Dinámica y navegación autónoma a través de cultivos de palma para dron hexarrotor basada en aprendizaje por refuerzo

Oscar Miguel Navas Torres

Trabajo de grado para optar al título de magíster en ingeniería mecánica

Director

Carlos Borrás Pinilla

PhD, y MSc en Ingeniería Mecánica

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingeniería Mecánica

Maestría en Ingeniería Mecánica

Bucaramanga

2025

Dedicatoria

A Dios y mi familia, por darme la fuerza y el apoyo para llevar a cabo mis estudios de pregrado y posgrado.

Agradecimientos

A mi padre y a mi madre, por sus buenos consejos y por permitirme tener un espacio en el cual llevar a cabo mi investigación sin interrupciones.

Al profesor Carlos Borrás, por el tiempo dedicado a la revisión de documentos, por su guía durante las fases del proyecto y por todo el conocimiento que nos compartió en sus asignaturas. Desde el punto de vista personal, también agradezco su apoyo en los momentos de dificultad, en los que estuvo presente, me respaldó y orientó para resolver todos los problemas.

A la Universidad Industrial de Santander por su apoyo económico en este programa de investigación subsidiado, por ayudarme con gastos de sostenimiento y asistencia a congresos, por permitirme trabajar como docente y por la formación que me ha brindado su planta profesoral, no solo en la maestría sino también en el pregrado.

Tabla de Contenido

Pág.

Lista de Tablas

Pág.

Lista de Figuras

Pág.

Lista de Apéndices

pág.

Resumen

Título: Dinámica y navegación autónoma a través de cultivos de palma para dron hexarrotor basada en aprendizaje por refuerzo[*]

Autor: Oscar Miguel Navas Torres[**]

Palabras Clave: VANT, Control LQG, Mecánica Lagrangiana, Inteligencia artificial, Aprendizaje por Refuerzo, dron hexacóptero, planeación de trayectoria local, palma de aceite

Descripción: Este trabajo detalla la metodología utilizada para desarrollar un sistema de navegación autónoma para un dron hexarrotor, diseñado para operar en plantaciones simuladas de palma de aceite. La primera fase del proyecto consistió en la creación de un modelo dinámico no lineal, utilizando la dinámica de Lagrange que incluye fuerzas de vientos externos y efectos giroscópicos. Matlab fue empleado para realizar operaciones simbólicas permitiendo de esta forma la obtención eficiente de las ecuaciones de movimiento del dron. El sistema de navegación está compuesto por dos controladores: un controlador LQG-Integral y un controlador basado en aprendizaje por refuerzo. El controlador LQG-Integral (Linear Quadratic Gaussian) se encargó del seguimiento preciso de la trayectoria para las coordenadas de posición del dron. Este controlador fue comparado con controladores PID y LQG en términos de precisión, estabilidad y respuesta dinámica. Los resultados demostraron que el LQG-Integral es el más efectivo, eliminando errores en estado estable y proporcionando un seguimiento de trayectoria superior. Por otro lado, el controlador de alto nivel basado en la estrategia de aprendizaje por refuerzo DDDQN (Dueling Double Deep Q-Network), fue responsable de la planificación de trayectorias locales y la evasión de obstáculos. Este controlador permitió al dron navegar de manera eficiente en el entorno simulado, evitando obstáculos de forma efectiva. La implementación y prueba del sistema en el simulador Gazebo permitieron validar el comportamiento del dron en un entorno con dinámica realista, sin necesidad de pruebas en drones y cultivo reales. Durante el entrenamiento, el agente autónomo mostró una notable mejora en la navegación, reduciendo la longitud de las rutas y minimizando la activación de las rutinas de evasión de emergencia, evidenciando un avance constante en su rendimiento y eficiencia.

[*] Trabajo de grado de Maestría

[**] Facultad de Ingenierías Físico Mecánicas. Escuela de Ingeniería Mecánica. Director: Carlos Borrás Pinilla Ph.D. MSc.

Abstract

Title: Dynamics and autonomous navigation through oil palm crops for hexacopter drones based on reinforcement learning [*]

Author(s): Oscar Miguel Navas Torres[**]

Key Words: UAV, LQG Control, Lagrangian Mechanics, Artificial Intelligence, Reinforcement Learning, Hexacopter Drone, Local Trajectory Planning, Oil Palm.

Description: This work details the methodology used to develop an autonomous navigation system for a hexacopter drone, designed to operate in simulated oil palm plantations. The first phase of the project involved creating a nonlinear dynamic model using Lagrangian dynamics, which includes external wind forces and gyroscopic effects. Matlab was used to perform symbolic operations, allowing for the efficient derivation of the drone's equations of motion. The navigation system consists of two controllers: an LQG-Integral controller and a reinforcement learning-based controller. The LQG-Integral (Linear Quadratic Gaussian) controller was responsible for precise trajectory tracking of the drone's position coordinates. This controller was compared to PID and LQG controllers in terms of precision, stability, and dynamic response. The results showed that the LQG-Integral was the most effective, eliminating steady-state errors and providing superior trajectory tracking. On the other hand, the high-level controller, based on the reinforcement learning strategy DDDQN (Dueling Double Deep Q-Network), was responsible for local trajectory planning and obstacle avoidance. This controller allowed the drone to navigate efficiently in the simulated environment, effectively avoiding obstacles. The implementation and testing of the system in the Gazebo simulator validated the drone's behavior in a realistic dynamic environment without the need for tests on real drones and plantations. During training, the autonomous agent showed significant improvement in navigation, reducing route lengths and minimizing the activation of emergency avoidance routines, demonstrating steady progress in its performance and efficiency.

---

[*] M.Sc. Thesis

[**]Faculty of Physical Mechanical Engineering. Mechanical engineering school. Director: Carlos Borrás Pinilla Ph.D. MSc.

1. Introducción

Los Vehículos Aéreos No Tripulados (VANT) de tipo multirrotor han adquirido una relevancia crucial en diversos sectores como la agricultura (Rejeb et al., 2022), el transporte (Raghunatha et al., 2023), la vigilancia (Mishra et al., 2020) y el monitoreo. Estos dispositivos presentan ventajas notables, como despegue vertical, movilidad omnidireccional y costos de fabricación asequibles. Lo cual ha incentivado a un aumento en la investigación enfocada hacia drones, abarcando desde el modelado matemático de la dinámica hasta el desarrollo de sistemas de control y la evaluación de su implementación práctica en distintas industrias. En este trabajo de investigación se dan las bases para la creación de controladores de vuelo para drones autónomos, y se utilizan técnicas de inteligencia artificial para el entrenamiento de un planificador de vuelo local, esto se realiza en un ambiente simulado de cultivos de palma de aceite. Está sección se divide en: motivación, planteamiento del problema, justificación de la investigación, alcance y metodología llevada a cabo para el desarrollo de la investigación.

1.1 Motivación

Este proyecto es desarrollado por miembros del grupo de investigación en Sistemas Dinámicos Multifísicos, Control y Robótica (DICBOT) de la Universidad Industrial de Santander. El grupo tiene como propósito investigar y generar conocimiento en las áreas de sistemas dinámicos, control y robótica, con el fin de contribuir activamente al avance científico e industrial del país.

En este contexto, el presente trabajo busca aportar a la industria agrícola colombiana mediante el desarrollo de un sistema de navegación autónoma para drones, enfocado en cultivos

de palma de aceite. Este proyecto surge de la necesidad de implementar tecnologías avanzadas que mejoren la productividad y sostenibilidad del campo colombiano, donde la automatización de algunas tareas permite incrementar la eficiencia de la gestión del cultivo.

Para ello, aprovechando el conocimiento adquirido en la maestría en áreas cómo dinámica, control e inteligencia artificial, se ha elaborado un modelo matemático del sistema dinámico de un dron, se han implementado estrategias de control para seguimiento de trayectoria y se han utilizado técnicas de inteligencia artificial para el entrenamiento de un controlador de alto nivel para la planificación de vuelo. Aunque la investigación se realizó en entornos simulados, los resultados obtenidos podrán aplicarse a drones reales en trabajos futuros.

Además, los paquetes desarrollados en ROS 2, junto con los resultados obtenidos, serán compartidos con la comunidad académica y científica. Esto permitirá que nuevos miembros del grupo de investigación o de la comunidad universitaria continúen la investigación en drones simulados, fomentando el avance en el área. Estos recursos también servirán como material didáctico en la enseñanza de sistemas dinámicos y robótica dentro de la universidad. Finalmente, los resultados se presentaron en dos ponencias en congresos internacionales tituladas *Modelado y seguimiento de trayectorias para dron hexacóptero: Evaluación comparativa de estrategias PID, LQG y PI-LQG* (Navas & Borrás, 2024) y *Obstacle avoidance and local path planning based on reinforcement learning for hexacopter drone in simulated environments* (Navas et al., 2024), y se documentan en este informe final.

1.2 Planteamiento del problema

Este trabajo de investigación tiene como objetivo desarrollar un sistema de navegación autónoma para un dron hexarrotor, capaz de navegar de manera eficiente a través de cultivos de

palma de aceite africana, con validación en entornos simulados. En esta sección se presenta el planteamiento del problema, comenzando con una descripción de los cultivos de palma, seguido de la explicación del problema que se busca abordar, las razones para la elección del vehículo, y los motivo por los cuales se decidió desarrollar la investigación utilizando un entorno simulado.

1.2.1 Descripción de los cultivos de palma

Los cultivos de palma de aceite producen entre 6 y 10 veces más aceite por hectárea que otras oleaginosas utilizadas para la extracción de aceite. Además del uso del aceite para la producción de alimentos, estos cultivos generan subproductos empleados en la fabricación de biodiésel y cosméticos. Por estas razones la palma de aceite se ha convertido en un cultivo de gran importancia a nivel global, donde Colombia destaca como el cuarto mayor productor de aceite de palma en el mundo y el primero en América (Fedepalma, 2024).

A nivel nacional, este cultivo se encuentra en 21 departamentos y 161 municipios, siendo un importante generador de empleo en las zonas rurales. A nivel regional, el departamento de Santander produjo 99.704 toneladas de aceite en 2020, lo que representa el 14,26% de la producción nacional (Fedepalma, 2022) constituyendo un aporte significativo a la producción agrícola del departamento.

La palma de aceite produce su fruto en la copa, y su altura varía según su edad. Estas palmas tienen una vida económica de 25 años, alcanzando aproximadamente 12 metros de altura antes de ser reemplazadas. En un cultivo de decenas de hectáreas, se pueden dividir las zonas por edades, lo que hace que las palmas produzcan frutos a diferentes alturas dentro del mismo cultivo.

Los cultivos de palma se suelen sembrar en una distribución regular de forma triangular, está distribución se conoce como tresbolillo y la distancia recomendada es de 9 metros entre

palmas (Grupo Jaremar, 2016, p. 12) . En la Figura 1 se muestra una fotografía aérea de un cultivo de palma de aceite.

Figura 1

Distribución de palmas dentro de un cultivo

Nota: Fotografía aérea de un cultivo de palma de aceite. Tomada de (Chiriacò et al., 2022)

Las actividades en el interior del cultivo, como la recolección de frutos, requieren desplazamientos a pie o mediante vehículos terrestres, ya sean motorizados o de tracción animal. Estos desplazamientos conllevan riesgos para los operarios, quienes pueden sufrir caídas, accidentes e incluso ataques de animales salvajes, comunes en áreas rurales (Lee et al., 2019). Además, el terreno irregular donde se cultivan las palmas de aceite acelera el desgaste de los componentes de los vehículos terrestres. Algunas actividades que se realizan al interior del cultivo que pueden llegar a automatizarse y que son esenciales incluyen: la polinización, fumigación, inspección del grado de madurez de los frutos, así como la vigilancia y monitoreo del estado de salud de las palmas.

1.2.2 Problema que se busca abordar

Este proyecto se enfoca en la navegación autónoma dentro de cultivos de palma de aceite, utilizando vehículos aéreos no tripulados para facilitar tareas que deben realizarse a diferentes alturas, como la inspección del grado de madurez del fruto o el estado de salud de las palmas, y evitar el desplazamiento terrestre en labores de vigilancia. Se busca que la navegación sea autónoma para eliminar la necesidad de un piloto para el dron, lo que también reduce los riesgos asociados al desplazamiento humano dentro del cultivo.

1.2.3 Hipótesis

Se plantea que la combinación de un controlador LQG-integral para el seguimiento de trayectoria, junto con un planificador de trayectoria local basado en aprendizaje por refuerzo, será eficaz para realizar desplazamientos dentro de los cultivos de palma de aceite utilizando un dron hexarrotor. Aunque el sistema se validará en un entorno simulado, se espera que los resultados proporcionen una base sólida que facilite la transferencia a entornos reales, mejorando la eficiencia y la seguridad en la gestión de estos cultivos.

1.2.4 Justificación del Vehículo Seleccionado.

Para la navegación dentro de estos cultivos, se seleccionó un vehículo aéreo no tripulado tipo dron hexarrotor. Los drones permiten desplazamientos omnidireccionales y pueden realizar despegue y aterrizaje vertical lo que facilita su movimiento en terrenos con poco espacio disponible y sin pista de despegue. La configuración de seis rotores mejora la resiliencia del sistema, permitiendo el desarrollo de controladores para mantener el vuelo en caso de fallo en uno de los rotores (Nguyen et al., 2019, 2022; Wen et al., 2021), lo que reduce el riesgo de caídas y garantiza la continuidad de la misión. Aunque no se contempla en el alcance del proyecto actual la

implementación de control ante fallas de rotores, se entrega el paquete de ROS 2 del dron para que futuras investigaciones desarrollen estas funcionalidades.

Se utilizará un modelo simulado de dron hexarrotor basado en un dron industrial disponible en el grupo de investigación. Además, el dron tiene las dimensiones adecuadas para navegar dentro del cultivo, ya que la distribución de las palmas proporciona un espacio adecuado para su navegación aérea, lo que facilita la implementación del planificador de vuelo.

1.2.5 Uso de ambientes simulados

Se cuenta con el modelo tridimensional CAD del dron (por sus siglas en inglés: Computer-Aided Design), que fue convertido al formato de Archivo de Descripción de Robots (URDF, por sus siglas en inglés: Unified Robot Description Format) para su implementación con las librerías del Sistema Operativo de Robots 2 (ROS 2, por sus siglas en inglés: Robot Operating System 2) y su integración en el simulador de física Gazebo.

El uso de un entorno simulado presenta varias ventajas. En primer lugar, elimina la necesidad de estar físicamente presente en los cultivos de palma, lo que ahorra tiempo de desplazamiento y evita la solicitud de permisos para probar drones en estos terrenos. Además, dado que se utilizan redes neuronales que se entrenan por medio de aprendizaje por refuerzo, el dron puede aprender de sus errores, como caídas o colisiones, sin riesgo de dañar el equipo disponible.

Inicialmente, desarrollar un controlador de vuelo y realizar pruebas en un entorno simulado es crucial para verificar su correcto funcionamiento antes de la integración en el hardware del dron. Esto no solo reduce los costos de investigación, sino que también permite probar y ajustar el sistema en condiciones controladas, asegurando que los controladores desarrollados sean robustos y estén preparados para enfrentar los desafíos de un entorno real.

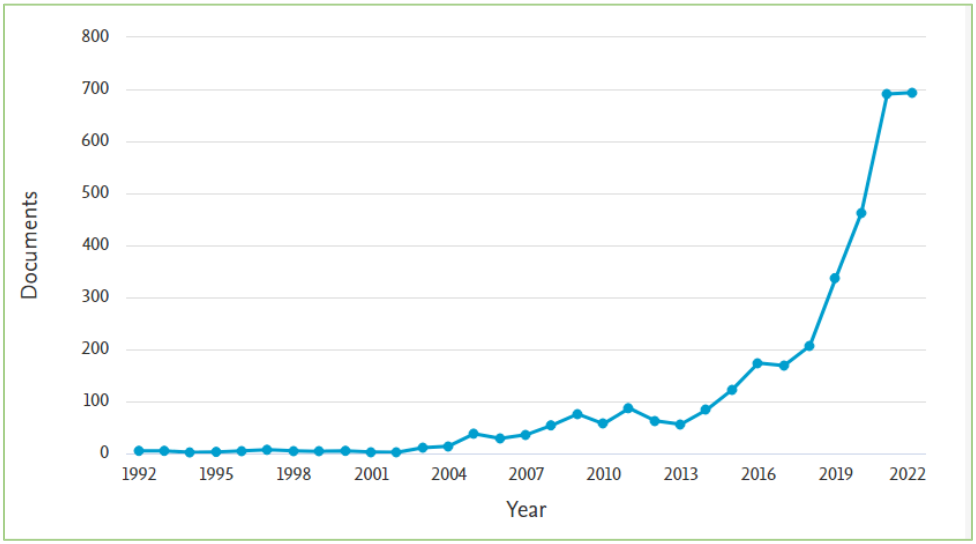
1.3 Justificación de la investigación

Se busca que la herramienta tecnológica a desarrollar contribuya al cumplimiento del objetivo de desarrollo sostenible "Cero Hambre" propuesto por la ONU (United Nations, 2015). El aceite de palma, clave en la industria alimentaria para la elaboración de productos como margarinas, helados, galletas, patatas fritas, panadería, pastillas de caldo y leche de continuación, entre otros (Lorca & Zapata, 2022, p. 21), realiza un gran aporte para alcanzar este objetivo de desarrollo sostenible. La automatización de tareas tediosas o riesgosas, como la inspección en cultivos de palma, aumenta la productividad y disminuye los riesgos laborales.

También se busca contribuir a la investigación en la aplicación de técnicas de inteligencia artificial (IA) en la agricultura, campo de investigación que ha crecido considerablemente en los últimos años, como lo demuestra la Figura 2, donde se realizó una búsqueda en la base de datos Scopus utilizando la siguiente ecuación: *artificial AND intelligence AND agriculture*, obteniendo 3509 resultados.

Figura 2

Artículos de investigación por año en inteligencia artificial aplicada a la agricultura

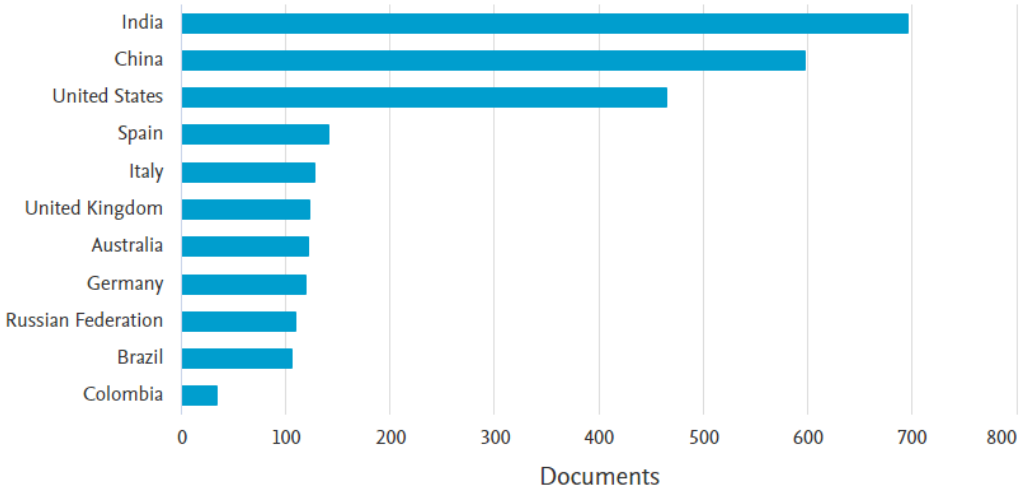


Nota: Gráfico generado por base de datos de Scopus.

En la Figura 3 se muestran los aportes por país de los principales investigadores (se añade a Colombia para efectos de comparación). India, China y Estados Unidos lideran la investigación en esta área, mientras que Colombia había producido solo 34 artículos relacionados con este tema hasta el 2022.

Figura 3

Documentos generados por país en inteligencia artificial aplicada a la agricultura



Nota: Gráfico generado usando Scopus.

Este proyecto busca aumentar la investigación en el área de inteligencia artificial aplicada a la agricultura y contribuir significativamente a la cantidad de investigaciones académicas de origen colombiano. Se enfoca en el cultivo de palma de aceite africana, en el cual Colombia destaca a nivel mundial como uno de los principales productores. A pesar de ser uno de los principales productores agrícolas del mundo, el país ha generado una cantidad limitada de investigaciones en este campo en comparación con otros países, lo que subraya la importancia de desarrollar estudios locales en esta área de aplicación tecnológica.

La investigación se enfoca en vehículos aéreos no tripulados (UAV, por sus siglas en inglés), ya que estos se han abierto campo en la agricultura, para labores como riego, fumigación y mapeo de cultivos, lo cual se debe a los beneficios económicos de su implementación. Entre los UAV disponibles, los drones multirotor ofrecen una mayor versatilidad por su capacidad de moverse en cualquier dirección o mantenerse estáticos en una posición determinada. Además, la investigación en UAV de tipo dron está en auge en áreas como el control de vuelo donde se han llevado a cabo investigaciones que abordan el problema desde enfoques como: la lógica difusa (Yazid et al., 2019) y la lógica difusa robusta (Sun et al., 2018), controladores diseñados por modos deslizantes (Oba et al., 2018), y SLAM (por sus siglas en inglés Simultaneous Localization and Mapping), combinado con el método de exploración basado en la frontera conocida (Bachrach et al., 2010).

El interés en la implementación de la inteligencia artificial ha aumentado, como se evidenció anteriormente. En particular, en cultivos de palma, se han desarrollado técnicas para el mapeo de cultivos, la identificación de la salud de las palmas (Yarak et al., 2021) y la detección de palmas mediante visión artificial al sobrevolar el cultivo (Lee et al., 2019). Sin embargo, no se han encontrado investigaciones que utilicen navegación autónoma dentro de cultivos de palma de aceite, lo cual se aborda en este proyecto mediante un entorno simulado. Esta investigación representa una contribución significativa al avance tecnológico en la agricultura, enfocándose en la automatización de procesos que pueden mejorar considerablemente las condiciones laborales de los trabajadores agrícolas y la productividad de los cultivos.

En este proyecto se implementaron herramientas para crear ambientes simulados, y en estos entornos probar y entrenar drones, con el objetivo de evaluar estrategias de navegación de drones y promover el uso de estas tecnologías en investigaciones futuras similares. Dado que implementar

inteligencia artificial en drones reales dentro de cultivos de palma de aceite sería costoso y prohibitivo en términos de tiempo y recursos, el uso de simulaciones resulta crucial. Para reducir los costos asociados con pruebas de prototipos físicos, se han desarrollado programas de simulación que permiten evaluar el comportamiento de robots en entornos virtuales. Ejemplos de esto incluyen el uso de ROS 2 (Robot Operating System) y Gazebo, programas de código libre que facilitan la creación de estos ambientes simulados. Un estudio relevante en este contexto es una investigación sobre carreras de drones, que utilizó una red neuronal convolucional para determinar la ubicación de un dron en un entorno simulado usando ROS y Gazebo y recorrer la pista de forma autónoma (Cocoma-Ortega & Martinez-Carranza, 2022)

1.4 Alcance del proyecto

Desarrollo de un modelo dinámico del dron hexarrotor, tanto lineal como no lineal, utilizando código en Matlab para su obtención eficiente. Además, se entrega una versión en diagramas de bloques en Simulink, la cual podrá ser utilizada para la prueba de controladores.

El proyecto incluye el diseño de un controlador LQG-integral, que incorpora un observador de Kalman para la estimación de estados y la matriz de ganancia asociada. Además, se entregan diagramas de bloques en Simulink con los elementos necesarios para implementar y comparar tres estrategias de control: LQG-integral, LQG y PID.

Se entrega el modelo del dron y su entorno simulado desarrollados en ROS2-Gazebo, el cual incluye el controlador de vuelo LQG-integral activo desde el momento en que el dron aparece en el ambiente del simulador Gazebo. Además, se proporciona el agente de navegación entrenado y el código utilizado para llevar a cabo dicho entrenamiento.

Memorias en las que se presenta el procedimiento y los resultados de la implementación del modelo de navegación autónoma en un cultivo simulado, analizando la estrategia y el comportamiento del vuelo del dron en diversas etapas del entrenamiento.

Se realizaron tres ponencias en congresos internacionales producto del desarrollo del proyecto y se encuentra en desarrollo un artículo de investigación.

1.5 Metodología

El desarrollo del proyecto se realiza utilizando el método científico y en diferentes etapas; estas etapas incluyen actividades que se pueden llevar a cabo de manera simultánea, como se puede ver en el cronograma de la Tabla 1. En las siguientes secciones se explica de forma breve algunas de las tareas llevadas a cabo en las distintas etapas.

Tabla 1

Cronograma

N	ACTIVIDAD	Semestre				
		2022-2	2023-1	2023-2	2024-1	2024-2
1	Revisión del estado del arte sobre navegación autónoma aplicada a la agricultura					
2	Identificación de la estrategia de control para seguimiento de trayectoria a usar					
3	Selección del método para diseñar el planificador de vuelo					
4	Estudio de las asignaturas electivas requeridas para el desarrollo del proceso					
5	Generación de las ecuaciones de movimiento para dron Hexarrotor					
6	Modelado de dron Hexarrotor mediante la herramienta CAD SolidWorks y exportación a ROS					
7	Desarrollo del cultivo de palma de aceite virtual en Gazebo					
8	Diseño del controlador robusto LQG-Integral					
9	Aplicación de técnica de aprendizaje por refuerzo para navegación					

10	Análisis de la navegación simulada en ambiente virtual					
11	Redacción del informe final					
12	Divulgación de resultados por medio de ponencia o artículo de investigación					

Nota. Las casillas en verde muestran los semestres en los que se ejecutaron las actividades.

1.5.1 Revisión del estado del arte y definición del alcance

En esta etapa se realizó la lectura de distintos artículos científicos, libros, videos y se cursaron asignaturas relacionadas con la línea de investigación, esto permitió ampliar los fundamentos teóricos y conocer el estado del arte relacionado con el vuelo autónomo de drones e inteligencia artificial, y de cómo estos vehículos se utilizan en la agricultura. Partiendo de esto se logró delimitar los alcances de la investigación, en aspectos claves como el tipo de controlador a usar y la estrategia a implementar para el planificador de vuelo, esto dio como resultado el título y objetivos presentados en la propuesta de investigación.

1.5.2 Generación de las ecuaciones de movimiento para dron Hexarotor

Para el desarrollo de los controladores fue necesario realizar el modelado dinámico del sistema, esto se logró al derivar las ecuaciones de movimiento de este. Para encontrar dicho modelo se realizó un estudio de la dinámica del cuerpo rígido con movimiento tridimensional, en las que para definir la orientación se utilizaron los ángulos de Euler, y con el formalismo de Lagrange, por medio de la caja de herramientas simbólica disponible en Matlab se obtuvieron las ecuaciones de movimiento de forma eficiente.

1.5.3 Creación del dron y el ambiente simulado

Para poder probar el modelo dinámico y desarrollar los controladores se creó un modelo 3D (tridimensional) de un dron hexarrotor disponible en el laboratorio del grupo de investigación DicBot, por medio del programa Solidworks. De este modelo se obtuvieron los parámetros del

dron, no obstante, las ecuaciones de movimiento obtenidas anteriormente pueden ser utilizadas para otros drones similares, posteriormente el modelo se exporta a el formato URDF (Unified Robot Description Format) para que puedan implementarse controladores en él usando ROS2.

A la vez se usó el software Gazebo para la creación de un cultivo simulado de palma simplificado, en el que se tienen los tallos de las palmas ubicados de acuerdo con las distancias a las que se recomienda que las palmas sean sembradas.

1.5.4 Diseño del controlador de seguimiento de trayectoria

Con los conocimientos adquiridos al cursar la asignatura sistemas dinámicos y control avanzado, se realizó la linealización de las ecuaciones de movimiento del dron y se diseñó un controlador robusto LQG-integral (objetivo de la investigación), también se desarrollaron controladores LQG y PID para comparación. Se realizaron pruebas numéricas y comparaciones entre los controladores por medio de Simulink. El controlador se tradujo al lenguaje de programación Python para ser implementado por medio de ROS2 en el dron simulado en Gazebo.

1.5.5 Diseño del planificador de vuelo usando aprendizaje por refuerzo

Se cursaron las asignaturas inteligencia artificial y reconocimiento de patrones, en las cuales se estudiaron los fundamentos de las redes neuronales, y el uso de aprendizaje por refuerzo para el entrenamiento de agentes (controladores). Se desarrolló un agente planificador de vuelo para el dron, este genera una trayectoria local que evita los obstáculos presentes en el ambiente, y llega al punto de destino. El agente de navegación se agregó al dron virtual y posteriormente se analizó el desempeño del dron guiado por este dentro del ambiente simulado.

1.5.6 Informe final y divulgación de resultados

El presente texto corresponde al informe final y se desarrolló a medida que transcurrieron los semestres de la maestría. La tarea final fue la divulgación de los resultados obtenidos en la

investigación, para esto se hicieron dos ponencias internacionales y se está trabajando en un artículo relacionado con los resultados de la investigación.

2. Objetivos

2.1 Objetivo General

Desarrollar un modelo dinámico de dron Hexa-rotor con seguimiento de trayectoria robusto y planificador de vuelo autónomo basado en aprendizaje por refuerzo para navegación en un ambiente no estructurado virtual de cultivo de palma de aceite.

2.2 Objetivos Específicos

Desarrollar un modelo dinámico para el dron Hexa-rotor, basado en la dinámica de Lagrange, a partir de su representación en el simulador Gazebo, con el propósito de diseñar y probar controladores.

Diseñar e implementar un sistema de control robusto, basado en la estrategia de control LQG-Integral (Linear Quadratic Gaussian), para el seguimiento de trayectoria de vuelo del dron Hexa-rotor.

Desarrollar un sistema de navegación autónoma para el dron Hexa-rotor, utilizando el algoritmo de aprendizaje por refuerzo DDDQN (Dueling Double Deep Q-Network), con el propósito de navegar a través de cultivos de palma de aceite.

Validar por medio de simulación el modelo propuesto en un ambiente no estructurado basado en cultivos de palma de aceite, creado en el simulador Gazebo, a fin de analizar la estrategia y comportamiento del vuelo del dron.

3. Marco teórico

A continuación, se da el marco teórico de algunos conceptos relevantes para el desarrollo de la investigación, como uso de ángulos de Euler para describir la orientación de un cuerpo rígido, fundamentos de los algoritmos de inteligencia artificial usados, y se da una breve explicación de las estrategias de control implementadas.

3.1 Ángulos de Euler

Los ángulos de Euler se emplean para determinar la orientación de un cuerpo rígido. Estos ángulos describen la orientación de un sistema de referencia fijo al cuerpo (SRC) no inercial en relación con un sistema de referencia inercial (SRI) a través de tres rotaciones secuenciales alrededor de los ejes del SRC. En este texto, se adopta la convención de guiñada-cabeceo-alabeo. De acuerdo con esta convención, el ángulo de rotación con respecto al eje z se denomina ángulo de guiñada ψ, el ángulo de rotación con respecto al eje y se denomina ángulo de cabeceo θ, y el ángulo de rotación con respecto al eje x se denomina ángulo de balanceo ϕ.

Para determinar la orientación del SRC respecto al SRI, se aplican las siguientes rotaciones en orden: primero para rotar alrededor del eje z un ángulo ψ se multiplica por la matriz de rotación R_z, dada en (1); luego para rotar alrededor del eje y un ángulo θ se multiplica por la matriz de rotación R_y dada en (2), y finalmente para rotar alrededor del eje x un ángulo ϕ, se multiplica por la matriz de rotación R_x dada en (3).

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

$$\mathrm{R}_\mathrm{y} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{2}$$

$$\mathrm{R}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \tag{3}$$

Los vectores unitarios del SRI se dan en la Ecuación (4). En este texto se usa el subíndice SRC, cuando se busca indicar que los vectores unitarios son del SRC.

$$\hat{\imath} = [1,0,0]^{\mathrm{T}}$$
$$\hat{\jmath} = [0,1,0]^{\mathrm{T}} \tag{4}$$
$$\hat{\mathrm{k}} = [0,0,1]^{\mathrm{T}}$$

Cada vez que se produce una rotación se tiene una nueva orientación de los ejes del cuerpo con respecto a los ejes de coordenadas inerciales, el sistema de referencia inercial hace uso de los vectores unitarios $\hat{\imath}$, $\hat{\jmath}$, $\hat{k}$, como base ortonormal. Para determinar cómo se expresan las coordenadas en un punto de un cuerpo luego de hacer cada una de las rotaciones es de utilidad definir vectores unitarios que permitan hacer la conversión.

Los vectores unitarios luego de realizar una rotación respecto al eje z se dan en (5).

$$\begin{bmatrix} \hat{\imath}_0 \\ \hat{\jmath}_0 \\ \hat{k}_0 \end{bmatrix} = R_\psi \begin{bmatrix} \hat{\imath} \\ \hat{\jmath} \\ \hat{k} \end{bmatrix} \tag{5}$$

Los vectores unitarios luego de realizar una rotación respecto al eje z, seguida por una rotación con respecto al eje y_0 se dan en (6).

$$\begin{bmatrix} \hat{\imath}_1 \\ \hat{\jmath}_1 \\ \hat{k}_1 \end{bmatrix} = R_\theta R_\psi \begin{bmatrix} \hat{\imath} \\ \hat{\jmath} \\ \hat{k} \end{bmatrix} \tag{6}$$

Los vectores unitarios luego de realizar una rotación respecto al eje z, seguida por una rotación con respecto al eje y_0, seguida por una rotación con respecto al eje x_1 se dan en (7).

$$\begin{bmatrix} \hat{\imath}_2 \\ \hat{\jmath}_2 \\ \hat{k}_2 \end{bmatrix} = R_\phi R_\theta R_\psi \begin{bmatrix} \hat{\imath} \\ \hat{\jmath} \\ \hat{k} \end{bmatrix} \tag{7}$$

3.2 Variación de los ángulos de Euler con respecto al tiempo y velocidad angular medida

Un giroscopio mide la velocidad angular con respecto al SRC $\vec{\omega}_{SRC}$, en este texto a los componentes con respecto a cada eje del SRC se les denomina ω_x, ω_y, y ω_z respectivamente. La ecuación de la velocidad angular medida por el giroscopio se puede escribir como en (8).

$$\vec{\omega}_{SRC} = \omega_x \hat{\imath}_{SRC} + \omega_y \hat{\jmath}_{SRC} + \omega_z \hat{k}_{SRC} \tag{8}$$

La forma en la que se obtiene la orientación de los cuerpos es con la aplicación de rotaciones sucesivas, recordando que se está usando la convención de guiñada-cabeceo-alabeo, se tiene entonces que primero ocurre un giro con respecto al eje z (dirección $\hat{k}$) del SRC, a una velocidad de cambio del ángulo de guiñada de $\dot{\psi}$; el SRC queda entonces con una nueva orientación y luego ocurre un giro con respecto a su eje y (dirección $\hat{\jmath}_0$) a una velocidad cambio del ángulo de cabeceo es $\dot{\theta}$; esto hace que el SRC se encuentre en una nueva orientación y finalmente se da una rotación con respecto a su eje x (dirección $\hat{\imath}_1$), a una velocidad cambio del ángulo de alabeo $\dot{\phi}$. De acuerdo con lo anterior la velocidad angular con respecto al SRI $\vec{\omega}_{SRI}$, es dada por (9).

$$\vec{\omega}_{SRI} = \dot{\phi}\hat{\imath}_1 + \dot{\theta}\,\hat{\jmath}_0 + \dot{\psi}\hat{k} \tag{9}$$

Para realizar la conversión y obtener una expresión que relacione las variables medibles con sensores y la tasa de cambio de cada ángulo con el tiempo, se deben expresar los vectores $\hat{\imath}_1$, $\hat{\jmath}_0$, y $\hat{k}$ como vectores del cuerpo, para esto se deben hacer rotaciones inversas de los vectores unitarios. El vector unitario $\hat{\imath}_1$ coincide con el vector unitario $\hat{\imath}_{SRC}$. Para los demás vectores

unitarios, se debe multiplicar por la matriz inversa de las rotaciones que ocurren en cada eje. Es decir, para expresar $\hat{k}$ en términos de $\hat{k}_{SRC}$ se deben hacer dos rotaciones inversas, y expresar $\hat{j}_0$ en términos de $\hat{j}_{SRC}$ una sola rotación inversa. Las matrices de rotación son ortogonales, por lo que su inversa coincide con su transpuesta.

La fórmula para obtener la velocidad angular en términos de los ángulos de Euler y sus variaciones en el tiempo usando los vectores unitarios del cuerpo está dada entonces por

$$\overrightarrow{\omega}_{SRC} = \begin{bmatrix}\omega_x \\ \omega_y \\ \omega_z\end{bmatrix} = \begin{bmatrix}\dot{\phi} \\ 0 \\ 0\end{bmatrix} + R_{\phi}^{T}\begin{bmatrix}0 \\ \dot{\theta} \\ 0\end{bmatrix} + R_{\phi}^{T}R_{\theta}^{T}\begin{bmatrix}0 \\ 0 \\ \dot{\psi}\end{bmatrix} = \begin{bmatrix}1 & 0 & -sin(\theta) \\ 0 & cos(\phi) & sin(\phi)cos(\theta) \\ 0 & -sin(\phi) & cos(\phi)cos(\theta)\end{bmatrix}\begin{bmatrix}\dot{\phi} \\ \dot{\theta} \\ \dot{\psi}\end{bmatrix} \quad (10)$$

En la práctica es más útil tener una expresión para tener las velocidades de los ángulos de Euler, en función de las velocidades angulares medidas, y los ángulos de Euler (Schaub & Junkins, 2018, p. 94). Esta expresión es dada en (10) se obtiene luego de despejar las velocidades en (11).

$$\begin{bmatrix}\dot{\phi} \\ \dot{\theta} \\ \dot{\psi}\end{bmatrix} = \begin{bmatrix}1 & \frac{sin(\phi)sin(\theta)}{cos(\theta)} & \frac{cos(\phi)sin(\theta)}{cos(\theta)} \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & \frac{sin(\phi)}{cos(\theta)} & \frac{cos(\phi)}{cos(\theta)}\end{bmatrix}\begin{bmatrix}\omega_x \\ \omega_y \\ \omega_z\end{bmatrix} \quad (11)$$

3.3 Control PID

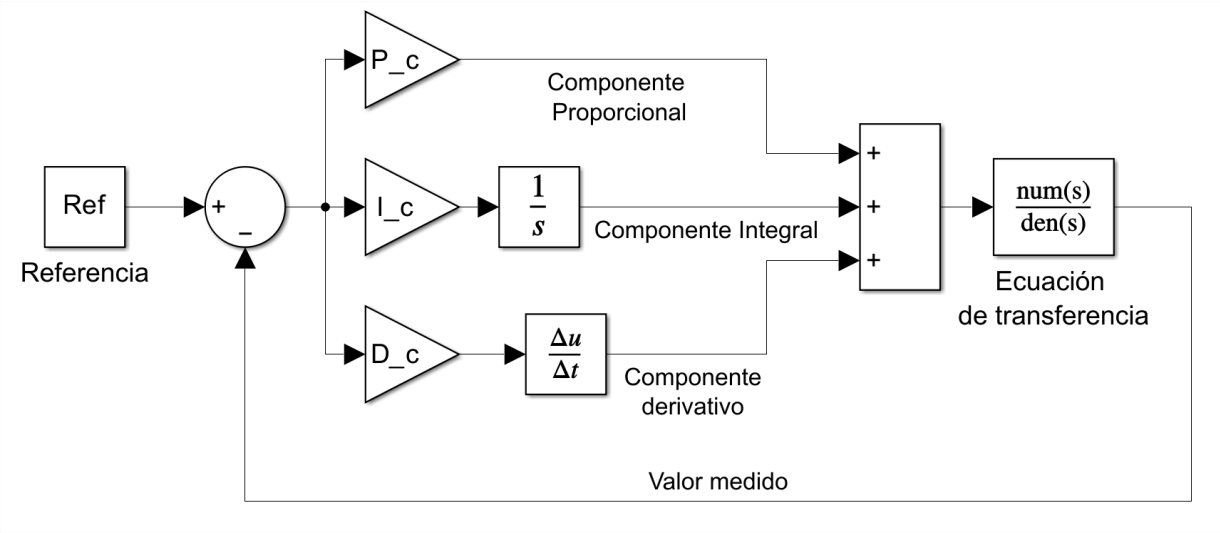
El control PID se basa en la magnitud del error, definido como la diferencia entre el valor medido y el valor de referencia. Este controlador ajusta la entrada al sistema o planta mediante operaciones sobre este error para minimizarlo. El controlador PID tiene tres componentes: proporcional (P), integral (I) y derivativo (D).

En la Figura 4 se muestra el diagrama de bloques del lazo cerrado de control para un sistema de una entrada y una salida, (SISO, por sus siglas en inglés: *Single Input, Single Output*).

Donde P_c es el factor de la acción proporcional, I_C es el factor de la acción integral, D_c es el factor de la acción derivativa.

Figura 4

Diagrama de bloques control de PID puro para un sistema SISO



La acción proporcional busca que el actuador envíe una respuesta directamente proporcional a la magnitud del error medido, lo que implica que la corrección se basa en el estado presente del error. Sin embargo, si solo se aplica una corrección proporcional, puede haber un error en estado estable u oscilaciones alrededor del valor de referencia. Para mitigar el error en estado estable, se incorpora un componente integral que considera los errores pasados y los acumula, es decir, actúa de manera proporcional al área bajo la curva de la función de error. Por otro lado, el componente derivativo anticipa la tendencia del error y toma una acción correctiva basada en la tasa de cambio del error. Esto permite que la acción derivativa ajuste el sistema antes de que el error se vuelva demasiado grande (Carlucho et al., 2017, p.2).

En el dominio de la frecuencia la ecuación de transferencia del controlador PID está dada por (12). Donde $U(s)$ es la salida del controlador y $E(s)$ es el error.

$$\frac{U(s)}{E(s)} = P_c + I_c \cdot \frac{1}{s} + D_c \cdot s \tag{12}$$

3.4 Control regulador linear cuadrático LQR

La estrategia de control óptimo llamado regulador linear cuadrático LQR, por sus siglas del inglés *Linear Quadractic Regulator*, busca el equilibrio entre la estabilidad y la agresividad del control, para lograr esto se cuantifican las necesidades de diseño mediante una función de costo (Carreño Sánchez & Ramirez Plata, 2023, p. 31).

La ecuación (13), se usa para describir la representación de un sistema en espacio de estados. Donde *A* es la matriz de estado, *B* es la matriz de entrada, $X(t)$ el vector de estados del sistema, $\dot{X}(t)$ la variación del vector de estado con respecto al tiempo y $u(t)$ es el vector de entradas del sistema.

$$\dot{X}(t) = A \cdot X(t) + B \cdot u(t) \tag{13}$$

La función de coste es dada por (14), donde Q es una matriz hermítica definida positiva (o semidefinida positiva) o simétrica real y R es una matriz hermítica definida positiva o simétrica real (Ogata, 2010, p. 794).

$$J = \int_0^\infty \left(X(t)^\top Q X(t) + u(t)^\top R u(t) \right) \, dt \tag{14}$$

La matriz Q representa el costo asociado a los estados del sistema, cada elemento de Q pondera la importancia de los diferentes estados en la función de coste. Por ejemplo, si la matriz Q es diagonal, los elementos en la diagonal representan los pesos asignados a cada estado. Un valor alto en un elemento diagonal de Q indica que el estado correspondiente es más crítico y debe ser estabilizado con mayor prioridad que otros estados.

La matriz R, por otro lado, representa el costo asociado a las acciones de control. Si R es diagonal, cada elemento de R pondera el costo de utilizar cada acción de control en el sistema. Un valor alto en un elemento diagonal de R indica que el uso de la acción de control correspondiente es costoso y, por lo tanto, se debe minimizar su uso en comparación con otras acciones de control.

Para poder encontrar *K* de la ley de control óptimo dada en (15) primero se debe resolver la ecuación matricial reducida de Ricatti dada por (16), para la matriz P. La resolución de esta ecuación se realiza numéricamente utilizando herramientas de software como MATLAB debido a su complejidad. La matriz P obtenida refleja cómo se ponderan los estados del sistema en la función de coste

$$u(t) = -K \cdot X(t) \tag{15}$$

$$A^\top P + PA - PBR^{-1}B^\top P + Q = 0 \tag{16}$$

Una vez que se ha encontrado la matriz P, se utiliza la ecuación (17), para calcular la matriz de ganancias K. La matriz K se ajusta para minimizar el índice de coste definido en el problema LQR, equilibrando el coste asociado a los estados del sistema y el coste de las acciones de control.

$$K = R^{-1}B^\top P \tag{17}$$

3.5 Observador de Kalman

Para implementar la estrategia de control LQR, es esencial conocer el vector de estados $X(t)$. Una forma de lograr esto es mediante sensores que midan directamente cada uno de los estados del sistema. Sin embargo, en la práctica, esto no siempre es factible debido a la complejidad de algunos sistemas, el costo elevado de los sensores, o la dificultad de medir ciertos estados directamente.

Para superar estas limitaciones, se utiliza un observador de Kalman. Este observador permite estimar el valor de todos los estados del sistema utilizando un subconjunto de estados medidos y las acciones de control aplicadas. De esta manera, incluso si no se pueden medir todos los estados directamente, el observador de Kalman proporciona estimaciones precisas y fiables de los estados necesarios para el control óptimo.

En sistemas lineales con múltiples entradas y múltiples salidas en tiempo discreto, donde las medidas están afectadas por errores estocásticos, es esencial estimar los estados del sistema de manera precisa. Para ello, se utiliza un observador de estado, que toma en cuenta el ruido presente tanto en las entradas como en las salidas del sistema.

El modelo del sistema, en tiempo discreto, se describe por las ecuaciones (18) y (19); donde $x(k)$ es el vector de estados en el instante k, $u(k)$ es el vector de entradas controladas, $y(k)$ es el vector de salidas observadas, $w(k)$ es el ruido de proceso (entrada), y $v(k)$ es el ruido de medida (salida) (Hoyos y Borrás, 2024). Se asume que se tiene un modelo del sistema que refleja correctamente su dinámica, y que las matrices A, B, C son conocidas.

$$x(k+1) = Ax(k) + Bu(k) + w(k) \tag{18}$$

$$y(k) = Cx(k) + v(k) \tag{19}$$

El estimador de Kalman requiere que el ruido tanto de la medida como del proceso puedan ser modelados como variables aleatorias independientes, con distribución normal y media cero. Además, se asume que el valor inicial del vector de estados, denotado como x_0, es conocido.

Si además se conoce el tamaño de los ruidos de medida (información que generalmente es proporcionada por los fabricantes de los sensores), y el tamaño del ruido del proceso, es posible determinar tanto la matriz de covarianza del ruido de proceso M como la matriz de covarianza del

ruido de medida N. Estas matrices son fundamentales en el proceso de estimación del filtro de Kalman, ya que influyen directamente en la precisión de las estimaciones del estado del sistema.

3.5.1 Etapas del filtro de Kalman

Se explica ahora el procedimiento para la estimación del estado, este consiste en dos etapas una de predicción y otra de corrección. Se usa la siguiente notación para las estimaciones en tiempo discreto: $\hat{x}(k+1|k)$, que quiere decir que se realiza la estimación del estado en el tiempo $k+1$, dado que se conoce hasta el tiempo k.

El primer paso consiste en realizar la predicción del estado por medio de (20).

$$\hat{x}(k+1 \mid k) = A\hat{x}(k \mid k) + Bu(k) \tag{20}$$

Luego se realiza la predicción de la covarianza del error usando (21), donde $P(k)$ es la matriz de covarianza del error de estimación en el tiempo k, y $P_{pred}(k+1)$ es la covarianza del error de predicción.

$$P_{pred}(k+1) = AP(k)A^T + M \tag{21}$$

Ahora se procede a realizar la actualización o corrección de la ganancia de Kalman $K(k+1)$ por medio de (22).

$$K(k+1) = P_{pred}(k+1)C^T\left(CP_{pred}(k+1)C^T + N\right)^{-1} \tag{22}$$

Para luego realizar la corrección de la $\hat{x}(k+1 \mid k+1)$ que representa la estimación del estado en el tiempo k+1 después de incorporar la medición $y(k+1)$ usando (23).

$$\hat{x}(k+1 \mid k+1) = \hat{x}(k+1 \mid k) + K(k+1)(y(k+1) - C\hat{x}(k+1 \mid k)) \tag{23}$$

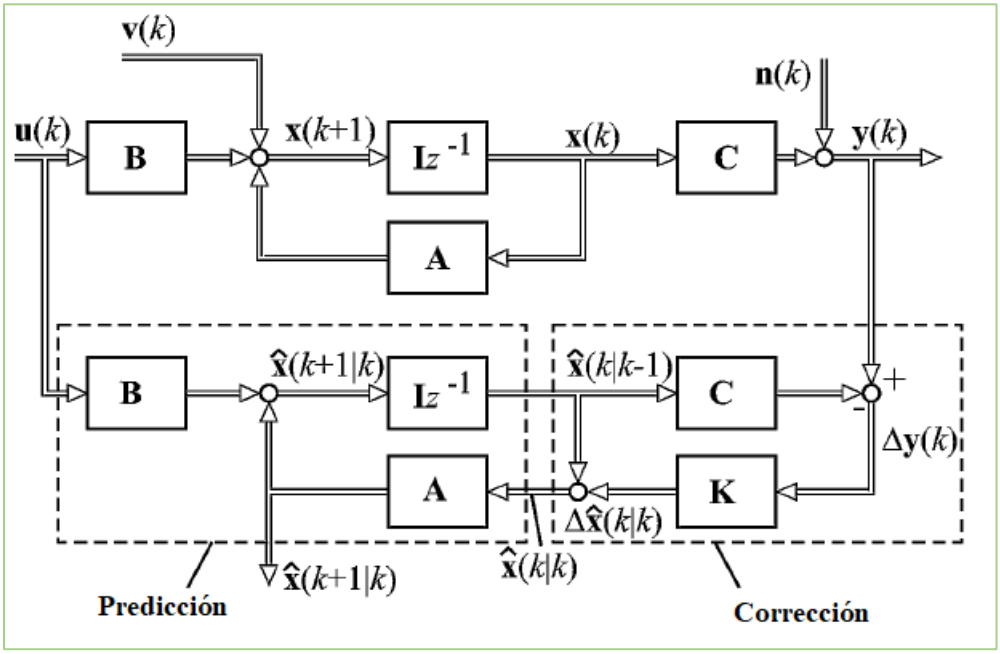
El último paso del proceso de corrección o actualización es actualizar el valor de la matriz de covarianza del error de estimación, para el tiempo $k+1$ usando (24).

$$P(k+1) = (I - K(k+1)C)P_{pred}(k+1) \tag{24}$$

Todo este proceso de predicción y actualización se repite para cada nuevo conjunto de datos de entrada, refinando continuamente las estimaciones de los estados del sistema. En la Figura 5 se muestra un diagrama esquemático del proceso de estimación de estado del observador de Kalman.

Figura 5

Diagrama de bloques del observador de Kalman



Fuente: Adaptado de (Isermann, 2006, p. 242)

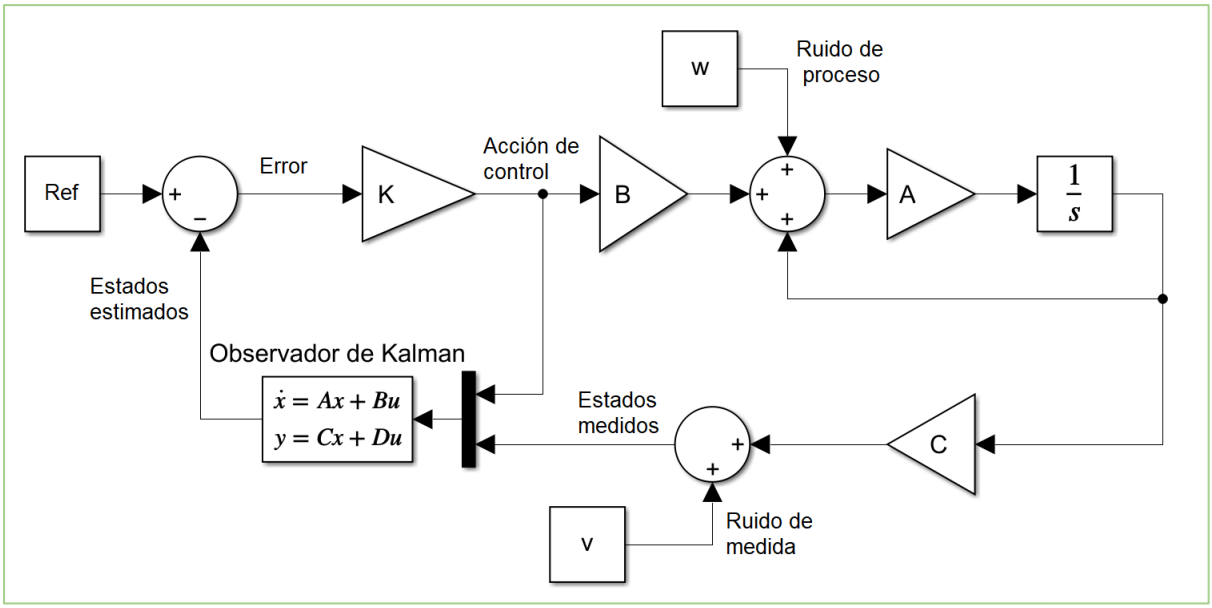
3.6 Control LQG

Como se mencionó anteriormente, el control LQR requiere conocer todo el vector de estados. Sin embargo, dado que no siempre es posible medir todos los estados directamente, puede ser necesario utilizar un observador de Kalman. La estrategia de control que combina un controlador óptimo LQR con un observador de Kalman para estimar los estados se denomina Lineal Cuadrático Gaussiano (LQG, por sus siglas en inglés *Linear Quadratic Gaussian*). En la

Figura 6 se muestra el diagrama de bloques del controlador LQG para seguimiento de trayectoria, incluyendo ruido de medida y ruido de proceso.

Figura 6

Diagrama de bloques controlador LQG



Este controlador suele ser robusto para un rango de perturbaciones cercanas a la media, siempre y cuando se disponga de un modelo adecuado del sistema y las perturbaciones de proceso, y recordando que el observador de Kalman requiere que los ruidos puedan ser modelados mediante una distribución normal. Sin embargo, es importante destacar que este controlador aún puede presentar error en estado estacionario, especialmente si no se han implementado técnicas adicionales como la acción integral para eliminar dicho error.

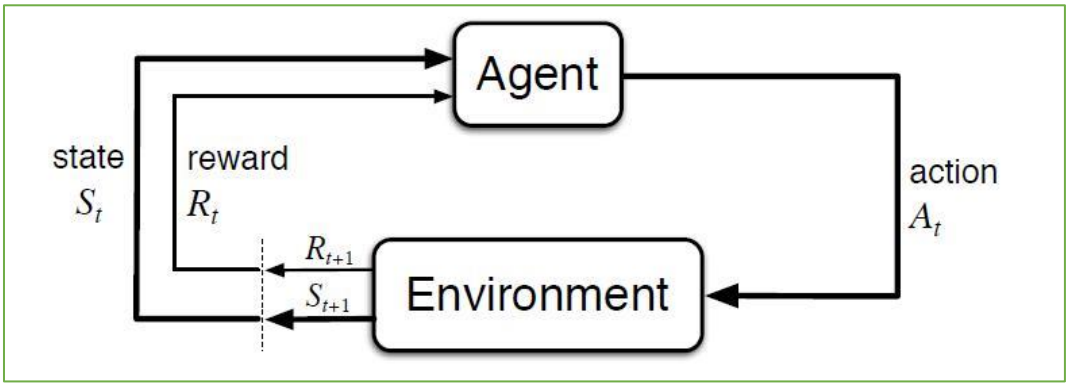
3.7 Aprendizaje por refuerzo

El aprendizaje por refuerzo es una rama del aprendizaje automático que se enfoca en cómo un agente aprende a actuar en un entorno para maximizar una señal de recompensa. El agente

aprende mediante la interacción continua con su entorno, donde cada acción genera una recompensa (Sutton & Barto, 2018, p.2), ver **Figura 7**. Además, el aprendizaje incluye un componente de recompensa futura, lo que significa que una acción tomada en un momento determinado puede influir en las recompensas recibidas más adelante.

Figura 7

Esquema de aprendizaje por refuerzo



Nota. Fuente: *Reinforcement Learning: An Introduction* (2nd ed., p. 48), por R. S. Sutton y A. G. Barto, 2018, The MIT Press.

Los dos elementos principales del aprendizaje por refuerzo son el agente y el ambiente. El agente interactúa constantemente con el ambiente, buscando modificarlo mediante sus acciones para alcanzar un objetivo. Un esquema de aprendizaje por refuerzo debe considerar tres aspectos: medición, acción y metas. Es decir, el agente debe ser capaz de percibir el estado de su ambiente, tomar acciones que alteren ese estado y trabajar hacia el logro de una meta.

A diferencia de otros tipos de aprendizaje, como el supervisado o no supervisado, el aprendizaje por refuerzo aborda el problema con un agente orientado a cumplir objetivos en un entorno incierto, sin la necesidad de tener etiquetas o datos previos. (Sutton & Barto, 2018, p. 2)

Los subcomponentes importantes en un sistema de aprendizaje por refuerzo son la política, la señal de recompensa, la función de valor y, opcionalmente, el modelo del entorno. La política define el comportamiento del agente, decidiendo que acción tomar en función del estado actual del ambiente. La señal de recompensa es un valor numérico que el agente intenta maximizar a largo plazo, es la que permite que el agente aprenda.

La función de valor estima la recompensa acumulada que se espera obtener a futuro a partir de un estado dado. A veces, una acción puede generar recompensas inmediatas bajas, pero puede llevar a una recompensa mayor en el largo plazo está función captura este comportamiento. Por último, el modelo del ambiente (cuando se utiliza) permite predecir cómo responderá el ambiente a las acciones del agente, lo que puede ser útil para planificación.

Existen métodos para resolver problemas de aprendizaje por refuerzo tanto con modelo (donde se predicen las recompensas y estados futuros) como sin modelo (donde se aprenden directamente las acciones óptimas a través de la experiencia).

3.8 Algoritmo DQN

Debido a que el algoritmo de inteligencia artificial utilizado requiere un entendimiento de algoritmos fundamentales, estos se explican brevemente aquí y en las secciones siguientes. Primero se introduce el DQN (Deep Q-Network), que combina el Q-Learning con el Aprendizaje Profundo.

3.8.1 Q-Learning

Este es un método de aprendizaje por refuerzo libre de modelo destinado a aprender la función de valor $Q(s, a)$, la cual permite determinar el valor de tomar una acción a en un estado s. Este método es de naturaleza tabular, lo que significa que requiere calcular el valor Q para todas las posibles combinaciones de acciones y estados. Debido a esta característica, es adecuado

principalmente para aplicaciones con un número limitado de combinaciones de estado-acción. La política entrenada se basa en el valor Q como criterio para seleccionar la mejor acción a en función del estado s.

3.8.2 Aprendizaje Profundo

Conocido como una subcategoría del aprendizaje automático, el aprendizaje profundo utiliza redes neuronales profundas con múltiples capas y neuronas para aprender funciones que relacionan entradas con salidas. Estas redes se pueden emplear tanto para tareas de predicción como de regresión. Para entrenar estas redes, se requieren datos de entrada junto con sus respectivas salidas etiquetadas.

En el algoritmo DQN, el uso de una tabla para almacenar los valores Q es reemplazado por una red neuronal profunda para aproximar la función $Q(s, a)$ (Sutton & Barto, 2018, p. 195). Esto permite manejar situaciones más complejas y aprender de entornos con numerosos estados y acciones.

En el aprendizaje por refuerzo, existe el dilema exploración-explotación, que se refiere a determinar cuándo explorar para acumular experiencias y cuándo explotar lo que ya se ha aprendido. Para la exploración, en este trabajo se utilizó exploración ϵ-greedy, donde se obtiene un número aleatorio y, si es mayor que ϵ, se selecciona la acción que produce el valor Q más alto; de lo contrario, se selecciona cualquier acción aleatoriamente. Este ϵ disminuye durante el entrenamiento para reducir la exploración y aumentar la explotación.

3.8.3 Procedimiento para entrenar la Red

Durante el entrenamiento de una red por medio del algoritmo DQN se ejecutan cientos de "episodios", cada uno compuesto por numerosos "pasos". A continuación, se describe el procedimiento llevado a cabo en cada paso:

- El ambiente inicialmente se encuentra en un estado s, la acción por tomar a se selecciona usando la exploración ϵ-greedy y se aplica.
- Se ingresa a la red el par (s, a) como entrada y se obtiene el valor Q predicho correspondiente a esa acción.
- Tras aplicar la acción a el ambiente pasa al nuevo estado s'. Se calcula la recompensa r por medio de una función de recompensa.
- En el nuevo estado s', se determina la acción a' que produce el valor Q máximo. Esta acción no se aplica, sino que se usa para calcular el valor Q objetivo, que se considera el equivalente a la salida etiquetada, del aprendizaje supervisado. Este valor Q objetivo se obtiene mediante la ecuación de Bellman dada en (25).

$$Q_{target}(s, a) = r + \gamma Q(s', a') \tag{25}$$

  Donde γ es conocido como el factor de descuento, que determina cuánta importancia se le da a la recompensa inmediata en comparación con la recompensa a largo plazo. Los valores típicos de γ oscilan entre 0.8 y 0.99, donde los valores más bajos otorgan mayor importancia a las recompensas inmediatas.
- La red neuronal se entrena de manera similar al aprendizaje supervisado. El valor Q objetivo se considera como el valor correcto y se compara con el valor Q predicho para la acción tomada. El algoritmo de retropropagación calcula los gradientes de la función de pérdida, que es la diferencia entre el valor Q objetivo y el valor Q predicho. Estos gradientes se utilizan para actualizar los parámetros de la red neuronal usando una tasa de aprendizaje lr.

3.9 Dueling DQN

Este método es una mejora sobre el algoritmo DQN explicado anteriormente. En este algoritmo, se utilizan dos redes: una calcula el valor del estado $V(s)$, que representa el beneficio de estar en un estado s sin considerar la acción tomada; la segunda red calcula la función de ventaja $A(s,a)$, que representa el beneficio de tomar la acción a en comparación con otras en el mismo estado s. Con los valores de estas dos redes, suponiendo que hay m acciones, la función de valor Q se puede calcular usando la ecuación (26) (Wang et al., 2016).

$$\mathrm{Q(s,a)} = \mathrm{V(s)} + \left(\mathrm{A(s,a)} - \frac{1}{m} \sum_{i=1}^{m} A(s, a_i) \right) \tag{26}$$

3.10 Algoritmo Double-DQN

Double-DQN es un algoritmo de aprendizaje por refuerzo que busca mejorar la estabilidad y precisión del aprendizaje en los algoritmos DQN al mitigar la sobreestimación de los valores Q, que ocurre cuando se utiliza una misma red para seleccionar y evaluar las acciones (Shin et al., 2019). Para resolver este problema, se utilizan dos redes neuronales: una para seleccionar la acción en el estado actual y otra para evaluar su rendimiento, conocidas como la "red en línea" y la "red objetivo," respectivamente.

Al comienzo del entrenamiento, las dos redes son idénticas, pero después de un cierto número de episodios, la red objetivo se reemplaza por una copia de la red en línea. La red en línea toma el estado actual como entrada y genera el valor Q de cada acción, que sirve como criterio de selección para la acción a que se debe elegir en el estado s, esta selección se hace usando ϵ-greedy. Después de ejecutar la acción seleccionada, se produce una transición al estado s', y el ambiente devuelve una recompensa r.

Para determinar el error y entrenar la red, se calcula el valor Q actualizado $Q_{update}(s,a)$, que se asume como el valor correcto para compararlo con el valor predicho por la red en línea. La ecuación de Bellman para DDQN dada en (27) se utiliza para obtener $Q_{update}(s,a)$, donde la acción a' es la que según la red en línea produciría el valor Q más alto en el estado s'; y el valor Q objetivo $Q_{target}(s',a')$ se obtiene usando la red objetivo (Hasselt et al., 2016).

$$Q_{update}(s,a) = r + \gamma Q_{target}(s',a') \tag{27}$$

Si la función de pérdida seleccionada es el Error Cuadrático Medio (MSE), y se utiliza un lote de datos (con entradas y salidas etiquetadas) para realizar el entrenamiento, se emplea la ecuación (28) para el cálculo del error. En esta ecuación $Q_{online}(s,a)$ representa el valor predicho por la red en línea y n es el tamaño del lote.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Q_{update,i}(s,a) - Q_{online,i}(s,a))^2 \tag{28}$$

3.11 Memoria de repetición (Replay buffer)

Las memorias de repetición son usadas para almacenar tuplas que contienen la información de las experiencias obtenidas al interactuar con el ambiente, estas son enviadas a la memoria de la siguiente manera $(s,a,r,s',done)$, donde $done$ es una bandera que indica si el estado s' es un estado terminal. El uso de la memoria de repetición permite almacenar experiencias y seleccionarlas aleatoriamente durante el entrenamiento. Este proceso rompe la correlación que ocurriría si el entrenamiento se realizara exclusivamente con nuevas experiencias, donde los estados consecutivos podrían ser muy similares (Lapan, 2018, p. 184). Además, la memoria de repetición permite reutilizar experiencias previamente obtenidas, mejorando la eficiencia y estabilidad del proceso de aprendizaje.

4. Estado del arte

En esta sección, se realiza una revisión del estado del arte en la navegación autónoma de drones, dividiéndola en tres tareas fundamentales. La primera tarea es el control de vuelo y seguimiento de trayectorias, que también abarca la odometría, esencial para que el dron pueda determinar su posición y orientación actual y así evaluar la distancia al objetivo. Esta área incluye el estudio de diversas técnicas de control y algoritmos de seguimiento, que permiten mantener la estabilidad del dron y optimizar su trayectoria.

La segunda tarea es la evasión de obstáculos, un aspecto crucial para la navegación segura en entornos complejos. La literatura revisada en esta área aborda el uso de sensores como LIDAR y cámaras (monoculares y de profundidad) para mapear el entorno del dron en tiempo real, lo que facilita la detección y evitación de obstáculos. Se han explorado diferentes enfoques y tecnologías para mejorar la precisión y velocidad de respuesta del dron en estas situaciones.

Finalmente, la tercera tarea es la planeación de trayectoria local, que está estrechamente relacionada con la evasión de obstáculos. Esta tarea implica la generación de rutas seguras y eficientes dentro de un entorno dinámico, teniendo en cuenta las limitaciones del dron y las características del entorno. La planeación de trayectorias locales es esencial para la adaptación en tiempo real a cambios en el entorno, lo que permite al dron navegar de manera autónoma en escenarios imprevistos.

La revisión del estado del arte se ha centrado en artículos originales y de conferencias publicados en los últimos cinco años. En total, se analizaron 38 artículos que cumplen con este criterio, seleccionados tanto por su relevancia en la aplicación de estrategias basadas en inteligencia artificial (IA) como por su contribución al conocimiento general en la navegación autónoma de drones. Aunque se dio prioridad a enfoques relacionados con la IA, también se

incluyeron trabajos que no emplean estas técnicas, con el fin de proporcionar una visión integral del campo.

La búsqueda de la literatura se realizó a través de la biblioteca virtual de la Universidad Industrial de Santander, que tiene acceso a bases de datos como IEEE, Elsevier, Proquest, entre otras. Se destaca la contribución significativa de la revista *Drones*, especializada en avances en la investigación de vehículos aéreos no tripulados, la cual proporcionó una gran cantidad de los artículos revisados en esta sección.

4.1 Control de vuelo y seguimiento de trayectoria

Orozco Soto et al. (2022) presentan un esquema de control robusto basado en el control por rechazo activo de perturbaciones (ADRC) para la regulación y seguimiento de trayectoria de un hexacóptero con rotor inclinado pasivamente. El esquema se fundamenta en un observador de estado extendido de modos deslizantes (SMESO) y fue validado utilizando un simulador realista. Los resultados indican que el controlador propuesto es capaz de alcanzar con éxito la trayectoria objetivo, manteniendo su eficacia incluso bajo condiciones de carga útil y perturbaciones de viento continuas y discontinuas. Este rendimiento se debe a la reconstrucción precisa de las perturbaciones externas mediante el observador, logrando una reducción significativa en la norma del error en estado estacionario en comparación con el control de modos deslizantes (SMC).

Por su parte, Xiao et al. (2022) desarrollan un modelo para un dron cuadricóptero utilizando el formalismo de Newton-Euler. Este modelo se aplicó en la realización de fotografías aéreas en entornos agrícolas, con el objetivo de mitigar la disminución de la calidad de la imagen debido a los efectos del viento y la carga de la batería. El sistema se divide en un sistema de accionamiento completo y un subsistema subactuado. Para controlar la coordenada z y el ángulo de guiñada, se diseñó un controlador de modos deslizantes rápido, mientras que para el control en

(x, y) y los ángulos de alabeo y cabeceo del subsistema subactuado, se desarrolló un controlador de modos deslizantes de segundo orden.

Fan (2022) emplea controladores PID para gestionar el vuelo de un cuadricóptero. A partir del formalismo de Newton-Euler, desarrollaron un modelo dinámico del dron y realizaron simulaciones en Matlab Simulink para controlar los movimientos verticales, de guiñada, cabeceo y alabeo. Estas simulaciones permitieron validar el diseño y funcionamiento de los controladores propuestos.

Choi et al. (2023) exploran un enfoque innovador de aprendizaje por refuerzo modular aplicado a UAVs de ala fija con 6 grados de libertad (GDL). Este método divide tareas complejas en sub-tareas más manejables, las cuales son aprendidas individualmente mediante el algoritmo SAC. Posteriormente, las sub-tareas se conectan, permitiendo un aprendizaje más rápido y estable gracias a la transferencia de información entre los módulos. Los experimentos realizados mostraron que este enfoque resultó en recompensas altas y una mayor estabilidad, con menos episodios de aprendizaje en comparación con métodos tradicionales.

Song y Scaramuzza (2022) combinan la técnica de aprendizaje por refuerzo basada en la búsqueda de políticas con el control predictivo basado en modelos (MPC) para el control de un dron cuadricóptero. En este enfoque, la búsqueda de políticas permite aprender automáticamente políticas complejas a partir de datos experimentales, mientras que el MPC proporciona un control óptimo utilizando modelos y optimización de trayectorias. La combinación de ambos enfoques facilita la optimización de políticas de manera auto supervisada, logrando un rendimiento robusto y en tiempo real en simulaciones y aplicaciones reales, como el control de un cuadricóptero a través de puertas en movimiento rápido.

Liu et al. (2022) desarrollan una estrategia de control para drones cuadricópteros utilizando un modelo de referencia basado en aprendizaje por refuerzo profundo (RL). Este enfoque innovador emplea redes neuronales profundas y el algoritmo deep deterministic policy gradient (DDPG) para diseñar un controlador de vuelo que traduce directamente los estados y objetivos del dron en comandos de control. La validación de esta estrategia se llevó a cabo mediante experimentos en simulación numérica en Matlab y pruebas de vuelo reales, demostrando su capacidad para eliminar oscilaciones y errores en estado estable, y ofreciendo un rendimiento robusto frente a diversas interferencias externas.

Sato y Iwase (2019) presentan una estrategia para lograr vuelos largos y estables de un dron hexacóptero destinado a la inspección forestal eficiente y segura. El modelo de comportamiento del dron se derivó utilizando el formalismo de Newton-Euler, y se modeló el comportamiento de las hélices de paso variable para obtener una dinámica precisa. Los experimentos avanzados de vuelo incluyeron el mantenimiento de altitud utilizando un sensor láser, confirmando la efectividad del control de altitud en las pruebas realizadas.

Ahn et al. (2023) diseñan un controlador de modo deslizante (CSMC) utilizando una ley de acercamiento adaptativa con un algoritmo super-twisting para un dron cuadricóptero. El objetivo del controlador es mejorar el control de vuelo y la estabilidad del dron mediante una ley adaptativa basada en propiedades exponenciales. El modelo dinámico se basa en la dinámica de Newton, y las pruebas de desempeño se llevaron a cabo en un entorno simulado utilizando Matlab y Simulink, enfocándose en el seguimiento de trayectoria de posición.

Finalmente, Hwang et al. (2023) proponen un nuevo método de entrenamiento denominado SeSAC (Stepwise learning with positive buffer and Cool-down alpha) para el aprendizaje eficiente de UAVs de ala fija en entornos de espacio continuo de estados y acciones. A diferencia de

enfoques anteriores que asumen entornos simplificados, SeSAC inicia con misiones fáciles y aumenta gradualmente la dificultad, abordando la ineficiencia de aprender tareas desafiantes desde el principio. Este algoritmo fue verificado en un entorno de vuelo de 6 GDL utilizando el simulador JSBSim.

4.2 Evasión de Obstáculos

Safa et al. (2021) presentan un innovador método de detección diseñado para diferenciar entre objetivos y ruido de fondo en aplicaciones interiores con drones equipados con Lidar. Aprovechando los avances recientes en la detección de objetivos no lineales, desarrollaron un detector de alto rendimiento y baja complejidad. Este método fue validado experimentalmente con datos adquiridos mediante un radar montado en un dron, demostrando su eficacia en entornos con numerosos reflejos.

Liang et al. (2023) proponen un método eficaz y seguro para la evasión de obstáculos en drones cuadricópteros en entornos complejos de baja altitud. Utilizan un sensor LiDAR para la percepción de obstáculos y procesan los datos con un algoritmo de histograma de campo vectorial (VFH) para determinar la velocidad deseada del dron. La velocidad calculada se envía al sistema de control de vuelo del cuadricóptero para realizar una evasión autónoma. La viabilidad de este método fue confirmada a través de simulaciones en Gazebo, mostrando su efectividad en la práctica.

Kalidas et al. (2023) desarrollan un sistema de planificación de trayectorias y evasión de obstáculos tanto estáticos como dinámicos para drones cuadricópteros, empleando tres estrategias de aprendizaje por refuerzo. Los entornos simulados, creados con Unreal Engine 4 en AirSim, permitieron la evaluación de redes neuronales convolucionales con una capa final completamente

conectada. Entre los algoritmos evaluados, SAC demostró el mejor rendimiento, seguido por DQN, mientras que PPO mostró limitaciones en entornos 3D grandes con actores dinámicos.

Aswini et al. (2022) presentan un enfoque de detección de obstáculos que combina un algoritmo de una sola etapa, YOLO V3, con un método de estimación de distancia para drones. La red preentrenada es capaz de identificar obstáculos y estimar distancias sin necesidad de sensores adicionales o GPS. Esto permite al dron desviar su vuelo de manera eficiente para evitar colisiones. La integración del módulo de medición de distancia en una red de aprendizaje profundo preentrenada representa un avance significativo en la detección y evasión de obstáculos.

Yang et al., (2021) introducen un sistema de predicción de profundidad monocular y evasión de obstáculos en tiempo real para drones cuadricópteros. Utilizan una red neuronal convolucional probabilística (pCNN) ligera que combina datos de profundidad escasa con imágenes RGB para obtener predicciones de profundidad precisas. Los resultados experimentales demuestran que esta técnica es de 1.8 a 5.6 veces más rápida que los métodos actuales, y mejora significativamente la precisión en la predicción de profundidad. La pCNN fue aplicada con éxito en simulaciones y entornos reales, reduciendo la probabilidad de colisión y aumentando la tasa de éxito en la evasión de obstáculos.

Panait (2022) exploran el uso de redes neuronales y redes de reflejos físicamente modeladas para generar comportamientos robustos en drones cuadricópteros autónomos o semi-autónomos. A pesar de que las redes neuronales modernas requieren grandes recursos computacionales, las redes biomiméticas de Tilden, que integran sensorialidad y lógica predefinida, ofrecen una alternativa eficiente en términos de consumo energético y fiabilidad para acciones rápidas. Este estudio sugiere que redes neuronales simples pueden ser implementadas en

sistemas de vuelo de drones, reduciendo la necesidad de hardware y mejorando la eficiencia energética.

Sang-Yun & Yong-Won (2019) investigan el uso del aprendizaje por refuerzo (RL) para entrenar drones en evasión de obstáculos, comparando algoritmos de RL con espacios de acción discretos (DD-DQN, Double DQN, Dueling DQN, DQN) y continuos (TRPO, PPO, ACKTR) en entornos simulados en AirSim y Unreal Engine, así como en pruebas con pilotos humanos. Los resultados indican que los algoritmos continuos superan a los discretos y tienen un rendimiento similar al de pilotos expertos. El estudio demuestra que combinar datos de imágenes RGB con mapas de profundidad produce el mejor rendimiento y propone una red de segmentación para el entrenamiento del espacio de acción continuo, eliminando la necesidad de etiquetado manual.

Kim et al. (2022) presentan un algoritmo de evasión de obstáculos para un pequeño dron hexacóptero que utiliza una cámara monocular junto con aprendizaje profundo por refuerzo (DRL). En lugar de imágenes RGB sin procesar, el DRL emplea imágenes de profundidad obtenidas mediante una red neuronal convolucional, lo que ayuda a mitigar las diferencias entre la simulación y el entorno real, mejorando la eficacia del algoritmo. Las pruebas realizadas tanto en simulación con ROS2 y Gazebo como en entornos reales muestran que el algoritmo D3QN supera a otros enfoques de Q-learning convencionales como DQN y DDQN.

Aldao et al. (2022) desarrollan un algoritmo de evasión de obstáculos para la navegación autónoma de cuadricópteros en entornos interiores de edificios para monitoreo de construcciones civiles. El algoritmo es capaz de evitar tanto obstáculos fijos como móviles, calcular trayectorias óptimas en función de desviaciones en tiempo y posición respecto a la ruta planificada, y considera las limitaciones de rendimiento de los UAV en los protocolos de evasión. Las pruebas en entornos

simulados demostraron un consumo de recursos computacionales asequibles para la implementación en tiempo real.

Tu y Juang, (2023) presentan un estudio sobre la planificación de trayectorias y la evasión de obstáculos para drones hexacópteros utilizando algoritmos de aprendizaje por refuerzo Q-learning y SARSA. La simulación del vuelo se realizó con AirSim para interactuar con el entorno y recolectar experiencias necesarias para el entrenamiento del agente. El algoritmo de Q-learning demostró un mejor rendimiento en términos de eficiencia computacional y longitud de ruta en comparación con SARSA. Además, se emplean redes neuronales profundas (DQN) y sensores ultrasónicos para mantener una distancia segura con los objetos durante la evasión de obstáculos.

Xue (2021) propone un método de aprendizaje profundo por refuerzo que utiliza un Variational Autoencoder (VAE) para el preprocesamiento de datos de imágenes. Este enfoque, validado a través de pruebas en Unreal Engine y AirSim, permite que un dron cuadricóptero aprenda de manera eficiente a evitar obstáculos utilizando una cámara de profundidad sin sensores adicionales. El uso del algoritmo SAC (soft actor-critic) como técnica de aprendizaje no dependiente de hiperparámetros demuestra que requiere una función de recompensa de alta calidad para optimizar la eficiencia. Sin embargo, el estudio revela que el dron tiene dificultades para evitar ciertos tipos de obstáculos que requieren ajustes de velocidad específicos.

Nhair y Al-Assadi (2020) desarrollan un sistema de navegación autónoma para un dron hexacóptero. El algoritmo se ejecuta en una pequeña computadora montada en el dron para reducir el tiempo de retardo entre la captura de imágenes y el procesamiento en la estación de control terrestre (GCS). La placa de control, equipada con varios sensores y utilizando el firmware Ardupilot, permite la comunicación bidireccional con la GCS a través del protocolo MAVLink. La cámara web frontal del dron captura imágenes que se procesan en tiempo real para detectar

obstáculos y generar un vector de velocidad adecuado, optimizando así la capacidad de evitar colisiones.

4.3 Planificación de trayectoria

Rojas-Perez y Martinez-Carranza (2020) presentan DeepPilot, una arquitectura de redes neuronales convolucionales (CNN) diseñada para permitir que un dron navegue autónomamente a través de una pista de carreras utilizando como entrada imágenes de cámara y como salida los comandos que tomaría un piloto para estas entradas. DeepPilot predice comandos de vuelo, como ángulos de cabeceo y alabeo, la velocidad rotacional de guiñada y la velocidad vertical. Estos comandos se envían al controlador interno del dron para facilitar su navegación autónoma. El sistema fue evaluado en pistas de carrera simuladas en Gazebo, a 25 fps, mostrando una navegación precisa al cruzar todos los marcos objetivos en orden, y recorrer la pista en tiempos competitivos.

Du et al. (2023) proponen una técnica sistemática de planificación y control para evasión de obstáculos autónoma y flexible para drones cuadricópteros, denominada EF-TTOA (Environment Feature-based Trajectory Tracking and Obstacle Avoidance). El planificador EF construye un frente rápido con un espacio de búsqueda más pequeño y un respaldo con una optimización concisa para suavizar y generar trayectorias libres de colisiones en esquinas. La entrada de control se obtiene resolviendo el problema CBF-QP (Control Barrier Function-Quadratic Problem) con una solución de forma cerrada. Para los obstáculos en movimiento, el controlador, en lugar del planificador, realiza las acciones más rápidamente. Las pruebas se realizaron en el simulador Gazebo.

Al Younes y Barczyk (2022) presentan un planificador de trayectorias para sistemas dinámicos no lineales, como el obtenido para un dron cuadrirotor, basado en aprendizaje profundo por refuerzo (DRL). La metodología se aplica para optimizar los parámetros del enfoque de planificación de trayectorias basado en horizonte predictivo de modelo no lineal (NMPH). El diseño resultante, denominado 'NMPH adaptativo', ajusta los parámetros de NMPH en tiempo real utilizando dos algoritmos de DRL basados en actor-crítico: DDPG y SAC. Ambos enfoques fueron entrenados y evaluados en un dron aéreo dentro de un entorno de simulación en AirSim, mostrando una mejora significativa en el rendimiento de vuelo en comparación con el NMPH convencional. SAC superó a DDPG en términos de velocidad de aprendizaje, capacidad para manejar un mayor conjunto de parámetros y rendimiento general de vuelo.

Hirota et al. (2020) proponen una nueva aplicación para lograr un dron altamente confiable en un entorno de edge computing. Este estudio se enfoca en situaciones inesperadas durante el vuelo del dron, calculando contramedidas para diversas situaciones previstas, centrándose especialmente en la planificación de trayectorias local. La paralelización de la planificación de trayectorias locales se propone para calcular múltiples rutas de escape frente a obstáculos esperados.

Hodgson et al. (2022) presentan un modelo innovador de planificación de vuelos de drones a baja altitud, basado en Sistemas de Información Geográfica (SIG) y respaldado por datos LiDAR de alta resolución espacial, diseñado para asistir en la recolección de muestras de agua en entornos estuarinos bajo diversas condiciones de marea. Este modelo de visibilidad permite simular diferentes escenarios de vuelo y ubicaciones de pilotos remotos, asegurando operaciones seguras y legales dentro del alcance visual, mientras se cumplen los objetivos de investigación. Se destaca la necesidad de datos geoespaciales precisos, especialmente para operaciones autónomas de baja

altitud sobre agua, donde la precisión vertical es crítica. El modelo se aplicó a un dron cuadricóptero con un recipiente para extracción de muestras, operado a una distancia de 1600 milímetros, teniendo en cuenta el caudal de agua.

Jarray et al. (2022) proponen una variante de optimización multiverso multiobjetivo (PMOMVO) basada en un modelo de CPU multicore maestro-esclavo para resolver el problema de planificación de rutas de UAV en un entorno dinámico con obstáculos en movimiento. Se introduce una paralelización eficiente del procesamiento basada en una arquitectura de CPU multicore maestro-esclavo para superar los límites de los algoritmos estándar MOMVO en términos de consumo de tiempo de cómputo. La reducción en el tiempo de cálculo del algoritmo PMOMVO paralelo contribuyó a la efectividad de la estrategia de planificación de rutas propuesta en términos de evitar colisiones con obstáculos en movimiento y zonas de paso estrechas. Los resultados muestran la efectividad y superioridad de los algoritmos PMOMVO paralelos propuestos para la planificación de rutas de UAV con problemas de evasión de colisiones en entornos 3D dinámicos simulados en Matlab.

Jung et al. (2021) proponen un marco detallado de vuelo autónomo de drones para la exploración de entornos desconocidos. El método utiliza un mapa ESDF para volar a través de una trayectoria de vuelo libre de colisiones, seguido de un método de selección de puntos finales. El marco de autonomía ligero se ha probado en diversas simulaciones usando ROS y Gazebo, y en experimentos en minas reales, demostrando con éxito el rendimiento del algoritmo. Se usa tanto información proveniente de LIDAR, como de una cámara 2D con una IMU para aumentar la robustez. Finalmente, el estudio se enfoca en planificadores locales y excluye información sobre planificadores globales.

Kurdel et al. (2022) buscan proporcionar herramientas para la decisión final de si es conveniente implementar un servicio adicional en operaciones de rescate en entornos alpinos. Estos UAVs podrían entregar medicamentos, desfibriladores y evaluar emergencias. Se presenta un método estadístico fiable para evaluar el éxito del vuelo en un corredor definido en los Altos Tatras. Se utilizó una simulación de un problema con pérdida local de señal GPS y se emplearon métodos de varianza estadística. La esperanza matemática ilustra la ruta ideal, mientras que la dispersión alrededor de esta ruta se considera como el corredor de vuelo. Los datos obtenidos pueden utilizarse para verificar y evaluar el éxito de los vuelos, así como para el proceso de toma de decisiones en la selección de varios UAV disponibles o para evaluar las habilidades de los operadores de UAV.

Ortner et al. (2021) resaltan la importancia de considerar la aceleración y desaceleración en la planificación de trayectorias, especialmente en aplicaciones con drones donde la velocidad no es lineal. Destacan que la aceleración y desaceleración tienen un impacto significativo en el tiempo total de vuelo, siendo crucial para aplicaciones críticas en tiempo, como búsqueda y rescate. Presentan un nuevo enfoque de planificación de trayectorias, Radial Gradient Accent (RGA), que tiene en cuenta la aceleración/desaceleración. Este método sigue una estrategia ascendente de gradiente local que minimiza los giros localmente mientras maximiza la acumulación de probabilidades. Las pruebas fueron hechas en un dron hexacóptero en un entorno real.

Tang et al. (2024) presentan un algoritmo de aprendizaje profundo por refuerzo mejorado para la planificación dinámica de trayectorias de UAVs. A través de un entrenamiento extenso por simulación, se desarrollan políticas de planificación de trayectorias para escenarios estáticos y dinámicos. La metodología combina la traducción del modelo MDP al marco de aprendizaje por refuerzo por medio del algoritmo D3QN, y el diseño de una estructura de red competitiva basada

en TensorFlow. Las pruebas realizadas en por simulación númerica muestran que la estrategia propuesta puede generar rutas seguras evitando zonas de obstáculos adversarios. En comparación con otros enfoques como A*, RRT, DQN y DDQN, el método exhibe un rendimiento superior en términos de longitud de trayectoria y tiempo de planificación en escenarios estáticos, mientras que en escenarios dinámicos demuestra una navegación exitosa evitando obstáculos.

F. Yang et al. (2022) proponen un algoritmo mejorado de RRT (Rapid-exploration Random Tree). El algoritmo RRT tiene problemas como alta aleatoriedad, velocidad de convergencia lenta, y trayectorias de vuelo curvadas. Para abordar estas limitaciones, se introduce la optimización de colonias de hormigas (ACO) para hacer que la planificación de trayectorias sea óptima asintóticamente. El algoritmo optimizado establece feromonas en la trayectoria obtenida por RRT y selecciona el siguiente punto de extensión según la concentración de feromonas, convergiendo así hacia una solución ideal a través de iteraciones. Se realizaron pruebas con MATLAB para verificar la efectividad y superioridad del algoritmo. El método propuesto garantiza la convergencia a la solución óptima, lo que lo hace adecuado para la planificación previa al despegue de UAVs.

Yu et al. (2023) introducen un planificador local agresivo basado en gradientes llamado SpeedFirst. Los métodos de planificación actuales son muy conservadores y desaprovechan la agilidad de los cuadricópteros. Introducen una estrategia de generación de información de gradiente de distancia que reemplaza los puntos de control en obstáculos con una ruta Hybrid-A* para garantizar la seguridad. Además, presentan un término de costo de duración de tiempo novedoso y agresivo para abordar la inviabilidad y mejorar la velocidad general de la trayectoria. Las pruebas de simulación y vuelo real demuestran que SpeedFirst mejora efectivamente la

velocidad de planificación y vuelo, permitiendo vuelos autónomos más agresivos en cuadricópteros.

N. Zhang et al. (2021) abordan la evasión de colisiones en misiones de drones en entornos urbanos complejos, considerando tanto obstáculos estáticos como dinámicos en vuelo de baja altitud. Para gestionar los obstáculos estáticos, introducen el método 3D voxel Jump Point Search (JPS) para generar un camino de referencia global que guía al dron a diferentes altitudes. El camino optimizado reduce el riesgo de colisiones estáticas, acorta el rango de vuelo y facilita los giros, en comparación con el camino generado directamente por el 3D voxel JPS. Para las amenazas dinámicas, desarrollan un esquema de resolución de colisiones en tiempo real basado en la formulación de un problema de programación cuadrática no lineal (NLP). Las pruebas se realizaron en entornos urbanos simulados con Gazebo y la implementación en ROS.

4.4 Síntesis de tendencias y avances en el control de vuelo y evasión de obstáculos en drones

Al analizar los trabajos revisados sobre control de vuelo, seguimiento de trayectoria y evasión de obstáculos en drones, se pueden identificar varios patrones y tendencias clave. La mayoría de los estudios emplean simuladores avanzados como Matlab/Simulink, AirSim, Gazebo y Unreal Engine para validar y probar sus algoritmos. Estos simuladores permiten la evaluación de estrategias de control y evasión en entornos realistas y dinámicos, facilitando la transición hacia aplicaciones del mundo real.

En cuanto a los algoritmos de control, los enfoques varían desde controladores clásicos como PID hasta métodos más avanzados como control por rechazo activo de perturbaciones (ADRC), control por modos deslizantes (SMC) y control predictivo basado en modelos (MPC). El uso de aprendizaje por refuerzo (RL) ha ganado popularidad, especialmente en tareas complejas

de vuelo y evasión de obstáculos, mostrando un desempeño robusto tanto en simulaciones como en aplicaciones reales.

El aprendizaje por refuerzo y redes neuronales se ha destacado como una técnica altamente efectiva para la navegación autónoma y la evasión de obstáculos. Algoritmos como SAC, DDPG, DQN y sus variantes (como D3QN) se utilizan para entrenar drones en entornos simulados. Estos enfoques han demostrado ser efectivos en la mejora de la estabilidad y el rendimiento frente a perturbaciones y obstáculos, gracias a su capacidad para aprender y adaptarse a diferentes condiciones.

En lo que respecta a la evasión de obstáculos, los métodos actuales combinan tecnologías de sensores como LiDAR, cámaras monoculares y redes neuronales para la detección y evasión eficiente. Algoritmos basados en histograma de campo vectorial (VFH), aprendizaje profundo y redes neuronales convolucionales (CNN) son comunes, mostrando un enfoque hacia la integración de múltiples tecnologías para mejorar la percepción y la toma de decisiones en tiempo real. En la se muestra un esquema del estado del arte de la navegación autónoma de acuerdo a las divisiones seleccionadas.

Figura 8

Resumen esquemático de investigaciones realizadas en navegación autónoma de drones

1Finalmente, la eficiencia computacional y energética ha sido un tema de interés en varios estudios, especialmente en contextos donde el procesamiento de datos en tiempo real es crítico. Se destacan las redes biomiméticas y el uso de algoritmos que optimizan el consumo de recursos, lo cual es fundamental para la implementación de sistemas autónomos en drones con capacidades limitadas.

En conclusión, el estado del arte revela un enfoque creciente hacia la integración de técnicas avanzadas de control y aprendizaje automático en drones, apoyado por simuladores robustos que permiten pruebas exhaustivas antes de la implementación en el mundo real. La tendencia hacia el uso de aprendizaje por refuerzo y la combinación de diferentes tecnologías de percepción sugiere un camino hacia sistemas más autónomos y eficientes en la navegación y control de UAVs en entornos complejos.

5. Modelado y diseño de controladores para dron hexacóptero

En esta sección se explica cómo se obtuvieron las EMS, y el procedimiento a seguir para diseñar los controladores. Para la obtención del modelo se utilizó la mecánica de Lagrange, cuya implementación se explica paso a paso hasta obtener las ecuaciones de movimiento. El código de Matlab con el que se obtienen de manera eficiente las ecuaciones de movimiento y las matrices requeridas para los controladores y el observador de Kalman se da en el Apéndice A.

5.1 Obtención de las ecuaciones del movimiento mediante el formalismo de Lagrange

El formalismo de Lagrange es un método basado en energías, que se utiliza para derivar directamente las EMS. Este método basado en energía requiere del cálculo de la energía cinética y potencial total del sistema que permite obtener una cantidad escalar conocida como Lagrangiano L, el cual es la diferencia entre la energía cinética $\mathrm{E_K}$ y potencial $\mathrm{E_P}$ del sistema, ver Ecuación (29).

$$\mathrm{L} = \mathrm{E_K} - \mathrm{E_P} \tag{29}$$

El enfoque de Lagrange además requiere el uso de coordenadas generalizadas que se seleccionan a conveniencia del problema, para el caso de un cuerpo rígido con movimiento de traslación y rotación en tres dimensiones se pueden elegir para el vector de coordenadas generalizadas los valores de las coordenadas de posición con respecto a SRI de un punto seleccionado en el cuerpo y los ángulos de Euler que permiten definir la orientación del SRC con respecto al SRI.

Además del Lagrangiano se requieren las fuerzas o torques generalizados para cada coordenada generalizada q_i para obtenerlos se puede usar (30), donde $\vec{r}$ es la posición del punto de aplicación de la fuerza externa $\vec{F}$. Cuando se trata de una coordenada generalizada lineal las

unidades de Q_{q_i} son de fuerza como el Newton, pero para coordenadas generalizades angulares Q_{q_i} tiene unidades de torque como el Newton-metro.

$$Q_{q_i} = -\vec{F} \cdot \frac{\partial \vec{r}}{\partial q_i} \tag{30}$$

Para el caso de un torque externo que mantiene su dirección respecto a los ejes del SRC, el componente del torque a lo largo de cada eje del SRC corresponde directamente al torque generalizado asociado con el ángulo de Euler correspondiente en ese sistema. Por ejemplo, si el torque está alineado con el eje x del SRC, el torque generalizado para el ángulo de Euler que controla la rotación alrededor del eje x es igual al componente del torque en esa dirección, τ_x. De manera similar, si el torque está alineado con el eje y, el torque generalizado para el ángulo de Euler asociado con el eje y es τ_y Finalmente, si el torque está alineado con el eje z, el torque generalizado para el ángulo de Euler asociado con el eje z es τ_z. Así, el componente del torque en cada eje del SRC se traduce directamente en el torque generalizado para la coordenada angular correspondiente.

También se suelen modelar los efectos de la disipación de energía que ocurren por la presencia de amortiguadores viscosos o fuerzas de fricción por medio de la función de disipación para un amortiguador viscoso dada en (31) (Szegleti & Márkus, 2020).

$$f_D = \frac{1}{2} c_a \dot{q}_i^{\,2} \tag{31}$$

Donde c_a es el coeficiente amortiguamiento viscoso, $\dot{q}_i$ es el cambio de la coordenada generalizada i con respecto al tiempo. La fuerza generalizada en este caso se puede obtener por medio de (32), esta fuerza siempre se opone al cambio de la velocidad.

$$Q_{q_i} = -\frac{\partial f_D}{\partial \dot{q}_\iota} \tag{32}$$

Por último, para obtener las ecuaciones de movimiento del sistema primero se utiliza la ecuación de Euler-Lagrange dada en (33), lo que da un sistema de ecuaciones lineales para los componentes del vector $\ddot{q}$, que se pueden despejar para obtener una ecuación para cada componente del $\ddot{q}$ las cuales son las ecuaciones de movimiento.

$$\frac{\mathrm{d}}{\mathrm{dt}}\left(\frac{\partial \mathrm{L}}{\partial \dot{\mathrm{q}}_\mathrm{i}}\right) - \frac{\partial \mathrm{L}}{\partial \mathrm{q}_\mathrm{i}} = \mathrm{Q}_\mathrm{i} \tag{33}$$

5.1.1 Variables Generalizadas del dron

El dron se modela como un cuerpo rígido con movimiento tridimensional de seis grados de libertad (GDL). Para indicar la posición del dron, basta con especificar la posición del SRC con respecto al SRI dando sus coordenadas cartesianas (x, y, z). Mientras que para definir la orientación se usan los ángulos de Euler. El origen del SRC se coloca en el CM del dron. Los GDL formaron el vector de coordenadas generalizadas q, mostrado en (34).

$$\mathrm{q} = [x, \mathrm{y}, \mathrm{z}, \phi, \theta, \psi]^\mathrm{T} \tag{34}$$

El vector de velocidades generalizadas $\dot{\mathrm{q}}$, dado en (35) corresponden a la variación instantánea de q con respecto al tiempo, es decir las tres velocidades lineales y las tres velocidades angulares.

$$\dot{\mathrm{q}} = \left[\dot{\mathrm{x}}, \dot{\mathrm{y}}, \dot{\mathrm{z}}, \dot{\phi}, \dot{\theta}, \dot{\psi}\right]^\mathrm{T} \tag{35}$$

Y a su vez la variación de $\dot{\mathrm{q}}$ con respecto al tiempo, es el vector de las aceleraciones generalizadas $\ddot{\mathrm{q}}$, dado en la Ecuación (36).

$$\ddot{\mathrm{q}} = \left[\ddot{\mathrm{x}}, \ddot{\mathrm{y}}, \ddot{\mathrm{z}}, \ddot{\phi}, \ddot{\theta}, \ddot{\psi}\right]^\mathrm{T} \tag{36}$$

5.1.2 Cálculo de la energía cinética del dron.

La energía cinética traslacional del dron $\mathrm{Ec_T}$, se obtiene mediante la Ecuación (37), donde $\mathrm{m_D}$ es la masa del dron y $\vec{V}_D$ es la velocidad de translación dada en la Ecuación (38).

$$\mathrm{Ec_T} = \frac{1}{2}\mathrm{m_D} \cdot \vec{\mathrm{V}}_\mathrm{D} \cdot \vec{\mathrm{V}}_D \tag{37}$$

$$\vec{\mathrm{V}}_\mathrm{D} = \dot{\mathrm{x}}\,\hat{\mathrm{i}} + \dot{\mathrm{y}}\,\hat{\mathrm{j}} + \dot{\mathrm{z}}\,\hat{\mathrm{k}} \tag{38}$$

La energía cinética rotacional $\mathrm{Ec_R}$, se calcula utilizando la ecuación (39). La matriz de inercia del dron con respecto a los vectores del cuerpo se simboliza con $\mathrm{I_{D/SRC}}$, el dron se considera simétrico con respecto a los planos X-Z y Y-Z, por lo que esta matriz es diagonal, ver Ecuación (40). La ecuación (41) permite obtener la velocidad angular como función de la variación instantánea de los ángulos de Euler con respecto al tiempo (Schaub & Junkins, 2018. p, 94).

$$\mathrm{Ec_R} = \frac{1}{2} \cdot \vec{\omega}^{\mathrm{T}}{}_{\mathrm{D/SRC}} \cdot \mathrm{I_{D/SRC}} \cdot \vec{\omega}_{\mathrm{D/SRC}} \tag{39}$$

$$\mathrm{I_{D/SRC}} = \begin{bmatrix} \mathrm{I_{xx}} & 0 & 0 \\ 0 & \mathrm{I_{yy}} & 0 \\ 0 & 0 & \mathrm{I_{zz}} \end{bmatrix} \tag{40}$$

$$\vec{\omega}_{\mathrm{D/SRC}} = \begin{bmatrix} 1 & 0 & -\sin(\theta) \\ 0 & \cos(\phi) & \sin(\phi)\cos(\theta) \\ 0 & -\sin(\phi) & \cos(\phi)\cos(\theta) \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{41}$$

5.1.3 Cálculo de la energía potencial del dron.

El cambio de energía potencial ocurre debido al desplazamiento $\Delta\vec{\mathrm{r}}$ del dron con masa $\mathrm{m_D}$ en un campo gravitatorio de gravedad g. En el escenario de un campo conservativo, la energía potencial se determina mediante la Ecuación (42), donde Δz representa el cambio en la altura del centro de masa (CM).

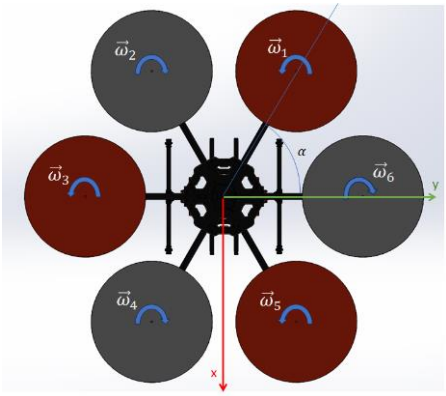
$$\mathrm{Ep} = -\left(-\,\mathrm{m_D} \cdot g \cdot \hat{\mathrm{k}}\right) \cdot \Delta\vec{\mathrm{r}} = \mathrm{m_D} \cdot \mathrm{g} \cdot \Delta\mathrm{z} \tag{42}$$

5.1.4 Fuerzas generalizadas

La Figura 9, muestra una vista superior del dron y la numeración de los propulsores además del sentido de giro de los motores, los brazos del hexarotor están colocados a cada 60°. El sistema de referencia mostrado está colocado en el centro de masa, y la dirección del eje z es hacia afuera de la página.

Figura 9

Vista superior del hexacóptero y numeración de los propulsores



Cada propulsor produce una fuerza de empuje y un torque de reacción a la rotación del motor, ambos paralelos al vector unitario del cuerpo $\hat{\mathrm{k}}_{SRC}$. La fuerza de empuje $\vec{\mathrm{F}}_E$ tiene una relación directamente proporcional con el cuadrado de la velocidad angular del propulsor, siendo K_T la constante de empuje, que depende del propulsor. La fuerza neta producida por los seis motores se calcula usando la Ecuación (43).

$$\vec{\mathrm{F}}_E = \mathrm{K_T}\left(\sum_{\mathrm{i}=1}^{6} \omega_i^2\right) \cdot \mathrm{R_z} \cdot \mathrm{R_y} \cdot \mathrm{R_x} \cdot \hat{\mathrm{k}} \tag{43}$$

La fuerza de arrastre parasita $\vec{F}_A$, tiene lugar cuándo hay una velocidad relativa del dron con respecto al aire. La $\vec{V}_{rel}$ (con v_x, v_y, v_z como sus componentes) es la diferencia entre la velocidad del dron $\vec{V}_D$ y la velocidad del viento $\vec{V}_w$. $\vec{V}_{rel}$ se obtiene mediante la Ecuación (44).

$$\vec{V}_{rel} = \left[v_x, v_y, v_z\right]^T = \vec{V}_D - \vec{V}_w \tag{44}$$

$\vec{F}_A$ se puede estimar como el producto de la $\vec{V}_{rel}$, multiplicada por su valor absoluto, y el coeficiente de arrastre en cada dirección del sistema de coordenadas, simbolizados por medio de c_x, c_y y c_z respectivamente, ver Ecuación (45). Debido a que los ángulos de Euler se mantienen cercanos a cero, no se tomaron en cuenta los efectos del cambio de orientación del dron.

$$\vec{F}_A = c_x v_x|v_x|\hat{\imath} + c_y v_y|v_y|\hat{\jmath} + c_z v_z|v_z|\hat{k} \tag{45}$$

La ecuación (46) muestra la relación entre las componentes de la fuerza total neta $\vec{F}_{Neta}$ y las fuerzas generalizadas de las coordenadas generalizadas de posición, Q_x, Q_y, Q_z.

$$\vec{F}_{Neta} = \left[Q_x, Q_y, Q_z\right]^T = \vec{F}_E - \vec{F}_A \tag{46}$$

5.1.5 Torques generalizados

El giro del motor produce un torque de reacción contrario al sentido de rotación del motor. El torque motor es directamente proporcional al cuadrado de la rotación del propulsor, con constante de proporcionalidad $\mathrm{K_Q}$, la cual se denomina constante de torque motor. La ecuación (47) permite calcular el torque del motor neto $\vec{\tau}_M$.

$$\vec{\tau}_M = \mathrm{K_Q}(\omega_2^2 + \omega_4^2 + \omega_6^2 - \omega_1^2 - \omega_3^2 - \omega_5^2) \cdot \hat{\mathrm{k}}_{SRC} \tag{47}$$

El torque giroscópico neto $\vec{\tau}_G$ surge debido a la rotación del dron combinada con otros cuerpos en rotación, como las hélices del dron que rotan sobre su eje a la vez que rota el marco. Este torque se define como el producto cruz entre el momento angular de las hélices $\vec{L}_{\mathrm{SRC}}$, y la velocidad angular del dron $\vec{\omega}_{D/SRC}$, según la ecuación (48). Aquí I_{prop} representa el momento de inercia de las hélices respecto a su eje de rotación, orientado en la dirección $\hat{k}_{\mathrm{SRC}}$.

$$\vec{\tau}_G = \vec{L}_{SRC} \times \vec{\omega}_{D/SRC} = I_{prop} \cdot (\omega_1 + \omega_3 + \omega_5 - \omega_2 - \omega_4 - \omega_6) \cdot \hat{k}_{\mathrm{SRC}} \times \vec{\omega}_{D/SRC} \tag{48}$$

Las fuerzas de empuje producen adicionalmente torques que afectan las coordenadas generalizadas θ, y ϕ, a los que se les denomina $\vec{\tau}_{F\theta}$ y $\vec{\tau}_{F\phi}$. Basado en la Figura 9 se deducen las ecuaciones (49) y (50) que permiten calcular estos torques.

$$\vec{\tau}_{F\theta} = K_T l_1 sin(\alpha) \cdot (\omega_1^2 + \omega_2^2 - \omega_4^2 - \omega_5^2) \cdot \hat{j}_{SRC} \tag{49}$$

$$\vec{\tau}_{F\phi} = K_T \big(l_1 cos(\alpha) \cdot (\omega_1^2 + \omega_5^2 - \omega_2^2 - \omega_4^2) + l_2 \cdot (\omega_6^2 - \omega_3^2)\big) \cdot \hat{\imath}_{SRC} \tag{50}$$

Donde la distancia en el plano X-Y desde el eje Z a los motores 1, 2, 4, 5 se denota por l_1 y a los motores 3, 6 se denota por l_2. La ecuación (51) muestra la relación entre los componentes del torque neto, $\vec{\tau}_{Neto}$, con el torque generalizado de cada una de las coordenadas generalizadas correspondiente a los ángulos de Euler.

$$\vec{\tau}_{Neto} = \big[Q_\phi, Q_\theta, Q_\psi\big]^T = \vec{\tau}_{F\phi} + \vec{\tau}_{F\theta} + \vec{\tau}_{motor} + \vec{\tau}_G \tag{51}$$

5.1.6 Ecuaciones de movimiento

Para obtener las EMS se utilizó la ecuación de Euler-Lagrange dada en la Ecuación (33). Recordando que el vector Q, ver Ecuación (52), en este caso corresponde a las fuerzas o torques generalizados, provenientes de fuerzas de los actuadores y perturbaciones.

$$Q = \big[Q_x, Q_y, Q_z, Q_\phi, Q_\theta, Q_\psi\big]^T \tag{52}$$

El resultado obtenido se puede escribir de forma compacta por medio de (53).

$$M\ddot{q} + H(q, \dot{q}) = Q \tag{53}$$

Donde M se conoce como la matriz de masa del sistema, para este caso es una matriz cuadrada de 6 filas donde la mayoría de sus valores son ceros, en el conjunto de ecuaciones (54) mostrado a continuación se dan los valores de los términos distintos de cero.

$$M(1,1) = m \tag{54}$$

$$M(2,2) = m$$

$$M(3,3) = m$$

$$M(4,4) = I_{xx}$$

$$M(4,6) = M(6,4) = -I_{xx}sin(\theta)$$

$$M(5,5) = I_{zz} + (I_{yy} - I_{zz})cos^2(\phi)$$

$$M(5,6) = M(6,5) = (I_{yy} - I_{zz})cos(\phi)cos(\theta)sin(\phi)$$

$$M(6,6) = I_{xx} + (I_{yy} - I_{xx})cos^2(\theta) + (I_{zz} - I_{yy})cos^2(\phi)cos^2(\theta)$$

El vector $H(q, \dot{q})$ representa los términos de aceleraciones centrifugas y de Coriolis del sistema, sus componentes se dan en el conjunto de ecuaciones (55).

$$H(1,1) = 0 \tag{55}$$

$$H(2,1) = 0$$

$$H(3,1) = -gm$$

$$H(4,1) = 0.5(\text{Izz} - \text{Iyy})\dot{\theta}^2 \sin(2\,\phi) + (\text{Ixx} - \text{Iyy} + \text{Izz})\,\dot{\psi}\,\dot{\theta}\cos(\theta) + (\text{Iyy} - \text{Izz})\,\dot{\psi}^2 \cos(\phi)\cos^2(\theta)\sin(\phi) + 2\,(\text{Iyy} - \text{Izz})\,\dot{\psi}\,\dot{\theta}\cos^2(\phi)\cos(\theta)\,\dot{\psi}\,\dot{\theta}\cos^2(\phi)\cos\theta$$

$$H(5,1) = \frac{\text{Ixx}\,\dot{\psi}^2 \sin(2\,\theta)}{2} - \text{Ixx}\,\dot{\phi}\,\dot{\psi}\cos(\theta) + (\text{Iyy} - \text{Izz})\,\dot{\phi}\,\dot{\theta}\sin(2\,\phi) - (\text{Izz}\cos^2(\phi) + \text{Iyy}\sin^2(\phi))\dot{\psi}^2 \cos(\theta)\sin(\theta) + (\text{Izz} - \text{Iyy})\,\dot{\phi}\,\dot{\psi}\cos^2(\phi)\cos(\theta) + (\text{Iyy} - \text{Izz})\,\dot{\phi}\,\dot{\psi}\cos(\theta)\sin^2(\phi)$$

$$H(6,1)=(\mathrm{Ixx}+\mathrm{Iyy}-\mathrm{Izz})\dot{\phi}\,\dot{\theta}\cos(\theta)\,Iyy-\mathrm{Ixx})\dot{\psi}\,\dot{\theta}\sin(2\,\theta)+(Iyy-Izz)\dot{\theta}^2\,cos(\phi)\,sin(\phi)\,sin(\theta)+2(Izz-Iyy)\dot{\phi}\,\dot{\theta}\,cos^2(\phi)\,cos(\theta)-2\,Iyy\,\dot{\phi}\,\dot{\psi}\,cos(\phi)\,cos^2(\theta)\,sin(\phi)+2(Izz-Iyy)\dot{\psi}\,\dot{\theta}\,cos^2(\phi)\,cos(\theta)\,sin(\theta)$$

La Ecuación (53) constituye un sistema lineal de ecuaciones para las aceleraciones, por lo que se pueden resolver para despejar los elementos del vector q̈, dando como resultado las EMS, esto se realizó de una manera eficiente por medio de la caja de herramienta de operaciones simbólicas del programa Matlab versión R2023b.

El código del Apéndice A en su parte 2 tiene un fragmento que permite calcular de forma rápida las seis ecuaciones de movimiento este se muestra en la Figura 10; durante esta etapa de la ejecución del código ya se tienen tanto el Lagrangiano como las fuerzas generalizadas, y las ecuaciones de movimiento quedan almacenadas en el conjunto de ecuaciones de la variable `aux`, las ecuaciones de movimiento por su extensión no se incluyen en este texto.

Figura 10

Fragmento de código desarrollado para obtener las ecuaciones de movimiento partiendo de la ecuación (33)

```
for ii=1:6
    dLdqdot(ii) = diff(L,qdot(ii));
    ddt_dLdqdot(ii) = diff(dLdqdot(ii),q(1))*qdot(1) + diff(dLdqdot(ii),qdot(1))*qddot(1)+...
                      diff(dLdqdot(ii),q(2))*qdot(2) + diff(dLdqdot(ii),qdot(2))*qddot(2)+...
                      diff(dLdqdot(ii),q(3))*qdot(3) + diff(dLdqdot(ii),qdot(3))*qddot(3)+...
                      diff(dLdqdot(ii),q(4))*qdot(4) + diff(dLdqdot(ii),qdot(4))*qddot(4)+...
                      diff(dLdqdot(ii),q(5))*qdot(5) + diff(dLdqdot(ii),qdot(5))*qddot(5)+...
                      diff(dLdqdot(ii),q(6))*qdot(6) + diff(dLdqdot(ii),qdot(6))*qddot(6);
    dLdq(ii) = diff(L,q(ii));
    EOM(ii) = simplify(ddt_dLdqdot(ii) - dLdq(ii) - T_ext(ii));
end
aux = solve(EOM,qddot);
```

5.2 Representación en espacio de estados y diseño de controladores

En esta sección se explica: la representación en espacio de estados del dron tanto no-lineal como lineal, el proceso de linealización y se diseñan tres controladores para seguimiento de trayectoria de posición usando las estrategias de control PID, control LQG y control LQG-Integral, estos tres se comparan entre sí.

5.2.1 Representación no lineal en espacio de estados

Hasta este punto se han encontrado las aceleraciones generalizadas en función de las velocidades y coordenadas generalizadas, o lo que es lo mismo las ecuaciones de movimiento, aunque solo usando variables simbólicas. El siguiente paso que se realizó fue sustituir las coordenadas y velocidades generalizadas por variables de estado. La Ecuación (56) muestra el vector de estados $X(t)$ y a que variable generalizada corresponde cada estado.

$$X(t) = [x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}]^T \tag{56}$$
$$= \left[x, \dot{x}, y, \dot{y}, z, \dot{z}, \phi, \dot{\phi}, \theta, \dot{\theta}, \psi, \dot{\psi}\right]^T$$

El vector de entradas al sistema $u(t)$, ver ecuación (57), está conformado por las acciones de control, que en este caso corresponden a las velocidades angulares al cuadrado de los motores.

$$u(t) = [u_1, u_2, u_3, u_4, u_5, u_6]^T = [\omega_1^2, \omega_2^2, \omega_3^2, \omega_4^2, \omega_5^2, \omega_6^2]^T \tag{57}$$

Se define el vector de las perturbaciones del sistema, dado en la Ecuación (59), que incluye las componentes de la velocidad del viento externo.

$$d(t) = \left[v_x, v_y, v_z\right]^T \tag{58}$$

La representación no lineal del sistema requiere también de la variación del vector de estados con respecto al tiempo, $\dot{X}(t)$. En la ecuación (59) se muestra a que variables de estado y a que aceleraciones generalizadas corresponden los componentes del vector $\dot{X}(t)$.

$$\dot{X}(t) = [\dot{x}_1, \dot{x}_2, \dot{x}_3, \dot{x}_4, \dot{x}_5, \dot{x}_6, \dot{x}_7, \dot{x}_8, \dot{x}_9, \dot{x}_{10}, \dot{x}_{11}, \dot{x}_{12}] \tag{59}$$
$$= \left[x_2, \ddot{x}, x_4, \ddot{y}, x_6, \ddot{z}, x_8, \ddot{\phi}, x_{10}, \ddot{\theta}, x_{12}, \ddot{\psi}\right]$$

Debido a que previamente se realizó el despeje de las aceleraciones de la ecuación (53) en función de las velocidades y coordenadas generalizadas, se puede realizar la integración numérica de $\dot{X}(t)$ por medio del programa simulación numérica Simulink R2023b, para obtener $X(t)$ en cada instante.

5.2.2 Determinación de parámetros del sistema

Se realizó un modelo tridimensional del dron hexacóptero que se tiene como referencia, usando el programa de diseño asistido por computadora SolidWorks 2023, con el objetivo de obtener las matrices de inercia y la posición del centro de masa. También se utilizó la herramienta de dinámica de fluido computacional disponible en este, con el objetivo de estimar los coeficientes de arrastre c_x, c_y, c_z. Las constantes K_T y K_Q fueron proporcionadas por el fabricante de los propulsores. Estos y otros valores como la masa y matriz de inercia del dron constituyen los parámetros del sistema que se dan en la Tabla 2.

Tabla 2

Parámetros del sistema

Constante	Valor	Unidad
m_D	0.884	kg
I_{prop}	1.058E-4	kg-m^2
l_1	0.357	m
l_2	0.343	m
c_x	1.616E-2	N- s^2/m

c_y	1.616E-2	N- s^2/m
c_z	4.98E-2	N- s^2/m
α	1.0472	rad
I_{xx}	3.814E-2	kg-m^2
I_{yy}	3.744E-2	kg-m^2
I_{zz}	7.109E-2	kg-m^2
g	9.81	m/s^2
K_T	2.19E-5	N- s^2
K_Q	1.99E-7	N-m-s^2

El resultado que se obtiene luego de remplazar los parámetros del sistema de la Tabla 2, en la ecuación (59) es el sistema no lineal en espacio de estados.

5.2.3 Linealización del sistema

Un sistema en espacio de estados linealizado se representa por medio de las ecuaciones (60) y (61). En donde *A* es la matriz de estado, *B* es la matriz de entrada, *C* es la matriz de salida, *D* es la matriz de transmisión directa, $y(t)$ es la salida linealizada del sistema y *E* es la matriz de perturbación.

$$\dot{\mathrm{X}}(t) = A \cdot \mathrm{X}(t) + B \cdot u(t) + E \cdot d(t) \tag{60}$$

$$y(t) = C \cdot X(t) + D \cdot \mathrm{u}(t) \tag{61}$$

El punto de equilibrio seleccionado es en el que el dron se encuentra en sobrevuelo estático o *hovering* en el que los valores de las variables generalizadas son cero, también se supone que las perturbaciones tienen un valor de cero. Para los valores iniciales se supone que el SRI está a una

altura que evita el contacto del dron con el suelo y que SRC y SRI coinciden en el instante inicial. El valor de las acciones de control para este punto de equilibrio se muestra en la ecuación (62).

$$u_1 = u_2 = u_3 = u_4 = u_5 = u_6 = \frac{g\,m}{6\,K_T} \tag{62}$$

Para obtener la matriz A se calculó el jacobiano entre el vector $\dot{\mathrm{X}}(t)$ con respecto a $X(t)$ esta se muestra en (63). Para obtener la matriz B se calculó el jacobiano entre el vector $\dot{\mathrm{X}}(t)$ con respecto al vector $u(t)$, esta se muestra en (64), la matriz E, debido a que los vientos son cero en el estado de equilibrio es una matriz de ceros de 12 filas y 3 columnas. Para todas las matrices se sustituyeron los valores del vector de entradas en equilibrio.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & g & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -g & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{63}$$

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{K_T}{m} & \frac{K_T}{m} & \frac{K_T}{m} & \frac{K_T}{m} & \frac{K_T}{m} & \frac{K_T}{m} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \sigma_1 & -\sigma_1 & -\frac{K_T l_2}{I_{xx}} & -\sigma_1 & \sigma_1 & \frac{K_T l_2}{I_{xx}} \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \sigma_2 & \sigma_2 & 0 & -\sigma_2 & -\sigma_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{K_Q}{I_{zz}} & \frac{K_Q}{I_{zz}} & -\frac{K_Q}{I_{zz}} & \frac{K_Q}{I_{zz}} & -\frac{K_Q}{I_{zz}} & \frac{K_Q}{I_{zz}} \end{bmatrix} \quad (64)$$

Donde:

$\sigma_1 = \frac{K_T l_1 sin\alpha}{I_{xx}}, \sigma_2 = \frac{K_T l_1 cos\alpha}{I_{yy}}$

Se sustituyeron las variables simbólicas por los valores de las constantes dados en la Tabla 2, para obtener las matrices *A* y *B* con valores numéricos. Los valores medidos (indirectamente) fueron las coordenadas generalizadas, pare este fin se usó una unidad de medición inercial, IMU por sus siglas en inglés, con esta información se procedió a crear la matriz C, que se muestra en (65). La matriz D es una matriz de ceros de 6 filas y 9 columnas.

$$\mathrm{C} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \quad (65)$$

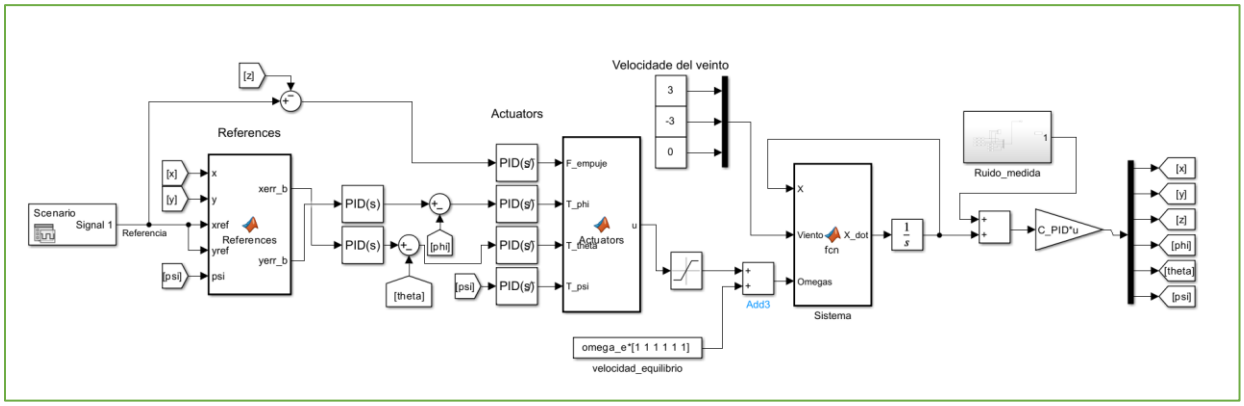
5.2.4 Controlador PID

Un controlador PID es propuesto para cada uno de los estados medidos. En la Figura 11, se muestra la estructura del controlador el cual fue diseñado usando el programa Simulink. El dron es un sistema subactuado en el que para que haya un desplazamiento en y o x se requiere una

rotación con respecto a ϕ y θ respectivamente, por lo que cuando se usa control PID se requiere que este sea en cascada. En el bloque de función llamado *references*, se reciben los valores de referencia para el seguimiento de trayectoria y los valores medidos de la posición y el ángulo de guiñada, para entregar como resultado el error de posición expresado de x e y en el SRC.

Figura 11

Diagrama de bloques del controlador PID y conexión a la planta



El control PID aplicado a drones, requiere que se desacoplen las acciones de control en cuatro, una fuerza de empuje: F, y tres torques: T_1, T_2 y T_3 con respecto a los ejes del SRC. Para realizar la conversión a las entradas al sistema que son las velocidades de rotación al cuadrado de los rotores, se requiere de la matriz de mezclado M_x, dada en la ecuación (66), que se multiplica por el vector formado por las cuatro acciones de control dadas por el controlador PID, ver ecuación (67), estás velocidades pasan por un limitador que restringe las velocidades en el rango de trabajo antes de que las entradas se apliquen al sistema.

$$M_x = \begin{bmatrix} \frac{1}{6K_T} & \frac{1}{2K_T l_1 cos(\alpha)} & \frac{1}{2K_T l_1 sin(\alpha)} & \frac{-1}{6K_Q} \\ \frac{1}{6K_T} & \frac{-1}{2K_T l_1 cos(\alpha)} & \frac{1}{2K_T l_1 sin(\alpha)} & \frac{1}{6K_T} \\ \frac{1}{6K_T} & \frac{-1}{2K_T l_2} & 0 & \frac{-1}{6K_Q} \\ \frac{1}{6K_T} & \frac{-1}{2K_T l_1 cos(\alpha)} & \frac{-1}{2K_T l_1 sin(\alpha)} & \frac{1}{6K_Q} \\ \frac{1}{6K_T} & \frac{1}{2K_T l_1 cos(\alpha)} & \frac{-1}{2K_T l_1 sin(\alpha)} & \frac{-1}{6K_Q} \\ \frac{1}{6K_T} & \frac{1}{2K_T l_2} & 0 & \frac{1}{6K_Q} \end{bmatrix} \quad (66)$$

$$[u_1, u_2, u_3, u_4, u_5, u_6]^T = M \cdot [F, T_1, T_2, T_3]^T \quad (67)$$

La ecuación de transferencia de los controladores PID (C_{PID}), está dada por ecuación (68), siendo P_c el factor de la acción proporcional, I_C el factor de la acción integral, D_c el factor de la acción derivativa y N el coeficiente del filtro.

$$C_{PID} = P_c + I_C \frac{1}{s} + D_c \frac{N}{1 + \frac{N}{s}} \quad (68)$$

Se realizó un proceso de sintonización para determinar las constantes del controlador PID usando el software Simulink R2023b, las constantes obtenidas se dan en la Tabla 3. El archivo de controlador PID en Simulink se incluye en el **Apéndice B**.

Tabla 3

Valores de los factores de los controladores PID para los controladores sintonizados

Variable controlada	P_c	I_C	D_c	N
$x_1 = x$	0.0993	0.00363	0.603	717
$x_3 = y$	-0.112	-0.00456	-0.659	768

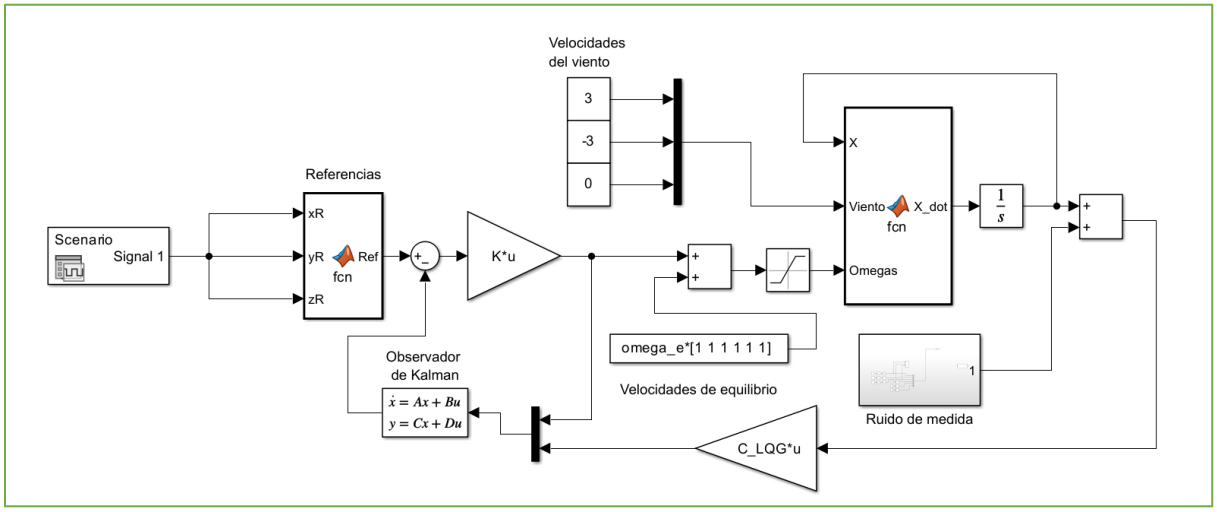
$x_5 = z$	3.43	0.734	3.94	24.9
$x_7 = \phi$	0.238	0.112	0.124	54.5
$x_9 = \theta$	0.348	0.163	0.182	54.5
$x_{11} = \psi$	-0.0526	-0.00491	-0.138	10.9

5.2.5 Controlador LQG

La estructura del controlador LQG desarrollado para el dron, se muestra en la Figura 12, este controlador está compuesto por un observador de Kalman en combinación con un controlador regulador cuadrático lineal LQR, por sus siglas en inglés. Un conjunto de variables de estados para los cuales el sistema es observable es: $x_1, x_3, x_5, x_7, x_9,\ x_{11}$, que corresponden a las coordenadas generalizadas, para estas se diseña el observador de Kalman, el cual basado en los estados medidos y los valores del vector de entradas logra estimar el valor de los estados restantes. Los valores estimados se restan a los de referencia y este error entra al controlador LQR, el cual determina el valor de las acciones de control que pasan por un limitador antes de entrar al sistema.

Figura 12

Diagrama de bloques del controlador LQG y conexión a la planta.



En la estrategia de control LQR se busca optimizar la acción de control con respecto al desempeño esperado, para esto se requieren seleccionar una matriz semidefinida positiva Q que penaliza el desempeño, y una matriz definida positiva R que penaliza la acción de control (Ogata, 2010). Las matrices Q y R fueron determinadas por prueba y error y se muestran en las Ecuaciones (69) y (70) respectivamente.

$$Q = diag([10^4, 1, 10^4, 1, 10^6, 1, 10^2, 1, 10^2, 1, 10^2, 1]) \tag{69}$$

$$R = 10^{-4} \cdot diag([1, 1, 1, 1, 1, 1]) \tag{70}$$

La ley de control viene dada por la ecuación (71) donde K es la matriz de ganancia de retroalimentación de estado. Inicialmente se resolvió la ecuación de Riccati, dada por la ecuación (72) asociada para P y luego se encontró K por medio de la ecuación (73) de forma numérica por medio de Matlab, esto se puede ver en la parte 5 del Apéndice A.

$$u = -K \cdot X \tag{71}$$

$$A^T P + PA - PBR^{-1}B^T P + Q = 0 \tag{72}$$

$$K = R^{-1}B^T P \tag{73}$$

La matriz K se da en la ecuación (74).

$$K^T = 1.0 \times 10^4 \times \begin{bmatrix} 0.50 & 0.50 & 0.00 & -0.50 & -0.50 & 0.00 \\ 0.54 & 0.54 & 0.00 & -0.54 & -0.54 & 0.00 \\ -0.30 & 0.30 & 0.57 & 0.30 & -0.30 & -0.57 \\ -0.32 & 0.32 & 0.61 & 0.32 & -0.32 & -0.61 \\ 4.08 & 4.08 & 4.08 & 4.08 & 4.08 & 4.08 \\ 2.46 & 2.46 & 2.46 & 2.46 & 2.46 & 2.46 \\ 1.69 & -1.69 & -3.25 & -1.69 & 1.69 & 3.25 \\ 0.53 & -0.53 & -1.03 & -0.53 & 0.53 & 1.03 \\ 2.84 & 2.84 & 0.00 & -2.84 & -2.84 & 0.00 \\ 0.90 & 0.90 & 0.00 & -0.90 & -0.90 & 0.00 \\ -0.04 & 0.04 & -0.04 & 0.04 & -0.04 & 0.04 \\ -0.69 & 0.69 & -0.72 & 0.69 & -0.69 & 0.72 \end{bmatrix} \quad (74)$$

Para el diseño del observador de Kalman se procede de manera similar primero se definen la matriz de covarianza de las mediciones R_{Kalman} y la matriz de covarianza del proceso Q_{Kalman}. En el caso del proceso las perturbaciones que se tienen son en la velocidad lineal ya que hay una presencia de perturbaciones con un valor de +-3 m/s, los demás valores se asumen con una precisión de varianza 0.01, por lo que la matriz Q_{Kalman} queda definida en (75).

$$Q_{kalman} = diag([0.1^2, 9, 0.1^2, 9, 0.1^2, 9, 0.1^2, 0.1^2, 0.1^2, 0.1^2, 0.1^2, 0.1^2]) \quad (75)$$

Las mediciones se realizan por medio de una IMU, que consta de un acelerómetro, un giroscopio y un magnetómetro. Para medir las coordenadas de posición se utiliza una doble integración de las medidas obtenidas del acelerómetro, y para obtener la orientación se trabaja con las medidas obtenidas del giroscopio. Teniendo en cuenta la información suministrada por el fabricante de la IMU BMI323 la varianza del acelerómetro v_{acc} es de $3.118 \cdot 10^{-6} \frac{m^2}{s^4}$, y la varianza del giroscopio v_{gir} es de $1.4926 \cdot 10^{-8} \frac{rad^2}{s^2}$, se multiplica por un factor de 100 para tomar en cuenta los efectos de la integración, por lo que la matriz de R_{Kalman} es dada por (76).

$$R_{kalman} = \text{diag}(3.12 \times 10^{-3}, 3.12 \times 10^{-3}, 3.12 \times 10^{-3}, 1.49 \times 10^{-5}, 1.49 \times 10^{-5}, 1.49 \times 10^{-5}, 1.49 \times 10^{-5}) \quad (76)$$

Para encontrar el sistema lineal del observador, primero se obtuvo la ganancia del filtro de Kalman K_{kalman} similarmente a como se hizo para obtener la matriz de ganancia, nuevamente se utilizaron las ecuaciones (72) y (73) pero en este caso se sustituyeron A, B, Q, R por A^T, C^T, Q_{Kalman} y R_{Kalman} respectivamente. Esta ganancia se usa para obtener el sistema del observador del Kalman dado por las ecuaciones (77), (78), (79), (80) donde I es la matriz identidad. El archivo en Simulink del diagrama de bloques del controlador LQG se da en el **Apéndice C**.

$$A_{kalman} = A - K_{Kalman}C \quad \textbf{(77)}$$

$$B_{kalman} = [B \quad K_{Kalman}] \quad \textbf{(78)}$$

$$C_{Kalman} = I(12) \quad \textbf{(79)}$$

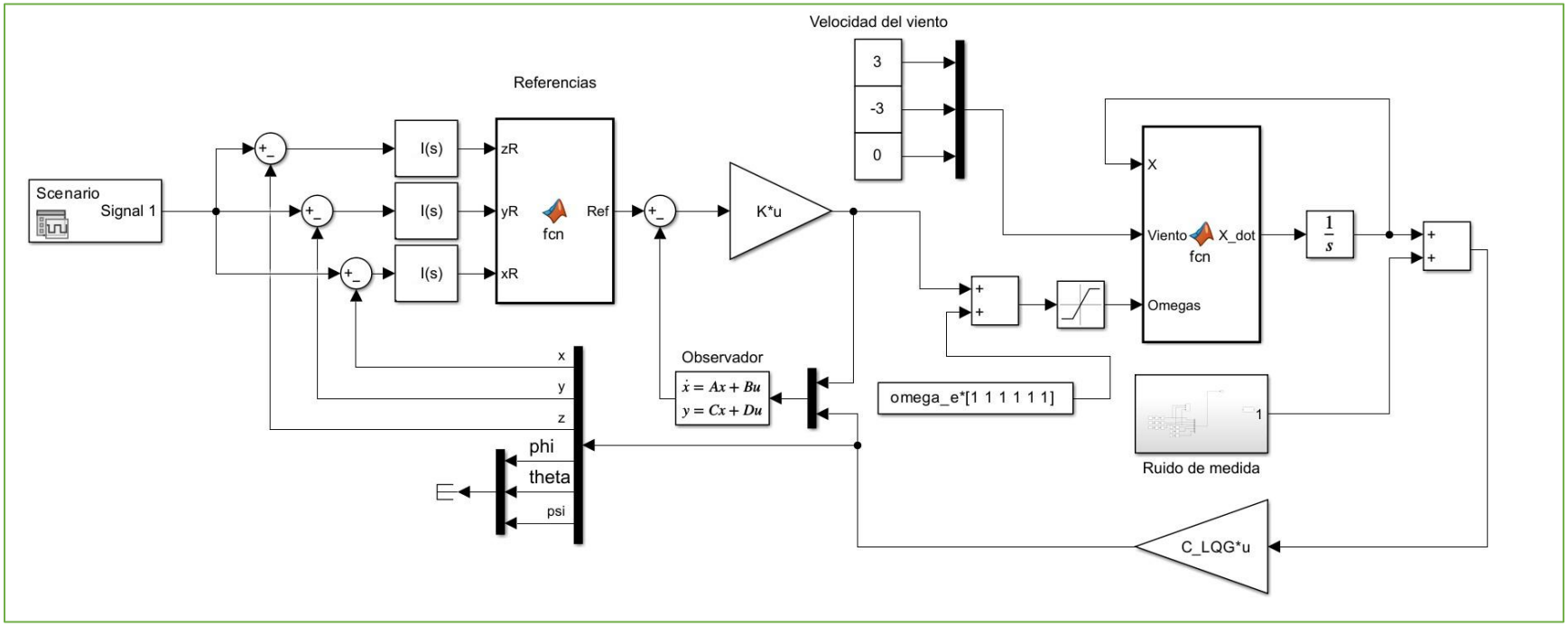
$$D_{kalman} = 0 * [B \quad K_{Kalman}] \quad \textbf{(80)}$$

5.2.6 Controlador LQG-Integral

En la tercera estrategia de control se toma como punto de partida el controlador LQG diseñado anteriormente y se agrega un controlador integral, con el objetivo de eliminar el error en estado estable en las coordenadas posición. La estructura del controlador y su conexión a la planta son mostrados en la Figura 13. El archivo del diagrama de bloques del controlador en simulink se incluye en el **Apéndice D**.

Figura 13

Diagrama de bloques del controlador LQG-integral y conexión a la planta



La ecuación de transferencia del controlador integral se da en la ecuación (81), los valores de la constante para cada uno de los controladores están dadas en la Tabla 4.

$$C_I = I_C \frac{1}{s} \tag{81}$$

Tabla 4

Valores de las constantes integrales para el controlador LQG-integral

Variable controlada	I_C
$x_1 = x$	0.472
$x_3 = y$	0.471
$x_5 = z$	0.876

5.3 Resultados y análisis comparativo de los controladores de seguimiento de trayectoria

Uno de los resultados obtenidos hasta ahora fue el procedimiento y código para la obtención eficiente del modelo no lineal del dron hexatorrotor, incorporando efectos giroscópicos y la fuerza de arrastre parasita, representado en las EMS. Este modelo se obtuvo de manera eficaz mediante el formalismo de Lagrange y la caja de operaciones simbólica de Matlab, el código puede ser reutilizado para otros drones hexarrotores modificando el valor de sus parámetros.

Los controladores fueron evaluados numéricamente utilizando el modelo no lineal implementado en Simulink. Para simular las mediciones de los estados se tomó como referencia la IMU BMI323, se incluyeron errores extraídos de la información del fabricante para los giroscopios y acelerómetros como ruido blanco de banda limitada, con una frecuencia de 100 Hz.

Dado que la IMU mide aceleraciones y velocidades angulares, se realizaron las integraciones correspondientes para obtener las coordenadas de posición y orientación del sistema,

esto junto con el error de medida agregado permite evaluar los controladores en condiciones más realistas.

Durante la realización del experimento, se llevó a cabo una prueba en la cual se buscaba trasladar al dron en un lapso de 120 segundos a una serie de puntos. Las coordenadas de dichos puntos se mantuvieron constantes en las tres dimensiones. En esta prueba, tanto ruidos de proceso como de medida fueron incorporados. Como ruido de proceso, se simularon velocidades de viento de 3 m/s en la dirección del eje x y de -3 m/s en la dirección del eje y, ambas referidas al SRI. El seguimiento de trayectoria fue implementado con el propósito de evaluar el rendimiento de los tres controladores en este escenario. Los comportamientos de las coordenadas de posición y angulares para las estrategias PID, LQG y LQG-Integral durante el experimento descrito se ilustran en las Figura 14, Figura 15, Figura 16, respectivamente.

Figura 14

Respuesta de la posición y orientación del sistema con control PID

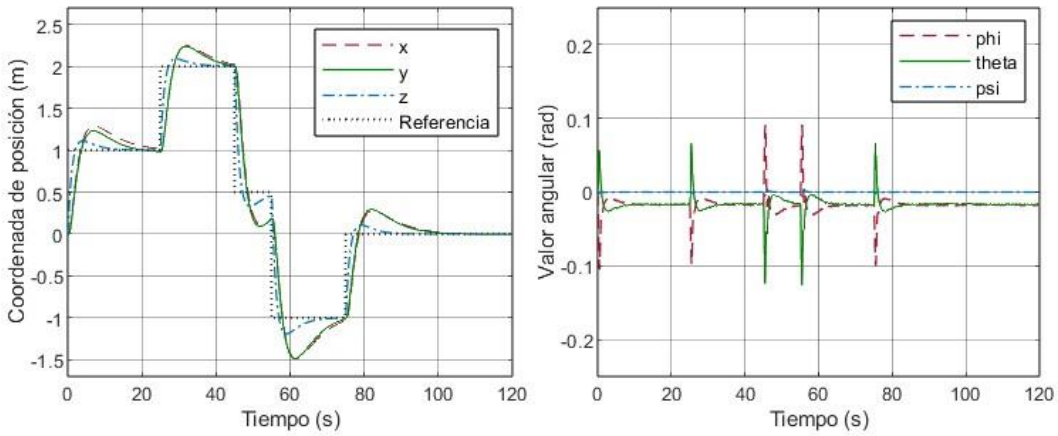


Figura 15

Respuesta de la posición y orientación del sistema con control LQG

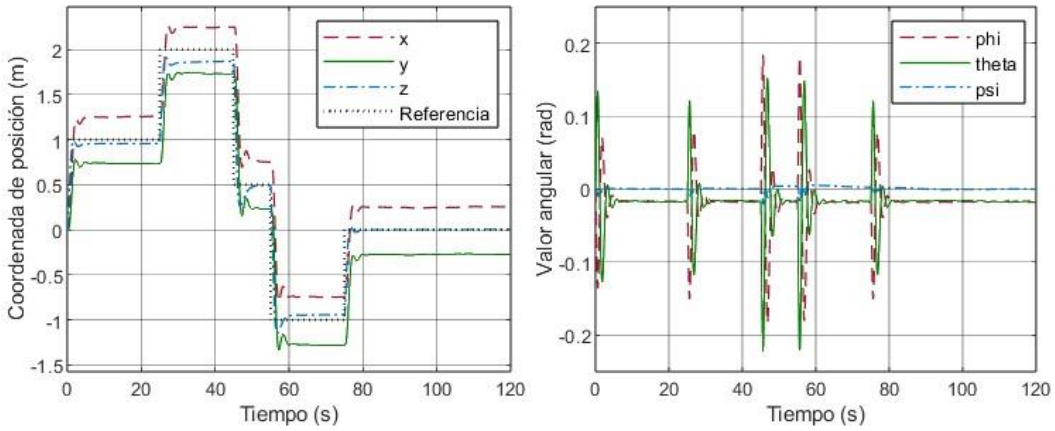
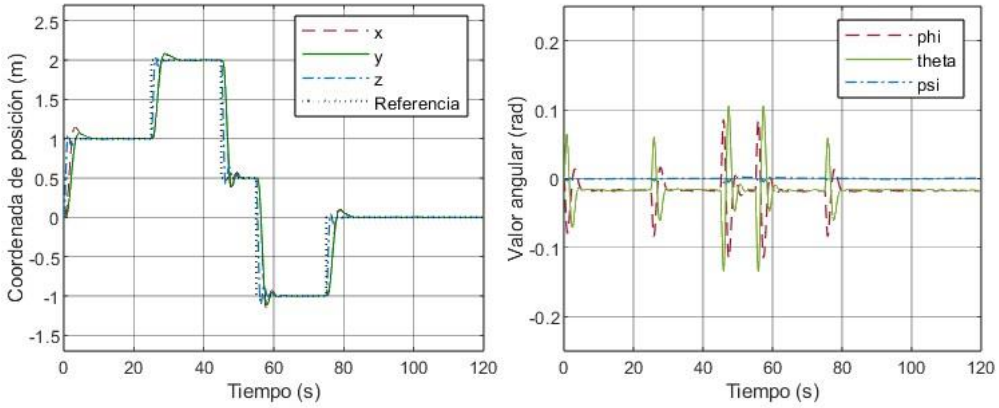


Figura 16

Respuesta de la posición y orientación del sistema con control LQG-Integral



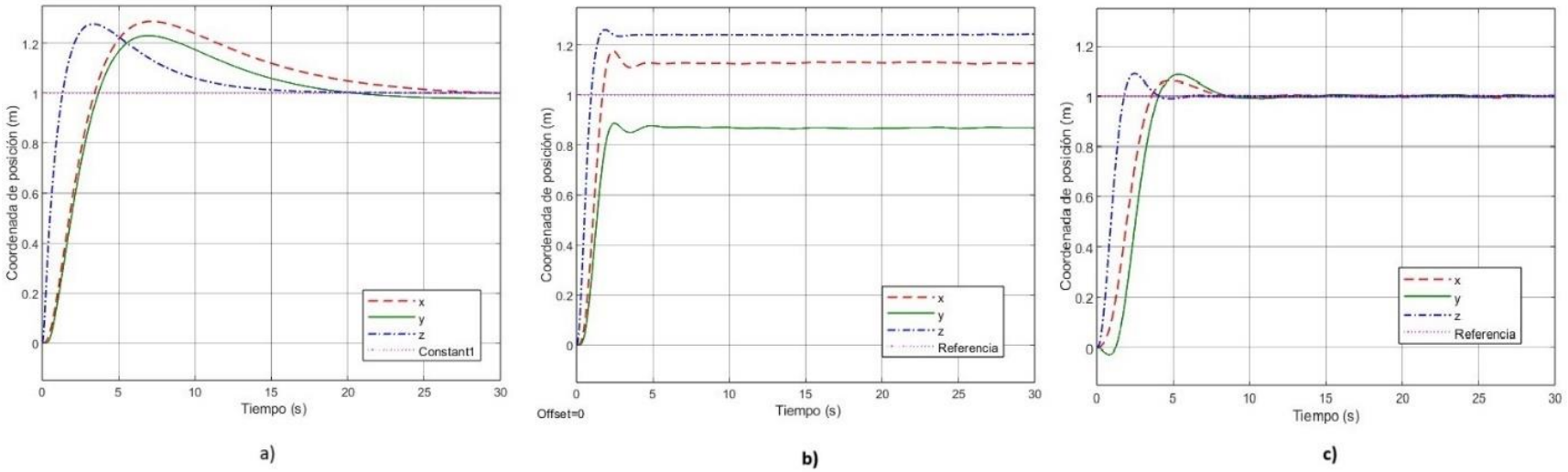
En la Figura 15 se evidencia un error en estado estable del controlador LQG, que para las coordenadas *x* e *y* no dependió del valor de la referencia, mientras que para la altura *z* sí lo hizo, esto es debido a las no linealidades del sistema.

En todas las estrategias de control, el dron mantuvo una ligera inclinación en los ángulos θ y ϕ para conservar la posición ante las perturbaciones causadas por vientos externos. La presencia de ruidos de medida no afectó considerablemente la respuesta del sistema gracias al filtrado eficiente proporcionado por el filtro de Kalman de la estrategia LQG y los componentes integral y derivativo en el control PID.

En la **Figura 17** se muestran tres gráficas, cada una representando la respuesta del estado para las coordenadas x, y y z respectivamente. Cada gráfica ilustra cómo varían estas coordenadas con el tiempo para los distintos controladores comparados, con el objetivo de alcanzar el punto (1,1,1). El controlador LQG-Integral, aunque no es el más rápido, destaca por su precisión al eliminar los errores en estado estable, superando al LQG en términos de precisión y siendo más rápido que el PID, que también elimina los errores en estado estable, pero con un desempeño menos eficiente.

Figura 17

Seguimiento de trayectoria hasta la coordenada (1,1,1)



Nota: Se presenta el comportamiento de las coordenadas en el tiempo usando: a) Controlador PID, b) Controlador LQG, c) Controlador LQG-Integral

La **Tabla 5** presenta los tiempos de establecimiento al 5% del valor en estado estable, el porcentaje de sobreelongación y el error en estado estable para cada estrategia de control durante el primer intervalo de 30 segundos, en los que el dron se desplaza al punto (1,1,1).

Tabla 5

Parámetros de respuesta en el tiempo del sistema con los distintos controladores

Controlador	Coordenada	Sobreelongación %	Tiempo de establecimiento (s)	Error en estado estable %
PID	x	28.7	19.8	0
	y	23.0	15.6	0
	z	27.7	10.7	0
LQG	x	3.98	2.2	13.2
	y	1.67	2.1	-13.1
	z	1.45	4.9	24
LQG-Integral	x	6.2	5.8	0
	y	8.8	6.77	0
	z	9.1	3.21	0

Aunque el controlador LQG exhibió menor sobreelongación y tiempo de asentamiento en comparación con el control PID, se detectó un error en estado estable tanto en la coordenada x estando un 13.2% por encima de la medida como en la coordenada y estando un 13.1% por debajo de la medida, lo cual es inaceptable para aplicaciones donde el control de posición del dron deba ser muy preciso, esto es debido a que el controlador LQG junto con la planta no presentan

integradores que puedan eliminar el error en estado estable, aunque logra optimizar la relación entre desempeño y energía de los actuadores.

No obstante, al incorporar el componente integral y transformar el controlador LQG en LQG-Integral, esta estrategia sobresalió al mostrar tiempos y sobreelongación inferiores al PID, sin presentar errores en estado estable, pero el tiempo de establecimiento mayor comparado con el controlador LQG (que se estabiliza, pero con error en estado estable alto). Esto se debe a que se combinan las ventajas del controlador óptimo con la supresión del error en estado estable de los integradores del control integral, se logró con esto realizar la tarea de seguimiento de trayectoria con rapidez y precisión, optimizando el uso de energía por parte de los actuadores.

5.4 Conclusiones de la sección

Se ha explorado el modelado eficiente del sistema no lineal de un dron hexacóptero, incorporando efectos giroscópicos y la fuerza de arrastre parasita mediante el método de Lagrange apoyado con Matlab para realizar las operaciones simbólicas de manera eficiente. Además, se ha evaluado y comparado el desempeño de las estrategias de control PID, LQG y LQG-Integral en términos de precisión y tiempos de establecimiento.

Durante las pruebas de seguimiento de trayectoria, se observó que el controlador LQG ofreció tiempos de asentamiento y sobreelongación menores en comparación con el PID, pero con errores en estado estable significativos en las coordenadas x e y. Estos errores son debido a las no linealidades del sistema y la falta de integradores para eliminar errores en estado estable, por lo que el uso de controladores LQG no se recomienda para aplicaciones de drones que requieren un control de posición altamente preciso.

El controlador LQG-Integral emergió como la estrategia más efectiva, mostrando tiempos de asentamiento y sobreelongación reducidos, sin errores en estado estable en ninguna de las coordenadas evaluadas. El LQG ofrece una solución óptima al minimizar una función de costo cuadrática, y al combinarlo con el componente integral, se eliminan los errores en estado estable, lo cual proporciona una ventaja significativa en la precisión del control del sistema.

Los tres controladores demostraron una eficaz capacidad para filtrar el ruido de medida, destacándose el filtro de Kalman en la estrategia LQG y los componentes integral y derivativo en el PID. Esta robustez asegura una respuesta confiable del sistema incluso en presencia de perturbaciones externas de baja magnitud.

Este trabajo de investigación contribuye al campo del control de drones al ofrecer una evaluación comparativa detallada entre estrategias de control convencionales y óptimas. La implementación del controlador LQG-Integral demuestra un seguimiento de trayectoria rápido y preciso en la simulación. Además, al optimizar el uso de energía por los actuadores en relación con el desempeño de los estados, mejora la eficiencia operativa del sistema.

6. Evasión de obstáculos y planificación de trayectoria local basadas en aprendizaje por refuerzo para drones hexarrotores en entornos simulados

Esta sección muestra el proceso llevado a cabo para entrenar un agente para las tareas de evasión de obstáculos y planificación de trayectoria local, basado en aprendizaje por refuerzo. Se detallan las características del ambiente, las variables que conforman el estado y la función de recompensa implementada. Además, se proporciona una explicación a profundidad de la estructura de la red neuronal del agente y el algoritmo de aprendizaje por refuerzo utilizado para el entrenamiento.

En el contexto del aprendizaje por refuerzo, el ambiente abarca todo lo que está externo al agente. Así, el ambiente incluye tanto el cultivo de palma como el dron simulado, y el agente será un controlador de alto nivel que envía coordenadas a un controlador de bajo nivel. El cultivo de palma es simplificado y tiene obstáculos que representa los estipes de las palmas de aceite colocados a las distancias recomendadas en plantaciones reales. En las siguientes secciones se explica el proceso para simular el dron en Gazebo, y la información que se extrae de la interacción con el cultivo.

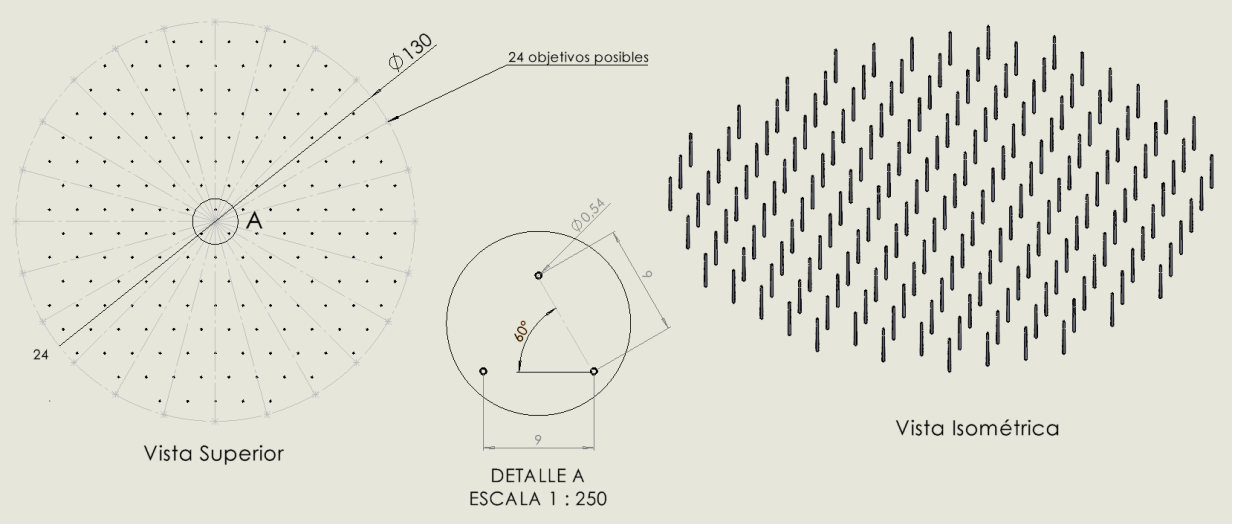
6.1 Cultivo de palma simplificado simulado

El cultivo de palma simplificado, mostrado en la Figura 18, fue desarrollado en Solidworks, respetando las dimensiones de las estipes de las palmas y su disposición según las recomendaciones de los manuales técnicos (Grupo Jaremar, 2016). Se conservaron solo las palmas dentro de un radio de 60 m para facilitar la localización de los puntos objetivo. Sin embargo, el dron aún debe recorrer una distancia de 65 m repleta de obstáculos, lo que resulta adecuado para su entrenamiento. En la Figura 18 se presenta una vista superior del cultivo, detallando la distancia

entre las palmas, una vista isométrica y la ubicación de las posibles posiciones objetivo que el dron debe alcanzar

Figura 18

Cultivo de palma de aceite simplificado



Nota: Distancia es metros

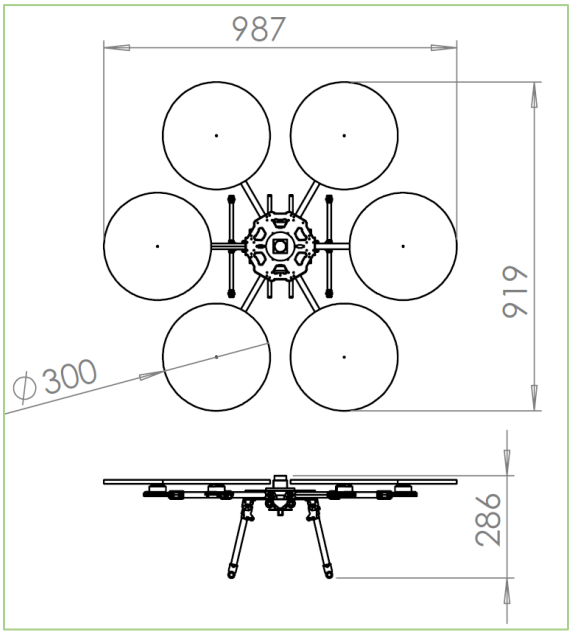
6.2 Simulación del dron en Gazebo

El dron hexacóptero simulado está basado en uno disponible en el laboratorio, que tiene el marco en el marco Tarot FY-680, algunas de sus dimensiones en milímetros se pueden ver en la Figura 19. El dron simulado incluye dos sensores disponibles en Gazebo para determinar su posición y velocidad en relación con un origen seleccionado: una unidad de medición inercial (IMU, por sus siglas en inglés, Inertial Measurement Unit), y un sensor de detección y medición mediante luz, LIDAR (por sus siglas en inglés, Light Detection and Ranging) bidimensional capaz de medir distancias en un rango de 1 a 12 metros respecto al centro del dron, con una separación de medidas de 1°. El LIDAR fue seleccionado porque puede detectar objetos en movimiento en

tiempo real y, al emitir su propia fuente de luz láser, lo cual reduce la interferencia de la luz externa (Liang et al., 2023).

Figura 19

Plano del dron hexarrotor con dimensiones generales



El dron fue modelado inicialmente en SolidWorks, de donde se extrajeron propiedades como la ubicación del centro de masa, la masa y la matriz de inercia. Utilizando un complemento de SolidWorks, se generó el modelo en el formato URDF (Unified Robot Description Format), exportando sólidos con un número reducido de polígonos para mantener la similitud con el modelo real sin incrementar el costo computacional.

El formato URDF, basado en XML, se utiliza para describir la estructura de robots en términos de eslabones (partes físicas) y juntas (conexiones). Es comúnmente empleado en simulación y control de robots en entornos como ROS2 (Macenski et al., 2022), donde se modelan propiedades físicas, geométricas, la masa, la inercia, y los sensores. En este caso, el primer eslabón

es el marco del hexarrotor, que está conectado a ocho eslabones: las seis hélices, la IMU y el LiDAR.

El marco y cada hélice está unida por una junta de tipo revolución, el marco se conecta a la IMU mediante una junta prismática, y al LiDAR a través de una junta de revolución. Para las ocho juntas, se establecen límites que simulan una junta fija, ya que no es posible utilizar las juntas fijas porque están unirían todos los eslabones como uno solo, lo cual impediría la implementación de los sensores y actuadores, que deben estar en eslabones independientes.

La representación del dron en URDF se organiza a través de varios archivos en el paquete de ROS 2 *hexarotor* dado en el **Apéndice E**. Este paquete, dentro de la carpeta "description", incluye los archivos: "hexarotor.URDF.xacro", que define los eslabones y las juntas del dron; "actuadores.xacro", que configura los actuadores; y "sensores.xacro", que establece los sensores.

6.2.1 Estructura de los Eslabones del dron

Como ejemplo en la Figura 20 se muestra un fragmento del código, que da la estructura del “chasis” (marco del hexarrotor) definido en el bloque <link name="chasis/>, en este se incluye información sobre su masa, geometría, e inercia. A continuación, se desglosan las partes más importantes de este bloque.

Figura 20

Código desarrollado para la definición de un eslabón

```
<link name="chasis">
  <inertial>
    <origin xyz="0.0058429 -0.0041212 0.2205" rpy="0 0 0" />
    <mass value="0.72386" />
    <inertia ixx="0.00067343" ixy="0" ixz="0" iyy="0.00082239" iyz="0" izz="0.0013425" />
  </inertial>
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="package://hexarotor/meshes/chasis.STL" />
    </geometry>
    <material name="">
      <color rgba="1 1 1 1" />
    </material>
  </visual>
  <collision>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <mesh filename="package://hexarotor/meshes/chasis.STL" />
    </geometry>
  </collision>
</link>
```

- El bloque <inertial> define las propiedades físicas del chasis, como la posición del centro de masa del sistema de referencia (xyz="0.00584 -0.00412 0.2205"), su masa total de 0.724 kg, y su matriz de inercia.
- Dentro del bloque <visual>, se especifica la forma que se verá en el simulador usando un modelo 3D del chasis que se encuentra en el archivo de tipo malla “chasis.STL”. El origen y la orientación son las mismas del archivo de malla y la apariencia está definida por un color blanco (rgba="1 1 1 1").
- El bloque <collision> también usa la misma malla que el visual para describir la geometría que se usará para detectar colisiones, entre eslabones, el terreno u obstáculos.

6.2.2 Estructura de las juntas

Las juntas conectan las distintas partes del dron, permitiendo movimiento (revoluciones o traslaciones) entre ellas. En el código, cada junta tiene un padre (el chasis) y un hijo, en este caso, los propulsores (hélices). A continuación, se muestra un ejemplo de la junta entre el *chasis* y el *propulsor1*, el fragmento del código que define junta de ejemplo se muestra en la Figura 21.

Figura 21

Código desarrollado para la definición de una junta

```
<joint name="motor1" type="revolute">
    <origin xyz="-0.3094 0.17866 0.26333" rpy="0 0 0" />
    <parent link="chasis" />
    <child link="propulsor1" />
    <axis xyz="0 0 1" />
    <limit lower="-0.0001" upper="0.0001" effort="100" velocity="1.0"/>
</joint>
```

Una explicación breve de cada uno de los componentes del bloque se da a continuación:

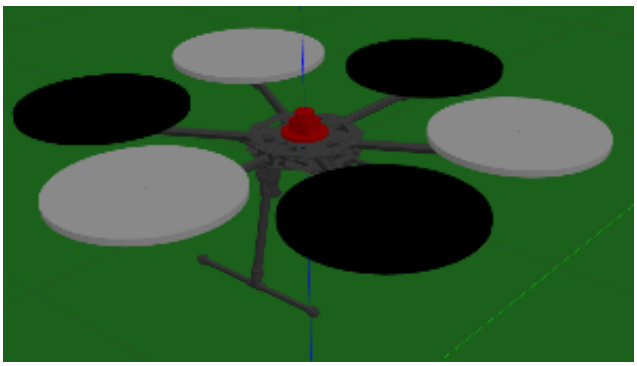
- El nombre de la junta es "motor1" y el tipo es "revolute", lo que indica que la junta permitirá un movimiento de rotación en torno a un eje, pero dentro de uno límites.
- La posición relativa de la junta se especifica por las coordenadas (xyz="-0.3094 0.1787 0.2633") respecto al chasis.
- La junta conecta dos partes, el *link* padre, que es el "chasis", y el *link* hijo, que es el "propulsor1".
- El eje alrededor del cual se produce el movimiento es (xyz="0 0 1"), indicando que la rotación ocurre alrededor del eje Z.

- Para simular que es una junta fija, se define un límite muy pequeño para la rotación de la hélice (lower="-0.0001" upper="0.0001") y también se especifica el esfuerzo y la velocidad máximos permitidos para la junta.

Como resultado final luego de incluir definir el URDF con las juntas y los eslabones, al hacer aparecer el dron en el simulador de Gazebo se tiene el resultado mostrado en la Figura 22.

Figura 22

Dron simulado desarrollado para ROS2-Gazebo



6.2.3 Estructura de los actuadores

Los actuadores incluidos en Gazebo en este caso pueden enviar un mensaje de tipo *Wrench* que representa una fuerza y un par de torsión (torque) aplicado a un objeto en un sistema tridimensional, en este caso estos enviaran una fuerza y un torque de acuerdo con la velocidad de rotación de las hélices al cuadrado. En la Figura 23 se muestra un fragmento del código para la creación del actuador en el propulsor 1.

Figura 23

Fragmento desarrollado para la creación de un actuador de tipo wrench en Gazebo

```
<gazebo>
    <plugin name="gazebo_ros_force1" filename="libgazebo_ros_force.so">
        <alwaysOn>true</alwaysOn>
        <ros>
            <namespace>/test1</namespace>
            <remapping>gazebo_ros_force1:=force_test1</remapping>
        </ros>
        <update>"${frecuencia}"</update>
        <updateRate>"${frecuencia}"</updateRate>
        <link_name>propulsor1</link_name>
        <force_frame>link</force_frame>
    </plugin>
</gazebo>
```

Las partes de la estructura del código son explicadas a continuación:

- name: Identificador único para el plugin.
- filename: Ruta al archivo del plugin (libgazebo_ros_force.so en este caso).
- alwaysOn: Configurado en true para que el plugin esté siempre activo.
- namespace: Espacio de nombres en ROS para el plugin.
- remapping: Mapea el nombre del topic en ROS.
- update y updateRate: Tasa de actualización, configurada con el valor de la propiedad frecuencia basado en la frecuencia con la el controlador envía la fuerza y el torque.
- link_name: *link* del robot donde se aplican la fuerza y el torque.
- force_frame: Define el marco de referencia para las fuerzas, en este caso es el de cuerpo.

6.2.4 Estructura de los sensores

En la Figura 24 se muestra un fragmento del código del sensor utilizado para la detección de obstáculos mediante un LIDAR 2D. Este sensor emite 360 rayos distribuidos equidistantemente en una circunferencia, con una capacidad de detección de 0.3 a 12 metros. A continuación, se explican los elementos de su estructura.

Figura 24

Fragmento del código desarrollado para implementar un sensor de láser de dos dimensiones

```
<gazebo reference="laser_frame_2D">
  <material>Gazebo/Red</material>
  <sensor name="laser2D" type="ray">
    <pose> 0 0 0.28 0 0 0 </pose>
    <visualize>false</visualize>
    <update_rate>100</update_rate>
    <ray>
      <scan>
        <horizontal>
          <samples>360</samples> <min_angle>-3.14</min_angle> <max_angle>3.14</max_angle>
        </horizontal>
      </scan>
      <range> <min>0.3</min> <max>12</max>
      </range>
    </ray>
    <plugin name="laser_controller" filename="libgazebo_ros_ray_sensor.so">
      <ros>
        <remap from="~/out" to="scan"/>
      </ros>
      <output_type>sensor_msgs/LaserScan</output_type>
      <frame_name>laser_frame_2D</frame_name>
    </plugin>
  </sensor>
</gazebo>
```

- <gazebo reference="laser_frame_2D">: Establece la referencia de Gazebo para el sensor, en este caso, el marco de referencia del sensor es laser_frame_2D.
- <material>Gazebo/Red</material>: Define el material visual del sensor en el simulador Gazebo, siendo el color rojo (Gazebo/Red).

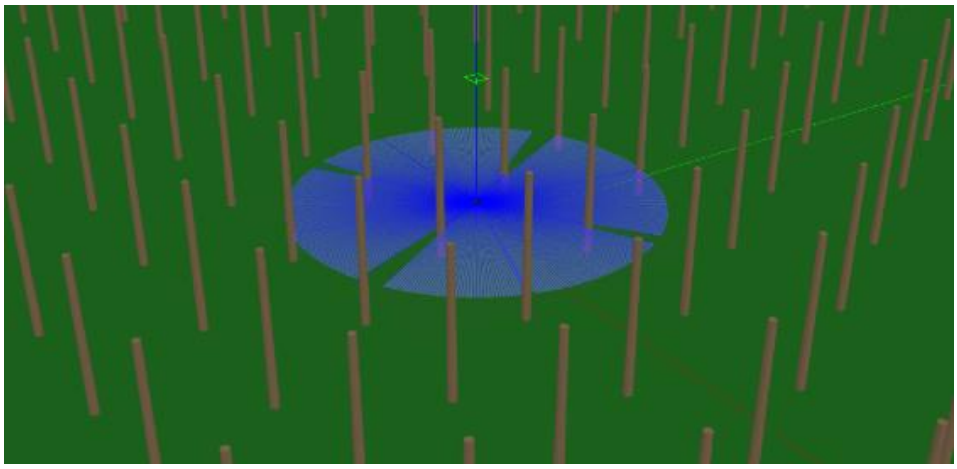
- <sensor name="laser2D" type="ray">: Inicia la definición del sensor. El nombre es laser2D y el tipo es *ray*, lo que indica que es un sensor de rayos (o láser).
- <pose> 0 0 0.28 0 0 0 </pose>: Define la posición y orientación del sensor en relación con el marco de referencia. En este caso, está a 0.28 metros en el eje Z, sin rotaciones.
- <visualize>false</visualize>: Indica si se visualizará o no los rayos que emite el sensor en Gazebo. Aquí se establece como false (no visible).
- <update_rate>100</update_rate>: Especifica la tasa de actualización del sensor, en este caso 100 Hz.
- <ray>: Comienza la configuración específica para el sensor de rayos láser.
- <scan>: Define los parámetros de escaneo.
- <horizontal>: Especifica la configuración horizontal del escaneo.
- <samples>360</samples>: Define el número de muestras o rayos emitidos, 360 en este caso (cubre un rango completo de 360°).
- <min_angle>-3.14</min_angle> y <max_angle>3.14</max_angle>: Establecen los ángulos mínimo y máximo en radianes para el escaneo, cubriendo un campo de visión de 360°.
- <range>: Especifica el rango en la dirección radial de detección del sensor.
- <min>0.3</min>: Rango mínimo de detección (0.3 metros).
- <max>12</max>: Rango máximo de detección (12 metros).
- <plugin name="laser_controller" filename="libgazebo_ros_ray_sensor.so">: Define un plugin para el sensor, que se utilizará para interactuar con ROS. El plugin se llama *laser_controller* y usa la biblioteca *libgazebo_ros_ray_sensor.so*.
- <ros>: Configuración relacionada con ROS.

- <remap from="~/out" to="scan"/>: Redirecciona la salida del sensor láser al tópico scan.
- <output_type>sensor_msgs/LaserScan</output_type>: Especifica el tipo de mensaje de salida en ROS, que es sensor_msgs/LaserScan.
- <frame_name>laser_frame_2D</frame_name>: Define el marco de referencia del sensor para ROS, en este caso, *laser_frame_2D*.

En la **Figura 25** se muestra el dron dentro del cultivo, y los rayos del LIDAR se hacen visibles para ilustrar su alcance de 12 metros. El LIDAR emiten rayos a cada grado en toda la circunferencia, lo que resulta en 360 datos. Sin embargo, solo se prestó atención para el entrenamiento al Rango de Detección (RD), que se define como los datos tomados a $\pm 90°$ con respecto al ángulo óptimo alfa α, este último se define como el ángulo entre la línea recta que une la posición actual con la posición objetivo y el eje x.

Figura 25

Dron con rayos de LiDAR visibles dentro de cultivo de palma simplificado simulado



6.3 Clase ambiente creada para el agente de aprendizaje por refuerzo

La meta del agente RL es mover al dron desde su posición actual hacia una señal emitida en una ubicación dentro del cultivo. Para lograrlo, el agente envía al controlador de vuelo y

seguimiento de trayectoria LQG-Integral (explicado en secciones anteriores), una nueva coordenada, ubicada a un metro de la posición actual, pudiendo tomar tres valores: el ángulo óptimo α, 45° grados a la derecha de α, o 45° a la izquierda de α.

Las coordenadas se actualizan a una frecuencia de 4 Hz, y dado que el controlador tarda aproximadamente 6 segundos en estabilizar las coordenadas de posición, la siguiente coordenada se envía antes de que el dron alcance la posición previamente asignada.

En esta sección se explican componentes de la clase que representa el ambiente (CustomEnv) del código en Python usado para el entrenamiento del agente DDDQN llamado *Apendice_F_Codigo_entrenamiento_agente_DDDQN.py* el cual está adjunto en el **Apéndice F**. Se describirán los estados que representan la situación actual del dron y su entorno, las acciones que puede tomar el agente, el modo de seguridad para evitar colisiones, la función de recompensa que sirve para entrenar la política de acción del agente y los estados terminales que determinan cuándo el episodio concluye. Estos elementos son clave para que el agente ser entrenado para guiar al dron hacia un objetivo de manera eficiente y segura.

6.3.1 Vector de estado s

El estado del ambiente en un momento dado se representa mediante el vector s, que consta de un total de 186 valores, los cuales fueron seleccionados por el autor luego de prueba y error, cada uno de los estos se explica en la siguiente lista utilizando la notación de vectores del lenguaje de programación Python. Los componentes de este vector están normalizados dividiéndolos por los valores indicados.

- s[0]: Velocidad lineal absoluta del dron, normalizada dividiéndola por 2.6 m/s.
- s[1]: Velocidad angular absoluta del dron, normalizada dividiéndola por 1 rad/s.

- s[2]: Distancia medida por el LIDAR al obstáculo más cercano dentro del rango de detección, normalizada dividiéndola por 12 m.
- s[3]: Posición del obstáculo más cercano dentro del rango de detección del LIDAR, normalizada dividiéndola por 180.
- s[4]: Distancia al objetivo, normalizada dividiéndola por 70 m.
- s[-181\:]: Los siguientes 181 valores son las distancias medidas por el LIDAR dentro del rango de detección, que envía 181 rayos con una separación entre sí de 1°. Estas distancias se normalizan dividiéndolas por 12 m.

6.3.2 Conjunto de acciones posibles a

El agente envía la nueva coordenada a la que el dron debe moverse. Esta nueva coordenada está a un metro de la posición actual, y el ángulo de movimiento se calcula añadiendo una desviación con respecto al ángulo óptimo. En el contexto del aprendizaje por refuerzo, esto corresponde al conjunto de posibles acciones *A* dado en (82), que representan las acciones de girar 45° a la derecha, continuar en línea recta, y girar 45° a la izquierda.

$$\mathrm{A} = \left\{-\frac{\pi}{4}, 0, \frac{\pi}{4}\right\} \tag{82}$$

Estas acciones se seleccionaron tras probar varios valores y cantidades. Se optó por tres acciones debido a que el algoritmo DDDQN que se va a implementar utiliza un conjunto discreto de acciones, el bajo número de posibles acciones se seleccionó para facilitar el entrenamiento del agente. La magnitud de las acciones fue elegida para asegurar que el giro del dron sea suave y no demasiado brusco con respecto a la línea recta.

6.3.3 Modo de seguridad

Para lograr un vuelo seguro del dron, se incluyó un modo de seguridad (MS) para evitar colisiones; este modo se activa cuando el centro del dron está a 1.44 metros de cualquier obstáculo. En este modo, el dron **no** utiliza la acción con el mayor valor-Q sugerida por la política del agente, sino que gira 45° en la dirección que lo aleja del obstáculo. Las tuplas de experiencia $(s, a, r, s', done)$ obtenidas mientras el dron se encuentra en el MS no se consideran para el entrenamiento del agente, excepto por el estado y la acción que activan este modo.

6.3.4 Recompensas r(s ,a, s')

La recompensa devuelta por el ambiente es una función tanto del estado s, la acción a aplicada por el agente, como del nuevo estado s' después de ejecutar la acción. La recompensa tiene diferentes componentes, los cuales son explicados en los siguientes párrafos. Dichos componentes fueron seleccionados luego de realizar un largo procedimiento de prueba y error por el autor.

El valor de R_1 se da en (83) y se usa para penalizar el estar muy cerca de cualquier obstáculo, cuanto más cerca esté el dron, menor será el valor de R_1. Además, este valor se vuelve cero para cuando $s'[2]$ es mayor que 0.36. Por último, si s' es un estado en modo de seguridad, R_1 se establece en -5 para penalizar alcanzar este estado.

$$\mathrm{R}_1 = -500 \cdot (\mathrm{s}'[2] - 0.36)^4 \tag{83}$$

El componente R_2, dado en (84), tiene como objetivo recompensar el estar lejos de los obstáculos dentro del rango de detección.

$$\mathrm{R}_2 = 0.25 \cdot \mathrm{s}'[2] \tag{84}$$

Para penalizar el estar lejos del objetivo, se utilizó la componente de recompensa R_3 mostrada en (85)

$$\mathrm{R}_3 = -0.1 \cdot \mathrm{s}'[4] \tag{85}$$

La componente de recompensa R_4 se usa para recompensar la acción de moverse en la dirección de α cuando $\mathrm{s}'[2] > 0.25$, si se cumplen dichas condiciones R_4 tiene un valor de 0.5, de lo contrario su valor es cero. Además, se otorga una recompensa de $R_5 = 10$ cuando se alcanza el objetivo. La recompensa total se calcula como la suma de las recompensas explicadas anteriormente utilizando (86).

$$r(s, a, s') = \mathrm{R}_1 + \mathrm{R}_2 + \mathrm{R}_3 + \mathrm{R}_4 + \mathrm{R}_5 \tag{86}$$

6.3.5 Estado terminal

Un estado terminal en el contexto de aprendizaje por refuerzo es un estado en el que el episodio concluye, es decir, no se realizan más pasos, como consecuencia el valor de este estado $V(s)$ no depende de recompensas futuras ya que el episodio ha terminado. En el problema que se está abordando de la planificación de trayectoria local, se alcanza un estado terminal cuando el dron llega exitosamente a la posición objetivo o se cae. Una variable booleana denominada $done$ señala si el estado actual es terminal. Cuando $done$ se establece en $True$, indica que el episodio ha concluido y se inicia un nuevo episodio.

6.3.6 Reinicio de la Simulación.

La clase ambiente se encarga de reiniciar los pasos cada vez que se alcanza un estado terminal. Al inicio de cada episodio el dron reaparece en el punto de origen y se desplaza hasta $x = 0$, $y = 0$, y $z = 1.5$ metros. El código de entrenamiento espera hasta que el dron alcance esta posición. Luego, para que el dron aprenda distintas rutas, se selecciona aleatoriamente un punto objetivo en el mismo plano horizontal. Este punto se elige entre 24 puntos distribuidos en un círculo de 65 metros de radio con centro en el origen, y una separación angular de 15°.

6.3.7 Ejecución de un paso

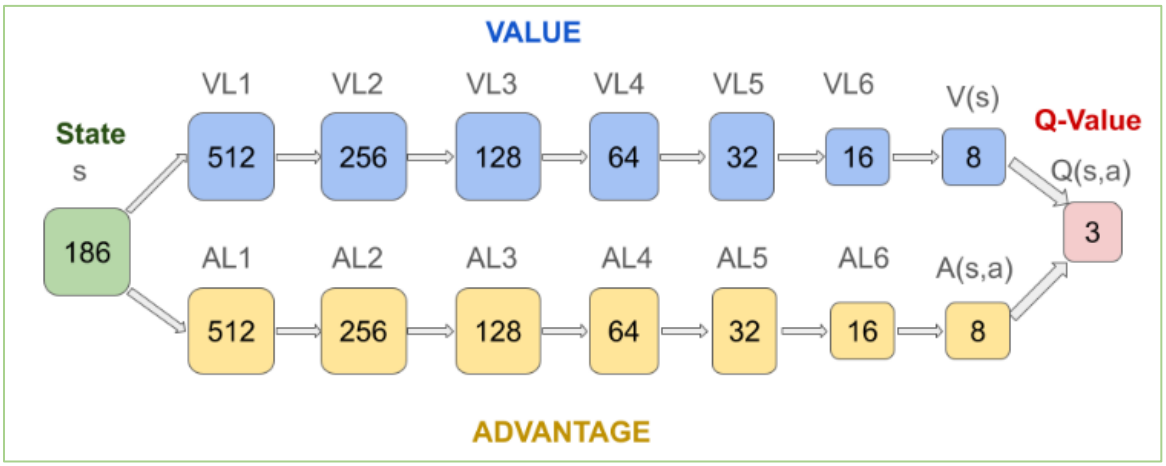
La clase ambiente es responsable de llevar a cabo los pasos del episodio siguiendo esta secuencia: primero, se detecta el estado actual s utilizando los sensores del dron. Luego, el agente selecciona la mejor acción según su política y una exploración ϵ-greedy, se aplica la acción a y se espera 0.25 segundos antes de enviar una nueva coordenada. A continuación, se detecta el nuevo estado s' con los sensores del dron, y se calcula la recompensa $r(s, a, s')$. Si la distancia del dron al objetivo es menor a 1 metro, la variable $done$ se establece en $True$, indicando que se ha alcanzado un estado terminal; de lo contrario, $done$ se establece en $False$. Un estado terminal también se puede alcanzar si el dron cae por debajo de una altura determinada. Finalmente, la clase ambiente envía al agente la tupla de experiencia $(s, a, r, s', done)$ para que se almacene en la memoria-cerca o memoria-lejos y pueda ser usada para su entrenamiento.

6.4 Algoritmo D3QN o DDDQN

El algoritmo utilizado para entrenar al agente para planificación de trayectoria local y evasión de obstáculos es el D3QN o DDDQN, que combina los algoritmos Double-DQN y Dueling-DQN explicados en la sección del marco teórico. Este enfoque emplea redes Dueling-DQN en lugar de redes DQN, tanto en las redes online como en las de objetivo del Double-DQN. La arquitectura utilizada para las redes del agente se muestra en la Figura 26. Esta arquitectura fue seleccionada basándose en la dimensión de las entradas y salidas, con el objetivo de lograr un aumento progresivo y luego una disminución en el número de neuronas. El uso de 8 capas se justifica porque ofrece la profundidad necesaria para capturar relaciones complejas en los datos, sin exagerar el número de capas, lo cual incrementaría los costos computacionales y el riesgo de sobreajuste, comprometiendo la eficiencia del modelo.

Figura 26

Arquitectura desarrollada para las dos redes neuronales Dueling-DQN usadas

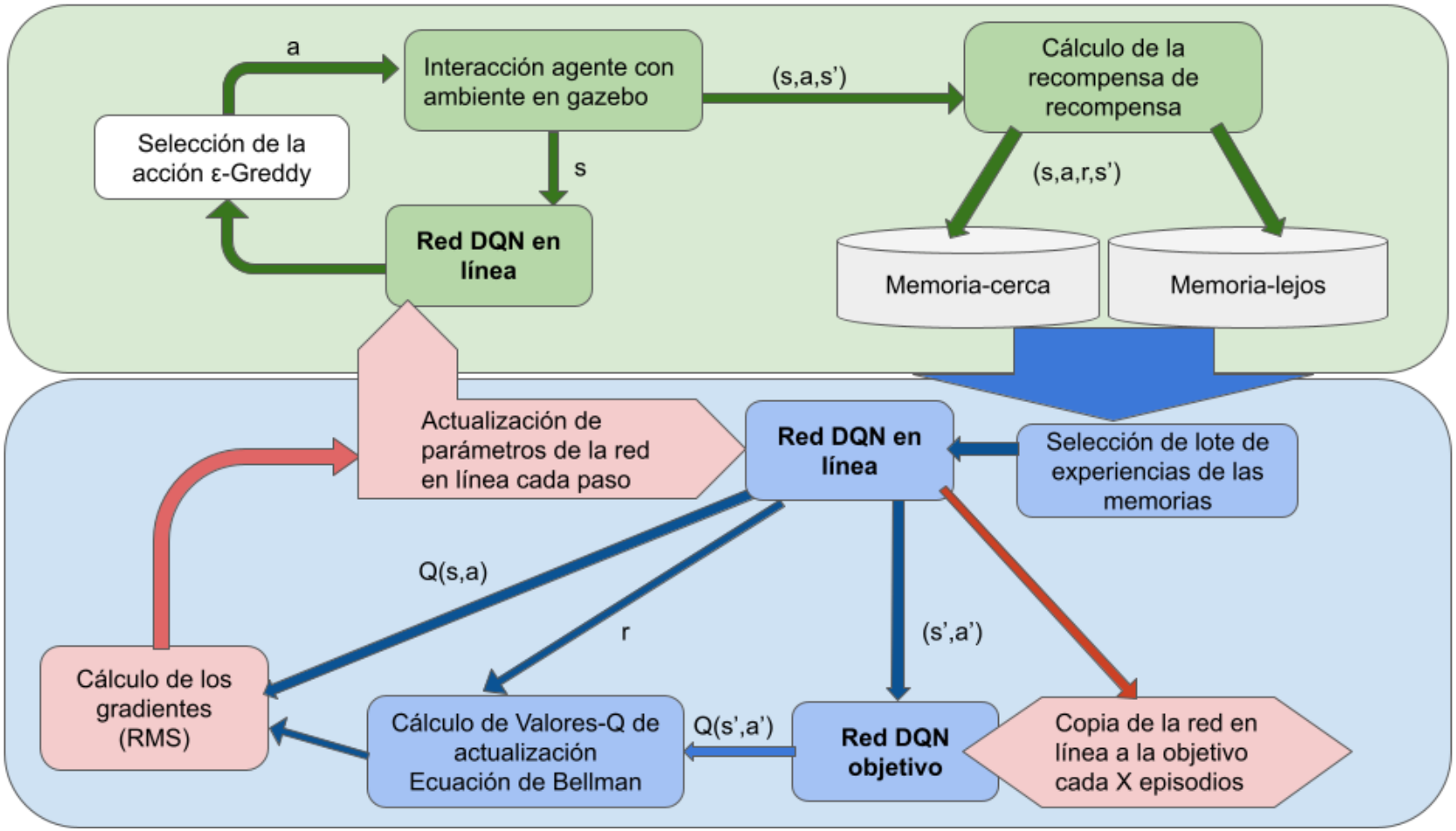


Se utilizaran dos memorias una de estas almacena las tuplas en las que el valor de la distancia normalizada al obstáculo más cercano detectado es menor a 0.36 (s[2]<0.36), a la que se le denomina memoria-cerca, y otra memoria con el resto de los datos a la que se le denomina memoria-lejos. Durante el entrenamiento, las tuplas se seleccionaron aleatoriamente de cada una de las dos memorias en cada paso en función del tamaño del lote, con el 60% de los datos provenientes de la memoria-cerca y el resto de la memoria-lejos. Esto se hizo con el objetivo de priorizar el aprendizaje del agente cerca a los obstáculos y que aprenda a evadirlos.

En la Figura 27 se muestra el diagrama esquemático del algoritmo DDDQN utilizado, donde las flechas azules representan el flujo de entrenamiento y las flechas verdes representan la interacción agente-entorno para la adquisición de experiencia.

Figura 27

Diagrama esquemático del algoritmo D3QN usado para el entrenamiento del agente



6.5 Resultados del entrenamiento del agente

En esta sección se explican las características del equipo usado para el entrenamiento, el procedimiento del entrenamiento, y se muestra la evolución de las rutas recorridas por el dron usando políticas de agentes guardados en distintas etapas del entrenamiento.

6.5.1 Materiales y método de entrenamiento

Una estación de trabajo con procesador AMD Ryzen 7 5700g de 4.6 GHz, gráficos integrados, 32 GB de RAM y sistema operativo Ubuntu 22.04 fue utilizada para el entrenamiento y las pruebas, que se realizaron con Gazebo 11.10.2 y ROS2 Humble.

Los programas de entrenamiento se desarrollaron en Python 3.10.12 utilizando PyTorch 2.1.2 (Paszke et al., 2019). El entrenamiento empleó los siguientes hiperparámetros: $\gamma = 0.995$, lr = 0.0005 y un tamaño de lote de 256. Para la exploración, se utilizó la selección ϵ-greedy, con un ϵ inicial de 1, que disminuía exponencialmente en cada episodio, multiplicándose por 0.995 hasta llegar a un valor mínimo de 0.05. La red en línea del agente se entrena, es decir actualiza sus parámetros en cada paso, y la red objetivo se copia de la red en línea cada 20 episodios.

Primero, se entrenó un agente sin modo de seguridad, este finalizaba los episodios cuando se alcanzaba la distancia de activación del modo de seguridad. Esta fase inicial involucró 2,000 episodios y 236,026 pasos, y el agente resultante fue denominado agente preentrenado.

Luego, se reanudó el entrenamiento utilizando al agente preentrenado como punto de partida, el código para realizar el entrenamiento se da en el **Apéndice F**, en esta ocasión el modo de seguridad estaba disponible para evitar colisiones. Esta segunda fase duró otros 1100 episodios y 533,500 pasos. La mayor cantidad de pasos por episodio se debe a que la duración del episodio con el modo de seguridad disponible es más larga, ya que el estado terminal se alcanzaba casi siempre al llegar al objetivo, aunque en algunas ocasiones, debido a fallos del simulador, el dron

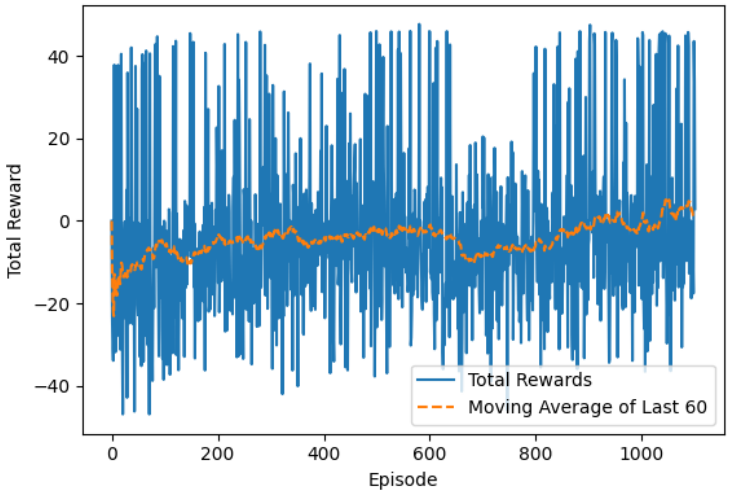
caía y se reiniciaba la simulación. El entrenamiento en cada fase tomó alrededor de 11 horas reales, con la simulación ejecutándose 4 veces más rápido que en la realidad, lo que resultó en un tiempo de simulación total de 44 horas por fase.

6.5.2 Evolución de las recompensas durante el entrenamiento

La evolución de las recompensas durante el entrenamiento partiendo del agente preentrenado se muestra en la Figura 28, donde la línea naranja es la tendencia del promedio de la recompensa de los últimos 60 episodios, en esta curva se refleja el progreso del aprendizaje del agente.

Figura 28

Evolución de la recompensa total por episodio



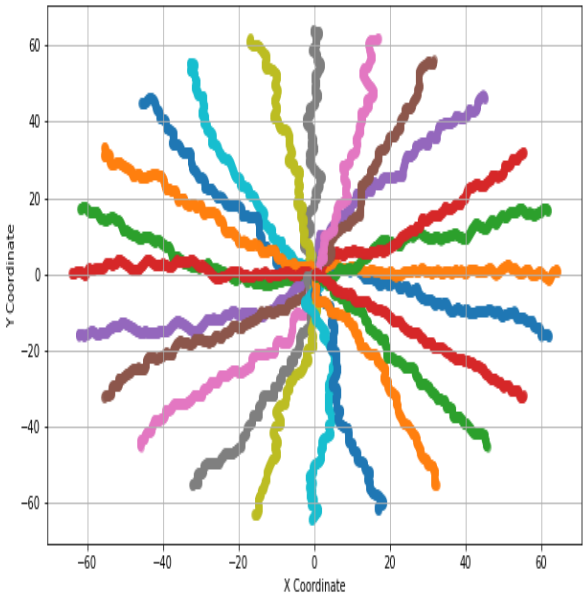
Al inicio, las recompensas fueron a menudo negativas debido a las frecuentes activaciones del Modo de Seguridad para prevenir choques. A medida que avanzó el entrenamiento, hubo un aumento notable en el promedio de las recompensas totales por episodio, lo que indica una mejora en el desempeño. En la Figura 28 se observa además una tendencia general ascendente en la curva de recompensas promedio, con fluctuaciones típicas del proceso de exploración-explotación.

6.5.3 Trayectorias seguidas por el dron durante el entrenamiento

Se realizó una prueba con el agente preentrenado, que consistió en llevar el dron a los 24 objetivos posibles, para esto se utilizó el código dado en el **Apéndice G**. Las trayectorias seguidas utilizando el agente preentrenado, pero con el modo de seguridad disponible para las pruebas, se muestran en la Figura 29. En esta fase de entrenamiento, el dron oscila demasiado entre las diferentes acciones posibles, incluso sin obstáculos cercanos, lo que no representa una ruta óptima.

Figura 29

Rutas seguidas por el agente preentrenado



Durante la segunda fase del entrenamiento se guardaron todos los agentes cada 20 episodios. Para visualizar el avance de la política de navegación del agente se realizaron pruebas con los agentes disponibles tomando en cuenta la curva de evolución de la recompensa para la selección de los agentes de prueba. Estás pruebas consisten en llevar al dron a los 24 objetivos

posibles. Las rutas seguidas por los agentes luego de: 300, 600, 1100 episodios se muestran en la Figura 30, Figura 31,

Figura 32, respectivamente.

Figura 30

Rutas seguidas por el agente luego de 300 episodios

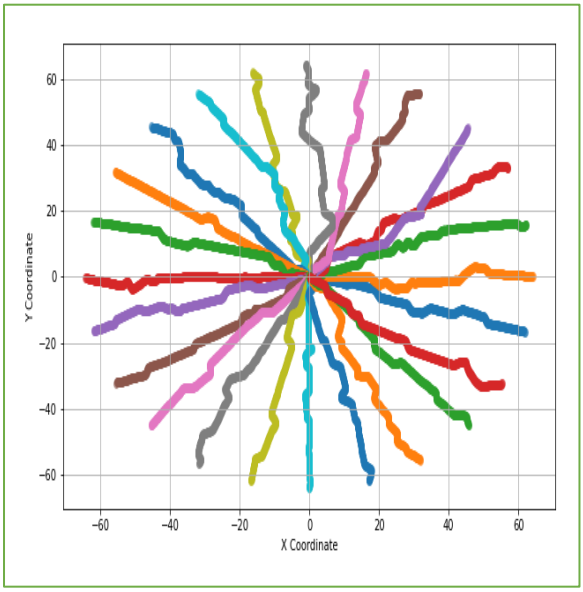


Figura 31

Rutas seguidas por el agente luego de 600 episodios

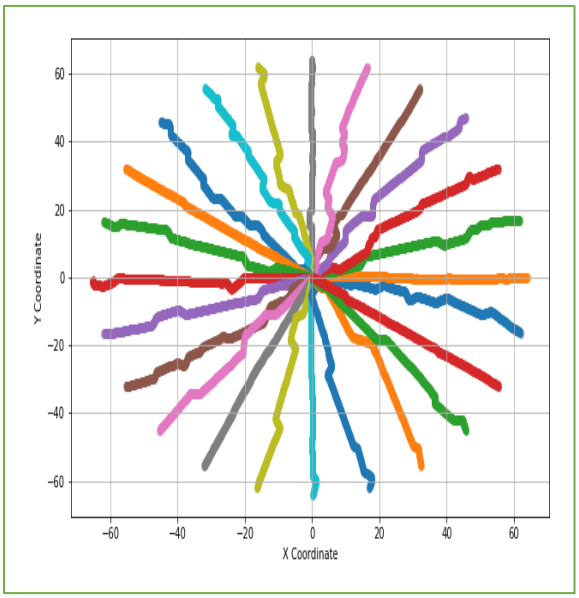
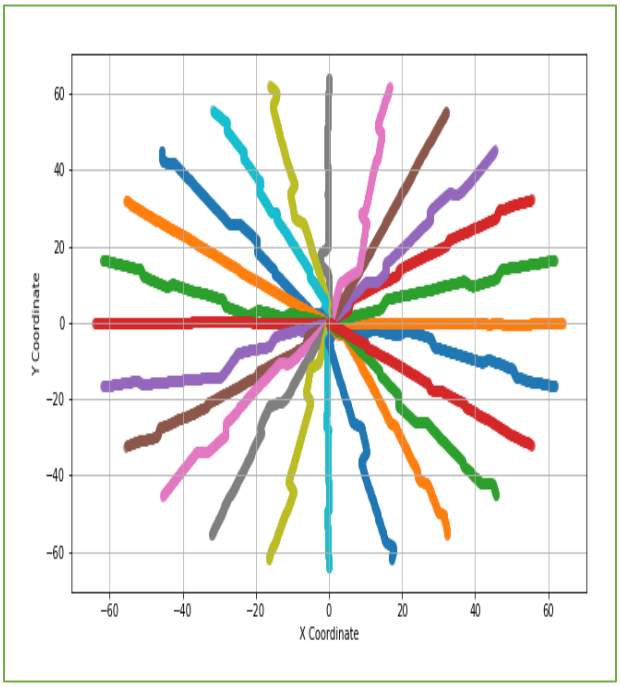


Figura 32

Rutas seguidas por el agente luego de 1100 episodios



De estas rutas se extraen algunas características mostradas en la Tabla 1, como la longitud de cada una de las rutas, excluyendo las rutas sin obstáculos. También se muestra el porcentaje de tiempo en que el Modo de Seguridad estuvo activo durante cada ruta. En la parte inferior de la tabla se proporciona la longitud total de todas las rutas, evidenciando una reducción del 13% en la distancia, de 1390 metros con el agente preentrenado a 1207 metros con el agente entrenado durante 1100 episodios. A partir de las gráficas de las rutas seguidas y la Tabla 1, se observa claramente cómo las trayectorias del dron evolucionaron hacia rutas más cortas y con menos oscilaciones con el tiempo.

Tabla 6

Características de las rutas seguidas por el dron

Episodios	Pre-entrenado		300		600		1100	
Ubicación de la posible meta X°	Longitud de la ruta (m)	% Modo de seguridad activo	Longitud de la ruta (m)	% Modo de seguridad activo	Longitud de la ruta (m)	% Modo de seguridad activo	Longitud de la ruta (m)	% Modo de seguridad activo
15	78.1	12.1	69.8	10.0	66.9	12.4	66.5	12.9
30	75.2	5.7	71.7	15.6	68.2	12.8	66.6	12.8
45	81.6	8.8	70.5	7.2	70.3	14.9	67.5	13.9
75	77.7	11.1	68.8	12.0	68.3	14.5	67.9	10.8
105	78.3	9.0	72.6	21.0	67.6	17.1	67.3	15.3
120	76.5	3.4	69.4	8.6	69.8	20.3	67.7	15.6
135	79.9	16.6	70.4	12.9	69.2	16.5	67.9	17.6
150	81.8	11.1	65.8	3.3	64.4	1.0	64.4	1.0
165	75.6	9.4	68.0	12.8	67.3	12.9	67.6	16.1
195	76.5	11.3	68.4	16.4	66.8	13.7	67.8	8.6
210	75.7	4.9	67.8	13.4	69.1	14.0	66.8	12.1
225	78.3	10.2	66.9	13.0	67.1	8.7	67.0	12.6
255	76.7	8.1	67.6	13.2	66.8	13.1	66.7	13.1
285	78.7	14.4	69.3	17.7	67.4	14.2	67.8	16.6
300	74.4	11.3	70.8	14.6	68.0	8.0	66.7	8.3
315	73.9	11.5	68.2	18.4	67.1	12.9	67.9	17.5
330	72.2	4.1	70.6	12.4	64.7	2.8	65.4	2.5
345	78.7	11.2	68.6	18.2	69.0	13.2	67.7	17.2
Total (m)	1389.9		1245.2		1218.1		1207.4	
Promedio (%)		9.7		13.4		12.4		12.5

En la Tabla 6 se muestra que el agente pre-entrenado estuvo el 9.7% de los pasos con el modo de seguridad activo. Este porcentaje más bajo se debe a que su método de entrenamiento consideraba la activación del modo de seguridad como un estado terminal, lo que mejoró la evasión de obstáculos, pero resultó en rutas más largas. En contraste, los agentes en las etapas posteriores del entrenamiento pasaron aproximadamente el 12.4% de los pasos con el modo de seguridad activo, pero lograron alcanzar rutas más cortas a medida que avanzaba el entrenamiento. Esta evolución coincide con el aumento de las recompensas generales observado en la sección anterior, indicando que la capacidad del agente para planificar trayectorias locales más eficientes mejoró con el tiempo.

6.5.4 Evaluación de la planificación de trayectoria local y evasión de obstáculos

Uno de los objetivos del proyecto es que el agente pueda realizar la planificación de la ruta local y la evasión de obstáculos utilizando el agente D3QN. En este caso, se logró la planificación de la ruta hasta el objetivo, pero la evasión de obstáculos no se logró de manera directa, ya que el agente solo tiene la capacidad de dar la próxima coordenada a seguir. Sin embargo, al observar la ruta final seguida por el dron y el porcentaje de activación del modo de seguridad, se notó una reducción en las activaciones del modo de seguridad en comparación con las fases iniciales del entrenamiento. Esto indica que el agente depende menos del modo de seguridad a medida que avanzan los episodios y trata de seguir rutas que no tengan obstáculos cerca.

6.6 Conclusiones de la sección y trabajos futuros

En esta sección, se aplicó con éxito el algoritmo D3QN para desarrollar un agente autónomo en una plantación simulada de palma de aceite. El agente daba coordenadas paso a paso para llegar a la meta, a medida que avanzó el entrenamiento, disminuyeron las oscilaciones en su trayectoria y se siguieron rutas más seguras. Durante el entrenamiento, el agente mostró una mejora progresiva en sus habilidades de navegación, siguiendo caminos cercanos a la línea recta y una tendencia consistente en el incremento de la recompensa acumulada promedio por episodio. Aunque el agente no aprendió directamente a evadir obstáculos, las rutas resultantes naturalmente minimizaron la proximidad a los obstáculos, lo que sugiere un potencial para aplicaciones prácticas en entornos complejos.

La simulación realista de la dinámica del dron en entornos complejos de plantaciones de palma de aceite por medio de Gazebo fue crucial para el entrenamiento y validación del agente de aprendizaje por refuerzo. El modelado preciso del comportamiento del dron y las interacciones

con los sensores permitió el desarrollo seguro y eficiente del agente de navegación, acelerando el entrenamiento del agente sin requerir pruebas en el mundo real y los costos que estas acarrean.

El agente entrenado durante 1100 episodios logró una reducción del 13% en la longitud de la ruta en comparación con el agente preentrenado. Sin embargo, el agente preentrenado tenía un 9.7% de activación del modo de seguridad debido a que el entrenamiento terminaba cuando se activaba este modo. En contraste, el agente entrenado durante 1100 episodios tuvo un 12.5% de activación del modo de seguridad, lo que indica que, aunque el preentrenado aprendió a esquivar mejor los obstáculos, seguía rutas más largas y oscilatorias. Esto mostró el impacto de la forma en que el entrenamiento es llevado a cabo en el equilibrio entre la eficiencia de la ruta y la gestión de la seguridad.

Los trabajos futuros que se pueden llevar a cabo incluyen: la implementación del agente de aprendizaje por refuerzo entrenado en un dron que navegue al interior de una plantación real de palma de aceite para validar su desempeño fuera del entorno simulado. Además, explorar algoritmos de aprendizaje por refuerzo más avanzados para optimizar aún más la eficiencia de la navegación del dron y sus capacidades de evasión de obstáculos. Para evaluar mejor la robustez del agente, sería beneficioso aumentar la complejidad del entorno simulado introduciendo obstáculos en movimiento o variando las velocidades del viento.

7. Conclusiones

Se desarrolló un modelo dinámico del dron hexarrotor utilizando la mecánica de Lagrange, lo que permitió obtener ecuaciones de movimiento precisas y representativas del sistema. Este modelo, que incluye las fuerzas de arrastre parasitas y los torques giroscópicos, fue derivado a partir del modelo simulado del dron en el entorno ROS2-Gazebo, proporcionando una base sólida para el diseño y evaluación de diferentes estrategias de control. La metodología empleada, basada en la dinámica de Lagrange, es adaptable a drones con configuraciones variadas, lo que permite modificar y extender el modelo para futuras investigaciones y aplicaciones.

Se diseñó e implementó un controlador LQG-Integral para el seguimiento de trayectoria del dron hexarrotor, demostrando un desempeño superior en comparación con los controladores LQG y PID. Las simulaciones realizadas en Simulink mostraron que el controlador LQG-Integral superó al PID en términos de tiempo de establecimiento, siendo un 41% más rápido en la coordenada x y un 57% más rápido en y. Además, todos los controladores fueron lo suficientemente estables para operar correctamente incluso con ruido de proceso, simulado mediante vientos de hasta 3 m/s en direcciones específicas. El componente integral del controlador LQG-Integral permitió eliminar el error en estado estable en todas las coordenadas, a diferencia del controlador LQG, que presentó errores de hasta un 24% en z y un 13% en x y y. Estos resultados confirman la efectividad del controlador LQG-Integral en el cumplimiento del objetivo de lograr un seguimiento preciso de la trayectoria del dron, incluso en presencia de perturbaciones externas.

Se desarrolló un sistema de navegación autónoma para el dron hexarrotor utilizando el algoritmo de aprendizaje por refuerzo DDDQN (Dueling Double Deep Q-Network). El sistema

fue entrenado y validado en un entorno simulado en ROS2-Gazebo, donde el dron, equipado con sensores LIDAR e IMU, interactuó en un escenario que replicaba la distribución de los cultivos de palma de aceite y obstáculos típicos del entorno real. El agente DDDQN fue capaz de generar trayectorias de navegación que evitaban los obstáculos, enviando las coordenadas correspondientes al controlador LQG-Integral. Este controlador se encargó de seguir con precisión las trayectorias proporcionadas, garantizando una navegación autónoma efectiva a través del entorno simulado. El entrenamiento se realizó a una velocidad de simulación cuatro veces superior al tiempo real, lo que permitió optimizar el uso de recursos y acelerar el proceso de validación. La simulación permitió tanto el entrenamiento como las pruebas sin la necesidad de un dron físico ni el desplazamiento a cultivos reales, optimizando así los tiempos de desarrollo.

Se validó por medio de simulación el sistema de navegación autónoma del dron hexarrotor en un entorno simplificado de Gazebo, basado en las dimensiones y posiciones de los obstáculos típicos en un cultivo de palma de aceite. Esta simulación permitió evaluar el desempeño del algoritmo DDDQN para la generación de trayectorias locales y la evasión de obstáculos en un escenario que, aunque simplificado, replica las características principales del entorno real. La simulación resultó ser una herramienta eficaz para el entrenamiento y evaluación del sistema sin necesidad de realizar pruebas en un entorno físico. Este enfoque no solo ahorró tiempo y recursos, sino que también proporcionó una base confiable para la futura transferencia del sistema de navegación autónoma a entornos reales.

Trabajos futuros

Como trabajos futuros, se busca utilizar el modelo del dron hexarrotor creado en ROS2-Gazebo para desarrollar controladores que consideren fallas parciales o totales en alguno de los rotores. Además, el entorno simulado puede ser reutilizado para entrenar y evaluar otras estrategias de aprendizaje por refuerzo, comparando sus resultados con los obtenidos en esta investigación. Finalmente, si se cuentan con los recursos y permisos necesarios, se podrían realizar pruebas del sistema de navegación en entornos reales para verificar el desempeño de los controladores en un dron que opere en cultivos de palma de aceite.

Referencias Bibliográficas

Ahn, H., Hu, M., Chung, Y., & You, K. (2023). Sliding-Mode Control for Flight Stability of Quadrotor Drone Using Adaptive Super-Twisting Reaching Law. *Drones*, *7*(8), Article 8. https://doi.org/10.3390/drones7080522

Al Younes, Y., & Barczyk, M. (2022). Adaptive Nonlinear Model Predictive Horizon Using Deep Reinforcement Learning for Optimal Trajectory Planning. *Drones*, *6*(11), 323. https://doi.org/10.3390/drones6110323

Aldao, E., González-deSantos, L. M., Michinel, H., & González-Jorge, H. (2022). UAV Obstacle Avoidance Algorithm to Navigate in Dynamic Building Environments. *Drones*, *6*(1), 16. https://doi.org/10.3390/drones6010016

Aswini, N., Uma, S. V., & Akhilesh, V. (2022). Drone to Obstacle Distance Estimation Using YOLO V3 Network and Mathematical Principles. *Journal of Physics: Conference Series*, *2161*(1), Article 1. https://doi.org/10.1088/1742-6596/2161/1/012022

Bachrach, A., de Winter, A., He, R., Hemann, G., Prentice, S., & Roy, N. (2010). RANGE - robust autonomous navigation in GPS-denied environments. *2010 IEEE International Conference on Robotics and Automation*, 1096–1097. https://doi.org/10.1109/ROBOT.2010.5509990

Carlucho, I., De Paula, M., Villar, S. A., & Acosta, G. G. (2017). Incremental *Q*-learning strategy for adaptive PID control of mobile robots. *Expert Systems with Applications*, *80*, 183–199. https://doi.org/10.1016/j.eswa.2017.03.002

Carreño Sánchez, L. F., & Ramirez Plata, L. H. (2023). *Diseño y simulación de un controlador y regulador cuadrático lineal (LQR) con aplicación de filtros de Kalman (LQG), para complementar las materias de sistemas dinámicos y control en la Universidad Industrial de Santander.* [UNIVERSIDAD INDUSTRIAL DE SANTANDER]. https://noesis.uis.edu.co/handle/20.500.14071/14253

Chiriacò, M. V., Bellotta, M., Jusić, J., & Perugini, L. (2022). Palm oil's contribution to the United Nations sustainable development goals: Outcomes of a review of socio-economic aspects. *Environmental Research Letters*, *17*(6), 063007. https://doi.org/10.1088/1748-9326/ac6e77

Choi, J., Kim, H. M., Hwang, H. J., Kim, Y.-D., & Kim, C. O. (2023). Modular Reinforcement Learning for Autonomous UAV Flight Control. *Drones*, *7*(7), Article 7. https://doi.org/10.3390/drones7070418

Cocoma-Ortega, J. A., & Martinez-Carranza, J. (2022). A compact CNN approach for drone localisation in autonomous drone racing. *Journal of Real-Time Image Processing*, *19*(1), 73–86. https://doi.org/10.1007/s11554-021-01162-3

Du, H., Wang, Z., & Zhang, X. (2023). EF-TTOA: Development of a UAV Path Planner and Obstacle Avoidance Control Framework for Static and Moving Obstacles. *Drones*, *7*(6), 359. https://doi.org/10.3390/drones7060359

Fan, Y. (2022). Flight Control System Simulation for Quadcopter Unmanned Aerial Vehicle (UAV) based on Matlab Simulink. *Journal of Physics: Conference Series*, *2283*(1), 012011. https://doi.org/10.1088/1742-6596/2283/1/012011

Fedepalma. (2022). *La palma de aceite en Colombia*. Fedepalma. https://fedepalma.org/zonas-palmeras/zona-norte/

Fedepalma. (2024, julio). Estas son las cifras más importantes del aceite de palma en Colombia – Empaquetadora del Norte. *Empaquetadora del norte*. https://empaquetadoradelnorte.com/2024/07/08/estas-son-las-cifras-mas-importantes-del-aceite-de-palma-en-colombia/

Grupo Jaremar. (2016). *Manual de Buenas Prácticas Agrícolas para la Producción Sostenible de la Palma Aceitera por Pequeños Productores*.

Hasselt, H. van, Guez, A., & Silver, D. (2016). Deep reinforcement learning with double Q-Learning. *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, 2094–2100.

Hirota, Y., Taniguchi, I., & Onoye, T. (2020). Parallelization of Local Path Planning for High Reliable Autonomous Drones. *2020 International SoC Design Conference (ISOCC)*, 67–68. https://doi.org/10.1109/ISOCC50952.2020.9333111

Hodgson, M. E., Vitzilaios, N. I., Myrick, M. L., Richardson, T. L., Duggan, M., Sanim, K. R. I., Kalaitzakis, M., Kosaraju, B., English, C., & Kitzhaber, Z. (2022). Mission Planning for Low Altitude Aerial Drones during Water Sampling. *Drones*, *6*(8), 209. https://doi.org/10.3390/drones6080209

Hoyos, N., & Borrás, C. (2024). *Modelo de detección y tolerante a fallas con observadores para un servosistema electrohidráulico de posición*. https://noesis.uis.edu.co/handle/20.500.14071/42256

Hwang, H. J., Jang, J., Choi, J., Bae, J. H., Kim, S. H., & Kim, C. O. (2023). Stepwise Soft Actor–Critic for UAV Autonomous Flight Control. *Drones*, *7*(9), 549. https://doi.org/10.3390/drones7090549

Isermann, R. (2006). *Fault-diagnosis systems: An introduction from fault detection to fault tolerance*. Springer.

Jarray, R., Bouallègue, S., Rezk, H., & Al-Dhaifallah, M. (2022). Parallel Multiobjective Multiverse Optimizer for Path Planning of Unmanned Aerial Vehicles in a Dynamic Environment with Moving Obstacles. *Drones*, *6*(12), 385. https://doi.org/10.3390/drones6120385

Jung, S., Lee, H., Shim, D. H., & Agha-mohammadi, A. (2021). Collision-free local planner for unknown subterranean navigation. *ETRI Journal*, *43*(4), 580–593. https://doi.org/10.4218/etrij.2021-0087

Kalidas, A. P., Joshua, C. J., Md, A. Q., Basheer, S., Mohan, S., & Sakri, S. (2023). Deep Reinforcement Learning for Vision-Based Navigation of UAVs in Avoiding Stationary and Mobile Obstacles. *Drones*, *7*(4), Article 4. https://doi.org/10.3390/drones7040245

Kim, M., Kim, J., Jung, M., & Oh, H. (2022). Towards monocular vision-based autonomous flight through deep reinforcement learning. *Expert Systems with Applications*, *198*, 116742. https://doi.org/10.1016/j.eswa.2022.116742

Kurdel, P., Češkovič, M., Gecejová, N., Adamčík, F., & Gamcová, M. (2022). Local Control of Unmanned Air Vehicles in the Mountain Area. *Drones*, *6*(2). https://doi.org/10.3390/drones6020054

Lapan, M. (2018). *Deep Reinforcement Learning Hands-On: Apply Modern RL Methods, with Deep Q-Networks, Value Iteration, Policy Gradients, TRPO, AlphaGo Zero and More*. Packt Publishing, Limited. http://ebookcentral.proquest.com/lib/bibliouis-ebooks/detail.action?docID=5434975

Lee, A. W. C., Yong, S.-P., & Watada, J. (2019). Global Thresholding for Scene Understanding Towards Autonomous Drone Navigation. *Journal of Advanced Computational Intelligence and Intelligent Informatics*, *23*(5), 909–919. https://doi.org/10.20965/jaciii.2019.p0909

Liang, Q., Wang, Z., Yin, Y., Xiong, W., Zhang, J., & Yang, Z. (2023). Autonomous aerial obstacle avoidance using LiDAR sensor fusion. *PLOS ONE*, *18*(6), e0287177. https://doi.org/10.1371/journal.pone.0287177

Liu, H., Suzuki, S., Wang, W., Liu, H., & Wang, Q. (2022). Robust Control Strategy for Quadrotor Drone Using Reference Model-Based Deep Deterministic Policy Gradient. *Drones*, *6*(9), 251. https://doi.org/10.3390/drones6090251

Lorca, A., & Zapata, S. (2022). *El aceite de Palma, análisis de su uso en España* [Universidad Politécnica de Cartagena]. https://repositorio.upct.es/entities/publication/13968ef1-d3d8-4c20-9be4-7868b64287d6

Macenski, S., Foote, T., Gerkey, B., Lalancette, C., & Woodall, W. (2022). Robot Operating System 2: Design, Architecture, and Uses In The Wild. *Science Robotics*, *7*(66), eabm6074. https://doi.org/10.1126/scirobotics.abm6074

Mishra, B., Garg, D., Narang, P., & Mishra, V. (2020). Drone-surveillance for search and rescue in natural disaster. *Computer Communications*, *156*, 1–10. https://doi.org/10.1016/j.comcom.2020.03.012

Navas, O.-M., & Borrás, C. (2024, octubre 22). Modelado y seguimiento de trayectorias para dron hexacóptero: Evaluación comparativa de estrategias PID, LQG y PI-LQG. *Actas del XV Congreso Iberoamericano de Ingeniería Mecánica*.

Navas, O.-M., Navas, J.-J., & Borrás, C. (2024). Obstacle avoidance and local path planning based on reinforcement learning for hexacopter drone in simulated environments. *Proceedings of the 2024 9th International Engineering, Sciences and Technology Conference (IESTEC)*.

Nguyen, N. P., Xuan Mung, N., Ha, L. N. N. T., & Hong, S. K. (2022). Fault-Tolerant Control for Hexacopter UAV Using Adaptive Algorithm with Severe Faults. *Aerospace*, *9*(6), Article 6. https://doi.org/10.3390/aerospace9060304

Nguyen, N. P., Xuan Mung, N., & Hong, S. K. (2019). Actuator Fault Detection and Fault-Tolerant Control for Hexacopter. *Sensors*, *19*(21), Article 21. https://doi.org/10.3390/s19214721

Nhair, R. R., & Al-Assadi, T. A. (2020). Vision-Based Obstacle Avoidance for Small Drone using Monocular Camera. *IOP Conference Series. Materials Science and Engineering*, *928*(3). https://doi.org/10.1088/1757-899X/928/3/032048

Oba, T., Bando, M., & Hokamoto, S. (2018). Controller performance for quad-rotor vehicles based on sliding mode control. *Journal of Robotics and Mechatronics*, *30*(3), 397–405. https://doi.org/10.20965/jrm.2018.p0397

Ogata, K. (2010). *Ingeniería de control moderna* (pp 793-803; 5a ed.). Pearson Educación.

Orozco Soto, S. M., Cacace, J., Ruggiero, F., & Lippiello, V. (2022). Active Disturbance Rejection Control for the Robust Flight of a Passively Tilted Hexarotor. *Drones*, *6*(9), 258. https://doi.org/10.3390/drones6090258

Ortner, R., Kurmi, I., & Bimber, O. (2021). Acceleration-Aware Path Planning with Waypoints. *Drones*, *5*(4). https://doi.org/10.3390/drones5040143

Panait, M. A. (2022). Neural Reflex Networks for Automating Quadcopter Drone Obstacle Avoidance. *INCAS Bulletin*, *14*(2), 65–73. https://doi.org/10.13111/2066-8201.2022.14.2.6

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems*, *32*. https://proceedings.neurips.cc/paper_files/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

Raghunatha, A., Thollander, P., & Barthel, S. (2023). Addressing the emergence of drones – A policy development framework for regional drone transportation systems. *Transportation Research Interdisciplinary Perspectives*, *18*, 100795. https://doi.org/10.1016/j.trip.2023.100795

Rejeb, A., Abdollahi, A., Rejeb, K., & Treiblmaier, H. (2022). Drones in agriculture: A review and bibliometric analysis. *Computers and Electronics in Agriculture*, *198*, 107017. https://doi.org/10.1016/j.compag.2022.107017

Rojas-Perez, L. O., & Martinez-Carranza, J. (2020). DeepPilot: A CNN for Autonomous Drone Racing. *Sensors*, *20*(16), Article 16. https://doi.org/10.3390/s20164524

Safa, A., Verbelen, T., Keuninckx, L., Ocket, I., Hartmann, M., Bourdoux, A., Catthoor, F., & Gielen, G. (2021, julio). *A Low-Complexity Radar Detector Outperforming OS-CFAR for Indoor Drone Obstacle Avoidance*. arXiv. https://doi.org/10.48550/arXiv.2107.07250

Sang-Yun, S., & Yong-Won, K. (2019). Obstacle Avoidance Drone by Deep Reinforcement Learning and Its Racing with Human Pilot. *Applied Sciences*, *9*(24). https://doi.org/10.3390/app9245571

Sato, M., & Iwase, M. (2019). Semi-autonomous flight control of forestry-use drone. *2019 58th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, 1232–1235. https://doi.org/10.23919/SICE.2019.8859900

Schaub, H., & Junkins, J. L. (2018). *Analytical mechanics of space systems* (4a ed.). American Institute of Aeronautics and Astronautics, Inc.

Shin, S.-Y., Kang, Y.-W., & Kim, Y.-G. (2019). Obstacle Avoidance Drone by Deep Reinforcement Learning and Its Racing with Human Pilot. *Applied Sciences*, *9*(24), Article 24. https://doi.org/10.3390/app9245571

Song, Y., & Scaramuzza, D. (2022). Policy Search for Model Predictive Control With Application to Agile Drone Flight. *IEEE Transactions on Robotics*, *38*(4), 2114–2130. https://doi.org/10.1109/TRO.2022.3141602

Sun, M., Liu, J., Wang, H., Nian, X., & Xiong, H. (2018). Robust fuzzy tracking control of a quad-rotor unmanned aerial vehicle based on sector linearization and interval matrix approaches. *ISA Transactions*, *80*, 336–349. https://doi.org/10.1016/j.isatra.2018.07.034

Sutton, R., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). The MIT Press. http://archive.org/details/rlbook2018

Szegleti, A., & Márkus, F. (2020). Dissipation in Lagrangian Formalism. *Entropy*, *22*(9), Article 9. https://doi.org/10.3390/e22090930

Tang, J., Liang, Y., & Li, K. (2024). Dynamic Scene Path Planning of UAVs Based on Deep Reinforcement Learning. *Drones*, *8*(2), 60. https://doi.org/10.3390/drones8020060

Tu, G.-T., & Juang, J.-G. (2023). UAV Path Planning and Obstacle Avoidance Based on Reinforcement Learning in 3D Environments. *Actuators*, *12*(2), 57. https://doi.org/10.3390/act12020057

United Nations. (2015, septiembre). *Objetivos de Desarrollo Sostenible | Naciones Unidas*. United Nations; United Nations. https://www.un.org/es/impacto-acad%C3%A9mico/page/objetivos-de-desarrollo-sostenible

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., & de Freitas, N. (2016). *Dueling Network Architectures for Deep Reinforcement Learning* (arXiv:1511.06581). arXiv. https://doi.org/10.48550/arXiv.1511.06581

Wen, F.-H., Hsiao, F.-Y., & Shiau, J.-K. (2021). Analysis and Management of Motor Failures of Hexacopter in Hover. *Actuators*, *10*(3), Article 3. https://doi.org/10.3390/act10030048

Xiao, M., Liang, J., Ji, L., Sun, Z., & Li, Z. (2022). Aerial photography trajectory-tracking controller design for quadrotor UAV. *Measurement and Control*, *55*(7–8), 738–745. https://doi.org/10.1177/00202940221115634

Xue, Z. (2021). Vision Based Drone Obstacle Avoidance by Deep Reinforcement Learning. *AI*, *2*(3). https://doi.org/10.3390/ai2030023

Yang, F., Fang, X., Gao, F., Zhou, X., Li, H., Jin, H., & Song, Y. (2022). Obstacle Avoidance Path Planning for UAV Based on Improved RRT Algorithm. *Discrete Dynamics in Nature and Society*, *2022*, e4544499. https://doi.org/10.1155/2022/4544499

Yang, X., Chen, J., Dang, Y., Luo, H., Tang, Y., Liao, C., Chen, P., & Cheng, K.-T. (2021). Fast Depth Prediction and Obstacle Avoidance on a Monocular Drone Using Probabilistic Convolutional Neural Network. *IEEE Transactions on Intelligent Transportation Systems*, *22*(1), Article 1. IEEE Transactions on Intelligent Transportation Systems. https://doi.org/10.1109/TITS.2019.2955598

Yarak, K., Witayangkurn, A., Kritiyutanont, K., Arunplod, C., & Shibasaki, R. (2021). Oil Palm Tree Detection and Health Classification on High-Resolution Imagery Using Deep Learning. *Agriculture*, *11*(2), Article 2. https://doi.org/10.3390/agriculture11020183

Yazid, E., Garratt, M., & Santoso, F. (2019). Position control of a quadcopter drone using evolutionary algorithms-based self-tuning for first-order Takagi–Sugeno–Kang fuzzy logic autopilots. *Applied Soft Computing*, *78*, 373–392. https://doi.org/10.1016/j.asoc.2019.02.023

Yu, J., Li, J., Zhang, T., Yan, B., Li, S., & Meng, Z. (2023). Speed-First: An Aggressive Gradient-Based Local Planner for Quadrotor Faster Flight. *Drones*, *7*(3), 192. https://doi.org/10.3390/drones7030192

Zhang, N., Zhang, M., & Low, K. H. (2021). 3D path planning and real-time collision resolution of multirotor drone operations in complex urban low-altitude airspace. *Transportation Research Part C: Emerging Technologies*, *129*, 103123. https://doi.org/10.1016/j.trc.2021.103123

Apéndices

Los apéndices están adjuntos y se pueden consultar en el Repositorio Institucional.

Apéndice A. Código de Matlab para obtener ecuaciones de movimiento y controlador LQG

El siguiente es el código que se encuentra en el archivo llamado *Apendice_A_Codigo_EOM_y_controlador_LQG.m*, se incluye completo porque se referencian algunas de sus partes en el contenido del texto.

```
clc
clear all
close all

% PARTE1 PARAMETROS DEL SISTEMA Y MATRICES DE ROTACION
%Variables símbolica a utilizar

%Posiciones en las coordenadas x,y,z y velocidades lineales
syms x y z real
syms xdot ydot zdot real

%Angulos de euler y su varición en el tiempo
syms phi theta psi real
syms phidot thetadot psidot real

% Aceleraciones de las variables generalizadas
syms xddot yddot zddot phiddot thetaddot psiddot real

%velocidad del viento en las direcciones x, y, z
syms v_wx v_wy v_wz real

%Velocidad de rotación de cada propulsor elevada al cuadrado
syms u1 u2 u3 u4 u5 u6 real


% Párametros del sistema

m = 0.9725;          % Masa del dron en kg
g = 9.81;            % Aceleración de la gravedad en m/s^2
Ixx = 3.7575e-2;     % Momento de inercia del CM del cuerpo con respecto a x,
unidades: kg-m^2
Iyy = 3.826e-2;      % Momento de inercia del CM del cuerpo con respecto a y,
unidades: kg-m^2
Izz = 7.113e-2;      % Momento de inercia del CM del cuerpo con respecto a z,
unidades: kg-m^2
```

```
Izr = 3.02e-4;      % Momento de inercia de la helice con respecto a z,
unidades: kg-m^2
l_1 = 0.357;        % Distancia en el plano horizontal desde el CM hasta los
propulsores  1, 2, 4, 5 unidades: m
l_2 = 0.343;        % Distancia en el plano horizontal desde el CM hasta los
propulsores  3, 6 unidades: m
K_T = 2.19e-5;      % Consante de fuerza de empuje del propulsor, unidad: N-
s^2
K_Q = 1.99e-7;      % Consante de torque del propulsor, unidad: N-m-s^2
cx  = 0.01616;      % Coeficiente de arrastre del motor en la dirección x,
unidad: N-s^2/m^2
cy  = 0.01616;      % Coeficiente de arrastre del motor en la dirección y,
unidad: N-s^2/m^2
cz  = 0.0498;       % Coeficiente de arrastre del motor en la dirección z,
unidad: N-s^2/m^2
alpha = pi/3;       % Ángulo entre brazos del hexarotor, unidad: rad
T_s = 0.01;         % Tiempo de muestreo, unidad: s

% Matriz de inercia del drone con respecto al SRC
I = [ Ixx   0     0;...
      0     Iyy   0;...
      0     0     Izz];

% Vectores unitarios del SRI
i = [1 0 0]';
j = [0 1 0]';
k = [0 0 1]';

% Matriz de rotación con respecto al ángulo de alabeo
R_x = [     1           0           0;          ...
            0           cos(phi)    -sin(phi);  ...
            0           sin(phi)    cos(phi)];

% Matriz de rotación con respecto al ángulo de cabeceo
R_y = [     cos(theta)        0     sin(theta);       ...
            0                 1           0;          ...
            -sin(theta)       0     cos(theta)];

% Matriz de rotación con respecto al ángulo de guiñada
R_z = [     cos(psi)    -sin(psi)   0; ...
            sin(psi)    cos(psi)    0; ...
            0                 0       1];

% Matriz de rotación total
R = R_z*R_y*R_x;

%% Parte 2. Obtención de las ecuaciones de movimiento por el método de
Lagrange
% Variables generalizadas
q    = [x,   y,    z,         phi,        theta,            psi];
qdot = [xdot,      ydot,       zdot,           phidot,           thetadot,
      psidot];
qddot= [xddot, yddot,    zddot,          phiddot,    thetaddot,  psiddot];

% Velocidad lineal
```

```
v = [xdot; ydot; zdot];

% Velocidad angular del dron con respecto al SRC
om_b = phidot*i +  R_x'*(thetadot*j) + R_x'*R_y'*(psidot*k);

%Velocidad relativa del dron con respecto al viento
V_rel =  [xdot-v_wx;    ydot-v_wy;  zdot-v_wz] ;

% Fuerzas externas del sistema

% Fuerza de arrastre
Drag = [cx*V_rel(1)*abs(V_rel(1));...
        cy*V_rel(2)*abs(V_rel(2));...
        cz*V_rel(3)*abs(V_rel(3))];

% Fuerza de empuje
Thrust = [0; 0; K_T*(u1+ u2+ u3+ u4+ u5+ u6)];

% Fuerza total
F_ext = R*Thrust - Drag;


% Torques externas del sistema

% Torques Giroscopicos
momentoAngTotal =   Izr*sqrt(abs(u1)) - Izr*sqrt(abs(u2))+ ...
                    Izr*sqrt(abs(u3)) - Izr*sqrt(abs(u4))+...
                    Izr*sqrt(abs(u5)) - Izr*sqrt(abs(u6));
Giroscopic = cross(momentoAngTotal*k,om_b);

% Torques producido por las fuerzas
torque1 = K_T*(l_1*cos(alpha)*(u1 + u5 - u2 - u4)+ l_2*(-u3 + u6));
torque2 = K_T*l_1*sin(alpha)*(u1 + u2 - u4 - u5);

% Torque motor
torque_motor = K_Q*(-u1 - u3 - u5 + u2 + u4 + u6);

% Torque en cada eje del SRI
tau_phi = torque1 + Giroscopic(1);
tau_theta = torque2  + Giroscopic(2);
tau_psi = torque_motor+ Giroscopic(3);

%Torque total
tau_ext = [tau_phi; tau_theta; tau_psi];


%Vector de fuerza y torques generalizados
T_ext = [F_ext; tau_ext];

% Cálculo de la energía cinética
T = 0.5*m*(v')*v + 0.5*om_b'*I*om_b;

% Cálculo de la energía potencial
V = m*g*z;
```

```
% Calculo del Lagrangiano
L = T-V;

for ii=1:6
    dLdqdot(ii) = diff(L,qdot(ii));
    ddt_dLdqdot(ii) = diff(dLdqdot(ii),q(1))*qdot(1) +
diff(dLdqdot(ii),qdot(1))*qddot(1)+...
                      diff(dLdqdot(ii),q(2))*qdot(2) +
diff(dLdqdot(ii),qdot(2))*qddot(2)+...
                      diff(dLdqdot(ii),q(3))*qdot(3) +
diff(dLdqdot(ii),qdot(3))*qddot(3)+...
                      diff(dLdqdot(ii),q(4))*qdot(4) +
diff(dLdqdot(ii),qdot(4))*qddot(4)+...
                      diff(dLdqdot(ii),q(5))*qdot(5) +
diff(dLdqdot(ii),qdot(5))*qddot(5)+...
                      diff(dLdqdot(ii),q(6))*qdot(6) +
diff(dLdqdot(ii),qdot(6))*qddot(6);
    dLdq(ii) = diff(L,q(ii));
    EOM(ii) = simplify(ddt_dLdqdot(ii) - dLdq(ii) - T_ext(ii));
end
aux = solve(EOM,qddot);


%% Parte 3 Representación del sistema en espacio de estados lineal
% Convertir sistema a espacio de estados y linealizacion del sistema
syms  x1 x2 x3 x4 x5 x6 ...
      x7 x8 x9 x10 x11 x12 real

x2d = aux.xddot;
x2d = subs(x2d, {x y z  phi theta psi}, {x1, x3, x5, x7, x9, x11});
x2d = subs(x2d, {xdot ydot zdot phidot thetadot psidot},{x2, x4, x6, x8,
xx10, x12});

x4d = aux.yddot;
x4d = subs(x4d, {x y z  phi theta psi}, {x1, x3, x5, x7, x9, x11});
x4d = subs(x4d, {xdot ydot zdot phidot thetadot psidot},{x2, x4, x6, x8, x10,
x12});

x6d  = aux.zddot;
x6d = subs(x6d, {x y z  phi theta psi}, {x1, x3, x5, x7, x9, x11});
x6d = subs(x6d, {xdot ydot zdot phidot thetadot psidot},{x2, x4, x6, x8, x10,
x12});

x8d  = aux.phiddot;
x8d = subs(x8d, {x y z  phi theta psi}, {x1, x3, x5, x7, x9, x11});
x8d = subs(x8d, {xdot ydot zdot phidot thetadot psidot},{x2, x4, x6, x8, x10,
x12});

x10d = aux.thetaddot;
x10d = subs(x10d, {x y z  phi theta psi}, {x1, x3, x5, x7, x9, x11});
x10d = subs(x10d, {xdot ydot zdot phidot thetadot psidot},{x2, x4, x6, x8,
x10, x12});

x12d = aux.psiddot;
x12d = subs(x12d, {x y z  phi theta psi}, {x1, x3, x5, x7, x9, x11});
```

```
x12d = subs(x12d, {xdot ydot zdot phidot thetadot psidot},{x2, x4, x6, x8,
x10, x12});

X =     [x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12]';
u =     [u1,u2,u3,u4,u5,u6]';
Xdot  = [x2; x2d;   x4; x4d;   x6; x6d;   x8; x8d;   x10; x10d;   x12; x12d];

% Linealizacion del sistema
A=jacobian(Xdot,X);
B=jacobian(Xdot,u);

%Valor del la velocidad angular al cuadrado de los rotores en equilibrio
omega_e=m*g/6/K_T;   %rad^2/s^2

AL=double(subs(A,{x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,  v_wx v_wy v_wz
u1,u2,u3,u4,u5,u6}, .
    {0,0, 0,0, 0,0, 0,0, 0,0  0,0,  0,0,0,
omega_e,omega_e,omega_e,omega_e,omega_e,omega_e}));
BL=double(subs(B,{x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12, v_wx v_wy v_wz
u1,u2,u3,u4,u5,u6}, .
    {0,0, 0,0, 0,0, 0,0, 0,0  0,0,  0,0,0,
omega_e,omega_e,omega_e,omega_e,omega_e,omega_e}));

% Verificar si el sistema es controlable debe dar doce
rank(ctrb(AL,BL))
```

%%Parte 4 Controlador LQR

```
%% CONTROLADOR LQR

% Definición de matrices para el controlador LQR
% CL: Matriz de salida del sistema, aquí es una matriz identidad de tamaño
12x12
% DL: Matriz de transmisión directa del sistema, aquí es una matriz de ceros
de tamaño 12x6

CL = eye(12);
DL = zeros(size(CL, 1), size(BL, 2));

% Matrices de ponderación para el LQR
% QQ: Matriz de ponderación para el estado en el diseño del controlador LQR
%     Pondera la importancia de cada estado en la función de costo
% RR: Matriz de ponderación para las entradas de control en el diseño del
controlador LQR
%     Pondera la importancia de cada entrada de control en la función de
costo

QQ = diag([1e4 1 1e4 1 1e6 1 1e2 1 1e2 1 1e6 1]);

RR = 0.0001 * diag([1 1 1 1 1 1]);

% Cálculo de la matriz de ganancia del controlador LQR
% K: Ganancia del controlador LQR que minimiza la función de costo
K = lqr(AL, BL, QQ, RR);
```

```
%%Parte 5 Observador de Kalman
% Definición de matrices para el observador LQG

% C_LQG: Matriz de salida del sistema del observador LQG.
% Aquí selecciona los estados del sistema que se van a observar. En este
caso,
% está configurada para observar las posiciones y velocidades en los ejes x,
y, z,
% y los ángulos de orientación phi, theta y psi, así como sus derivadas.
C_LQG = [1 0  0 0  0 0  0 0  0 0  0 0;
         0 0  1 0  0 0  0 0  0 0  0 0;
         0 0  0 0  1 0  0 0  0 0  0 0;
         0 0  0 0  0 0  1 0  0 0  0 0;
         0 0  0 0  0 0  0 0  1 0  0 0;
         0 0  0 0  0 0  0 0  0 0  1 0];

% D_LQG: Matriz de transmisión directa del observador LQG.
% Esta matriz de ceros de tamaño 6x6 representa que no hay transmisión
directa
% de las entradas al sistema de salida en el observador.
D_LQG = zeros(size(C_LQG, 1), size(BL, 2));

% Matrices para el filtro de Kalman
% Q_kalman: Matriz de covarianza del proceso para el filtro de Kalman.
% Define el nivel de incertidumbre en el modelo del sistema,
% con valores distintos para cada estado del sistema.
Q_kalman = diag([0.1 0.1 0.1 4 4 4 0.1 0.1 0.1 1.0 1.0 1.0]);

% NP_acc: Nivel de ruido en las mediciones de aceleración.
% Unidades: m^2/s^4 (cuadrado de la aceleración)
NP_acc = 3.118e-6;

% NP_gir: Nivel de ruido en las mediciones de velocidad angular.
% Unidades: rad^2/s^2 (cuadrado de la velocidad angular)
NP_gir = 1.4926e-8;

% R_kalman: Matriz de covarianza de las mediciones para el filtro de Kalman.
% Define el nivel de ruido en las mediciones con valores escalados
% por un factor de 100 (10e2).
R_kalman = 10e2 * diag([NP_acc NP_acc NP_acc NP_gir NP_gir NP_gir]);

% Cálculo del filtro de Kalman
% KF: Ganancia del filtro de Kalman que minimiza el error de estimación
% mediante la función LQR, usando las matrices Q_kalman y R_kalman.
KF = (lqr(AL', C_LQG', Q_kalman, R_kalman))';

% Sistema del observador de Kalman
% sysKF: Sistema de estado del observador de Kalman, que incluye la dinámica
ajustada
% por el filtro de Kalman. Se modela como un sistema de estado con la matriz
de dinámica
% (AL - KF * C_LQG), la matriz de entrada [BL KF], y matrices de salida y
transmisión directa.
sysKF = ss(AL - KF * C_LQG, [BL KF], eye(12), 0 * [BL KF]);
```

```
%% Parte 6 Matrices para el controlador PID
% Matriz de mezclado del sistema para convertir de fuerzas y torques, a
velocidades de rotación al cuadrado en controlador PID
M =   [     (1/(6*K_T))*[ 1                     1                       1
1                       1                     1    ];
            (1/(2*K_T))*[ 1/(l_1*cos(alpha))    -1/(l_1*cos(alpha))     -1/l_2
-1/(l_1*cos(alpha))     1/(l_1*cos(alpha))     1/l_2];
            (1/(2*K_T))*[1/(l_1*sin(alpha))     1/(l_1*sin(alpha))      0
-1/(l_1*sin(alpha))     -1/(l_1*sin(alpha))    0    ];
            (1/(6*K_Q))*[ -1                    1                       -1
1                       -1                    1   ]]';

C_PID  =   [1 0  0 0  0 0  0 0  0 0  0 0 ;
            0 0  1 0  0 0  0 0  0 0  0 0 ;
            0 0  0 0  1 0  0 0  0 0  0 0 ;
            0 0  0 0  0 0  1 0  0 0  0 0 ;
            0 0  0 0  0 0  0 0  1 0  0 0 ;
            0 0  0 0  0 0  0 0  0 0  1 0 ];

D_PID = zeros(size(C_PID,1),size(BL,2));


%% Discretizar el sistema por el metodo de Tustin
sys_d = c2d(sysKF, T_s, 'tustin');

%convertir a Python
matrixAsPython(sys_d.A)
matrixAsPython(K)

% Discretize the continuous-time system using Tustin's method

%% funciones auxiliares
function matrixAsPython(K)
    matrix_str = '[';
    for i = 1:size(K, 1)
        matrix_str = [matrix_str '[' sprintf('%d, ', K(i, 1:end-1))
num2str(K(i, end)) '], '];
    end
    matrix_str = [matrix_str(1:end-2) ']'];

    % Print the matrix string for Python
    disp('Python NumPy Array (Copy and paste the following line into your
Python code):');
    disp(matrix_str);
end
```

Apéndice B. Archivo en Simulink para prueba del controlador PID

El archivo incluido para probar el controlador es llamado *Apendice_B_Controlador_PID*. Para ser utilizado se requiere primero correr el Apéndice A, se incluye la ruta de prueba en el archivo *recorrido.mat*.

Apéndice C. Archivo en Simulink para prueba del controlador LQG

El archivo incluido para probar el controlador es llamado *Apendice_C_Controlador_LQG*. Para ser utilizado se requiere primero correr el Apéndice A, se incluye la trayectoria de prueba en el archivo *recorrido.mat*.

Apéndice D. Archivo en Simulink para prueba del controlador LQG-Integral

El archivo incluido para probar el controlador es llamado *Apendice_D_Controlador_LQG_Integral*. Para ser utilizado se requiere primero correr el Apéndice A, se incluye la ruta de prueba en el archivo *recorrido.mat*.

Apéndice E. Paquete de ROS2 hexarrotor con control LQG-Integral

En este apéndice se incluye el paquete de ROS2 del hexarrotor en la carpeta *hexarrotor.zip*. Para utilizarlo primero se debe instalar ROS2 y *Gazebo classic* de preferencia en el sistema operativo Ubuntu. Este paquete se debe colocar luego de extraído en el espacio de trabajo de ROS2, posteriormente se debe abrir una terminal desde la carpeta y construir el paquete del hexarrotor con el comando *colcon build*, por último, la línea para correr la simulación desde el terminal es *ros2 launch hexarrotor launch_sim.launch.py*. El dron aparecerá dentro del cultivo simulado con el controlador LQG-Integral activo, el dron se moverá a la coordenada por defecto (0,0,1.5).

Apéndice F. Código para el entrenamiento del agente de aprendizaje por refuerzo

Para realizar el entrenamiento del dron se utiliza el código "*Apendice_F_Codigo_entrenamiento_agente_DDDQN.py*", antes de ejecutar el código se debe modificar la velocidad de la simulación en Gazebo por un factor de cuatro veces el tiempo real, o

modificar en el código la línea del código `self.real_time_factor = 4,` al valor de la velocidad de la simulación con respecto al tiempo deseado. Luego se procede a ejecutar el código para entrenar el agente. Se guardan copias del agente entrenado cada 20 episodios.

Apéndice G. Agente entrenado y código de prueba

En el archivo comprimido "*Apéndice_G Agente entrenado y código de prueba.zip*" se incluye el agente entrenado y el código para probar el entrenamiento "*Codigo_prueba_agente_entrenado.py"*, la prueba por defecto llevará al agente a cada una de las posiciones posibles ubicadas cada 15° una después de otra y muestra una gráfica de las rutas que sigue, también se genera archivo .csv de cada una de las rutas seguidas, donde se entrega las coordenadas, y si se encontró o no con el modo de seguridad activo para esa coordenada enviada. Al igual que con el código de entrenamiento del dron se debe modificar la velocidad de la simulación en Gazebo por un factor de cuatro veces el tiempo real, o modificar en el código la línea del código self.real_time_factor = 4, al valor de la velocidad de la simulación con respecto al tiempo deseado.