del cuadricóptero con respecto al marco de coordenadas B; J representa la matriz de inercia con respecto al marco de coordenadas B. El vector de torques se representa como $\tau = \begin{bmatrix} \tau_\phi, & \tau_\theta, & \tau_\psi \end{bmatrix}^T$ es el vector de pares aplicados al cuadricóptero con respecto al marco de coordenadas B. $F = \begin{bmatrix} 0, & 0, & u \end{bmatrix}^T$ es el vector de fuerzas aplicadas al cuadricóptero con respecto al marco B, $u = \sum_{i=1}^{4} f_i$ es la fuerza de empuje total vertical aplicada al cuadricóptero generada por los cuatro rotores, donde f_i es la fuerza de empuje generada por el i-ésimo rotor. La fuerza u se asume que actúa solo en dirección Z_B. R es la matriz de rotación que representa la orientación del cuerpo rígido con respecto al marco inercial tal que $R : B \to G$ es ortogonal.

R se obtiene al realizar tres rotaciones sucesivas utilizando la convención $X_B\ Y_B\ Z_B$ y está dada por

$$R = R_z(\psi) R_y(\theta) R_x(\phi) \tag{2.4}$$

$$R = \begin{bmatrix} \cos\theta\cos\psi, & \operatorname{sen}\phi\operatorname{sen}\theta\cos\psi - \cos\phi\operatorname{sen}\psi, & \cos\phi\operatorname{sen}\theta\cos\psi + \operatorname{sen}\phi\operatorname{sen}\psi \\ \cos\theta\operatorname{sen}\psi, & \operatorname{sen}\phi\operatorname{sen}\theta\operatorname{sen}\psi + \cos\phi\cos\psi, & \cos\phi\operatorname{sen}\theta\operatorname{sen}\psi - \operatorname{sen}\phi\cos\psi \\ -\operatorname{sen}\theta, & \operatorname{sen}\phi\cos\theta, & \cos\phi\cos\theta \end{bmatrix}, \quad (2.5)$$

donde

$$\begin{gathered} R_z(\psi) = \begin{bmatrix} \cos\psi, & -\operatorname{sen}\psi, & 0 \\ \operatorname{sen}\psi, & \cos\psi, & 0 \\ 0, & 0, & 1 \end{bmatrix}, R_y(\theta) = \begin{bmatrix} \cos\theta, & 0, & \operatorname{sen}\theta \\ 0, & 1, & 0 \\ -\operatorname{sen}\theta, & 0, & \cos\theta \end{bmatrix}, \\ R_x(\phi) \begin{bmatrix} 1, & 0, & 0 \\ 0, & \cos\phi, & -\operatorname{sen}\phi \\ 0, & \operatorname{sen}\phi, & \cos\phi \end{bmatrix}. \end{gathered} \quad (2.6)$$

Existe la matriz $W(\eta)$ que relaciona las velocidades angulares con respecto al marco B, Ω, con las velocidades angulares generalizadas en la región donde los ángulos de Euler son válidos. Sin importar la convensión de rotación usada para calcular (2.5), para obtener $W(\eta)$ se tomará la convensión de rotación $Z_B\ Y_B\ X_B$, entonces se tiene que

$$\tilde{R} = R_x(\phi)R_y(\theta)R_z(\psi), \quad (2.7)$$

entonces,

$$\Omega = \dot{\phi}e_1 + \dot{\theta}R_x^{-1}e_2 + \dot{\psi}R_x^{-1}R_y^{-1}e_3 = W(\eta)\dot{\eta} \quad (2.8)$$

con

$$e_1 = [1,0,0]^T, \quad e_2 = [0,1,0]^T, \quad e_3 = [0,0,1]^T, \quad (2.9)$$

$$W(\eta) = \begin{bmatrix} 1, & 0, & -\operatorname{sen}\theta \\ 0, & \cos\phi, & \operatorname{sen}\phi\cos\theta \\ 0, & -\operatorname{sen}\phi, & \cos\phi\cos\theta \end{bmatrix} \quad (2.10)$$

Sustituyendo (2.8) en (2.3) se obtiene el modelo dinámico del cuadricóptero, expresado en términos de las coordenadas generalizadas y escrito de forma matricial, dado por

$$m\ddot{\xi} = R^B F - mg^G D, \quad (2.11)$$

$$JW\ddot{\eta} = -J\dot{W}\dot{\eta} - W\dot{\eta} \times JW\dot{\eta} + \tau \tag{2.12}$$

con las entradas de control dadas por ${}^{B}F = \begin{bmatrix} 0, & 0, & u \end{bmatrix}^T$ y $\tau = \begin{bmatrix} \tau_\phi, & \tau_\theta, & \tau_\psi \end{bmatrix}^T$.

2.3.2. Modelo cinemático de un cuadricóptero

Para utilizar el modelo cinemático del cuadricóptero, es necesario tomar en cuenta las siguientes suposiciones:

1. Se consideran bajas aceleraciones de traslación a lo largo de los ejes X_B Y_B Y_Z, esto es $\ddot{x}, \ddot{y}, \ddot{z} \approx 0$.
2. Se consdieran los ángulos de orientación (ϕ,θ) cercanos a cero. De esta manera, el eje vertical Z_B se puede suponer paralelo al eje vertical Z_G.
3. El cuadricóptero posee controladores internos tal que, para provocar un movimiento de traslación, se garantiza que las entradas del sistema son velocidades translacionales, en los ejes X_B Y_B Z_B.
4. El cuadricóptero posee un controlador interno tal que, para provocar un movimiento de rotación sobre su eje Z_B, se garantiza que la entrada de control es una velocidad angular.

Bajo las suposiciones anteriores, el modelo cinemático está dado por

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} \cos\psi, & -\operatorname{sen}\psi \\ \operatorname{sen}\psi, & \cos\psi \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix}, \tag{2.13}$$

$$\dot{z} = v_z, \tag{2.14}$$

$$\dot{\psi} = w_z, \tag{2.15}$$

donde V_x, V_y, V_z, W_z son las entradas de control, con respecto del marco de coordenada B.

2.4. Planificación de rutas

La planificación de rutas o trayectorias se refiere al proceso de encontrar una trayectoria factible y eficiente para que un vehículo autónomo se mueva desde un punto de inicio hasta un punto de destino, evitando obstáculos y cumpliendo con ciertas restricciones o criterios de optimización [26]. Este aspecto es crítico para todo vehículo autónomo, como los drones, para que operen de manera autónoma o semiautónoma en entornos complejos y dinámicos. Por ejemplo, en la Figura 2.6 se observa una trayectoria definida a partir de cajas que simulan edificios. Aquí el cuadricóptero aprende a seguir una ruta a base de aprendizaje Q en tiempo continuo, el cual es un tipo de aprendizaje por refuerzo [27].

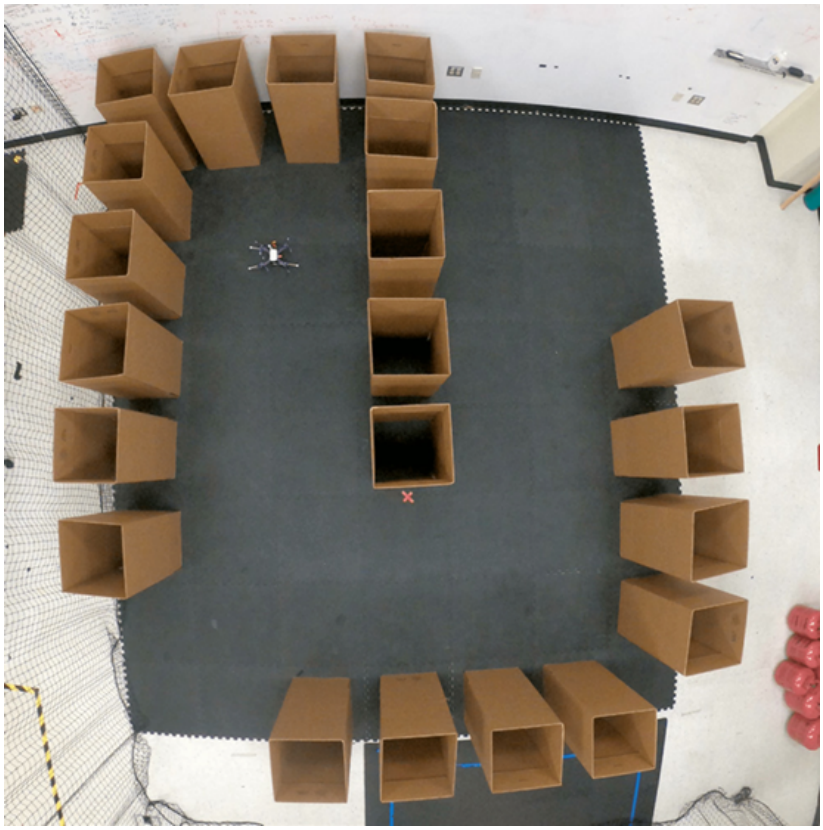


Figura 2.6: *Imagen de experimento de planificación de rutas con hardware real en laboratorio de vuelo [27].*

La planificación de trayectoria implica analizar datos del medio, detectar obstáculos y tomar decisiones en tiempo real. Para realizar esta tarea, se recurre a distintos métodos, entre los que

destacan los basados en muestreo, en búsqueda y en optimización. La elección del algoritmo depende de la complejidad del entorno, los requisitos de tiempo de cálculo y la precisión requerida en la trayectoria planificada [28].

Estos métodos pueden aplicarse en distintos niveles de planificación, dependiendo del alcance espacial y temporal del trayecto que se desea generar. La planificación de rutas se divide en planificación global y local. La planificación global genera una trayectoria completa antes de la ejecución, basándose en un conocimiento completo del entorno [28]. Esta planificación proporciona una ruta de alto nivel desde el punto de inicio hasta el destino, considerando los obstáculos conocidos y los objetivos generales de la misión [29]. En entornos no estructurados, como interiores, el robot debe explorar el entorno para crear un mapa y diseñar una trayectoria hacia el destino [30].

En contraste, la planificación local se enfoca en generar trayectorias en tiempo real a medida que el robot se desplaza por el entorno. La planificación local se basa en la información sensorial actual para ajustar la trayectoria a corto plazo, evitando obstáculos imprevistos y adaptándose a cambios en el entorno. Los algoritmos de planificación local son esenciales para manejar la incertidumbre y la dinámica en tiempo real, y suelen combinarse con la planificación global para proporcionar una navegación robusta y adaptable [31].

2.5. Controlador PID

El controlador PID (Proporcional-Integral-Derivativo) es ampliamente utilizado en la industria debido a su simplicidad, eficacia y facilidad de implementación [32]. Funciona ajustando la entrada de un sistema para minimizar la diferencia entre su estado actual y un estado deseado, al cual llamamos error [33], mediante sus tres componentes principales:

- **Proporcional (P)**: responde directamente al error actual, ajustando la salida proporcionalmente al tamaño de este.
- **Integral (I)**: Considera la suma acumulada de los errores pasados para corregir desviaciones prolongadas.
- **Derivativo (D)**: Actúa en función de la velocidad de cambio del error, ayudando a prevenir sobre correcciones o inestabilidad.

La ecuación del controlador PID está dada por:

$$U(t) = K_P e(t) + K_I \int e(t)dt + K_D \frac{de(t)}{dt} \tag{2.16}$$

Donde $U(t)$ es la salida del controlador, $e(t)$ es el error en el tiempo t, y K_P, K_I, K_D son parámetros que determinan el peso de cada componente del PID. En el caso de los cuadricópteros, el controlador PID se utiliza para regular variables como la altitud, la posición y la orientación (ángulos de cabeceo, balanceo y guiñada), permitiendo así la estabilización del vuelo y el seguimiento de trayectorias [34]. Estos controladores también pueden usarse en cascada, donde un controlador maneja variables primarias, como orientación, y otro controla variables secundarias, como velocidad de desplazamiento.

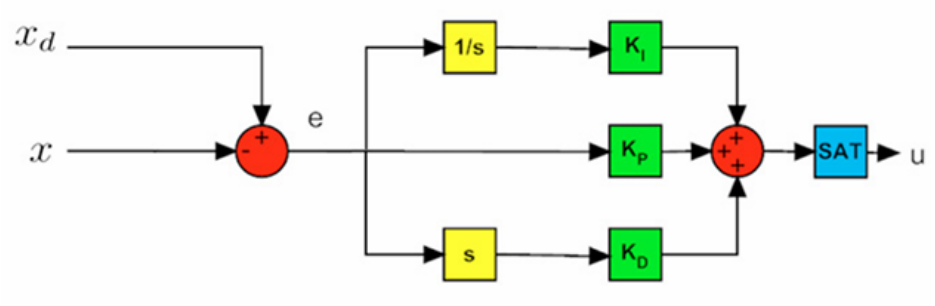


Figura 2.7: *Ejemplo de aplicación de PID en cuadricóptero Parrot Mambo [4].*

Un caso específico de la estructura de control PID aplicada al cuadricóptero Parrot Mambo se muestra en la Figura 2.7. En [4] se utilizan seis controladores PID para estabilizar las coordenadas espaciales (x, y, z) y los ángulos de rotación (ϕ, θ, ψ) en este modelo. Primero corrigen los errores de posición y después usan controladores adicionales para mantener la orientación deseada. Lo anterior, facilita la sintonización progresiva de los PID ajustando alabeo (roll) y cabeceo (pitch) con las demás variables fijas.

2.6. Controlador por aprendizaje por refuerzo

El aprendizaje por refuerzo es una técnica dentro de la inteligencia artificial que entrena a un agente para tomar decisiones óptimas en un entorno, a través de la interacción directa con este y mediante un sistema de recompensas. En esencia, el agente aprende a elegir acciones que maximicen una función de recompensa acumulada, basándose en la retroalimentación recibida del entorno. Esta metodología ha demostrado ser especialmente útil para abordar problemas donde el entorno es incierto, dinámico o parcialmente observable [35]. En la Figura 2.8 se observa un bosquejo de la estructura general del aprendizaje por refuerzo [36], donde un agente interactúa con un entorno sin supervisión explícita. El agente recibe recompensas positivas por realizar acciones "buenas" y recompensas negativas por acciones "malas".

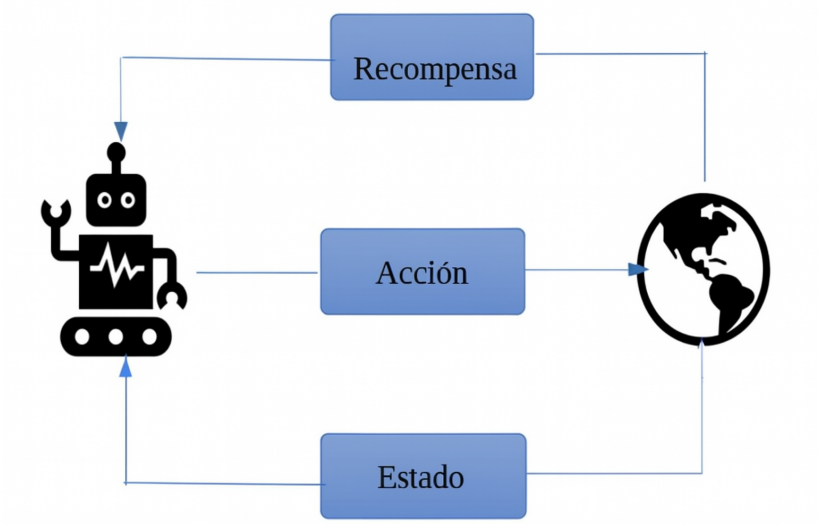


Figura 2.8: *Estructura de aprendizaje por refuerzo [36].*

En el contexto de los cuadricópteros, el aprendizaje por refuerzo se ha aplicado ampliamente de forma eficiente, incluso en entornos complejos o previamente desconocidos [37]. Gracias al uso de algoritmos de aprendizaje por refuerzo, como Q-learning y Deep Q-Networks, los cuadricópteros pueden adaptarse a cambios en el entorno, esquivar obstáculos estáticos y dinámicos, y ajustar su trayectoria en tiempo real con base en la retroalimentación sensorial [38,39]. Estos algoritmos han demostrado mejorar no solo la seguridad del vuelo, sino también la eficiencia en términos de tiempo de llegada y consumo energético, al encontrar rutas más cortas o menos exigentes para los motores del cuadricóptero [35, 40].

Capítulo 3

Estado del arte

La investigación en el campo de la planificación de rutas de cuadricópteros ha explorado diversas técnicas, desde controladores clásicos hasta enfoques más avanzados como el control óptimo y el aprendizaje por refuerzo. En esta sección se explican brevemente algunas de las contribuciones más recientes y la tendencia de investigación en este campo.

3.1. Trabajos relacionados

Se han propuesto varios enfoques en la literatura para mejorar la planificación de trayectorias en UAVs, particularmente en entornos dinámicos y con obstáculos. Uno de estos enfoques se presenta en el trabajo de Hayajneh et al. [41], donde se propone un método mejorado para la planificación de trayectorias de UAVs, el cual se basa en la extensión del método tradicional de campos potenciales, también conocido como campos de potencial artificial. Este método implica simular el entorno del vehículo como un campo de fuerzas, donde los obstáculos producen fuerzas repulsivas que dirigen el UAV lejos de las áreas de peligro y el objetivo ejerce una fuerza atractiva que lo dirige hacia su meta. El resultado de estas fuerzas determina cómo se mueve el vehículo. Debido a que se puede utilizar en tiempo real y ajustarse a los cambios en la posición, orientación y velocidad del UAV a lo largo de su misión, este tipo de planificación es especialmente útil en entornos dinámicos.

Los campos potenciales basados en la aceleración (ABPF, por sus siglas en inglés) son el

método que resulta de la mejora sugerida en este trabajo, que también incluye la aceleración relativa entre el UAV y el objetivo dentro del campo potencial. En comparación con las iteraciones anteriores de la técnica, esta integración permite una planificación más sensible al comportamiento dinámico del objetivo, produciendo trayectorias más suaves y precisas.

Por otro lado, el trabajo de Jahan et al. [42] consiste en un dron bombero. El dron está implementado con un microcontrolador Arduino Uno como el sistema de control central. Incluye una cámara ESP y varios sensores, como sensores ultrasónicos y de llama para la detección de incendios. Se utiliza una bomba de agua para extinguir incendios, así como se muestra en la Figura 3.1.

Figura 3.1: *Dron bombero con rutas dinámicas, usado en [42].*

La trayectoria del cuadricóptero depende de un método de control backstepping basado en lógica difusa, que se utiliza para mejorar la estabilidad del UAV, el seguimiento de trayectoria y reducir el parpadeo. La robustez de la posición y altitud del dron se mejora utilizando una técnica de optimización de control backstepping basada en lógica difusa. La red difusa procesa la retroalimentación y ajusta la entrada para el diseño de control backstepping.

Otro enfoque se presenta en el artículo de Abdelhay y Zakriti [33], donde se hace uso de un controlador PID en cascada para el seguimiento de trazadores en cuadricópteros. Este modelo de control se utiliza para variables regulares como la posición, actitud, velocidad y altitud del vehículo en el espacio. El método de Newton-Euler, que se basa en las ecuaciones

de movimiento y las fuerzas de momento, se utiliza para desarrollar el modelo matemático del sistema. El modelo está diseñado para funcionar a un nivel donde un sistema no lineal puede ser linealizado, lo que facilita el diseño del controlador. Además, se recomienda utilizar un PID en un caso simple para optimizar la implementación del seguimiento de trayectoria, permitiendo que los estados del cuadricóptero converjan hacia un conjunto de referencias temporales variables a través de un controlador específicamente diseñado para el modelo dinámico no lineal.

Por otro lado, en Quan et al. [43] diseñaron un control no lineal adaptativo y robusto, donde hacen uso de dos capas de control, una para la posición y otra para la actitud, haciendo uso del control de modo deslizante (*Sliding Mode Control*, SMC) para la primera y agregando la técnica de backstepping para la segunda; enfocado al seguimiento preciso de trayectorias bajo perturbaciones externas e internas. En la Figura 3.2 se documenta el diagrama a bloques del esquema de control del cuadricóptero.

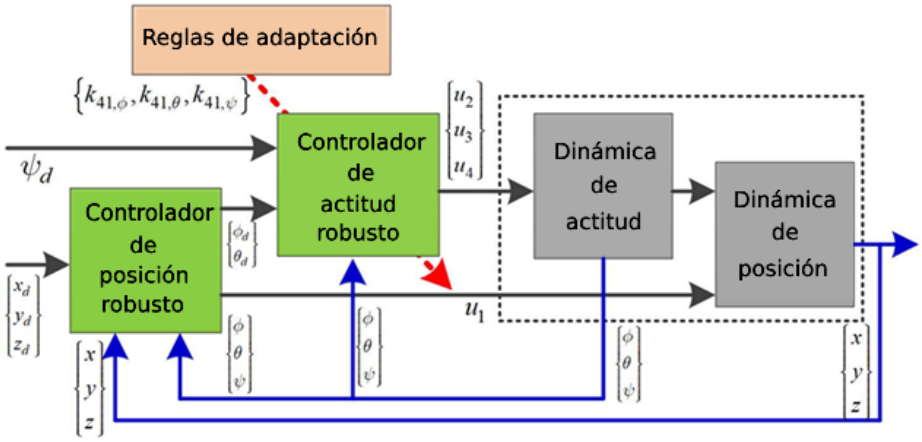


Figura 3.2: *Esquema de control para cuadricóptero en [43].*

En el trabajo de Asti et al. [44] se hace uso de aprendizaje automático (*Machine Learning*, ML) teniendo como objetivo evitar obstáculos considerando la eficiencia energética; haciendo uso del algoritmo KNN (vecino más cercano) muestra una alta precisión y tiempos bajos. La Figura 3.3 ilustra el diagrama de bloques del sistema de control del cuadricóptero, que consta de un bucle interno para el control de actitud y un bucle externo para el control de posición. El controlador proporcional-derivativo se utiliza para controlar el movimiento del cuadricóptero desde el punto de partida hasta el punto objetivo deseado en el entorno 3D.

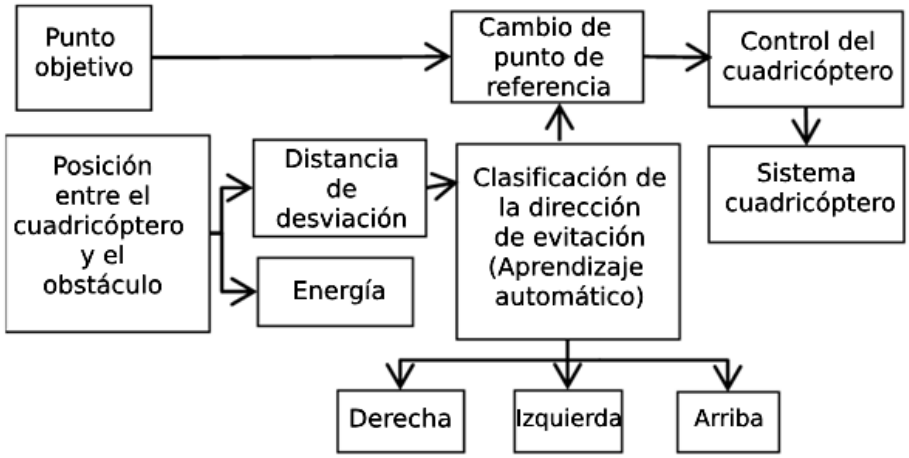


Figura 3.3: *Esquema propuesto en [44] para seguir una trayectoria y evitar obstáculos.*

Para validar el funcionamiento del sistema propuesto, se presentan distintos escenarios de navegación en la investigación. En la Figura 3.4, se ilustra uno de estos escenarios, mostrando la vista lateral del cuadricóptero navegando en un entorno 3D con un único obstáculo estático. En este escenario, el cuadricóptero parte de unas coordenadas y necesita alcanzar un punto objetivo, evitando el obstáculo estático.

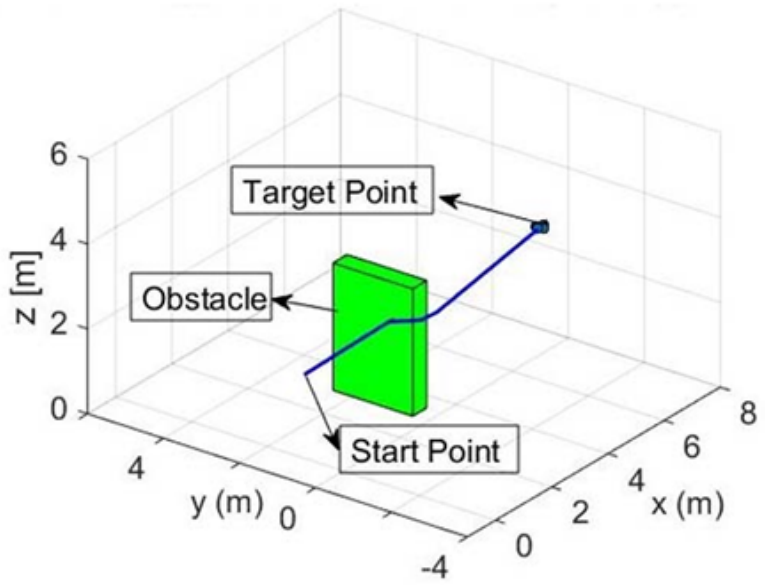


Figura 3.4: *Trayectoria de cuadricóptero en [44] evitando un obstáculo estático.*

El diseño de un controlador neuronal basado en aprendizaje por refuerzo es uno de los enfoques más avanzados. Se aplica al control de maniobras completas de aterrizaje y despegue, incluso cuando hay viento variable [15]. Asimismo, se ha explorado la aplicación de la optimización por cúmulo de partículas para ajustar los parámetros PID en el control de trayectoria y actitud de un hexarrotor UAV, con simulaciones numéricas que demuestran la efectividad de este enfoque en el seguimiento de trayectorias complejas. Este algoritmo optimiza los parámetros de control, mejorando la precisión y la eficiencia del sistema en el seguimiento de trayectorias tridimensionales [45]. También, en [46], se aborda el problema de planificación de trayectorias de un sistema de entrega con drones, mediante algoritmos de aprendizaje por refuerzo multiagente (*Multi-Agent Reinforcement Learning*, MARL), donde el dron recibe recompensas positivas por realizar entregas exitosas y penalizaciones por colisiones o acciones redundantes.

Las técnicas para la planificación de trayectorias son variadas en la literatura, y hablar de todas ellas en este documento puede tomar tiempo. Por ello, se presenta la Tabla 3.1 donde se presenta un resumen de diferentes categorías de control convencional, en las que se indican algunas técnicas y su descripción.

Tabla 3.1: Estado del arte, comparación de técnicas de control convencional.

Categoría	**Técnicas de control**	**Descripción general**
Control lineal clásico	- PID (Proporcional-Integral-Derivativo). - Control por realimentación. - Control por diferencia finita.	Técnicas basadas en sistemas lineales, ampliamente utilizadas, aunque presentan limitaciones ante no linealidades y perturbaciones fuertes [33].
Control de Estado	- Control por realimentación de estados. - Colocación de polos. - Observadores de estado.	Usa una representación en espacio de estados para diseñar el controlador [47].
Control Óptimo	- LQR (Linear Quadratic Regulator). - LQG (Linear Quadratic Gaussian).	Minimiza una función de costo cuadrática. Ideal para sistemas donde se busca minimizar el error de control [48].
Control Robusto Clásico	- H-infinito (H∞). - μ-síntesis.	Diseñados para mantener el rendimiento ante incertidumbres o perturbaciones [49].

También se presenta la Tabla 3.2 donde también se presentan categorías, técnicas de control y su descripción general. El propósito de esto es proporcionar un panorama general de lo hallado en el estado del arte durante la realización de esta investigación.

Tabla 3.2: Estado del arte, comparación de técnicas de control no convencional.

Categoría	**Técnicas de control**	**Descripción general**
Basados en lógica difusa	- Control difuso (Fuzzy Logic, Control, FLC)	Usa reglas "difusas"para manejar incertidumbres y decisiones aproximadas [50].
Basados en Redes Neuronales	- Controlador neuronal - Neurocontrol adaptativo	Usa redes neuronales para aprender dinámicamente cómo controlar un sistema [51].
Aprendizaje por Refuerzo (RL)	- Q-learning - DDPG (Deep Deterministic Policy Gradient) - PPO (Proximal Policy Optimization) - SAC (Soft Actor-Critic) - CACLA (Continuous Actor-Critic Learning Automaton)	Aprenden políticas óptimas a través de la interacción con el entorno y retroalimentación [52].

3.2. Tendencias de investigación

Si bien, se pueden emplear diversos métodos de control para la planificación de trayectorias en cuadricópteros, esta investigación se enfoca en dos: PID y el aprendizaje por refuerzo. Se reconoce que el control por PID continúa siendo ampliamente empleado para controlar la trayectoria, y también como referencia para nuevos algoritmos de control, particularmente aquellos que emplean algoritmos inteligentes. Es por ello que la tendencia del uso del PID se sigue manteniendo. Por otro lado, el uso de aprendizaje por refuerzo va tomando protagonismo en los últimos años. Por ello, a continuación se expone un análisis bibliométrico sobre el control de trayectoria utilizando algoritmos de aprendizaje por refuerzo.

3.2.1. Aprendizaje por refuerzo

Para esta investigación, se elige Scopus como la base de datos para la búsqueda. Se considera el período de tiempo comprendido entre 2018 y agosto del 2024. Se podría decir que el uso de métodos de control de aprendizaje por refuerzo en la planificación de rutas para cuadricópteros se ha incorporado recientemente.

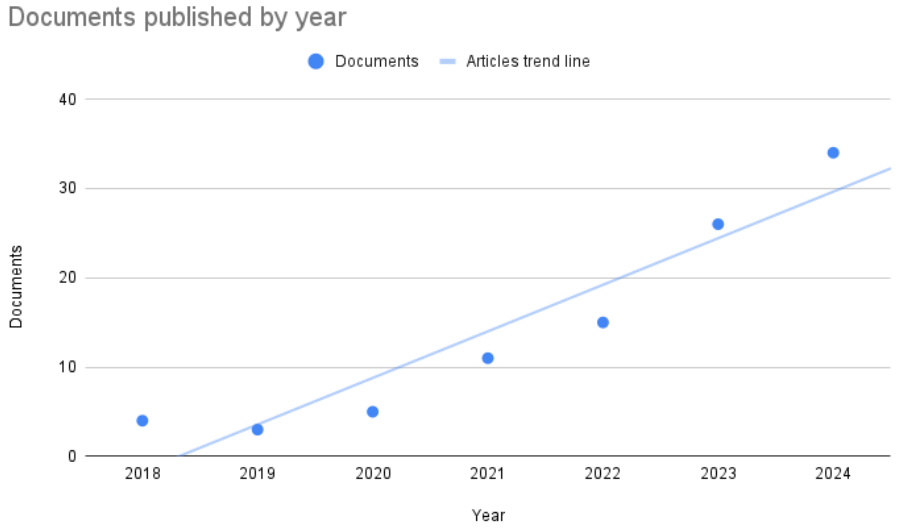


Figura 3.5: *Tendencia de publicaciones de documentos por año utilizando las palabras clave: reinforcement learning", "drone", "path planning", seleccionando la opción de búsqueda de palabras clave en título del artículo, resumen y palabras clave en Scopus.*

Según la Figura 3.5, las técnicas de aprendizaje por refuerzo para la planificación de rutas han ido en aumento en los últimos años, contando con la mayoría de publicaciones en 2024, con 34 publicaciones, seguido de 2023 con 26 publicaciones. En contraste, se publicaron menos de 10 documentos en los años 2018, 2019 y 2020. Sin embargo, la tendencia general indica un aumento incremental en las publicaciones en este tema.

Otra métrica bibliométrica es analizar documentos anuales publicados por fuente, lo cual ayuda a comprender cómo se distribuye y evoluciona el conocimiento en un área de investigación específica. Este análisis permite identificar las principales revistas y conferencias que aportan contribuciones significativas al campo de la planificación de rutas de cuadricópteros, así como detectar tendencias en la producción científica. Además, al examinar métricas de

impacto y calidad de cada fuente, como CiteScore, SJR y SNIP, se puede evaluar la influencia y relevancia de estas publicaciones dentro de la comunidad académica.

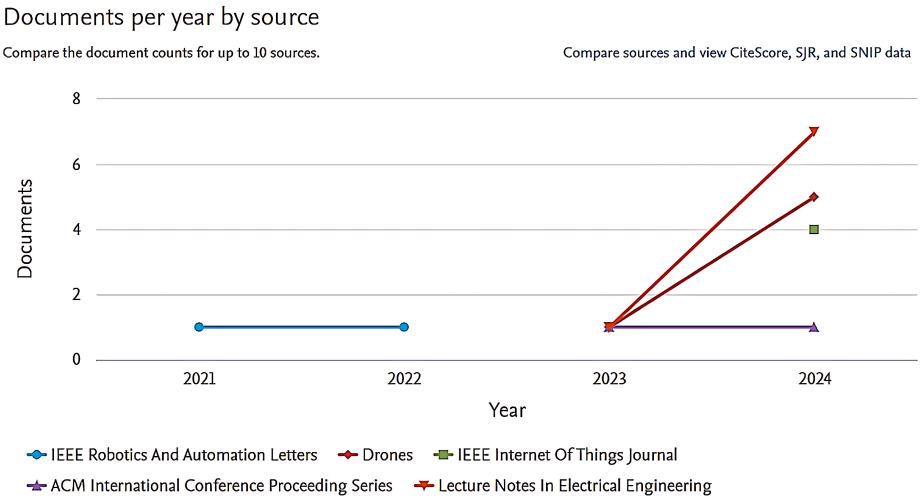


Figura 3.6: *Documentos anuales por fuente.*

En la Figura 3.6 se muestran los documentos anuales publicados por fuente, acotado a 5 fuentes en total. La fuente, Lecture Notes In Electrical Engineering cuenta con 8 publicaciones en total, una en 2023 y siete en 2024. Drones cuenta con 6 publicaciones en total, una en 2023 y cinco en 2024. IEEE Internet of Things cuenta con 4 publicaciones, todas en 2024. ACM International Conference Proceedings Series cuenta con 2 publicaciones, una en 2023 y otra en 2024. IEEE Robotics And Automation Letters cuenta también con 2 publicaciones, una en 2021 y otra en 2022. La gráfica también nos muestra las métricas de impacto y calidad de cada fuente en CiteScore, SJR y SNIP. CiteScore mide el promedio de citas por artículo en un período de 4 años y no pondera las citas. SJR mide la influencia de una revista basándose en la cantidad de citas ponderadas según la calidad de la revista citada. SNIP ajusta el impacto de las citas teniendo en cuenta las diferencias en las tasas de citación entre campos de investigación. Al hacer clic sobre cada fuente en la gráfica se muestran estás métricas, como en la imagen Figura 3.7.

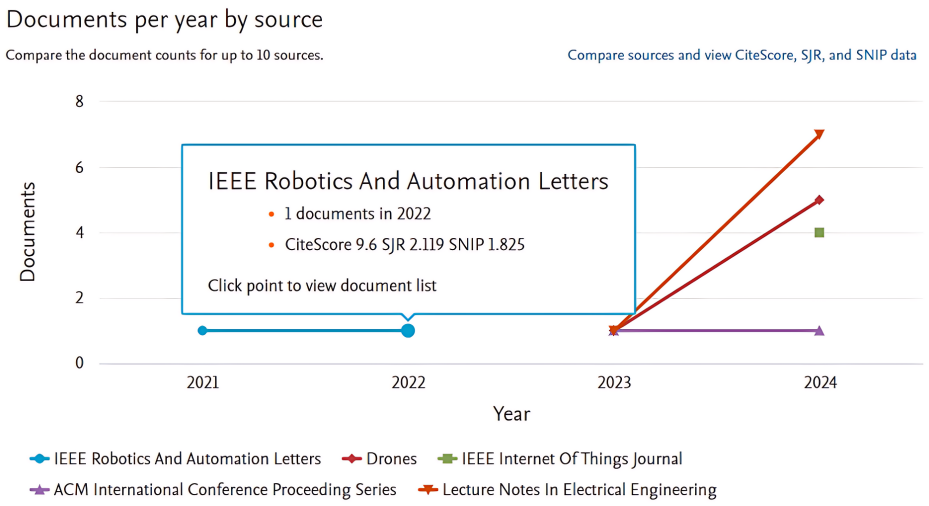


Figura 3.7: *Métricas de impacto y calidad de fuente IEEE Robotics And Automation Letters.*

De la Tabla 3.3 se observa que de todas las fuentes mencionadas, IEEE Internet of Things es una fuente altamente influyente con altas tasas de citación en las tres métricas a comparar.

Tabla 3.3: Comparativa de métricas de impacto con CiteScore, SJR y SNIP de fuentes.

Fuentes	**CiteScore**	**SJR**	**SNIP**
Lecture Notes In Electrical Engineering	0.7	147.000	145.000
Drones	5.6	0.760	1.509
IEEE Internet of Things	17.6	3.382	2.356
ACM International Conference Proceedings Series	1.5	253.000	233.000
IEEE Robotics And Automation Letters	9.6	2.119	1.825

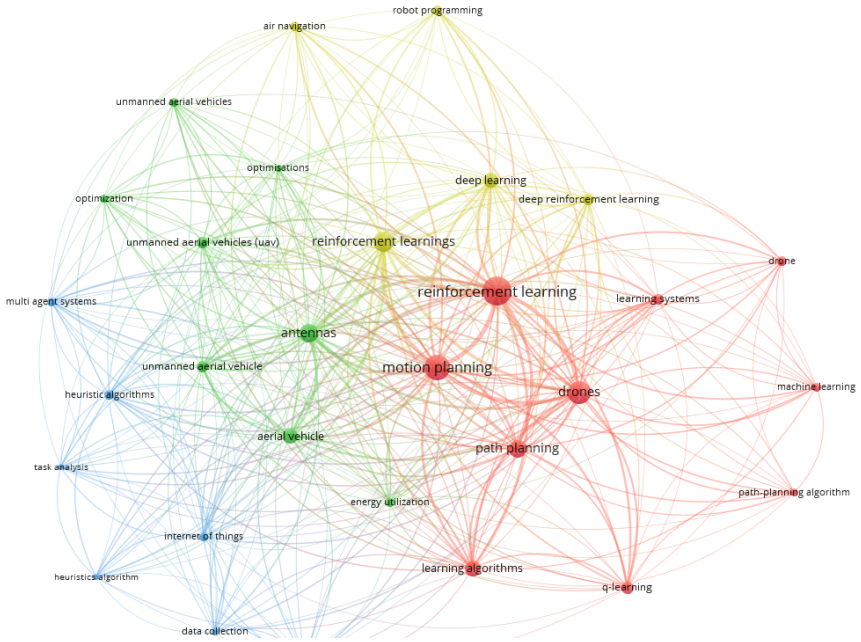


Figura 3.8: *Distribución de palabras clave.*

Además del análisis por fuente, es útil complementar la revisión bibliográfica mediante el estudio de relaciones temáticas, lo cual permite identificar tendencias y enfoques emergentes dentro del campo. En la Figura 3.8, el visor VOS (Visualization of Similarities) muestra las diferentes asociaciones con técnicas de aprendizaje por refuerzo para la planificación de rutas de drones en términos de palabras clave como aprendizaje por refuerzo, planificación de rutas, drones, drones autónomos, aprendizaje por refuerzo profundo, vehículos aéreos no tripulados, aprendizaje Q, aprendizaje automático, entre otros. Comprender estas relaciones es importante, ya que ayuda a detectar patrones, sinergias y lagunas en la bibliografía, orientando así la optimización de las técnicas y aumentando la eficacia de los próximos estudios.

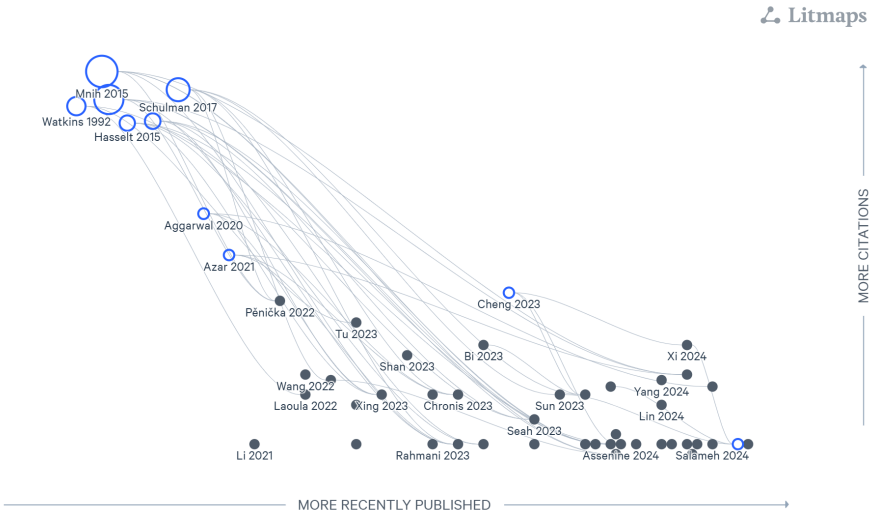


Figura 3.9: *Relaciones de citas entre artículos.*

Además, al centrarse en las conexiones directas entre las publicaciones científicas y su influencia dentro del campo, el análisis de las redes de citas ofrece otra perspectiva valiosa. En la Figura 3.9 se muestran los enlaces de citas entre 40 artículos, de los 98 de la búsqueda que arrojó Scopus. Litmap es una herramienta de visualización bibliométrica destinada a mostrar visualmente redes de citas y otros vínculos pertinentes entre publicaciones, facilitando así la investigación, el análisis y la comprensión de la literatura científica. Gracias a estas visualizaciones, los investigadores pueden identificar artículos clave, seguir la historia de los campos de estudio y descubrir tendencias emergentes. En este caso concreto, el mapa mostrado en Litmap organiza los artículos examinados en un paisaje académico estructurado, ilustrando cómo se conectan las publicaciones con los trabajos fundacionales.

Las esferas sólidas son los 40 artículos utilizados para el análisis en Litmap. Las esferas huecas representan los artículos recomendados por la aplicación. Las líneas muestran las referencias/citas entre artículos. El eje horizontal está dispuesto para destacar los artículos publicados recientemente, mientras que el eje vertical identifica los artículos más citados. Así, los artículos situados a la derecha son más recientes, y los de la parte superior tienen un mayor número de citas. Además, el artículo de la derecha hace referencia al de la izquierda. El tamaño de las esferas también se correlaciona con la frecuencia de citas, lo que subraya visualmente la prominencia de determinados estudios dentro del panorama académico cartografiado.

Capítulo 4

Materiales y métodos

En esta sección se detallan los recursos empleados para el desarrollo de las simulaciones de seguimiento de trayectorias del cuadricóptero, incluyendo el software utilizado para modelar y simular el comportamiento del sistema. Aunque en este punto de la investigación aún no se han obtenido resultados experimentales, se describen los materiales que se planean utilizar para la construcción del cuadricóptero en fases posteriores. Estos componentes físicos, como motores, baterías y estructuras, han sido considerados para obtener parámetros esenciales como peso, dimensiones, voltajes operativos y otros datos técnicos necesarios para una simulación precisa del comportamiento del cuadricóptero en entornos virtuales. Por esta razón, se incluyen aquí breves descripciones de estos elementos para contextualizar las condiciones bajo las cuales se desarrollan las simulaciones.

4.1. Selección de hardware

4.1.1. Marco de cuadricóptero S500

El S500 es un marco de cuadricóptero, así como se muestra en Figura 4.1. Es la estructura física principal que proporciona soporte y montaje para todos los componentes del dron. El S500 es popular entre los aficionados y desarrolladores de drones debido a su equilibrio entre costo, durabilidad y facilidad de montaje. Este marco proporciona una plataforma estable y

versátil para construir un cuadricóptero personalizado. Su diseño robusto y modular permite la integración de diversos componentes electrónicos y sensores, lo que lo convierte en una opción ideal para proyectos de investigación y desarrollo [20]. Además, el S500 ofrece suficiente espacio para alojar baterías de diferentes tamaños y capacidades, lo que permite ajustar la autonomía del vuelo según las necesidades específicas del proyecto [53].

Figura 4.1: *Armazón de cuadricóptero de fibra de vidrio S500[**20**].*

4.1.2. Motor DJI 2212

Para que el cuadricóptero pueda hacer todas sus tareas, se utilizan 4 motores, pero no de cualquier tipo, es necesario utilizar motores de tipo *brushless*. Los motores *brushless* son un tipo de motor eléctrico que no utiliza escobillas para conmutar la corriente del rotor. La correcta elección de motores y hélices es esencial para el desempeño de un cuadricóptero, dado que los motores constituyen los principales consumidores de energía de la batería [19]. En este trabajo, se emplean cuatro motores DJI 2212, mostrados en la Figura 4.2, los cuales son motores ampliamente utilizados en cuadricópteros debido a su eficiencia, durabilidad y capacidad para proporcionar un empuje adecuado para vuelos estables y maniobrables [54]. Estos motores son síncronos de corriente continua que reciben energía de una fuente eléctrica de corriente continua mediante un inversor integrado en la distribución de energía. Este flujo de corriente continua genera una corriente trifásica que excita las bobinas del motor. El rotor está constituido por imanes permanentes que se encuentran en movimiento cuando la corriente trifásica energiza el estator.

Figura 4.2: *Motor DJI 2212 [**42**].*

La Tabla 4.1 muestra una comparativa de parámetros eléctricos del motor en diferentes condiciones de voltaje y carga, proporcionados en el trabajo de Jahan et al. [42]. Incluye datos en condiciones sin carga y con carga, considerando voltajes de 11.1V, 14.8V y 18.5V.

Tabla 4.1: Comparativa de parámetros eléctricos de motor DJI 2212.

Voltaje [V]	**Sin carga**		**Con carga**				**Tipo de carga**
	Corriente [A]	**Velocidad [rpm]**	**Corriente [A]**	**Tracción/ Empuje [g]**	**Potencia [W]**	**EEP [%]**	**Batería**
11.1	0.3	10,200	8.5	600	94.4	6.4	3s Lipo/8045
14.8	0.3	13,610	13	940	192.4	4.9	4s Lipo/8045
18.5	0.4	17,000	17.2	1270	318.2	4.0	5s Lipo/8045

Los parámetros considerados son la corriente consumida por el motor sin aplicar ninguna carga, para ver reflejada su eficiencia en estado libre; la velocidad de giro sin carga, aumentando con el incremento del voltaje aplicado; la corriente consumida por el motor al soportar una carga específica, reflejando el consumo de energía en condiciones de vuelo real; la fuerza generada por el motor al operar bajo carga, permitiendo observar la capacidad de elevación del cuadricóptero; la potencia eléctrica consumida por el motor en condiciones de carga, calculada como el producto de corriente y voltaje; la eficiencia energética del motor, representada como el porcentaje de energía convertida en fuerza útil y la configuración de batería y hélice utilizada para alcanzar los valores especificados, que en este caso incluye combinaciones de baterías LiPo de 3, 4 y 5 celdas con hélices 8045.

4.1.3. Controlador electrónico de velocidad

El Controlador Electrónico de Velocidad (ESC) es un componente fundamental en los cuadricópteros, esto porque regulan la velocidad de los motores al controlar la cantidad de energía que se les suministra. Su función principal es convertir las señales de control que vienen del controlador de vuelo en señales eléctricas que pueden ser interpretadas por los motores *brushless* [19]. Básicamente, modula la potencia suministrada a los motores, permitiendo un control preciso de su velocidad, y por ende, del movimiento del cuadricóptero [19].

Figura 4.3: *Controlador electrónico de velocidad, modelo 40 A.*

Además, también incorporan características de seguridad importantes, como la protección contra sobrecorriente y sobretensión, lo que ayuda a prevenir daños en los motores y otros componentes electrónicos en caso de condiciones de funcionamiento anormales. Para este proyecto se pretende utilizar un ESC 40 A por cada motor, como el que se muestra en Figura 4.3.

4.1.4. Batería

Las baterías de polímero de iones de litio, comúnmente conocidas como LiPo, son ampliamente utilizadas en cuadricópteros debido a su alta densidad de energía y tasa de descarga [19]. Su excepcional densidad energética permite almacenar una gran cantidad de energía en un tamaño y peso reducidos [55]. Esta característica es crucial para maximizar el tiempo de

vuelo y la capacidad de carga útil de los cuadricópteros. Para la parte experimental de este proyecto se pretende utilizar la batería RC (Radio Control)de polímero de iones de litio (11.1 V, 30 C, 3000 mAh). Esta batería está diseñada para ser utilizada en modelos de control remoto, como coches, aviones o drones.

Figura 4.4: *Batería RC de polímero de iones de litio a 11,1 V 30 C 3000 mAh.*

4.1.5. Módulo GPS

El módulo de GPS UBLOX M8N (Figura 4.5) es un componente que permite en los cuadricópteros navegación precisa y funciones de geolocalización [56]. Este módulo se basa en la tecnología UBLOX M8N, que es un módulo GPS (Sistema de Posicionamiento Global) y GNSS (Sistema Mundial de Navegación por Satélite) avanzado, y gracias a que recibe señales de múltiples constelaciones de satélites, ofrece una alta sensibilidad y precisión en la recepción de señales GPS, lo que permite al cuadricóptero determinar su posición, velocidad y altitud con gran exactitud [57]. La incorporación de una brújula electrónica en el módulo GPS proporciona información adicional sobre la orientación del cuadricóptero, lo que facilita la navegación autónoma y la estabilización en vuelo.

Figura 4.5: *Módulo GPS UBLOX M8N con brújula para APM.*

La integración del módulo GPS UBLOX M8N con sistemas de control de vuelo, por ejemplo APM (ArduPilot Mega) y Pixhawk, permite a los cuadricópteros realizar tareas complejas, como el vuelo autónomo a través de puntos de referencia predefinidos, el seguimiento de objetos en movimiento y el regreso automático al punto de partida en caso de pérdida de señal o batería baja [20].

4.1.6. Módulo de potencia

El módulo de potencia, como el de la Figura 4.6, es un componente que utilizan los sistemas de control de vuelo, como Pixhawk y APM. Este módulo proporciona una medición precisa del voltaje y la corriente suministradas a los componentes electrónicos del cuadricóptero, lo que permite monitorear el consumo de energía y optimizar la gestión de la batería [20].

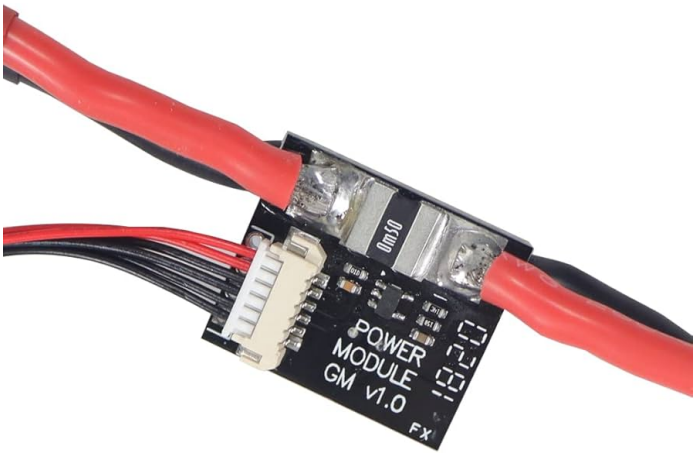


Figura 4.6: *Módulo de potencia del sensor amperímetro para Pixhawk y APM.*

Además de la medición de voltaje y corriente, el módulo de potencia integra un regulador de voltaje BEC (Circuito Eliminador de Batería), que proporciona una fuente de alimentación estable y regulada para los componentes electrónicos del cuadricóptero.

4.1.7. Codificador/decodificador PPM

El codificador PPM (Modulación por Posición de Pulso) es un dispositivo que convierte señales PWM (Modulación por Ancho de Pulso) en una señal PPM. Este componente facilita la conexión de receptores de radio control tradicionales al controlador de vuelo. El módulo convierte las señales PWM de hasta 8 canales del receptor en una única señal PPM, que puede ser interpretada directamente por el controlador de vuelo Pixhawk.

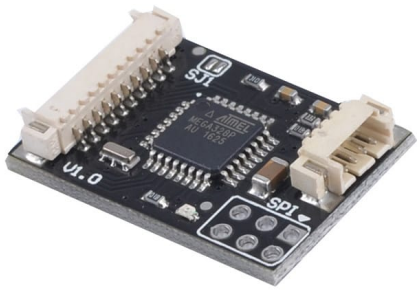


Figura 4.7: *Codificador/decodificador PPM*

Para la parte experimental de este proyecto, se utiliará el codificador PPM de la Figura 4.7.

4.1.8. Telemetría por radio Pixhawk

La telemetría por radio Pixhawk establece un enlace de comunicación confiable con el controlador de vuelo utilizado. Se destaca como uno de los controladores de diseño más rentables, proporcionando capacidades completas de control GPS. Este enlace permite la monitorización en tiempo real de diversos parámetros de vuelo, como la altitud, la velocidad, la ubicación y el estado de la batería.

Figura 4.8: *Controlador de vuelo Pixhawk PX4 2.4.8.*

Además, permite la transmisión de comandos desde una estación terrestre al cuadricóptero, lo que facilita el control remoto y la modificación de la trayectoria de vuelo en tiempo real. En sistemas más avanzados, esta telemetría puede ser crucial para la implementación de algoritmos de control adaptativo y para la supervisión del estado del vehículo aéreo no tripulado [20].

4.2. Selección de software

En esta sección se describe el entorno de simulación utilizado para el desarrollo y prueba de los algoritmos de control aplicados en la trayectoria del cuadricóptero. Se detalla la configuración del simulador, el cual permite modelar tanto el comportamiento dinámico del dron como las condiciones del entorno en escenarios tridimensionales. Además, se presentan los enfoques de planificación de trayectorias que se evalúan en esta investigación.

4.2.1. ROS

ROS 2 (*Robot Operating System*) es una plataforma de código abierto ampliamente utilizado en robótica para el desarrollo de aplicaciones complejas y distribuidas. Proporciona un conjunto de herramientas, bibliotecas y convenciones que facilitan la comunicación y coordinación entre diferentes componentes de un sistema robótico [22]. La arquitectura modular de ROS 2 permite la integración de diversos sensores, actuadores y algoritmos de control en un entorno unificado. Facilita la creación de sistemas robóticos modulares y escalables [22].

ROS permite construir sistemas robustos y adaptables a diferentes entornos y tareas [58]. En este trabajo, se utiliza ROS 2 (versión *Humble*) para implementar el sistema de control del cuadricóptero, lo que permite aprovechar las capacidades de procesamiento de una computadora para ejecutar los algoritmos de control y así realizar las tareas de percepción y planificación [59].

4.2.2. Gazebo

Gazebo es un simulador robótico de código abierto que ofrece un entorno virtual realista, como se muestra en Figura 4.9, para probar y validar algoritmos de control y planificación antes de su implementación en un sistema real. La capacidad de simular la dinámica del vuelo, el comportamiento de los sensores y las interacciones con el entorno es crucial para el desarrollo y la evaluación de sistemas robóticos aéreos [60].

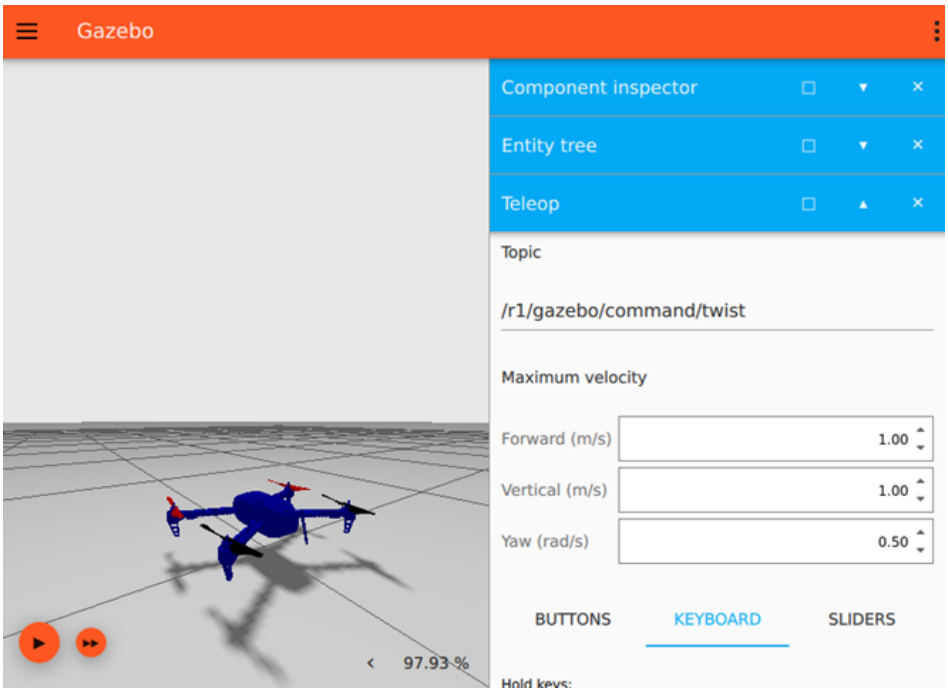


Figura 4.9: *Cuadricóptero en entorno Gazebo.*

Gazebo ofrece una amplia gama de modelos de robots, sensores y entornos, lo que facilita la creación de simulaciones personalizadas para diferentes aplicaciones. El uso de simuladores robóticos como Gazebo reduce el tiempo de prueba en campo y simplifica la depuración [61].

4.2.3. RViz

RViz (ROS Visualization) es una herramienta de visualización 3D que permite visualizar los datos generados por ROS 2 en tiempo real. RViz proporciona una interfaz gráfica intuitiva para mostrar información proveniente de sensores, como cámaras y LiDARs, así como datos de estado del robot, como la posición, la orientación y la velocidad.

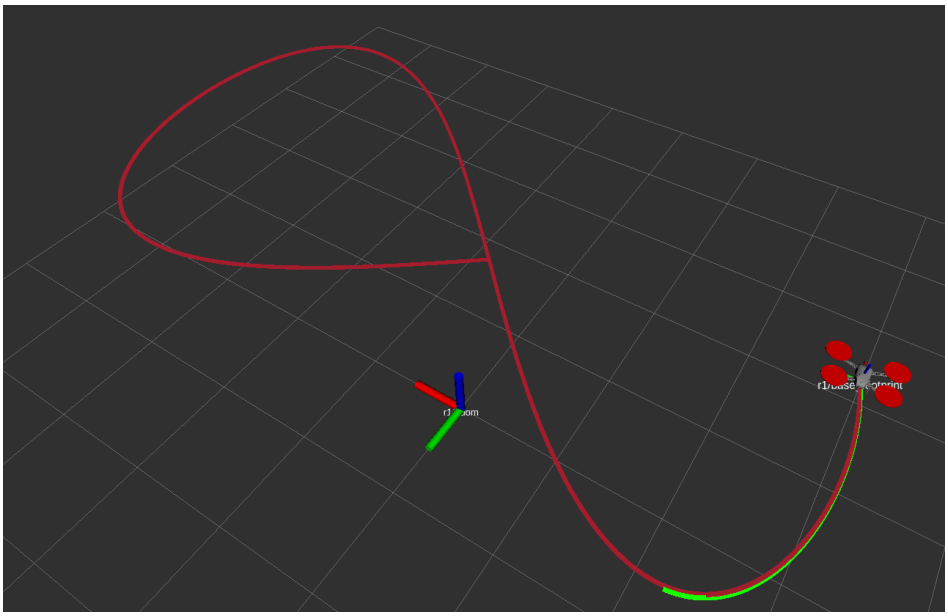


Figura 4.10: *Cuadricóptero siguiendo trayectoria de tipo lemniscata en RViz.*

En la Figura 4.10 se observa que RViz se utiliza para visualizar la trayectoria del cuadricóptero y la información del entorno en un entorno 3D interactivo.

4.3. Simulación del cuadricóptero

En esta sección se describen los componentes que permiten simular el comportamiento del cuadricóptero en Ignition Gazebo y ROS 2, así como su integración con RViz y los scripts de control. Cada subsección detalla un grupo de archivos y su función en el flujo global de la simulación.

4.3.1. Datos experimentales

Previo a generar los archivos para simular el cuadricóptero, se obtuvo experimentalmente el peso aproximado del dron pesando todos los elementos que serán parte de él. Los pesos se muestran en la Tabla 4.2. Un análisis del peso de todo el sistema garantiza que todos los componentes elegidos no excedan el empuje que los motores pueden producir [20]. Además, es fundamental equilibrar el peso del robot aéreo para lograr un vuelo estacionario estable [62].

El S500 tiene aproximadamente una capacidad de carga útil de 1.678 kg [19]. El peso del cuadricóptero se determina sumando cada componente, dando un total de 1.313 kg.

Tabla 4.2: Masa de los elementos que conforman el cuadricóptero.

Descripción	**Total [kg]**
Marco de cuadricóptero S500	0,576
Motor DJI 2212 (4 en total)	0,208
ESC 40 A (4 en total)	0,120
Batería RC de polímero de iones de litio	0,267
Módulo GPS, de potencia, codificador PPM	0,103
Pixhawk	0,039
TOTAL	**1,313**

4.3.2. Modelado de la dinámica y geometría

Para detallar la representación virtual del cuadricóptero se generó el archivo *quadcopter.sdf*. Este documento es fundamental para reproducir el comportamiento dinámico del S500 equipado con los motores DJI 2212, ESC 40A, la batería, GPS y demás periféricos. A través de sus parámetros, el simulador conoce las características de masa, inercia, posición de montaje y constantes de motor/hélice, esto para reflejar cuáles serían los resultados del cuadricóptero en la vida real. Este archivo describe en Ignition Gazebo el modelo físico considerando:

- **Inerciales y colisiones**:
  El link r1/base_link incluye la masa (1.313 kg) y el tensor de inercia realista, así como una geometría de colisión tipo caja ($0.47 \times 0.47 \times 0.15$ m).
- **Visuales**:
  Se carga la malla 3D iris.dae para la representación visual del fuselaje.
- **Plugins de motor**:
  Cuatro instancias (una por rotor) definen la dinámica de aceleración/decadencia de velocidad angular, la constante de motor y las velocidades máximas.
- **Plugin de control de velocidad**:
  Expone el tópico `gazebo/command/twist`, donde se envían comandos de velocidad lineal y angular para el dron.

- **Plugin de odometría**:
  Publica la posición y orientación en un tópico interno que luego se mapea a ROS 2.

La velocidad máxima del motor no se encuentra en la hoja de especificaciones, sin embargo, sí se especifica el KV del motor. KV es la constante que relaciona el voltaje aplicado con la velocidad de giro sin carga en revoluciones por minuto. A partir de ahí, se calcula la velocidad angular máxima teórica con la fórmula:

$$\omega_{\text{máx}} = \text{KV} \left[\frac{\text{rpm}}{\text{V}}\right] \times V_{\text{bat}} \times \frac{2\pi}{60} \tag{4.1}$$

Siendo que KV tiene un valor de 920 rpm/V y el voltaje de la batería es de 11.1 V, se tiene que:

$$\omega_{\text{máx}} \approx 920 \times 11.1 \times \frac{2\pi}{60} \approx 1068\text{rad/s} \tag{4.2}$$

Sin embargo, cuando se realizan simulaciones, es recomendable elegir un valor más bajo para tener en cuenta pérdidas de carga, resistencia interna, eficiencia real de las hélices y condiciones aerodinámicas que pueden reducir la velocidad máxima alcanzable bajo esfuerzo. Por ello, para la simulación se elige una velocidad máxima de 850 rad/s. El código del archivo *quadcopter.sdf* se encuentra en la sección de **Anexos**.

4.3.3. Definición visual para ROS 2

El archivo `r1.xacro` constituye la representación gráfica del cuadricóptero en el entorno de ROS 2 y tiene como finalidad principal generar, de manera parametrizada, el modelo URDF (*Unified Robot Description Format*) que consumirá el nodo `robot_state_publisher` para publicar el tópico `robot_description`. En este XACRO se definen tres propiedades (`package_name`, `length`, `radius`) que centralizan la ruta del paquete y las dimensiones de los cilindros. Al procesar este archivo, ROS expande el XACRO en un URDF completo, publicándolo como descripción del robot, lo que permite a RViz renderizar el dron con colores, texturas y formas realistas sin necesidad de incluir información de colisión o física.

4.3.4. Configuración del entorno simulado

Con el objetivo de crear un entorno de pruebas reproducible y controlar las variables del entorno, se define el archivo `my_custom_world.sdf`, definiendo en Gazebo Ignition el espacio virtual donde se probará el cuadricóptero, estableciendo tanto propiedades físicas del mundo como las herramientas gráficas e interactivas necesarias para el desarrollo y validación de los controladores.

Se especifica el motor de física con un paso de integración de 4 ms y un factor de tiempo real de 1.0, garantizando realismo y estabilidad en la simulación; además, se cargan los plugins de sensores para la renderización con Ogre2, y se configuran varios paneles GUI, incluyendo la vista 3D, los controles de reproducción, las estadísticas de simulación y un teleoperador que publica comandos de velocidad. Asimismo, se incluye un plano de suelo de 500 x 500 m con colisiones y una fuente de luz direccional que aporta sombreado y referencia espacial. Finalmente, el dron se inserta en la posición deseada mediante `<include><uri>model://r1</uri></include>`.

4.3.5. Integración y ejecución de la simulación

Para coordinar todos los componentes descritos en las secciones anteriores y poner en marcha la simulación de extremo a extremo, se emplea el archivo de lanzamiento `ign_world_launch.py` junto con los archivos de configuración de compilación `CMakeLists.txt` y `package.xml`. El archivo `ign_world_launch.py` sirve como punto de entrada: inicia Ignition Gazebo cargando el mundo definido en `my_custom_world.sdf`, arranca el puente entre los tópicos de Gazebo y ROS 2 para traducir `/r1/gazebo/command/twist` ↔ `/r1/cmd_vel` y `/model/r1/odometry` ↔ `/r1/odom`, publica el URDF parametrizado generado por `r1.xacro` a través de `robot_state_publisher` y lanza RViz con la configuración `one_drone.rviz`.

Por su parte, `CMakeLists.txt` especifica cómo instalar los scripts de Python, asegurando que tras ejecutar `colcon build` queden disponibles tanto los ejecutables como los *launches files* y los recursos (URDF, SDF, RViz, modelos). El archivo `package.xml` declara la metadata del paquete (nombre, versión) y las dependencias de compilación y de ejecución (*rclcpp*, *rclpy*, `geometry_msgs`, `tf2_ros`, *xacro*, `launch_ros`, etc.), de modo que ROS 2 pueda resolver automáticamente todos los paquetes necesarios al invocar `ros2 run` o `ros2 launch`. Juntos, permiten construir, instalar y lanzar la simulación de maenra reproducible y sencilla, inte-

grando modelos, controladores y herramientas de visualización en un único flujo de trabajo.

4.3.6. Simulación y monitoreo en RViz

El archivo de configuración `one_dron.rviz` define la interfaz de RViz utilizada para supervisar en tiempo real la simulación del cuadricóptero, facilitando tanto la inspección de la pose y la trayectoria como la interacción manual con el sistema. Aquí se establece como **fixed frame** el tópico `r1/odom`, de manera que todos los elementos visuales se referencian al marco de odometría del dron, garantizando coherencia espacial. Se habilita una malla de referencia, los ejes y nombrs de los *frames*, el modelo URDF y dos trazas de camino (color verde para la trayectoria real y rojo para la trayectoria deseada). Este archivo permite cargar una vista predefinida de RViz con un solo comando: `ros2 launch my_uavs ign_world_launch.py`.

4.3.7. Control clásico PID y registro de datos

Así como en Pinto et. al. [63], en esta investigación se implementa un controlador PID que coordina el despegue, seguimiento de trayectoria y aterrizaje del cuadricóptero, pero todo desde scripts de Python y C++. Paralelamente, se registran de manera sistemática las variables de estado y de mando para su posterior análisis. El nodo Python se suscribe al tópico **r1/odom** para leer en tiempo real la posición (x,y,z), la velocidad lineal (v_x,v_y,v_z) y la orientación (**guiñada**) del cuadricóptero, y publica comandos de velocidad en `r1/cmd_vel`, esto tras el remapeo realizado por el puente ROS-Ignition.

Se eligieron y programaron tres trayectorias para que el cuadricóptero siguiera: lemniscata, circular y aleatoria. Para la programación de las trayectorias, primero se definen las ecuaciones que representan a cada una. La trayectoria deseada de tipo lemniscata se define mediante las siguientes ecuaciones paramétricas:

$$X_d(t) = X_0 + a\sin(\omega t), \quad Y_d(t) = Y_0 + b\sin(2\omega t), \quad Z_d(t) = Z_0 + c\sin(\omega t) \tag{4.3}$$

donde $X_0 = 0$, $Y_0 = -0.5$, $Z_0 = 1.5$, $a = 3$, $b = 1.5$, $c = 0.5$ y $\omega = 2\pi/T$ con $T = 100$. A partir del error de posición $e = (x - X_d, y - Y_d, z - Z_d)$, su integral y su derivada, se calcula

la acción de control

$$U = \dot{\mathbf{X}}_d - \left(K_p \mathbf{e} + K_i \int \mathbf{e} dt + K_d \frac{d\mathbf{e}}{dt} \right) \tag{4.4}$$

y, tras rotarlo al marco de referencia del dron, se publican las velocidades lineales y angulares del cuadricóptero.

Para programar la trayectoria circular, se definen sus ecuaciones paramétricas:

$$\begin{aligned} X_d(t) &= R\cos(\omega t), \quad R = 1.5 \text{ m}, \\ Y_d(t) &= R\sin(\omega t), \\ Z_d(t) &= 1.5 + c\sin(\omega t), \quad c = 0.5 \text{ m}. \end{aligned} \tag{4.5}$$

donde ω y T conservan el mismo valor que en la lemniscata, y donde también se publican las velocidades lineales y angulares del cuadricóptero.

Para la trayectoria aleatoria se programaron 5 tramos aleatorios. La trayectoria se guardó y conta de una recta de 2.5 m, un giro de 90° contra las manecillas del reloj, un tramo en zig-zag, un giro de 45° contra las manecillas del reloj y un retorno al inicio en forma de curvas sinoidales.

Para ilustrar la relación de los archivos de simulación, se muestra el diagrama de flujo de la Figura 4.11, donde se visualiza desde el lanzamiento de la simulación hasta la visualización y el control PID, señalando qué archivos intervienen en cada bloque y qué tópicos ROS 2 circulan entre ellos.

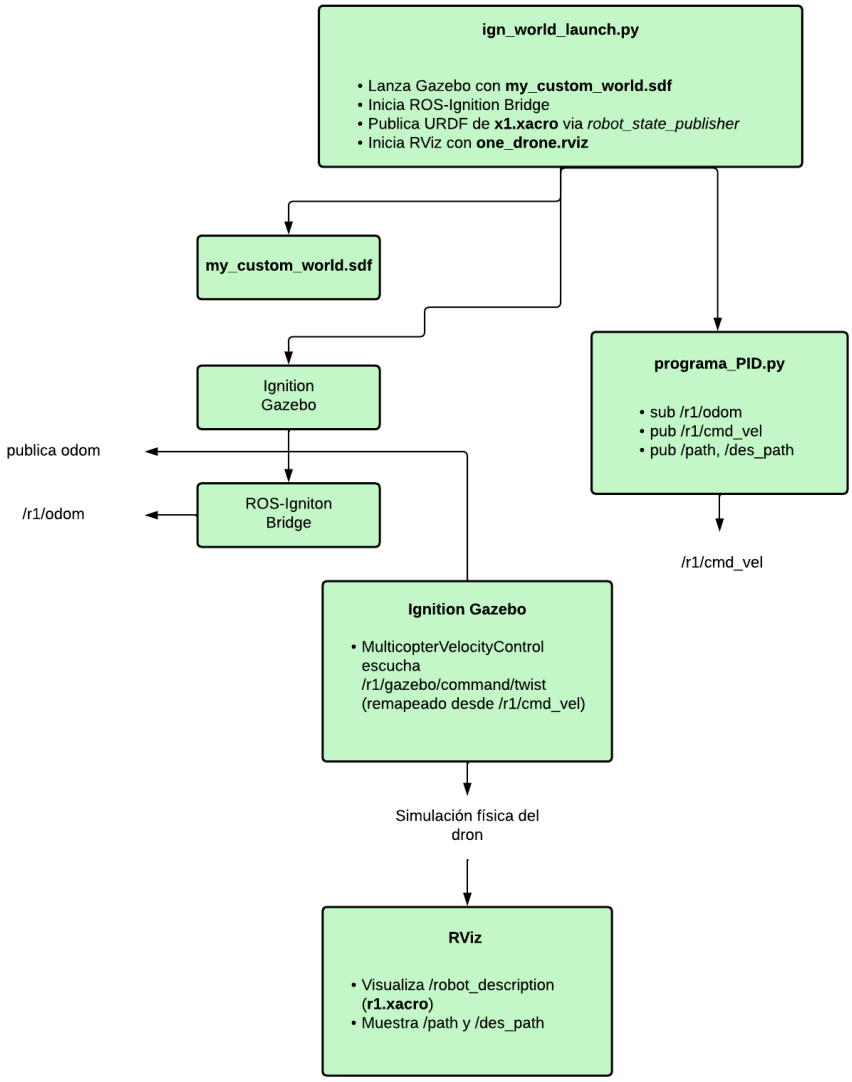


Figura 4.11: *Diagrama a bloques de simulación del cuadricóptero con control clásico PID.*

4.3.8. Control por aprendizaje por refuerzo

Mediante un script de Python se implementa un agente de aprendizaje por refuerzo basado en el algoritmo SAC (Soft Actor-Critic) con el objetivo de entrenar un agente capaz de controlar el cuadricóptero, en un entorno simulado. El entrenamiento consiste en la elevación del cuadricóptero a 1.2 m sobre el suelo y permanecer lo más cerca posible al punto (0,0,1.2), lo que se conoce como *hover*.

Para esta sección, se desarrolló un script de entrenamiento en Python. El script implementa un ciclo de entrenamiento de aprendizaje por refuerzo, con SAC. La relación de las plataformas y archivos se desglosan en forma de diagrama de bloques en la Figura 4.12.

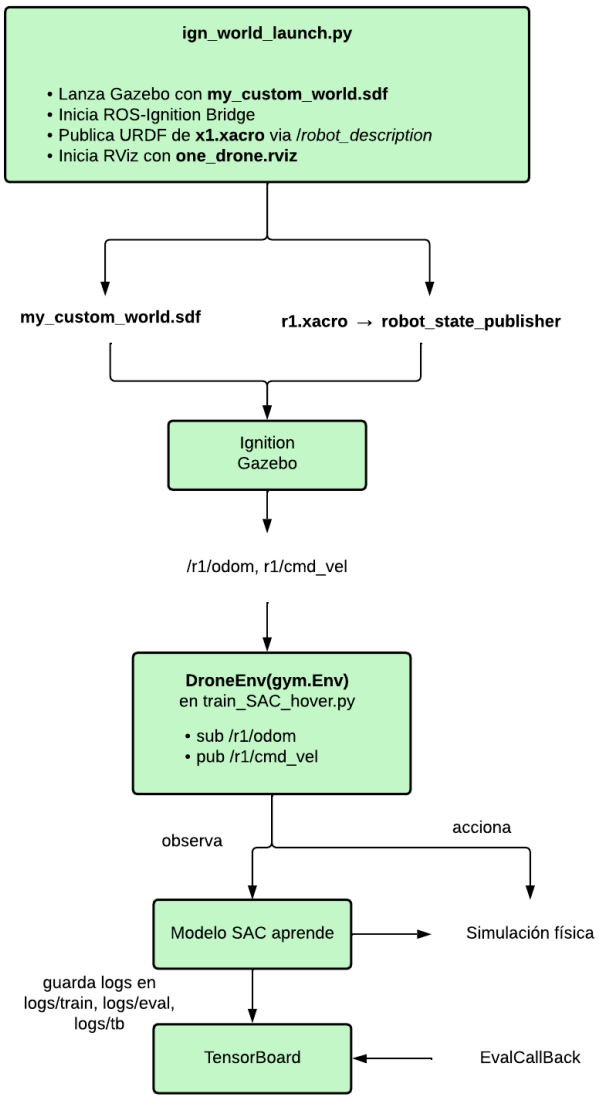


Figura 4.12: *Diagrama a bloques de entrenamiento con SAC para que el cuadricóptero aprenda a realizar hover.*

Capítulo 5

Resultados experimentales

En este capítulo se presentan los resultados obtenidos a partir de las simulaciones implementadas con dos enfoques de control: el método clásico PID y el controlador basado en aprendizaje por refuerzo, SAC.

5.1. Resultados de la simulación con PID

Para los resultados de la simulación con PID, primero se definieron los valores de las ganancias K_p, K_i y K_d para cada uno de los controladores PID de posición y actitud. Las trayectorias que se implementaron son de tipo lemniscata, circular y aleatoria. Para cada una de estas trayectorias se aplican los mismos sets de ganancias en el PID. Desde el script de control de Python, se crearon tablas y se guardaron datos de tiempo, posición deseada (x,y,z), posición real (x,y,z), y errores de posición (x,y,z). Todo se guarda en un archivo de tipo csv, que posteriormente se utilizó en otro script de Pyhton para calcular las métricas de evaluación. En las siguientes subsecciones se explican dichas métricas.

5.1.1. Error medio, ME

El error medio (*mean error*) es el valor promedio del error signado a lo largo de la trayectoria. Mide la tendencia central de la diferencia (con signo) entre la posición deseada y la real.

$$\mathrm{ME} = \frac{1}{N}\sum_{i=1}^{N} e_i \tag{5.1}$$

donde N es el número total de muestras en la simulación, e_i es el error en la muestra i, que se obtiene mediante la resta del valor real menos el valor deseado. Esta métrica indica si existe un sesgo permanente (por ejemplo, tendencia a quedarse atrás o pasarse del punto deseado). Un ME cercano a cero sugiere que, en promedio, el cuadricóptero no acumula desviación en una dirección preferente.

5.1.2. Error absoluto medio, MAE

Promedio de la magnitud del error, sin tomar en cuenta su signo. Captura la desviación típica en posición sin cancelaciones.

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N} |e_i| \tag{5.2}$$

Esta métrica evalúa directamente cuánta distancia, en promedio, se aleja el cuadricóptero de la trayectoria planeada, sea por defecto o por exceso.

5.1.3. Error cuadrático medio, RMSE

Cuantifica la dispersión de los errores alrededor de cero, dando mayor peso a los errores más grandes.

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} e_i^2} \tag{5.3}$$

donde e_i^2 es el error al cuadrado en la muestra i. Esta métrica destaca picos de error pronunciados, que pueden provenir de colisiones o fuertes desvíos repentinos.

5.1.4. Indice de error absoluto integral, IAE

Suma discreta del valor absoluto del error multiplicado por el intervalo de tiempo. Aproxima el área bajo la curva de $|e(t)|$.

$$\text{IAE} = \sum_{i=1}^{N} |e_i| \, \Delta t_i \tag{5.4}$$

donde,

- $\Delta t_i = t_i - t_{i-1}$: paso de tiempo entre muestras.
- $|e_i|$: valor absoluto del error en la muestra i.

Esta métrica mide el error acumulado a lo largo del tiempo, dando el mismo peso a todos los errores, independientemente de su magnitud o signo.

5.1.5. Overshoot máximo

El *overshoot* es un fenómeno que se manifiesta como una oscilación antes de estabilizarse, lo que puede afectar la precisión y la suavidad de la trayectoria. Se define como el valor máximo que alcanza el error (con signo) por encima de la referencia deseada antes de estabilizarse.

$$\text{Overshoot} \ = \max_i (e_i) \qquad \text{o bien} \qquad \max_i |e_i| \tag{5.5}$$

Un overshoot bajo es deseable para seguir rutas precisas sin rebasar excesivamente el punto objetivo.

5.1.6. Variabilidad

La variabilidad mide la dispersión de los errores alrededor de su valor medio. Cuantifica qué tan inestable es el seguimiento.

$$\sigma_e = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(e_i - \bar{e})^2} \quad \text{donde} \quad \bar{e} = \text{ME} \tag{5.6}$$

La variabilidad en el cuadricóptero se manifiesta como pequeñas oscilaciones o desviaciones alrededor de la trayectoria deseada, lo que puede ser causado por factores como el ruido en los sensores, las perturbaciones externas o la propia dinámica del sistema. Un dron con baja variabilidad mantiene un seguimiento más suave y predecible.

5.1.7. Resultados con trayectoria lemniscata

La trayectoria lemniscata, que se muestra en la Figura 5.1, también conocida como el símbolo de infinito, presenta un desafío significativo para el control del cuadricóptero debido a su forma compleja y sus cambios de dirección.

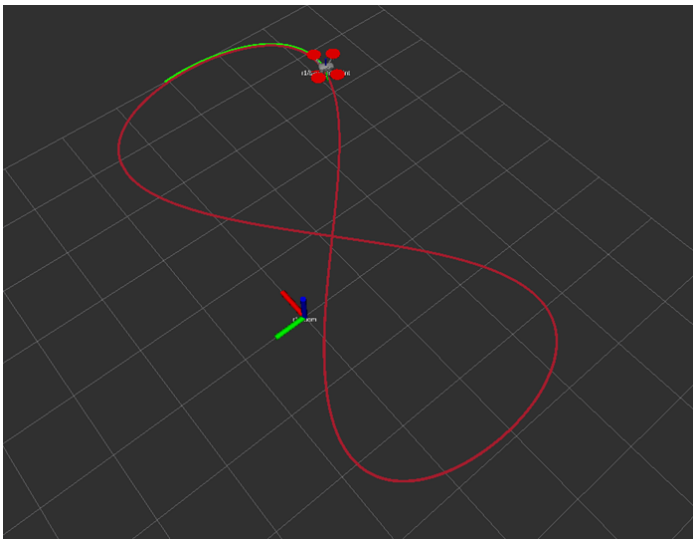


Figura 5.1: *Trayectoria deseada (rojo) vs. trayectoria real (verde) en RViz.*

Mantener la estabilidad y la precisión en los puntos de inflexión de la lemniscata requiere un ajuste preciso de los parámetros del controlador PID y una capacidad de respuesta rápida a los cambios en la dinámica del sistema.

Se realizaron varias simulaciones con diferentes tipos de ganancias PID, con el propósito de

observar el comportamiento del cuadricóptero. Después de las observaciones, se eligieron 5 sets de ganancias PID, cercanas entre sí, para encontrar cuál de estas es la mejor opción. En la Tabla 5.1 se presentan las métricas por PID aplicado al seguimiento de trayectoria de la lemniscata.

Tabla 5.1: Resultados de métricas para trayectoria lemniscata.

Ganancias PID	ME	MAE	RMSE	IAE	Overshoot	Variabilidad
Kp=0.5, Ki=0.1, Kd=0.5	0.105413	0.105413	0.186535	11.030468	1.165005	0.153912
Kp=0.8, Ki=0.05, Kd=0.1	0.058152	0.058152	0.133666	5.795062	1.163799	0.120367
Kp=1.0, Ki=1.0, Kd=1.0	0.075004	0.075004	0.175895	7.579283	1.165207	0.159120
Kp=2.0, Ki=0.05, Kd=0.5	0.039016	0.039016	0.123004	4.693652	1.162597	0.116665
Kp=2.0, Ki=1.0, Kd=1.0	0.043143	0.043143	0.138554	4.847465	1.162749	0.131681

Para la navegación de cuadricópteros es crucial evitar grandes errores instantáneos. Si se toma como criterio principal el **RMSE**, que penaliza fuertemente grandes desviaciones, el mejor desempeño lo da el PID con $K_p = 2$, $K_i = 0.05$, $K_d = 0.5$ con un **RMSE** de 0.1230, siendo este el error cuadrático medio más pequeño. También se observan errores bajos en **ME** y **MAE**, muy proximos a 0.04, lo que indica que el dron sigue la trayectoria sin desviarse. El **IAE** reducir también refleja que el error acumulado a lo largo de toda la trayectoria es bajo, con 4.69. El *Overshoot*, de 1.1626, presenta un pico del 16 % por encima de la trayectoria objetivo, lo que apunta a un sistema estable sin oscilaciones exageradas. La variabilidad se puede considerar también baja, con un 0.1167, siendo observable que la respuesta del error cuenta con poca dispersión.

Haciendo uso de los datos guardados de la simulación, se pueden gráficar las posiciones

deseadas contra las posiciones reales del cuadricóptero para visualizar el comportamiento en x, y, z. Primero se grafican los datos de posición y error de posición del PID con mejor rendimiento. En la Figura 5.2 se muestra la posición deseada contra la posición real en X del cuadricóptero respecto al tiempo, donde se observa que en este eje hay muy poco margen de error.

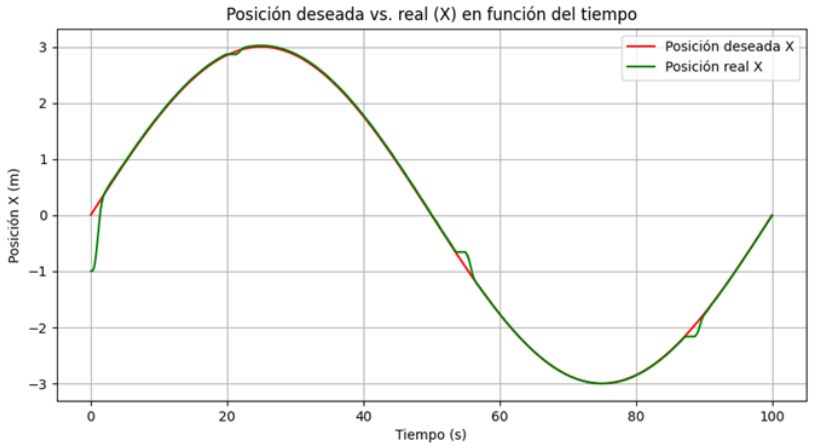


Figura 5.2: *Trayectoria deseada (color rojo) y trayectoria del cuadricóptero (color verde) en X, mejor caso.*

En la Figura 5.3 se muestra la posición deseada contra la posición real en Y del cuadricóptero respecto al tiempo, donde se observa que en lo general el cuadricóptero sigue sin problemas la trayectoria en este eje.

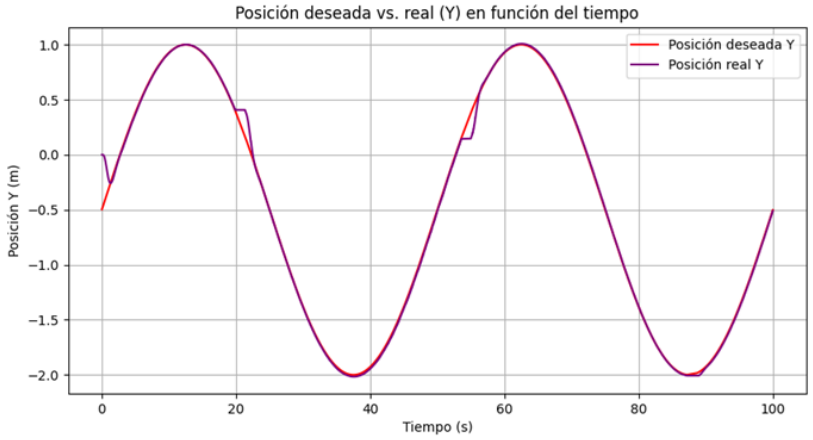


Figura 5.3: *Trayectoria deseada (color rojo) y trayectoria del cuadricóptero (color morado) en Y, mejor caso.*

En la Figura 5.4 se muestra la posición deseada contra la posición real en Z del cuadricóptero respecto al tiempo. En los primeros instantes se observa que el cuadricóptero se va corrigiendo en el despegue para seguir la trayectoria de tipo lemniscata. En lo general, el cuadricóptero sigue se comporta adecuadamente.

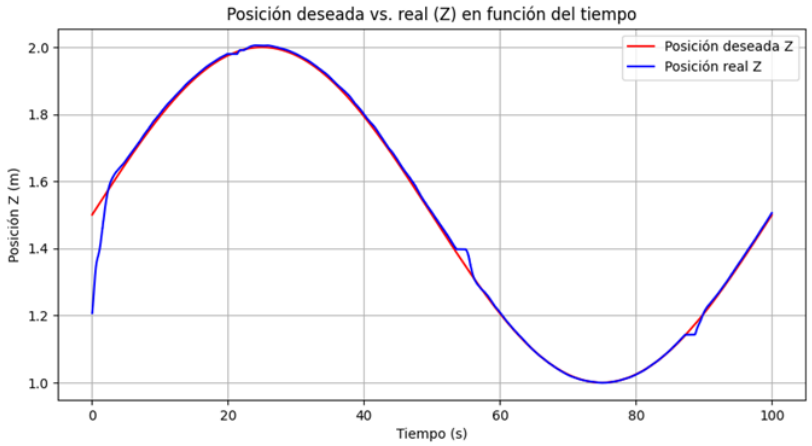


Figura 5.4: *Trayectoria deseada (color rojo) y trayectoria del cuadricóptero (color azul) en Z, mejor caso.*

En la Figura 5.5 se grafican los errores de posición de x, y y z. Se observan errores de posición menores a 0.25 m, los cuales son pequeños y coinciden con el cambio de dirección en la lemniscata.

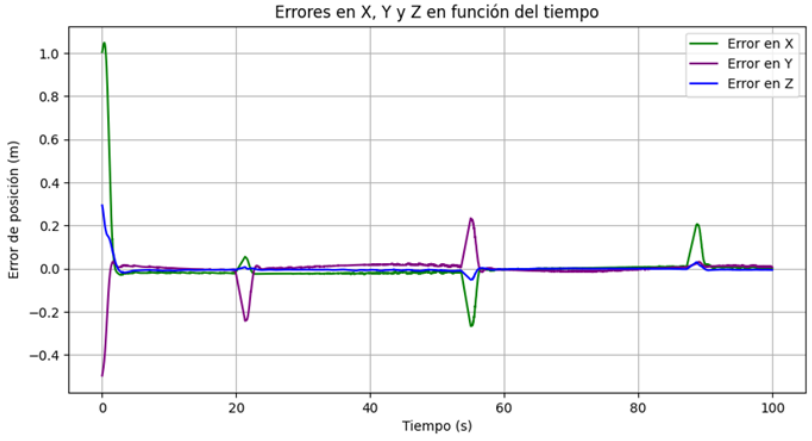


Figura 5.5: *Errores de posición en x,y,z de la trayectoria de tipo lemniscata, mejor caso.*

Siguiendo la misma línea de pensamiento, el controlador con el peor desempeño es el PID

con $K_p = 0.5$, $K_i = 0.1$, $K_d = 0.5$, ya que cuenta con un **RMSE** de 0.1865, siendo este el que peor desempeño presenta. Cuenta con un error medio elevado, ≈ 0.10, lo que significa que el cuadricóptero se aleja con más frecuencia de la trayectoria de tipo lemniscata. El **IAE**, de 11.03, es un valor muy alto a comparación de los demás, lo que nos dice que el error se acumula de forma significativa, evidenciando un mal seguimiento. El *Overshoot* es similar a la opción de mejor desempeño, pero la combinación de ganancias en K_i y K_p hace que la corrección sea demasiado lenta, manteniendo el error residual. Respecto a la variabilidad, esta es mayor, con 0.1539; por lo que la respuesta del sistema es más inestable, con picos de error menos predecibles.

A continuación, se muestran tres gráficas de la posición deseada vs. la posición real, respecto al tiempo, de x, y y z, pero ahora de los datos obtenidos del conjunto de ganancias con peor desempeño.

En la Figura 5.6 se grafica la posición deseada contra la posición real del dron en X respecto al tiempo. Al inicio se observa que al dron le cuesta un poco seguir la trayectoria, pero a mitad de la simulación se observa que sigue la trayectoria sin problemas.

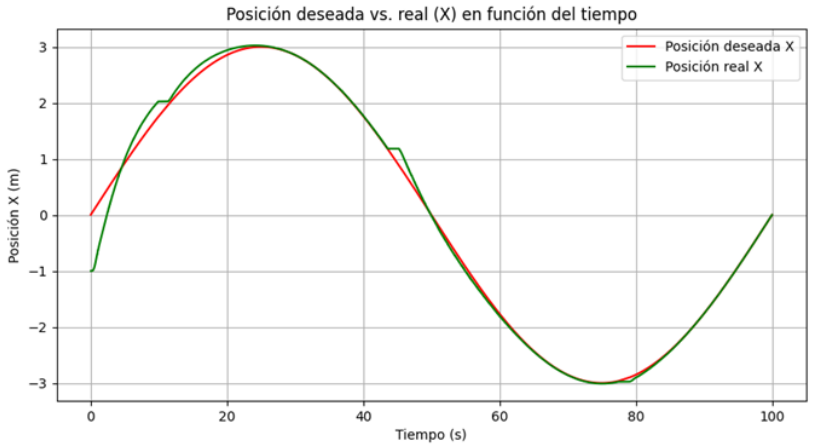


Figura 5.6: *Trayectoria deseada (color rojo) y trayectoria del cuadricóptero (color verde) en X, peor caso.*

En la Figura 5.7 se observa que los desvíos son evidentes en los cambios de dirección que debe hacer el dron al seguir la lemniscata.

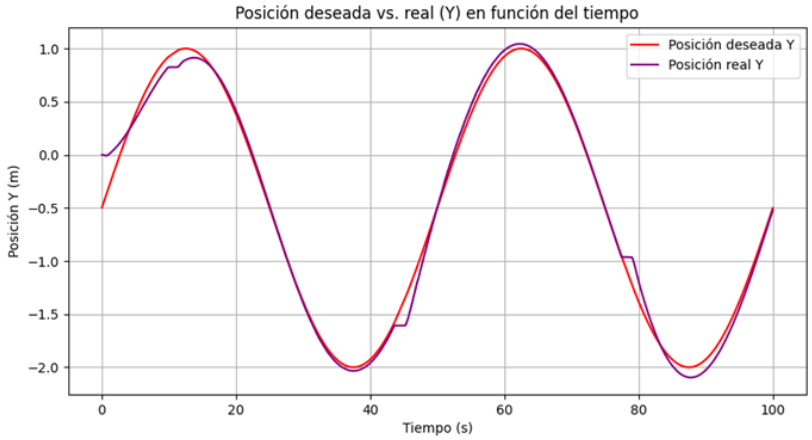


Figura 5.7: *Trayectoria deseada (color rojo) y trayectoria del cuadricóptero (color morado) en Y, peor caso.*

En la Figura 5.8 se observa que al cuadricóptero le cuesta al inicio seguir la trayectoria de la lemniscata, pero después el dron sigue la trayectoria sin mayor problema.

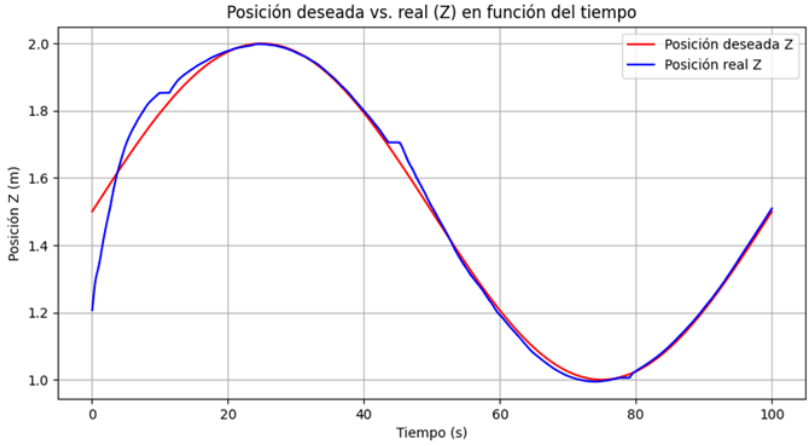


Figura 5.8: *Trayectoria deseada (color rojo) y trayectoria del cuadricóptero (color azul) en Z, peor caso.*

En la Figura 5.9 se muestran los errores de posición en x, y, z del PID con peor desempeño. Después del despegue, estos errores son menores a la unidad, pero tienen mayor ocurrencia a lo largo del vuelo.

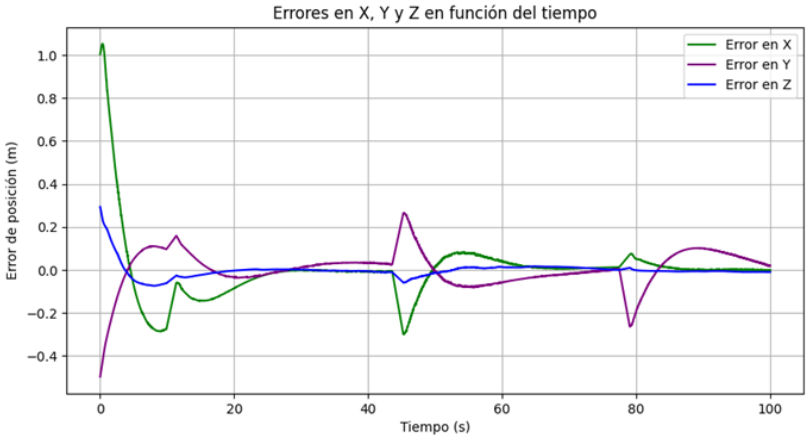


Figura 5.9: *Errores de posición en x,y,z de la trayectoria de tipo lemniscata, peor caso.*

5.1.8. Resultados con trayectoria circular

En esta sección se presentan las métricas cuantitativas obtenidas a partir de las simulaciones en las que el cuadricóptero sigue una trayectoria de tipo circular, como en la Figura 5.10, bajo el control de un PID.

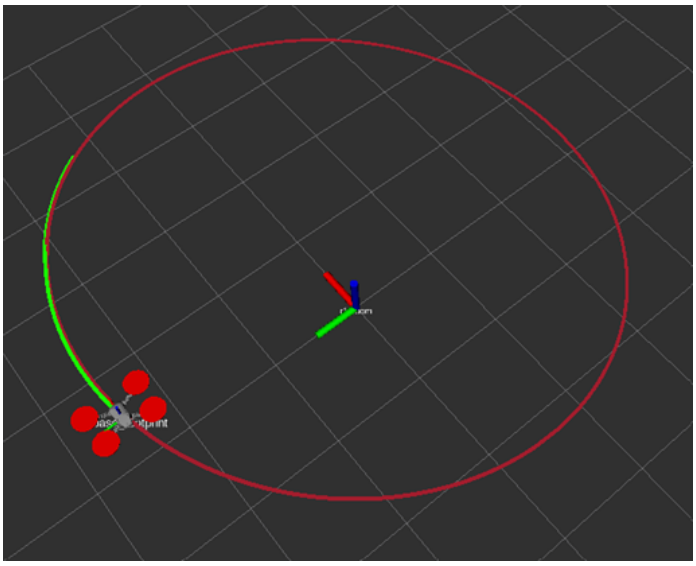


Figura 5.10: *Trayectoria deseada (rojo) vs. trayectoria real (verde) en RViz.*

La Tabla 5.2 resume los indicadores de desempeño para las cinco combinaciones de ganancias PID que se utilizaron en la simulación del seguimiento de trayectoria de tipo circular.

Tabla 5.2: Resultados de métricas para trayectoria circular.

Ganancias PID	ME	MAE	RMSE	IAE	Overshoot	Variabilidad
Kp=0.5, Ki=0.1, Kd=0.5	0.236710	0.236710	0.552209	23.537155	3.512355	0.498959
Kp=0.8, Ki=0.05, Kd=0.1	0.164330	0.164330	0.469330	15.974981	3.512489	0.439670
Kp=1.0, Ki=1.0, Kd=1.0	0.288820	0.288820	0.722399	27.360631	3.512488	0.662225
Kp=2.0, Ki=0.05, Kd=0.5	0.146480	0.146480	0.456818	15.614769	3.512287	0.432746
Kp=2.0, Ki=1.0, Kd=1.0	0.176665	0.176665	0.580426	17.995416	3.512353	0.552950

El set de ganancias con mejor desempeño, según su **RMSE**, es $K_p = 2.0$, $K_i = 0.05$, $K_d = 0.5$. El cuadricóptero con este ajuste presenta el menor **RMSE**, lo que indica una mayor precisión global en el seguimiento del círculo. Los valores bajos de **ME** y **MAE** reflejan desviaciones promedio reducidas. Además, el **IAE** es el mas pequeño (15.61), lo que implica que el error acumulado a lo largo de toda la trayectoria es menor. Aunque el *Overshoot* se mantiene alrededor de 3.5 %, su variabilidad es la más baja, señal de una respuesta consistente y estable, son oscilaciones pronunciadas tras las correcciones.

A continuación, se muestran las gráficas de la posición deseada vs. la posición real, respecto al tiempo, de x, y y z y la gráfica de error de posición para el PID con mejor desempeño.

En la Figura 5.11 se observa cómo al inicio el cuadricóptero en su despegue inicia a corregirse para intentar seguir la trayectoria circular en X, dando como resultado un seguimiento de trayectoria aceptable en este eje.

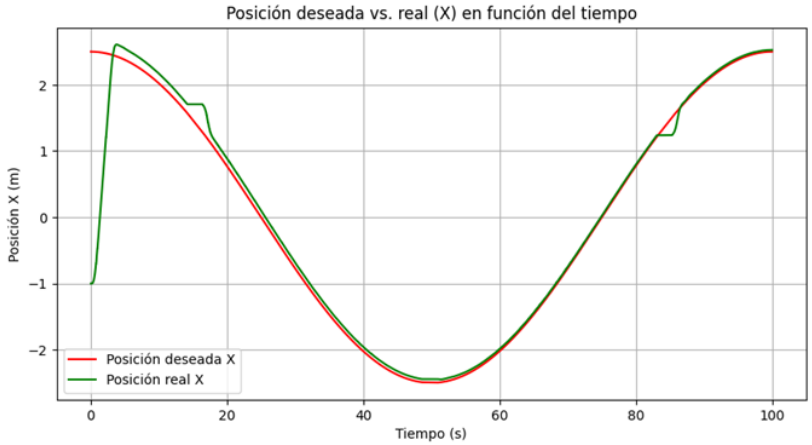


Figura 5.11: *Trayectoria deseada (color rojo) y trayectoria del cuadricóptero (color verde) en X, mejor caso.*

En la Figura 5.12 se grafica la trayectoria deseada y la trayectoria real en Y del cuadricóptero. Se observa que en este eje el dron sigue la trayectoria sin problemas, presentando pequeñas desviaciones.

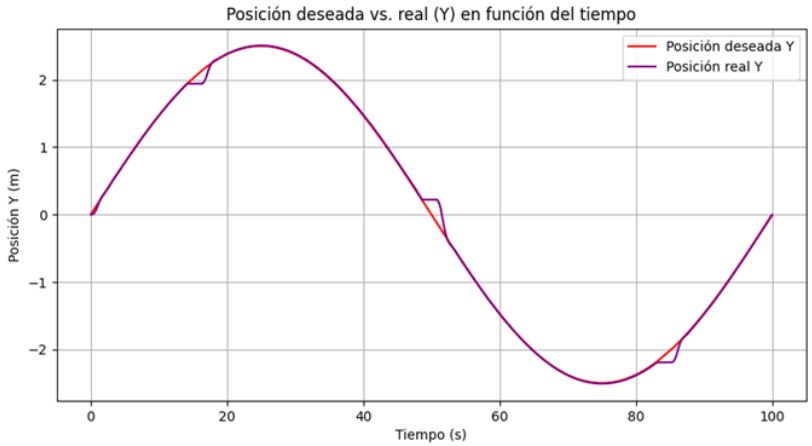


Figura 5.12: *Trayectoria deseada (color rojo) y trayectoria del cuadricóptero (color morado) en Y, mejor caso.*

En la Figura 5.13 se observa que la trayectoria deseada y la trayectoria real en Z del cuadricóptero tiene pequeñas desviaciones, pero en lo general, el dron sigue la trayectoria sin mayor problema.

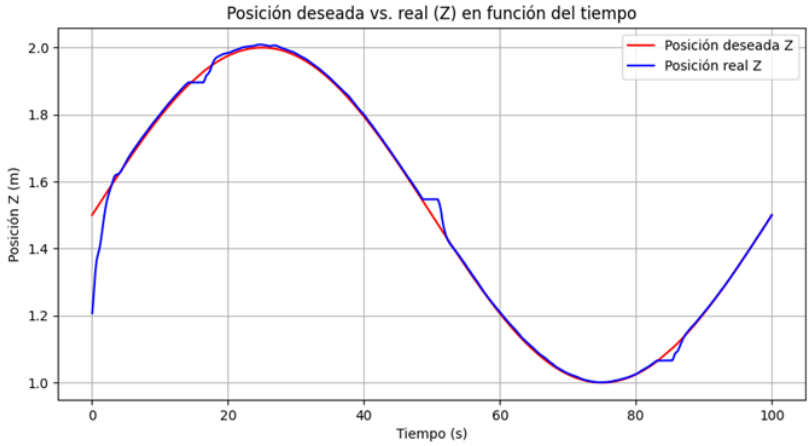


Figura 5.13: *Trayectoria deseada (color rojo) y trayectoria del cuadricóptero (color azul) en Z, mejor caso.*

En la Figura 5.14 se despliegan los errores de posición en x, y y z, siendo estos bastante pequeños después del despegue del cuadricóptero.

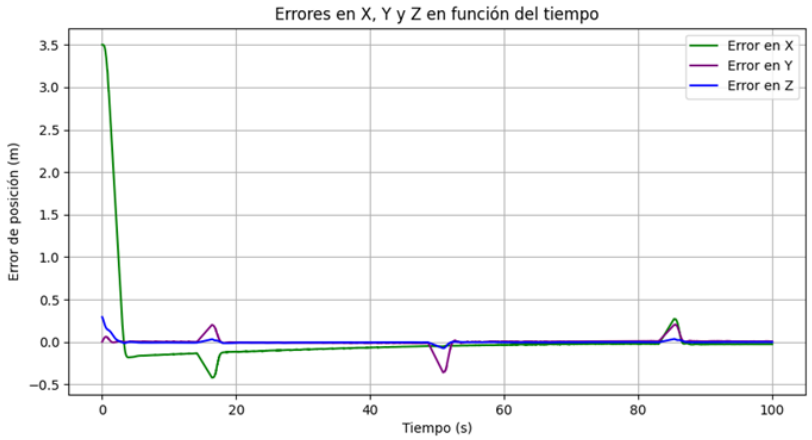


Figura 5.14: *Errores de posición en x,y,z de la trayectoria de tipo lemniscata, mejor caso.*

El set de ganancias con el peor desempeño, según su **RMSE** de 0.7224, es $K_p = 1$, $K_i = 1$, $K_d = 1$. Con este **RMSE** se muestra un seguimiento menos preciso. El **ME** y **MAE** son los más altos, ≈ 0.26, lo que demuestra desviaciones promedio significativas. El IAE, de 27.36, evidencia un gran acumulado de error y la variabilidad sugiere una respuesta errática y menos predecible. El *Overshoot* es muy similar al resto, pero no compensa la pobre corrección integral y proporcional, resultando en un control menos eficaz. A continuación se comparten

las gráficas de posición deseada contra la posición real del cuadricóptero y los errores de posición, todo respecto al tiempo.

En la Figura 5.15 se observa el cuadricóptero en el despegue, presentando una alta desviación en el intento de corregir la posición en X para seguir la trayectoria circular. Cuando consigue seguir la trayectoria, el dron ya no presenta desviaciones tan acentuadas como en el inicio.

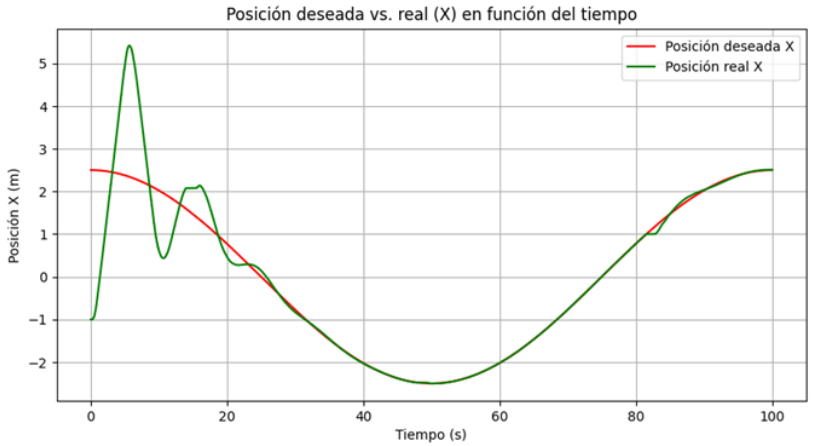


Figura 5.15: *Trayectoria deseada (color rojo) y trayectoria del cuadricóptero (color verde) en X, peor caso.*

En la Figura 5.16 se observan desviaciones mínimas de la posición real en Y respecto a la posición deseada.

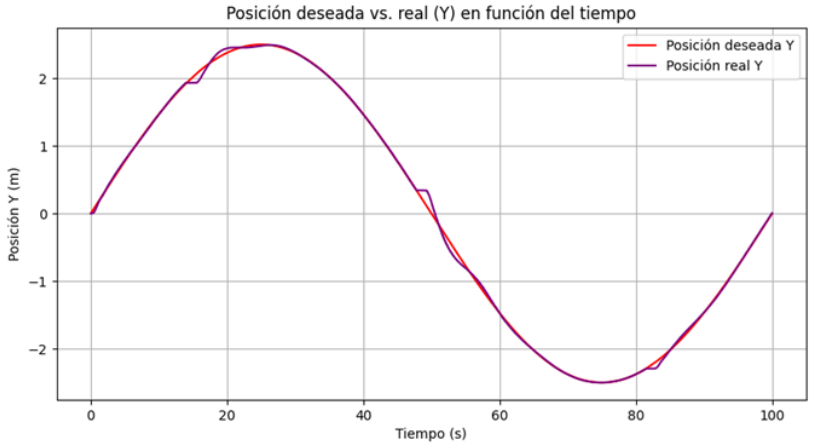


Figura 5.16: *Trayectoria deseada (color rojo) y trayectoria del cuadricóptero (color morado) en Y, mejor caso.*

En la Figura 5.17 se observa en el despegue como el cuadricóptero presenta desviaciones en el intento de corregirse y seguir la trayectoria circular, presentando oscilaciones que van decrementando conforme se corrige. Después de presentar este comportamiento y conseguir seguir la trayectoria en Z, el dron presenta desviaciones mínimas.

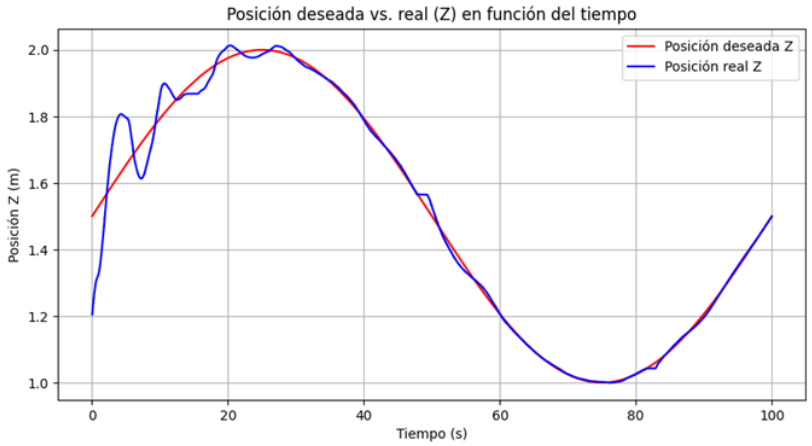


Figura 5.17: *Trayectoria deseada (color rojo) y trayectoria del cuadricóptero (color azul) en Z, mejor caso.*

En la Figura 5.18 se grafican los errores de posición en x, y, z del PID con peor rendimiento, de la trayectoria de tipo circular. Presenta errores significativos, especialmente en X.

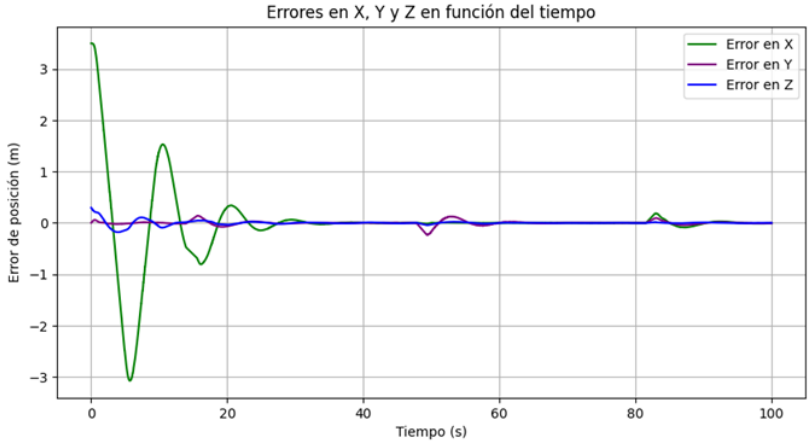


Figura 5.18: *Errores de posición en x,y,z de la trayectoria de tipo circular, peor caso.*

5.1.9. Resultados con trayectoria aleatoria

Se presentan las métricas obtenidas de las simulaciones en las que el cuadricóptero, bajo un control clásico de tipo PID, sigue una trayectoria aleatoria cerrada, la cual se muestra en la Figura 5.19.

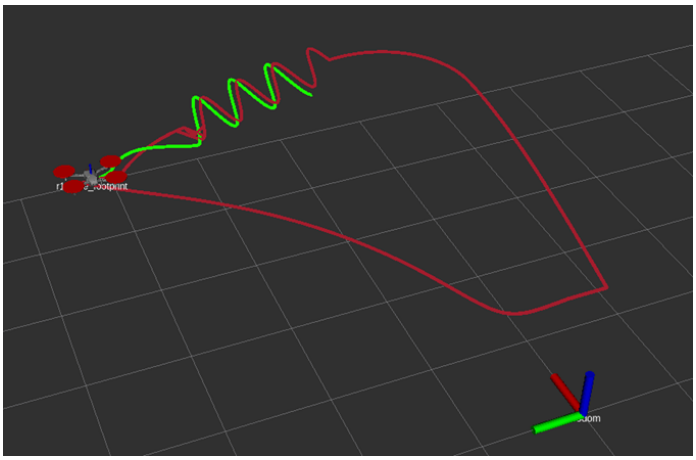


Figura 5.19: *Trayectoria deseada (rojo) vs. trayectoria real (verde) en RViz.*

La trayectoria aleatoria se generó a partir de un script de Python, donde se programó que esta trayectoria estuviera dividida en 5 secciones y que al final la última sección regresara a la primera sección. Posteriormente, la trayectoria resultante se agrega al código de Python donde se realiza la simulación con Gazebo Ignition. La Tabla 5.3 agrupa las métricas de desempeño para cada combinación de ganancias evaluadas.

Tabla 5.3: Resultados de métricas para trayectoria aleatoria.

Ganancias PID	ME	MAE	RMSE	IAE	Overshoot	Variabilidad
Kp=0.5, Ki=0.1, Kd=0.5	0.107101	0.107101	0.187187	11.635340	1.146084	0.153536
Kp=0.8, Ki=0.05, Kd=0.1	0.058759	0.058759	0.135025	8.259747	1.112227	0.121583
Kp=1.0, Ki=1.0, Kd=1.0	0.101459	0.101459	0.178404	11.084277	1.114523	0.146761
Kp=2.0, Ki=0.05, Kd=0.5	0.058748	0.058748	0.131572	6.675662	1.128559	0.117740
Kp=2.0, Ki=1.0, Kd=1.0	0.071338	0.071338	0.153384	9.581504	1.112084	0.135799

El ajuste con $K_p = 2$, $K_i = 0.05$ y $K_d = 0.5$ obtuvo un **RMSE** bajo, lo que indica un seguimiento muy preciso de la trayectoria aleatoria. Tanto el **ME** como el **MAE** son mínimos (≈ 0.059), reflejando pequeñas desviaciones promedio. La **IAE**, también la más baja (6.68), muestra que el error acumulativo es escaso a lo largo de toda la maniobra. Con un *Overshoot* contenido ($\approx 1.13\,\%$) y la variabilidad de error más baja, la respuesta del cuadricóptero es tanto estable como predecible, corrigiendo rápidamente sin generar oscilaciones innecesarias. Se presentan a continuación las gráficas de la posición deseada y la posición del cuadricóptero, y de los errores de posición en x, y y z.

En la Figura 5.20 se observan leves desviaciones en la trayectoria real en X respecto a la trayectoria deseada, en especial en el segmento oscilatorio.

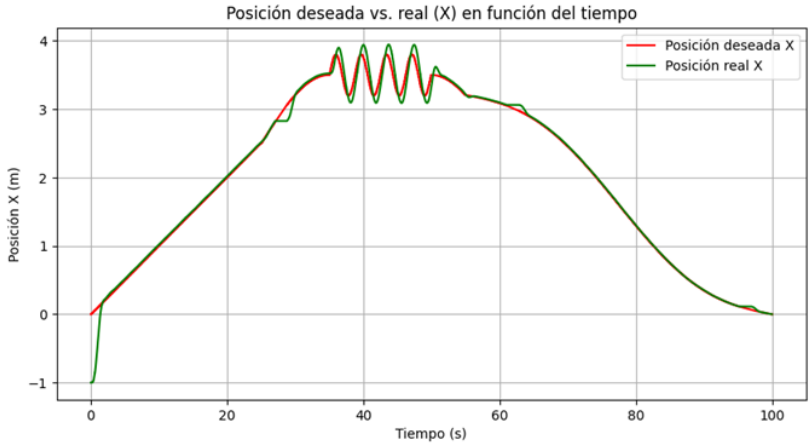


Figura 5.20: *Trayectoria deseada (color rojo) y trayectoria del cuadricóptero (color verde) en X, mejor caso.*

En la Figura 5.21 se puede observar que el dron en Y sigue sin problemas la trayectoria aleatoria.

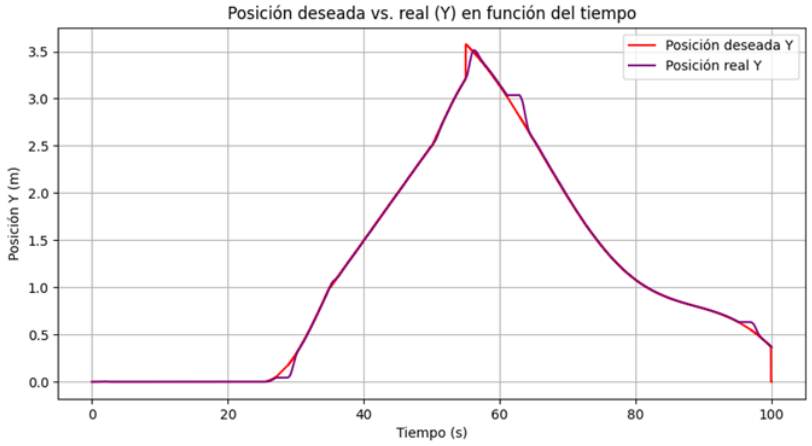


Figura 5.21: *Trayectoria deseada (color rojo) y trayectoria del cuadricóptero (color morado) en Y, mejor caso.*

En la Figura 5.22 se observa una desviación mantenida al despegue del cuadricóptero, para después seguir la trayectoria, pero después presenta leves desviaciones en el segmento oscilatorio, esto en Z.

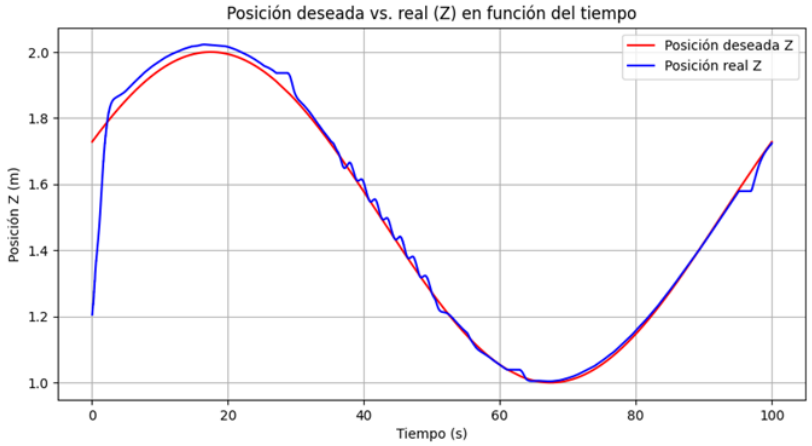


Figura 5.22: *Trayectoria deseada (color rojo) y trayectoria del cuadricóptero (color azul) en Z, mejor caso.*

En la siguiente gráfica se despliegan los errores de posición en x, y y z, los cuales son pequeños después del despegue del cuadricóptero. Donde hay más ocurrencia de error de posición es en el segmento oscilatorio de la trayectoria aleatoria.

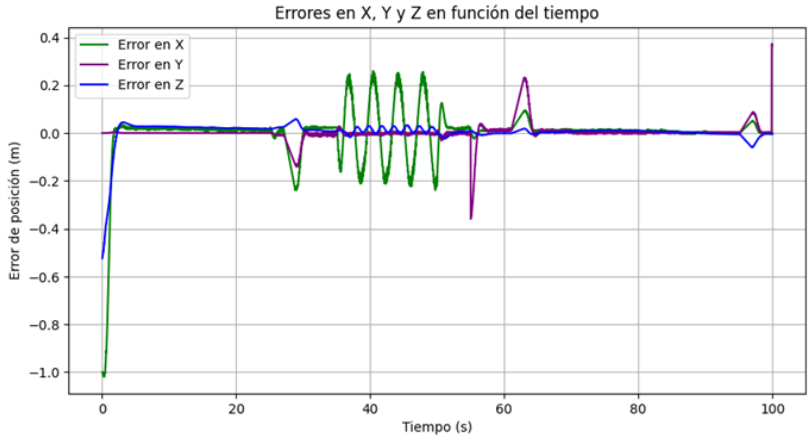


Figura 5.23: *Errores de posición en x,y,z de la trayectoria de tipo lemniscata, mejor caso.*

El conjunto $K_p = 0.5$, $K_i = 0.1$ y $K_d = 0.5$ presentó el **RMSE** más elevado, indicando un seguimiento menos fiel a la trayectoria deseada. El **ME** y **MAE**, los mayores (≈ 0.107), denotan desviaciones promedio significativas. El **IAE** elevado, con 11.64, señala un notable acumulado de error. Además, la variabilidad más alta, con 0.1535 y el *Overshoot* ligeramente superior evidencian qeu las correcciones son lentas y algo inestables, con picos de error más

pronunciados. Se presentan las gráficas de la posición deseada contra la posición real del cuadricóptero en x, y y z; así como también la gráfica de los errores de posición.

En la Figura 5.24 se observa que en el despegue le cuesta un poco más de tiempo seguir la trayectoria en X, a comparación del mejor caso. También, en el segmento oscilatorio, al dron le cuesta un poco más de trabajo seguir la trayectoria.

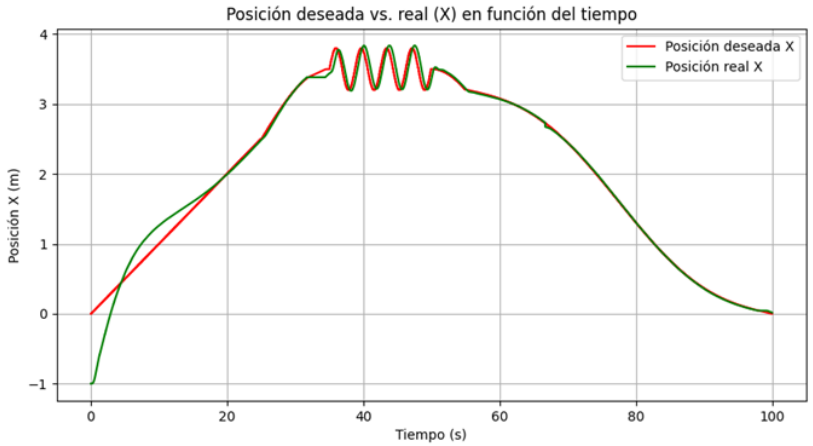


Figura 5.24: *Trayectoria deseada (color rojo) y trayectoria del cuadricóptero (color verde) en X, peor caso.*

En la Figura 5.25 se observan desviaciones de la posición real en Y respecto a la posición deseada, donde a comparación del mejor caso, en este hay presencia de mayores desviaciones.

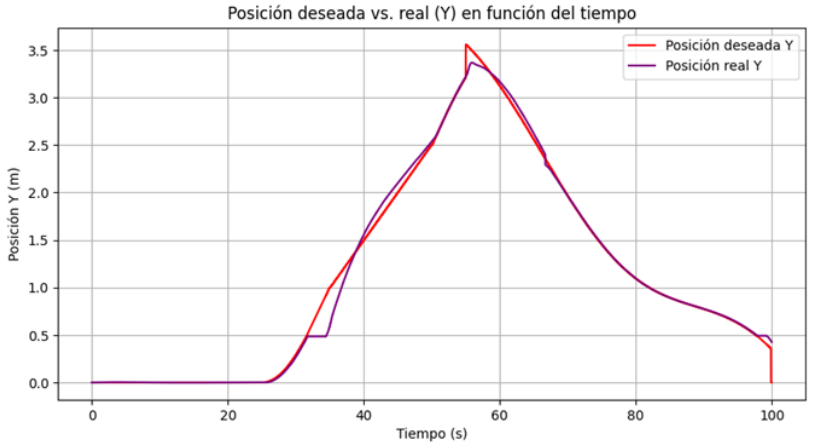


Figura 5.25: *Trayectoria deseada (color rojo) y trayectoria del cuadricóptero (color morado) en Y, mejor caso.*

En la Figura 5.26 el cuadricóptero presenta una desviación importante al despegue, esto en Z. A lo largo de la simulación presenta más desviaciones, en especial cuando se presentan cambios de dirección.

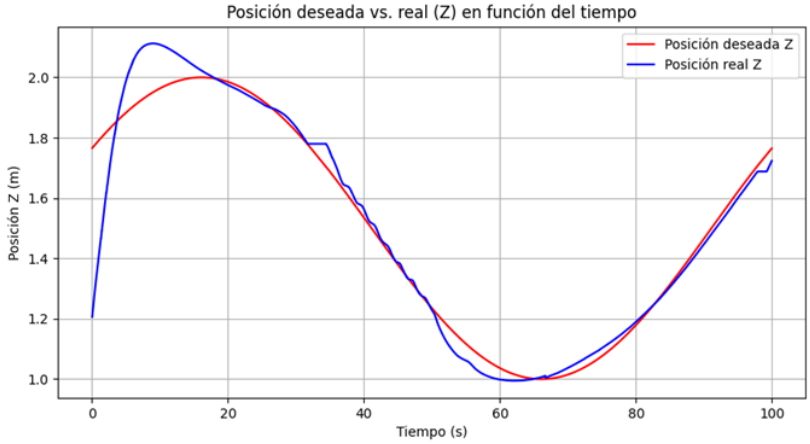


Figura 5.26: *Trayectoria deseada (color rojo) y trayectoria del cuadricóptero (color azul) en Z, mejor caso.*

En la Figura 5.27 se presentan los errores de posición en x, y, z del PID con el peor desempeño. Este caso presenta errores más pronunciados, a comparación del PID con mejor desempeño. Los errores más pronunciados se presentan en el despegue y en el segmento oscilatorio de la trayectoria aleatoria.

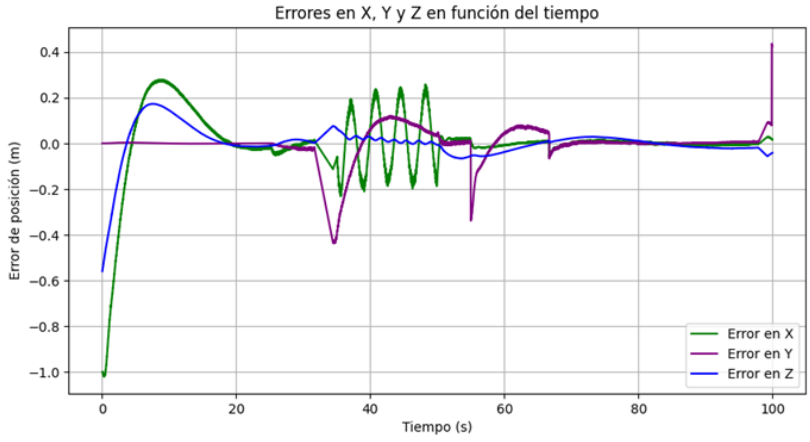


Figura 5.27: *Errores de posición en x,y,z de la trayectoria de tipo circular, peor caso.*

5.2. Resultados de la simulación con SAC

El algoritmo *Soft Actor-Critic*, SAC, es un algoritmo de aprendizaje por refuerzo fuera de la política que maximiza una recompensa esperada mientras maximiza la entropía. Esto fomenta la exploración y puede conducir a políticas más sólidas [64].

Para el entrenamiento se programa en Python, como se muestra en la Figura 5.28. Se programa para que el agente aprenda a elevarse hasta 1.2 m, estabilizándose allí. Se minimizan desviaciones y movimientos innecesarios para que el cuadricóptero pueda mantenerse en *hover*. Cada episodio tiene un máximo de $2,000$ pasos; se reinicia si transcurren todos sin éxito o si se rompe la condición de seguridad, por ejemplo, la altura fuera de rango.

```python
# Comprobar y actualizar el estado del despegue
if not self.taken_off and self.pos[2] >= self.takeoff_alt:
    self.taken_off = True

# Bloquear XY hasta que se complete el despegue
if not self.taken_off:
    cmd.linear.x = 0.0
    cmd.linear.y = 0.0
    cmd.linear.z = float(action[2])
else:
    cmd.linear.x = float(action[0])
    cmd.linear.y = float(action[1])
    # Bloquear el ascenso/descenso tras alcanzar el punto de planeo
    cmd.linear.z = 0.0

cmd.angular.z = float(action[3])
self.cmd_pub.publish(cmd)
```

Figura 5.28: *Segmento de código en Python para entrenamiento con SAC para que el cuadricóptero despegue a 1.2 m y aprenda a hacer hover.*

5.2.1. Espacios de observación y acción

En SAC, el espacio de observación es el conjunto de todas las representaciones posibles que el agente puede recibir del entorno en un instante dado [64]. Cada observación es un vector de números reales que sintetiza la información relevante para la toma de decisiones. En SAC, la política (la red neuronal que decide qué acción tomar) recibe como entrada una muestra en este espacio de observaciones, de modo que aprenda a identificar, a partir del estado actual, cuál es la mejor maniobra que lo acerque a su objetivo [65].

$$\text{Observación } \mathbf{o} = [x, y, z, v_x, v_y, v_z] \in R^6 : \text{ posición y velocidad lineal del dron.} \tag{5.7}$$

El espacio de acción es el conjunto de todos los comandos que el agente puede emitir para interactuar con el entorno [35]. SAC trabaja normalmente en espacios continuos, de modo que la política devuelve una distribución continua sobre acciones.

$$\begin{aligned} \text{Acción } \mathbf{a} = &\left[v_x^{\text{cmd}}, v_y^{\text{cmd}}, v_z^{\text{cmd}}, \omega_z^{\text{cmd}}\right], \text{ con límites:} \\ &v_x, v_y \in [-1.5, 1.5], \quad v_z \in [-0.5, 0.5], \quad \omega_z \in [-1, 1]. \end{aligned} \tag{5.8}$$

5.2.2. Función de recompensa y ecuaciones

En el aprendizaje por refuerzo, la función de recompensa es un elemento fundamental, ya que define el objetivo que el agente debe alcanzar. En el entorno del cuadricóptero, la función de recompensa se diseña para guiar al dron hacia el punto objetivo, mientras se penalizan las acciones que lo alejan de él o que comprometen su estabilidad. Cada paso recibe un bonus de supervivencia +0.1, más un término principal según la fase. Si se encuentra en la fase de ascenso ($z < 1.2$), la recompensa está dada por

$$r = 0.1 + 10z - \sqrt{x^2 + y^2}. \tag{5.9}$$

En la fase de *hover* se utiliza la siguiente recompensa

$$r = 0.1 - K_{xy}\sqrt{x^2 + y^2} - K_z|z - 1.2| - K_{\text{mse}}\frac{(x^2 + y^2 + (z - 1.2)^2)}{3} - K_{\text{eff}}\|\mathbf{a}\|^2 - K_{\text{ang}}\left|\omega_z^{\text{cmd}}\right|, \tag{5.10}$$

donde los pesos usados en el código son

$$K_{xy} = 1.0, K_z = 1.0, K_{\text{mse}} = 0.5, K_{\text{eff}} = 0.01, K_{\text{ang}} = 0.1. \tag{5.11}$$

Un episodio termina con éxito si $\|pos - [0, 0, 1.2]\| < 0.05$ m y $\|vel\| < 0.05$ m/s.

Además, si tras despegar $\mathbf{z} < 0.1$ m o $|\mathbf{z} - 1.2| > 0.5$ m, se aplica una penalización de -50 y se da por fallido el episodio.

5.2.3. Resultados del entrenamiento *hover* con SAC

Previo a definir la función de recompensa anterior, se trabajó con algunas otras funciones de recompensa para el entrenamiento del despegue y *hover* del cuadricóptero, de las cuales no hubo mucho éxito. La función de recompensa explicada anteriormente nace de ir agregando elementos a la recompensa y seccionar dicha recompensa en dos estados, el despegue y *hover*. En la gran mayoría de los entrenamientos, el cuadricóptero salía del espacio establecido, ya sea subiendo infinitamente o avanzando en horizontal sin detenerse, hasta perderse.

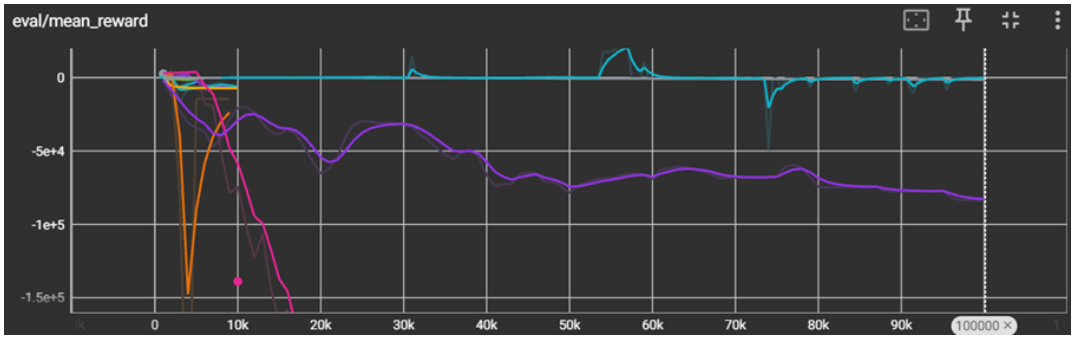


Figura 5.29: *Grafica en TensorBoard del promedio de la recompensa acumulada por episodio de 37 entrenamientos con SAC*

Como se observa en la Figura 5.29, varias de las curvas se estancaron o bajaron abruptamente, lo cual pudo ser un sobreajuste, un cambio demasiado brusco en la política o que la función de recompensa no está incentivando el comportamiento deseado. La única que no tiene una caída abrupta es la línea azul, la cual representa el entrenamiento número 37 con SAC, y cuya configuración se explicó previamente.

En la Figura 5.30 se ilustra la evolución de la recompensa promedio en evaluación durante el propio entrenamiento. Cuando la recompensa promedio sube, significa que el agente aprende correctamente y mejora en resolver la tarea. Pero si la recompensa decrece, el agente probablemente se está sobreajustando o presenta problemas en generalizar lo aprendido. En el

entrenamiento ejecutado se observa que la recompensa promedio es relativamente estable a lo largo del entrenamiento, pero en los últimos pasos se observa que la recompensa promedio cambia drásticamente, decrementando considerablemente.

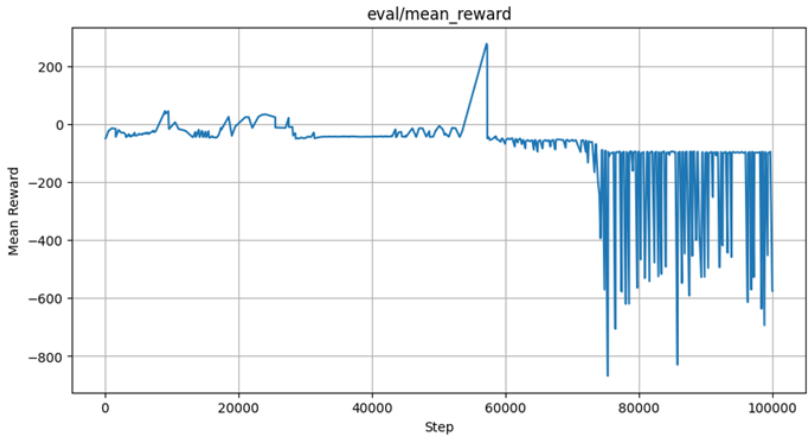


Figura 5.30: *Gráfica de la recompensa promedio en evaluación del entrenamiento con SAC.*

En la Figura 5.31 se presenta la gráfica de la recompensa promedio durante el entrenamiento, reflejando el rendimiento promedio del agente mientras aprende activamente. Se observa que al inicio se comienza con recompensas negativas (≈ -49). La recompensa va fluctuando y mejorando, donde en algunos momentos incrementa. Pero, hacia el final, se observa una caída de la recompensa promedio por debajo de -500, mostrando que la política aprendida se desestabilizó al alargar demasiado el entrenamiento sin considerar reajustes de hiperparámetros.

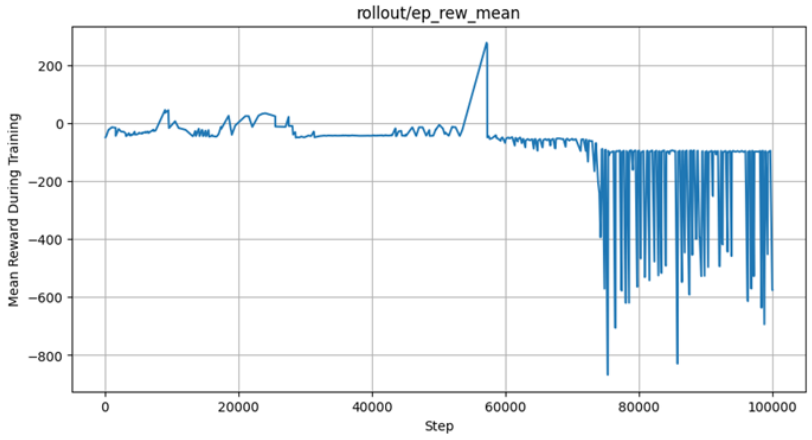


Figura 5.31: *Gráfica de la recompensa durante el entrenamiento con SAC.*

En la Figura 5.32 se muestra la gráfica de longitud promedio de episodio en evaluación, donde se muestra la cantidad promedio que dura un episodio durante las pruebas periódicas del entrenamiento con SAC. El episodio más largo ocurre previo a los $60,000$ pasos, indicando que en este segmento el agente mejora su capacidad de controlar la tarea asignada y evitando así estados negativos. Se podría decir que los valores al inicio permanecen bajos y constantes, lo cual se debe a que en este punto el agente termina sus episodios muy rápido, posiblemente por errores o falta de conocimiento del entorno. Pero, al observar al final que los valores vuelven a bajar y se mantienen, significa que el agente perdió el desempeño consistente tras su período estable, o que tal vez el agente se sobreajusta y no vuelve a mantener un episodio.

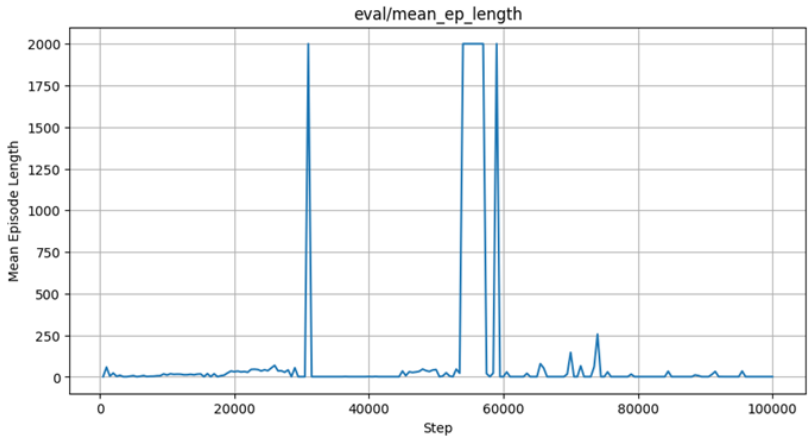


Figura 5.32: *Gráfica de longitud promedio del episodio respecto a los pasos del entrenamiento SAC.*

En la Figura 5.33 se muestra el valor de pérdida (error) del crítico, que estima el valor esperado de las acciones tomadas en el entrenamiento. En SAC el crítico (critic) es una red que estima el valor esperado de una acción en un estado dado. En la gráfica se observa que al inicio los valores se mantienen estables. Esto puede ser porque se están recibiendo experiencias sencillas, y a falta de cambios en el entorno, el error entre lo que el crítico predice y lo que ocurre es bajo y constante. También puede ser que el agente aún no enfrenta estados complejos y esos valores estables son por la poca exploración efectiva. El pico que se observa previo a los $60,000$ indica cambios en la política del actor, esto puede ser porque ocurre una trayectoria exitosa. El decremento de los picos sugiere que el crítico se ajusta a los nuevos valores esperados generados por la política del actor.

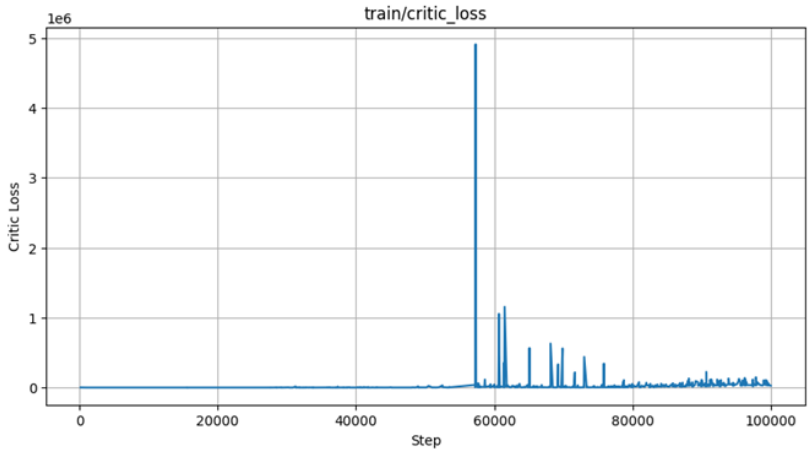


Figura 5.33: *Gráfica de pérdida del crítico durante el entrenamiento con SAC.*

En la Figura 5.34 se muestra la gráfica de pérdida del actor durante el entrenamiento. Los datos desplegados sirven para medir qué tan va el actor (política) selecciona acciones que maximizan las recompensas. Una pérdida estable y disminuida indica que el actor encuentra mejores decisiones con el tiempo. Pero al final se observa como la pérdida incrementa, lo que indica dificultades para estabilizar la política.

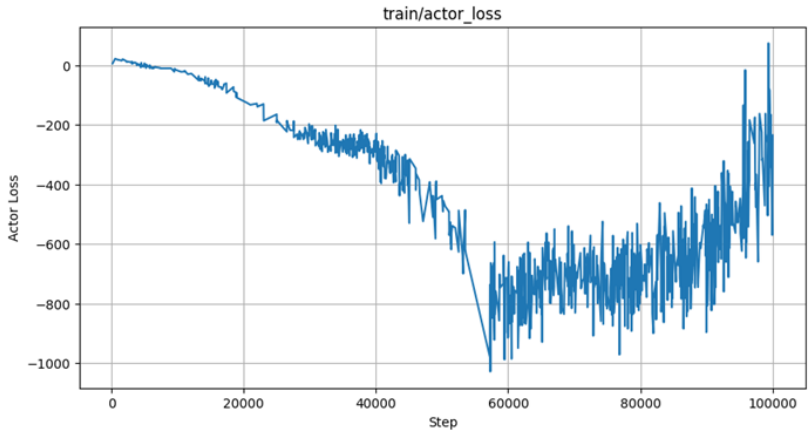


Figura 5.34: *Gráfica pérdida del actor durante el entrenamiento con SAC.*

En general, el agente muestra mejora en recompensa y duración de episodios durante las primeras decenas de miles de pasos, pero en las etapas finales diverge y se vuelve inestable.

Capítulo 6

Conclusiones

6.1. Conclusiones generales

Durante la realización de esta investigación, se evaluaron tanto métodos tradicionales como no tradicionales para el control y monitoreo de trayectorias en un cuadricóptero, empleando el ambiente de simulación ROS 2. Se utilizaron controladores PID y de refuerzo por aprendizaje basado en el algoritmo Soft Actor-Critic (SAC), utilizados en simulaciones con tres tipos de trayectorias: lemniscata, circular y aleatoria para el PID, despegue y hover para el algoritmo con SAC.

Los hallazgos obtenidos indican que el controlador PID exhibió robustez y estabilidad para trayectorias preestablecidas, particularmente para movimientos de menor complejidad como la trayectoria circular. No obstante, su rendimiento disminuyó frente a movimientos de mayor complejidad o dinámica, lo que evidencia una mayor sensibilidad a las ganancias seleccionadas y subraya la relevancia del ajuste meticuloso de sus parámetros para optimizar el seguimiento exacto.

En contraste, el método de aprendizaje por refuerzo SAC evidenció inicialmente un potencial considerable en tareas complejas como el *hover*, evidenciando mejoras significativas en la recompensa y en la duración de los episodios durante las fases iniciales de formación. Sin embargo, se registró una notable inestabilidad y divergencia en las fases avanzadas del entrenamiento, lo que señala la necesidad imperativa de ajustes apropiados en los hiperparámetros

y en la función de recompensa para prevenir situaciones como el sobreajuste o la desestabilización de la política adquirida. Además de lo anterior, el entrenamiento necesita de recurso computacional para poder llevarse a cabo.

Las métricas de evaluación implementadas (ME, MAE, RMSE, IAE, Overshoot y Variabilidad) establecieron un fundamento robusto para el análisis comparativo en los resultados del PID, permitiendo una identificación precisa del grupo de ganancias que se ejecutaron a las diferentes trayectorias. Se deduce que, a pesar de que el aprendizaje por refuerzo proporciona sencillez y eficiencia en determinados escenarios, esté presenta un amplio margen de mejora y promete un rendimiento superior en contextos dinámicos y menos estructurados, siempre que se aborden los retos asociados con la estabilidad del aprendizaje.

6.2. Futuras líneas de investigación y mejoras

Al término de esta investigación, existe el interés de mejorar los algoritmos de control aplicados a la planificación de trayectoria del cuadricóptero. Lo anterior debido a que existen otras rutas a considerar en la ejecución y análisis del control PID. También, a pesar de que hay un avance en el desarrollo de algoritmos de entrenamiento por aprendizaje por refuerzo, aún este es limitado, por lo que existe un gran campo de exploración en este ramo, sin considerar que todos los algoritmos aquí utilizados se pueden optimizar. A continuación se comparten futuras líneas de investigación a raíz del trabajo aquí presentado:

1. Para el entrenamiento con SAC, definir una mejor función de recompensa para el *hover* y para las trayectorias ejecutadas en el control PID.
2. Mejorar la precisión en trayectorias complejas mediante algoritmos inteligentes como optimización por cúmulo de partículas, para optimizar el ajuste automático de ganancias PID.
3. Explorar arquitecturas híbridas que combinen PID y SAC para aprovechar la estabilidad del control clásico y la adaptabilidad del control inteligente al profundizar en estrategias avanzadas de aprendizaje por refuerzo.
4. Evaluar la robustez de los controladores propuestos mediante la inclusión de perturbaciones realistas en las simulaciones, como variaciones climáticas o fallos de componentes.

5. Comprobar los modelos creados en ambientes experimentales con equipos reales, lo que facilitará la comparación directa de los descubrimientos simulados con resultados prácticos.

La integración de todos estos esfuerzos facilitará el progreso hacia sistemas autónomos más sólidos, fiables y eficaces, contribuyendo de esta manera a la planificación de rutas de cuadricópteros.

Apéndice A

Publicaciones derivadas del trabajo de tesis

1. **Producción científica**
   Susana Borrego-Domínguez, Laura Jiménez-Beristain, Enrique Efrén García-Guerrero, José Jaime Esqueda-Elizondo, César Ortega-Corral, Ulises Jesús Tamayo-Pérez, Diego Armando Trujillo-Toledo, Oscar Adrián Aguirre-Castro, César Alberto López-Mercado, Everardo Inzunza-González. 2025, *Reinforcement Learning for Drone Path Planning: A Bibliometric Analysis*. Sometido, en estatús aprobado.

2. **Acceso Universal del conocimiento**
   Susana Borrego-Domínguez, Laura Jiménez-Beristain, Enrique Efrén García-Guerrero, José Jaime Esqueda-Elizondo, César Ortega-Corral, Ulises Jesús Tamayo-Pérez, Diego Armando Trujillo-Toledo, Oscar Adrián Aguirre-Castro, César Alberto López-Mercado, Everardo Inzunza-González. 2024, *Reinforcement Learning for Drone Path Planning: A Bibliometric Analysis*. Congreso internacional EDIESCA, 2024, Aguascalientes, México.

   Susana Borrego-Domínguez et al. *Robot autónomo con inteligencia artificial* 2024. XXXI Jornadas FIAD de la Expo Ciencia y Tecnología. Ensenada, Baja California.

   Susana Borrego-Domínguez et al. *Pensamiento computacional para todos.* 2023. Presentación en Expo Ciencia y Tecnología - FIAD. Ensenada, Baja California. Enlace: Pensamiento computacional para todos

Susana Borrego-Domínguez et al. *Evaluación de métodos convencionales y no convencionales de control para un brazo helicóptero de dos grados de libertad* 2023. X Simposio de Investigación FIAD. Ensenada, Baja California.

Apéndice B

Códigos fuente

Código del archivo ign world launch.py.

```
#!/usr/bin/env python3

from launch import LaunchDescription
from launch_ros.actions import Node
from launch.actions import ExecuteProcess
from ament_index_python.packages import get_package_share_directory
from pathlib import Path
import xacro

# Gestión de eventos de simulación
from launch.actions import RegisterEventHandler, EmitEvent
from launch.event_handlers import OnProcessExit
from launch.events import Shutdown

# Variables de trayectoria
pkg_path = get_package_share_directory("my_uavs") # ~/colcon_ws/install/my_uavs/share/my
simulation_world_file_path = Path(pkg_path, "worlds/my_custom_world.sdf").as_posix()
rviz_config_file_path = Path(pkg_path, "rviz/one_drone.rviz").as_posix()
```

```
robot_description_path = Path( pkg_path, "urdf/r1.xacro")
robot_description = {"robot_description": xacro.process_file(robot_description_path).toxi

simulation_cmd = ExecuteProcess(
    cmd=['ign', 'gazebo', '-r', simulation_world_file_path],
    output='screen'
)

open_rviz = Node(
        package="rviz2",
        executable="rviz2",
        arguments=['-d', rviz_config_file_path],
        output="screen"
)

robot_state_publisher_node = Node(
        package="robot_state_publisher",
        executable="robot_state_publisher",
        output="both",
        parameters=[robot_description],
)

def generate_launch_description():
    return LaunchDescription([
        simulation_cmd,

        Node(
            package="ros_gz_bridge",
            executable="parameter_bridge",
            arguments=[
                "/r1/gazebo/command/twist@geometry_msgs/msg/Twist@ignition.msgs.Twist",
                "/model/r1/odometry@nav_msgs/msg/Odometry@ignition.msgs.Odometry"
            ],
            remappings=[
                ("/r1/gazebo/command/twist", "/r1/cmd_vel"),
```

```
                ("/model/r1/odometry", "/r1/odom")
            ],
            output="screen"
        ),

        open_rviz,

        robot_state_publisher_node,

        RegisterEventHandler(
            event_handler=OnProcessExit(
                target_action=simulation_cmd,
                on_exit=[EmitEvent(event=Shutdown())],
            )
        )
    ])
```

Código del archivo my drone.sdf.

```
<?xml version="1.0" ?>
<sdf version="1.6">
  <model name="r1">
    <pose>0 0 0.05 0 0 0</pose>

    <!-- Base -->
    <link name="r1/base_link">
      <pose frame="">0 0 0 0 0 0</pose>
      <inertial>
        <mass>1.313</mass>
        <inertia>
          <ixx>0.0386</ixx>
          <iyy>0.0778</iyy>
          <izz>0.1086</izz>
        </inertia>
```

```
  </inertial>
  <collision name="r1/base_link_inertia_collision">
    <pose frame="">0 0 -0.03 0 0 0</pose>
    <geometry>
      <box>
        <size>0.47 0.47 0.15</size>
      </box>
    </geometry>
  </collision>
  <visual name="r1/base_link_inertia_visual">
    <pose frame="base_frame">0 0 0 0 0 0</pose>
    <geometry>
      <mesh>
        <scale>1 1 1</scale>
        <uri>meshes/iris.dae</uri>
      </mesh>
    </geometry>
    <material>
      <diffuse>0 0 0.7 1</diffuse>
    </material>
  </visual>
</link>

<!-- Rotor 0 -->
<link name="r1/rotor_0">
  <pose frame="">0.13 -0.22 0.023 0 0 0</pose>
  <inertial>
    <pose frame="">0 0 0 0 -0 0</pose>
    <mass>0.005</mass>
    <inertia>
      <ixx>9.75e-07</ixx>
      <iyy>4.17041e-05</iyy>
      <izz>4.26041e-05</izz>
    </inertia>
  </inertial>
```

```
  <collision name="r1/rotor_0_collision">
    <pose frame="">0 0 0 0 0 0</pose>
    <geometry>
      <cylinder>
        <length>0.005</length>
        <radius>0.1</radius>
      </cylinder>
    </geometry>
    <surface>
      <contact><ode/></contact>
      <friction><ode/></friction>
    </surface>
  </collision>
  <visual name="r1/rotor_0_visual">
    <pose frame="">0 0 0 0 -0 0</pose>
    <geometry>
      <mesh>
        <scale>0.1 0.1 0.1</scale>
        <uri>meshes/propeller_ccw.dae</uri>
      </mesh>
    </geometry>
    <material>
      <diffuse>1 0 0 1</diffuse>
    </material>
  </visual>
  <gravity>1</gravity>
  <velocity_decay/>
</link>
<joint name="r1/rotor_0_joint" type="revolute">
  <child>r1/rotor_0</child>
  <parent>r1/base_link</parent>
  <axis>
    <xyz>0 0 1</xyz>
    <limit>
      <lower>-1e+16</lower>
```

```
      <upper>1e+16</upper>
    </limit>
    <dynamics>
      <spring_reference>0</spring_reference>
      <spring_stiffness>0</spring_stiffness>
    </dynamics>
    <use_parent_model_frame>1</use_parent_model_frame>
  </axis>
</joint>

<!-- Rotor 1 -->
<link name="r1/rotor_1">
  <pose frame="">-0.13 0.2 0.023 0 0 0</pose>
  <inertial>
    <pose frame="">0 0 0 0 -0 0</pose>
    <mass>0.005</mass>
    <inertia>
      <ixx>9.75e-07</ixx>
      <iyy>4.17041e-05</iyy>
      <izz>4.26041e-05</izz>
    </inertia>
  </inertial>
  <collision name="r1/rotor_1_collision">
    <pose frame="">0 0 0 0 0 0</pose>
    <geometry>
      <cylinder>
        <length>0.005</length>
        <radius>0.1</radius>
      </cylinder>
    </geometry>
    <surface>
      <contact><ode/></contact>
      <friction><ode/></friction>
    </surface>
  </collision>
```

```
  <visual name="r1/rotor_1_visual">
    <pose frame="">0 0 0 0 -0 0</pose>
    <geometry>
      <mesh>
        <scale>0.1 0.1 0.1</scale>
        <uri>meshes/propeller_ccw.dae</uri>
      </mesh>
    </geometry>
    <material>
      <script>
        <name>Gazebo/Red</name>
        <uri>file://media/materials/scripts/gazebo.material</uri>
      </script>
    </material>
  </visual>
  <gravity>1</gravity>
  <velocity_decay/>
</link>
<joint name="r1/rotor_1_joint" type="revolute">
  <child>r1/rotor_1</child>
  <parent>r1/base_link</parent>
  <axis>
    <xyz>0 0 1</xyz>
    <limit>
      <lower>-1e+16</lower>
      <upper>1e+16</upper>
    </limit>
    <dynamics>
      <spring_reference>0</spring_reference>
      <spring_stiffness>0</spring_stiffness>
    </dynamics>
    <use_parent_model_frame>1</use_parent_model_frame>
  </axis>
</joint>
```

```
<!-- Rotor 2 -->
<link name="r1/rotor_2">
  <pose frame="">0.13 0.22 0.023 0 0 0</pose>
  <inertial>
    <pose frame="">0 0 0 0 -0 0</pose>
    <mass>0.005</mass>
    <inertia>
      <ixx>9.75e-07</ixx>
      <iyy>4.17041e-05</iyy>
      <izz>4.26041e-05</izz>
    </inertia>
  </inertial>
  <collision name="r1/rotor_2_collision">
    <pose frame="">0 0 0 0 0 0</pose>
    <geometry>
      <cylinder>
        <length>0.005</length>
        <radius>0.1</radius>
      </cylinder>
    </geometry>
    <surface>
      <contact><ode/></contact>
      <friction><ode/></friction>
    </surface>
  </collision>
  <visual name="r1/rotor_2_visual">
    <pose frame="">0 0 0 0 -0 0</pose>
    <geometry>
      <mesh>
        <scale>0.1 0.1 0.1</scale>
        <uri>meshes/propeller_ccw.dae</uri>
      </mesh>
    </geometry>
    <material>
      <diffuse>1 0 0 1</diffuse>
```

```
    </material>
  </visual>
  <gravity>1</gravity>
  <velocity_decay/>
</link>
<joint name="r1/rotor_2_joint" type="revolute">
  <child>r1/rotor_2</child>
  <parent>r1/base_link</parent>
  <axis>
    <xyz>0 0 1</xyz>
    <limit>
      <lower>-1e+16</lower>
      <upper>1e+16</upper>
    </limit>
    <dynamics>
      <spring_reference>0</spring_reference>
      <spring_stiffness>0</spring_stiffness>
    </dynamics>
    <use_parent_model_frame>1</use_parent_model_frame>
  </axis>
</joint>

<!-- Rotor 3 -->
<link name="r1/rotor_3">
  <pose frame="">-0.13 -0.2 0.023 0 0 0</pose>
  <inertial>
    <pose frame="">0 0 0 0 -0 0</pose>
    <mass>0.005</mass>
    <inertia>
      <ixx>9.75e-07</ixx>
      <iyy>4.17041e-05</iyy>
      <izz>4.26041e-05</izz>
    </inertia>
  </inertial>
  <collision name="r1/rotor_3_collision">
```

```
    <pose frame="">0 0 0 0 0 0</pose>
    <geometry>
      <cylinder>
        <length>0.005</length>
        <radius>0.1</radius>
      </cylinder>
    </geometry>
    <surface>
      <contact><ode/></contact>
      <friction><ode/></friction>
    </surface>
  </collision>
  <visual name="r1/rotor_3_visual">
    <pose frame="">0 0 0 0 -0 0</pose>
    <geometry>
      <mesh>
        <scale>0.1 0.1 0.1</scale>
        <uri>meshes/propeller_ccw.dae</uri>
      </mesh>
    </geometry>
    <material>
      <script>
        <name>Gazebo/Red</name>
        <uri>file://media/materials/scripts/gazebo.material</uri>
      </script>
    </material>
  </visual>
  <gravity>1</gravity>
  <velocity_decay/>
</link>
<joint name="r1/rotor_3_joint" type="revolute">
  <child>r1/rotor_3</child>
  <parent>r1/base_link</parent>
  <axis>
    <xyz>0 0 1</xyz>
```

```
    <limit>
      <lower>-1e+16</lower>
      <upper>1e+16</upper>
    </limit>
    <dynamics>
      <spring_reference>0</spring_reference>
      <spring_stiffness>0</spring_stiffness>
    </dynamics>
    <use_parent_model_frame>1</use_parent_model_frame>
  </axis>
</joint>

<!-- Plugins de modelos de motor -->
<plugin filename="ignition-gazebo-multicopter-motor-model-system"
        name="gz::sim::systems::MulticopterMotorModel">
  <robotNamespace>r1</robotNamespace>
  <jointName>r1/rotor_0_joint</jointName>
  <linkName>r1/rotor_0</linkName>
  <turningDirection>ccw</turningDirection>
  <timeConstantUp>0.0125</timeConstantUp>
  <timeConstantDown>0.025</timeConstantDown>
  <maxRotVelocity>850.0</maxRotVelocity>
  <motorConstant>1.28192e-05</motorConstant>
  <momentConstant>0.016</momentConstant>
  <commandSubTopic>gazebo/command/motor_speed</commandSubTopic>
  <motorNumber>0</motorNumber>
  <rotorDragCoefficient>0.0004</rotorDragCoefficient>
  <rollingMomentCoefficient>1e-06</rollingMomentCoefficient>
  <motorSpeedPubTopic>motor_speed/0</motorSpeedPubTopic>
  <rotorVelocitySlowdownSim>10</rotorVelocitySlowdownSim>
  <motorType>velocity</motorType>
</plugin>
<plugin filename="ignition-gazebo-multicopter-motor-model-system"
        name="gz::sim::systems::MulticopterMotorModel">
  <robotNamespace>r1</robotNamespace>
```

```
  <jointName>r1/rotor_1_joint</jointName>
  <linkName>r1/rotor_1</linkName>
  <turningDirection>ccw</turningDirection>
  <timeConstantUp>0.0125</timeConstantUp>
  <timeConstantDown>0.025</timeConstantDown>
  <maxRotVelocity>850.0</maxRotVelocity>
  <motorConstant>1.28192e-05</motorConstant>
  <momentConstant>0.016</momentConstant>
  <commandSubTopic>gazebo/command/motor_speed</commandSubTopic>
  <motorNumber>1</motorNumber>
  <rotorDragCoefficient>0.0004</rotorDragCoefficient>
  <rollingMomentCoefficient>1e-06</rollingMomentCoefficient>
  <motorSpeedPubTopic>motor_speed/1</motorSpeedPubTopic>
  <rotorVelocitySlowdownSim>10</rotorVelocitySlowdownSim>
  <motorType>velocity</motorType>
</plugin>
<plugin filename="ignition-gazebo-multicopter-motor-model-system"
        name="gz::sim::systems::MulticopterMotorModel">
  <robotNamespace>r1</robotNamespace>
  <jointName>r1/rotor_2_joint</jointName>
  <linkName>r1/rotor_2</linkName>
  <turningDirection>cw</turningDirection>
  <timeConstantUp>0.0125</timeConstantUp>
  <timeConstantDown>0.025</timeConstantDown>
  <maxRotVelocity>850.0</maxRotVelocity>
  <motorConstant>1.28192e-05</motorConstant>
  <momentConstant>0.016</momentConstant>
  <commandSubTopic>gazebo/command/motor_speed</commandSubTopic>
  <motorNumber>2</motorNumber>
  <rotorDragCoefficient>0.0004</rotorDragCoefficient>
  <rollingMomentCoefficient>1e-06</rollingMomentCoefficient>
  <motorSpeedPubTopic>motor_speed/2</motorSpeedPubTopic>
  <rotorVelocitySlowdownSim>10</rotorVelocitySlowdownSim>
  <motorType>velocity</motorType>
</plugin>
```

```
<plugin filename="ignition-gazebo-multicopter-motor-model-system"
        name="gz::sim::systems::MulticopterMotorModel">
  <robotNamespace>r1</robotNamespace>
  <jointName>r1/rotor_3_joint</jointName>
  <linkName>r1/rotor_3</linkName>
  <turningDirection>cw</turningDirection>
  <timeConstantUp>0.0125</timeConstantUp>
  <timeConstantDown>0.025</timeConstantDown>
  <maxRotVelocity>850.0</maxRotVelocity>
  <motorConstant>1.28192e-05</motorConstant>
  <momentConstant>0.016</momentConstant>
  <commandSubTopic>gazebo/command/motor_speed</commandSubTopic>
  <motorNumber>3</motorNumber>
  <rotorDragCoefficient>0.0004</rotorDragCoefficient>
  <rollingMomentCoefficient>1e-06</rollingMomentCoefficient>
  <motorSpeedPubTopic>motor_speed/3</motorSpeedPubTopic>
  <rotorVelocitySlowdownSim>10</rotorVelocitySlowdownSim>
  <motorType>velocity</motorType>
</plugin>

<!-- Multicopter velocity control plugin -->
<plugin
  filename="ignition-gazebo-multicopter-control-system"
  name="gz::sim::systems::MulticopterVelocityControl">
  <robotNamespace>r1</robotNamespace>
  <commandSubTopic>gazebo/command/twist</commandSubTopic>
  <enableSubTopic>enable</enableSubTopic>
  <comLinkName>r1/base_link</comLinkName>
  <velocityGain>2.7 2.7 2.7</velocityGain>
  <attitudeGain>2 3 0.15</attitudeGain>
  <angularRateGain>0.4 0.52 0.18</angularRateGain>
  <maximumLinearAcceleration>2 2 2</maximumLinearAcceleration>
  <rotorConfiguration>
    <rotor>
      <jointName>r1/rotor_0_joint</jointName>
```

```
          <forceConstant>8.54858e-06</forceConstant>
          <momentConstant>0.016</momentConstant>
          <direction>1</direction>
        </rotor>
        <rotor>
          <jointName>r1/rotor_1_joint</jointName>
          <forceConstant>8.54858e-06</forceConstant>
          <momentConstant>0.016</momentConstant>
          <direction>1</direction>
        </rotor>
        <rotor>
          <jointName>r1/rotor_2_joint</jointName>
          <forceConstant>8.54858e-06</forceConstant>
          <momentConstant>0.016</momentConstant>
          <direction>-1</direction>
        </rotor>
        <rotor>
          <jointName>r1/rotor_3_joint</jointName>
          <forceConstant>8.54858e-06</forceConstant>
          <momentConstant>0.016</momentConstant>
          <direction>-1</direction>
        </rotor>
      </rotorConfiguration>
    </plugin>

    <!--  Editor de odometría -->
    <plugin
      filename="ignition-gazebo-odometry-publisher-system"
      name="ignition::gazebo::systems::OdometryPublisher">
      <dimensions>3</dimensions>
    </plugin>

  </model>
</sdf>
```

Código del archivo my custom world.sdf.

```
<sdf version="1.6">
  <world name="multicopters">
    <physics name="4ms" type="ignored">
      <max_step_size>0.004</max_step_size>
      <real_time_factor>1.0</real_time_factor>
    </physics>
    <plugin
      filename="ignition-gazebo-physics-system"
      name="gz::sim::systems::Physics">
    </plugin>
    <plugin
      filename="ignition-gazebo-scene-broadcaster-system"
      name="gz::sim::systems::SceneBroadcaster">
    </plugin>
    <plugin
      filename="ignition-gazebo-user-commands-system"
      name="gz::sim::systems::UserCommands">
    </plugin>
    <plugin
      filename="ignition-gazebo-sensors-system"
      name="gz::sim::systems::Sensors">
      <render_engine>ogre2</render_engine>
    </plugin>

    <gui fullscreen='true'>
      <plugin name='3D View' filename='GzScene3D'>
        <ignition-gui>
          <title>3D View</title>
          <property type='bool' key='showTitleBar'>false</property>
          <property type='string' key='state'>docked</property>
        </ignition-gui>
        <engine>ogre2</engine>
```

```
  <scene>scene</scene>
  <ambient_light>0.4 0.4 0.4</ambient_light>
  <background_color>0.8 0.8 0.8</background_color>
  <camera_pose>-4 4 3 0 0 -0.7071</camera_pose>
</plugin>
<plugin name='World control' filename='WorldControl'>
  <ignition-gui>
    <title>World control</title>
    <property type='bool' key='showTitleBar'>false</property>
    <property type='bool' key='resizable'>false</property>
    <property type='double' key='height'>72</property>
    <property type='double' key='width'>121</property>
    <property type='double' key='z'>1</property>
    <property type='string' key='state'>floating</property>
    <anchors target='3D View'>
      <line own='left' target='left'/>
      <line own='bottom' target='bottom'/>
    </anchors>
  </ignition-gui>
  <play_pause>true</play_pause>
  <step>true</step>
  <start_paused>false</start_paused> <!-- '-r' must be used -->
</plugin>
<plugin name='World stats' filename='WorldStats'>
  <ignition-gui>
    <title>World stats</title>
    <property type='bool' key='showTitleBar'>false</property>
    <property type='bool' key='resizable'>false</property>
    <property type='double' key='height'>110</property>
    <property type='double' key='width'>290</property>
    <property type='double' key='z'>1</property>
    <property type='string' key='state'>floating</property>
    <anchors target='3D View'>
      <line own='right' target='right'/>
      <line own='bottom' target='bottom'/>
```

```
    </anchors>
  </ignition-gui>
  <sim_time>true</sim_time>
  <real_time>true</real_time>
  <real_time_factor>true</real_time_factor>
  <iterations>true</iterations>
</plugin>
<plugin name='Component inspector' filename='ComponentInspector'>
  <ignition-gui>
    <property key='x' type='double'>0</property>
    <property key='y' type='double'>0</property>
    <property key='width' type='double'>400</property>
    <property key='height' type='double'>375</property>
    <property key='state' type='string'>docked_collapsed</property>
  </ignition-gui>
</plugin>
<plugin name='Entity tree' filename='EntityTree'>
  <ignition-gui>
    <property key='x' type='double'>0</property>
    <property key='y' type='double'>0</property>
    <property key='width' type='double'>400</property>
    <property key='height' type='double'>375</property>
    <property key='state' type='string'>docked_collapsed</property>
  </ignition-gui>
</plugin>
<plugin name='Teleop' filename='Teleop'>
  <ignition-gui>
    <property key='x' type='double'>0</property>
    <property key='y' type='double'>0</property>
    <property key='width' type='double'>400</property>
    <property key='height' type='double'>900</property>
    <property key='state' type='string'>docked</property>
  </ignition-gui>
  <topic>/r1/gazebo/command/twist</topic>
</plugin>
```

```
</gui>

<light type="directional" name="sun">
  <cast_shadows>true</cast_shadows>
  <pose>0 0 10 0 0 0</pose>
  <diffuse>0.8 0.8 0.8 1</diffuse>
  <specular>0.2 0.2 0.2 1</specular>
  <attenuation>
    <range>1000</range>
    <constant>0.9</constant>
    <linear>0.01</linear>
    <quadratic>0.001</quadratic>
  </attenuation>
  <direction>-0.5 0.1 -0.9</direction>
</light>

<model name="ground_plane">
  <static>true</static>
  <link name="link">
    <collision name="collision">
      <geometry>
        <plane>
          <normal>0 0 1</normal>
          <size>500 500</size>
        </plane>
      </geometry>
    </collision>
    <visual name="visual">
      <geometry>
        <plane>
          <normal>0 0 1</normal>
          <size>100 100</size>
        </plane>
      </geometry>
      <material>
```

```
            <ambient>0.8 0.8 0.8 1</ambient>
            <diffuse>0.8 0.8 0.8 1</diffuse>
            <specular>0.8 0.8 0.8 1</specular>
          </material>
        </visual>
      </link>
    </model>

    <include>
     <pose>-1 0 0.1 0 0 0</pose>
     <!--name>drone</name--> <!-- Esto reasignará el nombre de todos los temas -->
      <uri>model://r1</uri>
    </include>

  </world>
</sdf>
```

Código del archivo PID circular.py.

```
#!/usr/bin/env python3

import rclpy
from rclpy.node import Node
from geometry_msgs.msg import Twist
from nav_msgs.msg import Odometry, Path
from geometry_msgs.msg import TransformStamped, PoseStamped
import tf_transformations
import tf2_ros
import math
import sys
import select
import termios
import tty
import csv
```

```
from time import time

# límites de velocidad
Vxy_max = 1.5
Vz_max  = 0.5

# periodo y radio de la trayectoria circular
T = 100
R = 2.5

# ganancias PID
Kp = 2
Ki = 0.05
Kd = 0.5

# estados globales de error
int_ex = int_ey = int_ez = 0.0
last_ex = last_ey = last_ez = 0.0

def kbhit():
    fd = sys.stdin.fileno()
    old = termios.tcgetattr(fd)
    try:
        tty.setcbreak(fd)
        return bool(select.select([sys.stdin], [], [], 0)[0])
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old)

class TrajControl(Node):
    def __init__(self):
        super().__init__('quadrotor_controller_paths_urdf')
        # Publicador de velocidad
        self.publisher = self.create_publisher(Twist, 'cmd_vel', 2)
        # Suscripción a odometría
        self.subscriber = self.create_subscription(Odometry, 'odom', self.odom_callback,
```

```
# Temporizador para el loop
self.timer = self.create_timer(0.02, self.loop)

# TF y visualización de trayectorias en RViz
ns = self.get_namespace()
self.frame_id       = ns + '/odom'           if len(ns)>1 else '/odom'
self.child_frame_id = ns + '/base_footprint' if len(ns)>1 else '/base_footprint'
self.tf_broadcaster = tf2_ros.TransformBroadcaster(self)
self.pub_path       = self.create_publisher(Path, 'path', 1)
self.pub_des_path   = self.create_publisher(Path, 'des_path', 1)
self.path_msg       = Path()
self.des_path_msg   = Path()
self.robot_pt       = PoseStamped()
self.des_pt         = PoseStamped()

# CSV de trayectoria con line buffering
self.file = open('trayectoria_circular.csv', 'w', newline='', buffering=1)
self.csv_writer = csv.writer(self.file)
self.csv_writer.writerow([
    'tiempo','des_x','des_y','des_z',
    'x','y','z','err_x','err_y','err_z'
])

# Estado inicial
self.mode              = 0    # 0=takeoff, 1=trayectoria, 2=aterrizando
self.landing_requested = False
self.takeoff_alt       = 1.2
self.start_time        = time()
self.t0                = None

# Pose y velocidad actuales
self.x = self.y = self.z = 0.0
self.vx = self.vy = self.vz = 0.0
self.yaw = 0.0
```

```
    # Mensajes reutilizables
    self.vel_msg = Twist()
    self.tranStamp = TransformStamped()

    self.get_logger().info("Presiona 'a' para aterrizar o 'q' para salir.")

def __del__(self):
    if hasattr(self, 'file'):
        self.file.close()
    self.get_logger().info("Nodo terminado.")

def odom_callback(self, msg):
    # Actualizar pose y twist actuales
    p = msg.pose.pose.position
    t = msg.twist.twist.linear
    self.x, self.y, self.z = p.x, p.y, p.z
    self.vx, self.vy, self.vz = t.x, t.y, t.z

    # Orientación en yaw
    q = msg.pose.pose.orientation
    _, _, self.yaw = tf_transformations.euler_from_quaternion((q.x, q.y, q.z, q.w))

    # Publicar transformada
    self.tranStamp.header.stamp = self.get_clock().now().to_msg()
    self.tranStamp.header.frame_id    = self.frame_id
    self.tranStamp.child_frame_id     = self.child_frame_id
    self.tranStamp.transform.translation.x = self.x
    self.tranStamp.transform.translation.y = self.y
    self.tranStamp.transform.translation.z = self.z
    self.tranStamp.transform.rotation.x    = q.x
    self.tranStamp.transform.rotation.y    = q.y
    self.tranStamp.transform.rotation.z    = q.z
    self.tranStamp.transform.rotation.w    = q.w
    self.tf_broadcaster.sendTransform(self.tranStamp)
```

```
    # Actualizar headers para paths en RViz
    self.path_msg.header = msg.header
    self.des_path_msg.header = msg.header

def movement(self, x, y, z, turn):
    # Asegurar floats para ROS
    self.vel_msg.linear.x  = float(x)
    self.vel_msg.linear.y  = float(y)
    self.vel_msg.linear.z  = float(z)
    self.vel_msg.angular.z = float(turn)

def velocity_controller(self):
    global int_ex, int_ey, int_ez, last_ex, last_ey, last_ez

    # Tiempo transcurrido
    t_now = time() - (self.t0 if self.t0 else self.start_time)
    w     = 2 * math.pi / T
    c     = 0.5

    # Trayectoria circular deseada y velocidades feedforward
    Xd  = R * math.cos(w * t_now)
    Yd  = R * math.sin(w * t_now)
    Zd  = 1.5 + c * math.sin(w * t_now)
    Xdp = -R * w * math.sin(w * t_now)
    Ydp =  R * w * math.cos(w * t_now)
    Zdp =  c * w * math.cos(w * t_now)

    # Errores (real - deseado)
    ex = self.x - Xd
    ey = self.y - Yd
    ez = self.z - Zd

    # Integrales y derivadas
    int_ex += ex * 0.02
    int_ey += ey * 0.02
```

```
int_ez += ez * 0.02
der_ex = (ex - last_ex) / 0.02
der_ey = (ey - last_ey) / 0.02
der_ez = (ez - last_ez) / 0.02
last_ex, last_ey, last_ez = ex, ey, ez

# Control PID + feedforward
Ux = Xdp - (Kp * ex + Ki * int_ex + Kd * der_ex)
Uy = Ydp - (Kp * ey + Ki * int_ey + Kd * der_ey)
Uz = Zdp - (Kp * ez + Ki * int_ez + Kd * der_ez)

# Velocidades en cuerpo
Vx = max(min(Ux * math.cos(self.yaw) + Uy * math.sin(self.yaw), Vxy_max), -Vxy_m
Vy = max(min(-Ux * math.sin(self.yaw) + Uy * math.cos(self.yaw), Vxy_max), -Vxy_
Vz = max(min(Uz, Vz_max), -Vz_max)

# Aplicar vel.
self.movement(Vx, Vy, Vz, 0.0)
self.publisher.publish(self.vel_msg)

# Construcción de paths en RViz
self.robot_pt.pose.position.x = self.x
self.robot_pt.pose.position.y = self.y
self.robot_pt.pose.position.z = self.z
self.des_pt.pose.position.x   = Xd
self.des_pt.pose.position.y   = Yd
self.des_pt.pose.position.z   = Zd
self.path_msg.poses.append(self.robot_pt)
self.des_path_msg.poses.append(self.des_pt)
if len(self.path_msg.poses) > 800:
    self.path_msg.poses.pop(0)

self.pub_path.publish(self.path_msg)
self.pub_des_path.publish(self.des_path_msg)
```

```
        # Cálculo de errores signados para logging (deseado - real)
        err_x = -ex
        err_y = -ey
        err_z = -ez

        # Escritura en CSV y flush
        self.csv_writer.writerow([
            t_now, Xd, Yd, Zd,
            self.x, self.y, self.z,
            err_x, err_y, err_z
        ])
        self.file.flush()

    def loop(self):
        # Lectura de teclado
        if kbhit():
            c = sys.stdin.read(1)
            if c == 'a' and not self.landing_requested:
                self.landing_requested = True
                self.mode = 2
                self.get_logger().info("==> Aterrizando...")
            elif c == 'q':
                self.get_logger().info("[EMERGENCIA] abortando nodo.")
                self.file.flush()
                rclpy.shutdown()
                return

        # Estados
        if self.mode == 0:
            if self.z >= self.takeoff_alt:
                self.mode = 1
                self.t0 = time()
                self.get_logger().info("==> Altitud alcanzada, comenzando trayectoria.")
            else:
                self.movement(0.0, 0.0, 0.2, 0.0)
```

```
                self.publisher.publish(self.vel_msg)

        elif self.mode == 1:
            if (time() - self.t0) >= T:
                self.get_logger().info("Trayectoria completada: iniciando aterrizaje.")
                self.mode = 2
            else:
                self.velocity_controller()

        elif self.mode == 2:
            self.movement(0.0, 0.0, -0.2, 0.0)
            self.publisher.publish(self.vel_msg)
            if self.z < 0.2:
                self.get_logger().info("Aterrizaje completado. Cerrando nodo.")
                rclpy.shutdown()


def main(args=None):
    rclpy.init(args=args)
    node = TrajControl()
    rclpy.spin(node)
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Código del archivo train SAC.py.

```
#!/usr/bin/env python3

import os
import numpy as np
```

```
import rclpy
from geometry_msgs.msg import Twist
from nav_msgs.msg import Odometry

import gymnasium as gym
from gymnasium import spaces
from gymnasium.wrappers import TimeLimit
from stable_baselines3 import SAC
from stable_baselines3.common.monitor import Monitor
from stable_baselines3.common.callbacks import EvalCallback, BaseCallback
from stable_baselines3.common.logger import configure

class ConsoleCallback(BaseCallback):
    def __init__(self, print_freq: int = 500):
        super().__init__(verbose=0)
        self.print_freq = print_freq

    def _on_step(self) -> bool:
        if self.num_timesteps and self.num_timesteps % self.print_freq == 0:
            print(f"[Train] step {self.num_timesteps}")
        return True

class DroneHoverEnv(gym.Env):
    metadata = {'render.modes': []}

    def __init__(self, ros_node_name: str):
        super().__init__()
        # Nodo ROS2, editor y abonado
        self.node     = rclpy.create_node(ros_node_name)
        self.cmd_pub  = self.node.create_publisher(Twist, '/r1/cmd_vel', 1)
        self.odom_sub = self.node.create_subscription(
            Odometry, '/r1/odom', self._odom_cb, 1
        )

        # Límites del espacio de acción
```

```
        Vxy, Vz = 1.5, 0.5
        self.action_space = spaces.Box(
            low  = np.array([-Vxy, -Vxy, -Vz, -1.0], dtype=np.float32),
            high = np.array([ Vxy,  Vxy,  Vz,  1.0], dtype=np.float32),
            dtype=np.float32
        )
        # Observacion: [x, y, z, vx, vy, vz]
        self.observation_space = spaces.Box(
            low=-np.inf, high=np.inf, shape=(6,), dtype=np.float32
        )

        # Altitud objetivo de vuelo
        self.takeoff_alt = 1.2

        # Variables estatales
        self.pos       = np.zeros(3, dtype=np.float32)
        self.vel       = np.zeros(3, dtype=np.float32)
        self.taken_off = False

        # Para el cálculo de las sanciones
        self.last_action = np.zeros(4, dtype=np.float32)

        # Gira una vez para obtener el odom inicial
        rclpy.spin_once(self.node, timeout_sec=0.1)

    def _odom_cb(self, msg: Odometry):
        p = msg.pose.pose.position
        t = msg.twist.twist.linear
        self.pos = np.array([p.x, p.y, p.z], dtype=np.float32)
        self.vel = np.array([t.x, t.y, t.z], dtype=np.float32)

    def reset(self, *, seed=None, options=None):
        self.taken_off   = False
        zero = Twist()
        self.cmd_pub.publish(zero)
```

```
        self.last_action = np.zeros(4, dtype=np.float32)
        rclpy.spin_once(self.node, timeout_sec=0.02)
        obs = np.concatenate([self.pos, self.vel]).astype(np.float32)
        return obs, {}

    def step(self, action):
        # Ajustar las acciones a los límites físicos
        action = np.clip(action, self.action_space.low, self.action_space.high)
        self.last_action = action.copy()

        cmd = Twist()

        # Comprobar y actualizar el estado del despegue
        if not self.taken_off and self.pos[2] >= self.takeoff_alt:
            self.taken_off = True

        # Bloquear XY hasta que se complete el despegue
        if not self.taken_off:
            cmd.linear.x = 0.0
            cmd.linear.y = 0.0
            cmd.linear.z = float(action[2])
        else:
            cmd.linear.x = float(action[0])
            cmd.linear.y = float(action[1])
            # Bloquear el ascenso/descenso tras alcanzar el punto de planeo
            cmd.linear.z = 0.0

        cmd.angular.z = float(action[3])
        self.cmd_pub.publish(cmd)

        rclpy.spin_once(self.node, timeout_sec=0.02)

        obs, reward, done = None, None, None
        obs, reward, done = None, None, None
        obs = np.concatenate([self.pos, self.vel]).astype(np.float32)
```

```
    reward, done = self._compute_reward()
    return obs, reward, done, False, {}

def _compute_reward(self):
    # Bonificación por supervivencia
    reward = 0.1

    z = float(self.pos[2])
    xy_err = np.hypot(self.pos[0], self.pos[1])
    target = np.array([0.0, 0.0, self.takeoff_alt], dtype=np.float32)
    pos_err = np.linalg.norm(self.pos - target)
    vel_err = np.linalg.norm(self.vel)
    mse_err = float(np.mean((self.pos - target)**2))
    effort  = float(np.sum(self.last_action**2))
    ang_cmd = float(abs(self.last_action[3]))

    # Weights
    K_xy = 1.0     # error horizontal
    K_z  = 1.0     # error vertical
    K_mse = 0.5    # mse
    K_eff = 0.01   # esfuerzo
    K_ang = 0.1    # angular cmd

    # Computar recompensa
    if not self.taken_off:
        reward += 10.0 * z - 1.0 * xy_err
        done = False
    else:
        # Penalizaciones en la fase de planeo
        reward -= K_xy  * xy_err
        reward -= K_z   * abs(z - self.takeoff_alt)
        reward -= K_mse * mse_err
        reward -= K_eff * effort
        reward -= K_ang * ang_cmd
        # Condición de éxito
```

```
            done = (pos_err < 0.05) and (vel_err < 0.05)

        # Rescisión anticipada en caso de fallo
        if z < 0.1 or abs(z - self.takeoff_alt) > 0.5:
            reward -= 50.0
            done = True

        return float(reward), bool(done)

    def close(self):
        self.node.destroy_node()

if __name__ == '__main__':
    rclpy.init()
    os.makedirs('logs/train',      exist_ok=True)
    os.makedirs('logs/eval',       exist_ok=True)
    os.makedirs('logs/best_model', exist_ok=True)
    os.makedirs('logs/tb',         exist_ok=True)

    configure("logs", ["stdout","tensorboard"])

    raw_train = DroneHoverEnv('train')
    train_env = Monitor(TimeLimit(raw_train, max_episode_steps=2000), 'logs/train')

    raw_eval = DroneHoverEnv('eval')
    eval_env = Monitor(TimeLimit(raw_eval, max_episode_steps=2000), 'logs/eval')

    eval_cb = EvalCallback(
        eval_env,
        best_model_save_path='logs/best_model',
        log_path='logs/eval',
        eval_freq=500,
        n_eval_episodes=5,
        deterministic=True,
        verbose=1
```

```
)
console_cb = ConsoleCallback(print_freq=100)

model = SAC(
    policy='MlpPolicy',
    env=train_env,
    verbose=1,
    tensorboard_log='logs/tb'
)
model.learn(
    total_timesteps=100_000,
    callback=[eval_cb, console_cb]
)

model.save('logs/best_model/sac_drone_hover')
train_env.close()
eval_env.close()
rclpy.shutdown()
```

Bibliografía

[1] A. Yousaf, "Review on the stabilization of non linear systems achieved by output feedback control technique," *arXiv (Cornell University)*, Jan. 2022.

[2] L. Physics, "Introduction to nonlinear systems and chaos," Nov. 2017.

[3] S. Saat, S. K. Nguang, and A. Nasiri, "Chapter 1 - introduction - non-linear systems," 2017.

[4] I. S. Trenev, , and A. Y. Kustov, "Movement stabilization of the parrot mambo quadcopter along a given trajectory based on pid controllers," *IFAC-PapersOnLine*, vol. 54, pp. 227–232, Jan 2021.

[5] A. Saxena and R. Negi, "Trajectory tracking of quadcopter using type-2 fuzzy-aided supertwisting algorithm," in *2023 IEEE 12th International Conference on Communication Systems and Network Technologies (CSNT)*, pp. 357–362, Apr. 2023.

[6] B. P. Amiruddin, E. Iskandar, A. Fatoni, and A. Santoso, "Deep learning based system identification of quadcopter unmanned aerial vehicle," Nov. 2020.

[7] K. Agrawal and P. Shrivastav, "Multi-rotors: A revolution in unmanned aerial vehicle," *International Journal of Science and Research (IJSR)*, vol. 4, pp. 1800–1804, Nov. 2015.

[8] I. Lopez-Sanchez and J. Moreno-Valenzuela, "Pid control of quadrotor uavs: A survey," Jan. 2023.

[9] N. P. Agustina and P. A. Darwito, "Autonomous quadcopter trajectory tracking and stabilization using control system based on sliding mode control and kalman filter," vol. 8, pp. 489–493, Jul. 2023.

[10] A. Eltayeb, M. F. Rahmat, M. A. M. Eltoum, M. Ibrahim, and M. A. M. Basri, "Trajectory tracking for the quadcopter uav utilizing fuzzy pid control approach," in *2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, vol. 42, pp. 1–6, 02 2021.

[11] R. Guardeño, M. J. López, and V. M. Sánchez, "Mimo pid controller tuning method for quadrotor based on lqr/lqg theory," *Robotics*, vol. 8, pp. 36–36, 05 2019.

[12] V. P. Tran, F. Santoso, M. Garratt, and I. R. Petersen, "Fuzzy self-tuning of strictly negative-imaginary controllers for trajectory tracking of a quadcopter unmanned aerial vehicle," *IEEE Transactions on Industrial Electronics*, vol. 68, pp. 5036–5045, 04 2020.

[13] T. Shan, Y. Wang, C. Zhao, Y. Li, G. Zhang, and Q. Zhu, "Multi-uav wrsn charging path planning based on improved heed and ia-drl," 01 2023.

[14] B. Yan, Y. Wei, S. Liu, W. Huang, R. Feng, and X. Chen, "A review of current studies on the unmanned aerial vehicle-based moving target tracking methods," 01 2025.

[15] X. Olaz, D. Aláez, M. Prieto, J. Villadangos, and J. J. Astráin, "Quadcopter neural controller for take-off and landing in windy environments," *Expert Systems with Applications*, vol. 225, pp. 120146–120146, 04 2023.

[16] R. Harisankar, A. Kaushik, and S. S. Muni, "Path planning for multi-quadrotor 3d boundary surveillance using non-autonomous discrete memristor hyperchaotic system," *Franklin Open*, pp. 100239–100239, 03 2025.

[17] F. Çakmak Bolat and M. C. Avci, "Development and validation of a human-machine interface for unmanned aerial vehicle (uav) control via hand gesture teleoperation," *Expert Systems with Applications*, pp. 126828–126828, 02 2025.

[18] A. R. Tajammal and M. Habib, "Autonomous control of a quadcopter using machine learning algorithm," pp. 1–8, 12 2021.

[19] Y. Belhadjer, B. Naceri, H. A. Abbas, S. Haroun, K. Laroussi, and O. Fergani, "Dynamic modeling and control of an s500 uav using proportional-integral-derivative (pid) algorithm," pp. 1–8, 05 2023.

[20] R. T. Dan, U. Shah, and W. Hussain, "Development process of a smart uav for autonomous target detection," 01 2018.

[21] X. Zhang, X. Li, K. Wang, and Y. Lu, "A survey of modelling and identification of quadrotor robot," *Abstract and Applied Analysis*, vol. 2014, pp. 1–16, 01 2014.

[22] R. Beneder, P. Schmitt, and C. Környefalvy, "A model-based approach for remote development of embedded software for object avoidance applications," 02 2023.

[23] Z. He and L. Zhao, "A simple attitude control of quadrotor helicopter based on zieglernichols rules for tuning pd parameters," *The Scientific World JOURNAL*, vol. 2014, pp. 1–13, 01 2014.

[24] C. Kownacki, "Artificial potential field based trajectory tracking for quadcopter uav moving targets," *Sensors*, vol. 24, pp. 1343–1343, 02 2024.

[25] S. Bertrand, N. Guénard, T. Hamel, H. Piet-Lahanier, and L. Eck, "A hierarchical controller for miniature vtol uavs: Design and stability analysis using singular perturbation theory," *Control Engineering Practice*, vol. 19, pp. 1099–1108, 07 2011.

[26] D. L. Vinokursky, , P. V. Samoylov, K. Ganshin, and O. A. Baklanova, "Model predictive control for path planning of uav group," vol. 1155, pp. 12092–12092, IOP Publishing, 06 2021.

[27] C. Llanes, J. Netter, K. G. Vamvoudakis, and S. Coogan, "Experimental validation on aerial vehicles of real-time motion planning with continuous-time q-learning," *IFAC-PapersOnLine*, vol. 56, pp. 31–36, 01 2023.

[28] S. Singh, K. Kishore, S. Dalai, M. Irfan, S. Singh, S. A. Akbar, G. Sachdeva, and R. Yechangunja, "Cacla-based local path planner for drones navigating unknown indoor corridors," *IEEE Intelligent Systems*, vol. 37, pp. 32–41, 05 2022.

[29] S. Teng, D. Peng, Y. Li, B. Li, X. Hu, Z. Xuanyuan, L. Chen, Y. Ai, L. Li, and F. Wang, "Motion planning for autonomous driving: The state of the art and future perspectives," *arXiv (Cornell University)*, 01 2023.

[30] T. T. Hoang, D. Hiep, P.-P. Duong, N. T. Van, B. G. Duong, and V. Tran-Quang, "Proposal of algorithms for navigation and obstacles avoidance of autonomous mobile robot," *2022 IEEE 17th Conference on Industrial Electronics and Applications (ICIEA)*, pp. 1308–1313, 06 2013.

[31] T. T. Mac, C. Copot, D. T. Tran, and R. D. Keyser, "Heuristic approaches in robot path planning: A survey," *Robotics and Autonomous Systems*, vol. 86, pp. 13–28, 09 2016.

[32] S. I. Abdelmaksoud, M. Mailah, and A. M. Abdallah, "Improving disturbance rejection capability for a quadcopter uav system using self-regulating fuzzy pid controller," vol. 40, pp. 1–6, 02 2021.

[33] S. Abdelhay and A. Zakriti, "Modeling of a quadcopter trajectory tracking system using pid controller," *Procedia Manufacturing*, vol. 32, pp. 564–571, 01 2019.

[34] T. K. Priyambodo, A. Dharmawan, and A. E. Putra, "Pid self tuning control based on mamdani fuzzy logic control for quadrotor stabilization," *AIP conference proceedings*, vol. 1705, pp. 20013–20013, 01 2016.

[35] C. Chronis, G. C. Anagnostopoulos, E. Politi, G. Dimitrakopoulos, and I. Varlamis, "Dynamic navigation in unconstrained environments using reinforcement learning algorithms," *IEEE Access*, vol. 11, pp. 117984–118001, 01 2023.

[36] N. Khlif, N. Khraief, and S. Belghith, "Reinforcement learning for mobile robot navigation: An overview," pp. 1–7, 07 2022.

[37] A. Beishenalieva and S. Yoo, "Uav path planning for data gathering in wireless sensor networks: Spatial and temporal substate-based q-learning," *IEEE Internet of Things Journal*, vol. 11, pp. 9572–9586, 10 2023.

[38] T. Yang, F. Yang, and D. Li, "A new autonomous method of drone path planning based on multiple strategies for avoiding obstacles with high speed and high density," *Drones*, vol. 8, pp. 205–205, 05 2024.

[39] G.-T. Tu and J. Juang, "Path planning and obstacle avoidance based on reinforcement learning for uav application," 08 2021.

[40] G.-T. Tu and J. Juang, "Uav path planning and obstacle avoidance based on reinforcement learning in 3d environments," *Actuators*, vol. 12, p. 57, 01 2023.

[41] M. Hayajneh, M. H. Garibeh, A. B. Younes, and M. Garratt, "Unmanned aerial vehicle path planning using acceleration-based potential field methods," *Electronics*, vol. 14, pp. 176–176, 01 2025.

[42] N. Jahan, T. B. M. Niloy, J. F. Silvi, M. Hasan, I. J. Nashia, and R. Khan, "Development of an iot-based firefighting drone for enhanced safety and efficiency in fire suppression," *Measurement and Control*, vol. 57, pp. 1464–1479, 04 2024.

[43] P. V. Quan, D. X. Ba, C. Truong, N. P. Luu, V. Q. Huy, and N. T. Duc, "An adaptive robust nonlinear control approach of a quadcopter with disturbance observer," pp. 476–481, 11 2022.

[44] I. S. Asti, T. Agustinah, and A. Santoso, "Obstacle avoidance with energy efficiency and distance deviation using knn algorithm for quadcopter."

[45] B. Çetinsaya, D. Reiners, and C. Cruz-Neira, "From pid to swarms: A decade of advancements in drone control and path planning - a systematic review (2013–2023)," 06 2024.

[46] Z. Bi, X. Guo, J. Wang, S. Qin, and G. Liu, "Truck-drone delivery optimization based on multi-agent reinforcement learning," *Drones*, vol. 8, pp. 27–27, 01 2024.

[47] A. Brito, V. Brito, L. B. Palma, F. Coito, and P. Gil, "Trajectory control approaches for a fault tolerant quadcopter," pp. 13–18, 05 2018.

[48] C. M. E, O. GOUGEON, D.-T. NGUYEN, and D. S. E, "Modeling and control of a quadcopter flying in a wind field: A comparison between lqr and structured h control techniques."

[49] N. T. Hegde, A. C. Vaz, and C. G. Nayak, "Closed loop performance analysis of classical pid and robust h-infinity controller for vtol unmanned quad tiltrotor aerial vehicle," *International Journal of Mechanics*, vol. 15, pp. 211–222, 10 2021.

[50] A. S. Salem, C. M. Elias, O. M. Shehata, and E. I. Morgan, "Investigation of various optimization algorithms in tuning fuzzy logic-based trajectory tracking control of quadcopter," pp. 82–87, 11 2020.

[51] B. Breese, M. Kumar, M. Bolender, and D. W. Casbeer, "Physics-based neural networks for modeling control of aerial vehicles."

[52] S. A. Hoseini, J. Hassan, A. Bokani, and S. S. Kanhere, "Trajectory optimization of flying energy sources using q-learning to recharge hotspot uavs," pp. 683–688, 07 2020.

[53] F. Ahmad, P. Kumar, and P. P. Patil, "Vibration characteristics based pre-stress analysis of a quadcopter's body frame," *Materials Today Proceedings*, vol. 46, pp. 10329–10333, 01 2021.

[54] M. F. Zulkifli, Z. M. Razlan, A. B. Shahriman, I. Zunaidi, W. Khairunizam, and N. Z. Noriman, "Dynamic analysis of uav's motor support bar length control system," vol. 557, pp. 12053–12053, IOP Publishing, 06 2019.

[55] T. Zhao and W. Li, "Design configuration and technical application of rotary-wing unmanned aerial vehicles," *Deleted Journal*, vol. 1, 11 2022.

[56] Z. Longchao, J. Xu, and L. Yan, "Design of four axis aircraft based on arduino used in city-wide food delivery," *Advance Journal of Food Science and Technology*, vol. 11, pp. 153–157, 04 2016.

[57] Y. Sun, M. Yu, L. Wang, T. Li, and M. Dong, "A deep-learning-based gps signal spoofing detection method for small uavs," *Drones*, vol. 7, pp. 370–370, 06 2023.

[58] G. A. B. D. L. Rios, G. Urrea, and J. A. N. Navia, "Propuesta para la aplicación de métodos ágiles en el diseño conceptual de un dron," *Ciencia y Poder Aéreo*, vol. 15, pp. 110–121, 11 2020.

[59] L. Colombo and J. I. Giribet, "Learning-based fault-tolerant control for an hexarotor with model uncertainty," *IEEE Transactions on Control Systems Technology*, vol. 32, pp. 672–679, 10 2023.

[60] Y. Z. Jembre, Y. W. Nugroho, M. T. R. Khan, M. Attique, R. Paul, S. B. H. Shah, and B. Kim, "Evaluation of reinforcement and deep learning algorithms in controlling unmanned aerial vehicles," *Applied Sciences*, vol. 11, pp. 7240–7240, 08 2021.

[61] T. Chaffre, J. Moras, A. Chan-Hon-Tong, J. Marzat, K. Sammut, G. L. Chenadec, and B. Clément, "Learning-based vs model-free adaptive control of a mav under wind gust," *arXiv (Cornell University)*, 01 2021.

[62] C. Guerrero, F. M. E. Montero, and G. B. C. Nieto, "Diseño y desarrollo de gimbal impreso en 3d para la adaptación de métodos de escaneo 3d sobre drones," *Killkana Técnica*, vol. 6, pp. 29–39, 12 2022.

[63] W. L. Pinto, L. E. S. A. Filho, C. L. Nascimento, S. R. B. dos Santos, and W. C. Cunha, "Planning of the coordination of multiple quadrotors applied to the transport of materials," pp. 1–7, 04 2021.

[64] N. Bernini, M. Bessa, R. Delmas, A. Gold, Éric Goubault, R. Pennec, S. Putot, and F. X. Sillion, "Reinforcement learning with formal performance metrics for quadcopter attitude control under non-nominal contexts," *Engineering Applications of Artificial Intelligence*, vol. 127, pp. 107090–107090, 10 2023.

[65] C. Chronis, G. C. Anagnostopoulos, E. Politi, A. Garyfallou, I. Varlamis, and G. Dimitrakopoulos, "Path planning of autonomous uavs using reinforcement learning," *Journal of Physics Conference Series*, vol. 2526, pp. 12088–12088, 06 2023.