Universitat Autònoma de Barcelona

Entornos de aprendizaje por refuerzo para aplicaciones de tráfico aéreo con BlueSky-Gym

Memoria del Trabajo Fin de Grado en Gestión Aeronáutica

realizado por

Francisco José Muñoz

y dirigido por

Laura Calvet Liñan

Escuela de Ingeniería

Sabadell, junio de 2025

Modelo de certificado del director/a

La abajo firmante, Laura Calvet Liñan director/a del Trabajo de Fin de Grado, profesora de la Escuela de Ingeniería de la UAB,

CERTIFICA:

Que el trabajo al que corresponde la presente memoria ha sido realizado bajo su dirección por

Francisco José Muñoz Alsina

Y para que conste firma la presente en Sabadell, junio de 2025

Firmado: Laura Calvet Liñan

Índice

Términos y acrónimos utilizados

AI	Artificial Intelligence	Inteligencia artificial
API	Application Programming Interface	Interfaz de programación de aplicaciones
ATC	Air Traffic Control	Control del tráfico aéreo
ATM	Air Traffic Management	Gestión del tráfico aéreo
CWI	Centrum Wiskunde & Informatica	Instituto nacional de investigación en matemáticas e informática
DDPG	Deep Deterministic Policy Gradient	Algoritmo de RL para acciones continuas
DQN	Deep Q-Network	Algoritmo que combina Q-learning con redes neuronales
FAA	Federal Aviation Administration	Administración federal de aviación
GPU	Graphics Processing Unit	Unidad de procesamiento gráfico
GUI	Graphical User Interface	Interfaz gráfica de usuario
HRL	Hierarchical Reinforcement Learning	Aprendizaje por refuerzo jerárquico
IEEE	Institute of Electrical and Electronics Engineers	Instituto de ingenieros eléctricos y electrónicos
IRL	Inverse Reinforcement Learning	Aprendizaje por refuerzo inverso
MARL	Multi-Agent Reinforcement Learning	Aprendizaje multi-agente
MCTS	Monte Carlo Tree Search	Método de búsqueda para planificación en RL
ML	Machine Learning	Aprendizaje automático
PPO	Proximal Policy Optimization	Algoritmo de optimización de políticas en RL
RL	Reinforcement Learning	Aprendizaje por Refuerzo
SAC	Soft Actor-Critic	Algoritmo de RL que equilibra eficiencia y exploración
TD3	Twin Delayed Deep Deterministic Policy Gradient	Algoritmo de RL sucesor de DDPG
TU Delft	Delft University of Technology	Universidad Tecnológica de Delft
TPU	Tensor Processing Unit	Unidad de procesamiento de Google para IA
UAV	Unmanned Aerial Vehicle	Vehículo Aéreo no Tripulado
XAI	Explainable AI	IA Explicable

Agente (RL): Entidad que toma decisiones en un entorno de aprendizaje por refuerzo, interactuando con el entorno para maximizar recompensas.

Algoritmo Actor-Crítico: Método de RL que combina un "actor" (toma decisiones) y un "crítico" (evalúa decisiones) para mejorar políticas.

Batch Size: Número de muestras utilizadas en una iteración de entrenamiento de un modelo de RL.

Buffer de Reproducción: Memoria que almacena experiencias pasadas del agente para reutilizarlas durante el entrenamiento.

Coeficiente de Entropía (ent_coef): Parámetro que regula el balance entre exploración (probabilidad de probar acciones nuevas) y explotación (usar acciones conocidas).

Deep Q-Network (DQN): Algoritmo de RL que combina Q-learning con redes neuronales profundas para manejar entornos complejos.

Entorno (RL): Contexto simulado o real donde el agente interactúa y aprende.

Espacio de Acciones: Conjunto de todas las acciones posibles que un agente puede tomar en un entorno.

Espacio de Estados: Conjunto de todas las situaciones posibles en las que puede encontrarse un entorno.

Función de Valor: Estimación de la recompensa acumulada esperada desde un estado o acción específica.

Hiperparámetros: Parámetros configurables que controlan el proceso de entrenamiento (ej. tasa de aprendizaje).

Política (RL): Estrategia que define cómo el agente selecciona acciones en función del estado actual.

Q-learning: Algoritmo de RL basado en valores que aprende una política óptima mediante una tabla Q.

Recompensa Descontada: Método para ponderar recompensas futuras en RL, dando más peso a las recompensas inmediatas.

Red Neuronal Profunda: Modelo computacional inspirado en el cerebro humano, usado en RL para aproximar funciones complejas.

Stable-Baselines3: Biblioteca de Python que implementa algoritmos de RL estables y optimizados.

Tasa de Aprendizaje (learning_rate): Parámetro que determina cuánto ajusta el modelo sus pesos en cada iteración de entrenamiento.

Índice de figuras

Índice de tablas

1. Introducción

El aprendizaje por refuerzo (RL, por sus siglas en inglés) [1,2] se ha consolidado como una de las áreas más prometedoras de la inteligencia artificial, especialmente en aplicaciones que requieren toma de decisiones en tiempo real y adaptabilidad a entornos dinámicos. El RL consiste en entrenar a un agente a tomar decisiones óptimas mediante la interacción reiterada con entornos, recibiendo recompensas o penalizaciones según sus acciones. Su objetivo es maximizar una recompensa acumulada a lo largo del tiempo, utilizando estrategias de exploración y explotación para mejorar su desempeño en la tarea asignada. En el ámbito del control de tráfico aéreo (ATC) y la gestión del tráfico aéreo (ATM), el RL ofrece un potencial significativo para optimizar operaciones, mejorar la seguridad y aumentar la eficiencia en espacios aéreos cada vez más congestionados. Por ejemplo, avisando al controlador de posibles conflictos futuros (aviones al mismo nivel de altitud con una ruta donde se cruzan) o automatizando procesos como peticiones de "vuelo en ruta directa" comprobando si efectuar la maniobra supone riesgos en la seguridad. Sin embargo, la falta de entornos estandarizados y herramientas accesibles para probar y comparar algoritmos de RL ha sido un obstáculo para el avance de la investigación en este campo.

Aun estando todavía en desarrollo BlueSky-Gym [3] surge como una solución innovadora para poder afrontar este desafío. Basado en el simulador de tráfico aéreo de código abierto BlueSky [4,5], proporciona una plataforma estandarizada y flexible para la investigación y aplicación de algoritmos de RL en tareas relacionadas con el control de tráfico aéreo. BlueSky-Gym se basa en la popular API Gymnasium [6], lo que permite a los investigadores y desarrolladores utilizar una amplia gama de algoritmos de RL disponibles en bibliotecas. Además, ofrece una colección de entornos predefinidos que abarcan desde tareas básicas de control vertical y horizontal hasta escenarios más complejos, como la resolución de conflictos y la fusión de flujos de tráfico.

La importancia de BlueSky-Gym radica en su capacidad para simplificar y estandarizar el proceso de investigación en RL aplicado al tráfico aéreo. Al proporcionar entornos reproducibles y bien documentados, como los del entorno de la figura 1, donde el agente debe trazar la ruta más corta y realizarla en el menor tiempo posible hasta el nodo (circulo blanco) teniendo en cuenta y evitando los obstáculos estáticos del escenario, este escenario junto con otros facilita la comparación de distintos algoritmos y la validación de resultados, lo que permite el avance de la investigación en este campo.

Figura 1: Visualización utilizada para el entorno StaticObstacleEnv-v0 dentro de BlueSky-Gym.

Además, su enfoque de generación procedural asegura que los algoritmos no se adapten únicamente a escenarios específicos, promoviendo la generalización y robustez de las políticas aprendidas.

Este trabajo tiene como objetivo explorar las funcionalidades y aplicaciones de BlueSky-Gym en el contexto del control de tráfico aéreo. A través de una revisión exhaustiva de la literatura, un análisis detallado del paquete y la realización de simulaciones computacionales se busca demostrar cómo el paquete puede ser utilizado para entrenar y evaluar algoritmos de RL en tareas de ATC/ATM. Además, se propone la creación de un tutorial, con el fin de hacer esta herramienta accesible a una audiencia más amplia, incluyendo estudiantes, investigadores y profesionales del sector.

1.1. Contexto

El tráfico aéreo mundial ha experimentado una notable evolución en los últimos años, marcada por un crecimiento sostenido hasta 2019, una drástica caída en 2020 debido a la pandemia de COVID-19 y una recuperación gradual que culminó en niveles récord en 2024.

Actualmente, el tráfico aéreo mundial ha aumentado un 10,4% con respecto a 2023, superando en un 3,8% los niveles de 2019. Este crecimiento ha llevado al tráfico aéreo a superar los niveles pre-COVID, evidenciando una recuperación completa del sector. Este aumento también se ha visto reflejado en los aeropuertos españoles, donde se han alcanzado cifras históricas de pasajeros. Por ejemplo, el Aeropuerto de Valencia cerró 2024 con 10,8 millones de pasajeros, un 8,7% más que el año anterior.

Figura 2: Gráfica de número de operaciones anuales de aeropuertos españoles contabilizados por AENA[8].

Este crecimiento en el tráfico aéreo no solo responde a la recuperación de la demanda de pasajeros, sino también a una serie de factores clave que han impulsado el sector. Entre ellos, destacan la apertura de nuevas rutas internacionales, el aumento del turismo y la consolidación de aerolíneas de bajo costo, que han facilitado el acceso a los viajes aéreos para un mayor número de personas.

A nivel operativo, la recuperación del tráfico ha supuesto nuevos retos para la gestión del espacio aéreo. La congestión en aeropuertos y rutas de alto tráfico ha generado una creciente necesidad de optimizar la planificación y el control de vuelos.

1.2. Conceptos básicos

1.2.1. Origen de la Inteligencia Artificial

La inteligencia artificial (AI) es el campo que estudia cómo hacer que las computadoras imiten la inteligencia humana, lo que ha permitido avances increíbles en diversos sectores, desde la medicina hasta la aviación.

La idea de máquinas inteligentes no es nueva. Ya en la antigüedad, los filósofos imaginaban autómatas capaces de razonar. No obstante, no fue hasta el siglo XX cuando se dieron los primeros pasos. En la década de 1940, durante la Segunda Guerra Mundial, el matemático Alan Turing desarrolló una de las primeras computadoras y propuso la idea de que una máquina podía "pensar" si lograba imitar el comportamiento humano en una conversación. Este concepto dio lugar al Test de Turing [7], que aún hoy se usa para evaluar si una IA puede engañar a una persona haciéndole creer que está hablando con otro humano.

Posteriormente, en 1956, en la Conferencia de Dartmouth, John McCarthy acuñó el término "Inteligencia Artificial", marcando el inicio oficial de esta disciplina. Durante esta época, los científicos eran optimistas y creían que en poco tiempo las computadoras podrían alcanzar el nivel de inteligencia humana. Sin embargo, en las décadas de 1960 y 1970, aunque se desarrollaron algunos de los primeros programas de AI, como ELIZA, un chatbot rudimentario, las limitaciones tecnológicas llevaron a una pérdida de interés.

Figura 3: Conversación con el chatbot ELIZA.

En los años 80 y 90, gracias a la mejora de los procesadores y al desarrollo de redes neuronales artificiales, la AI volvió a cobrar importancia. Se crearon sistemas expertos capaces de resolver problemas complejos en medicina e industria. No obstante, fue en el siglo XXI cuando la AI experimentó un crecimiento explosivo, impulsado por el acceso a grandes volúmenes de datos (Big Data) y el aumento del poder computacional.

Durante este período, aparecieron asistentes virtuales como Siri y Alexa, sistemas de reconocimiento facial y vehículos autónomos como los de Tesla.

1.2.2. Tipos de Aprendizaje en Inteligencia Artificial

Para entender cómo funciona la AI, es fundamental conocer sus diferentes formas de aprendizaje. Estas no "nacen" sabiendo qué hacer, sino que deben ser entrenadas para tomar decisiones y mejorar con el tiempo. Existen tres principales tipos de aprendizaje:

1. Aprendizaje Supervisado: En este método, el modelo aprende a partir de datos etiquetados. Por ejemplo, si quieres enseñar a una AI a reconocer gatos en fotos, le muestras imágenes etiquetadas como "con gato" o "sin gato". Con suficiente entrenamiento, puede predecir si una nueva imagen contiene un gato. Este tipo de aprendizaje se aplica en reconocimiento facial, diagnóstico médico y predicción del clima.
2. Aprendizaje no Supervisado: Aquí, la AI analiza datos sin etiquetar para encontrar patrones y organizar la información. Un ejemplo es Spotify, que observa las canciones que escuchas y te recomienda nuevas basadas en tus gustos. Este método se utiliza en segmentación de clientes, detección de fraudes y sistemas de recomendación.

Figura 4: Ejemplo visual de aprendizaje supervisado (regresión lineal) y no supervisado (Clustering).

3. Aprendizaje por Refuerzo (Reinforcement Learning - RL): En este caso, la AI aprende por prueba y error. Un "agente" realiza acciones en un entorno y recibe recompensas o penalizaciones según sus decisiones. Por ejemplo, un robot que aprende a jugar ajedrez recibe puntos positivos por buenos movimientos y negativos por malos. Este enfoque se aplica en juegos, robótica y optimización.

1.2.3. BlueSky

BlueSky es un simulador de tráfico aéreo de código abierto diseñado para investigar y desarrollar soluciones innovadoras en la gestión del tráfico aéreo. BlueSky fue creado en 2015 por el Instituto Nacional de Investigación Matemática e Informática (CWI) de los Países Bajos, en colaboración con la Universidad Tecnológica de Delft (TU Delft). Este proyecto surgió como una iniciativa de código abierto para proporcionar una herramienta flexible y accesible para la investigación en gestión del tráfico aéreo. Este software permite recrear escenarios realistas de vuelo, simulando el comportamiento de aeronaves, controladores aéreos y otros elementos clave del espacio aéreo. Su flexibilidad y capacidad de personalización lo convierten en una herramienta invaluable para investigadores, desarrolladores y profesionales de la aviación que buscan optimizar la eficiencia y seguridad en los cielos.

Una de las aplicaciones más destacadas de BlueSky es su integración con sistemas de AI. En este contexto, surge BlueSky-Gym, una plataforma que combina el simulador con técnicas de RL para entrenar modelos de AI en la gestión del tráfico aéreo. BlueSky-Gym permite a los investigadores desarrollar y probar algoritmos que optimicen rutas, reduzcan demoras y mejoren la toma de decisiones en tiempo real, marcando un hito en la modernización de la aviación.

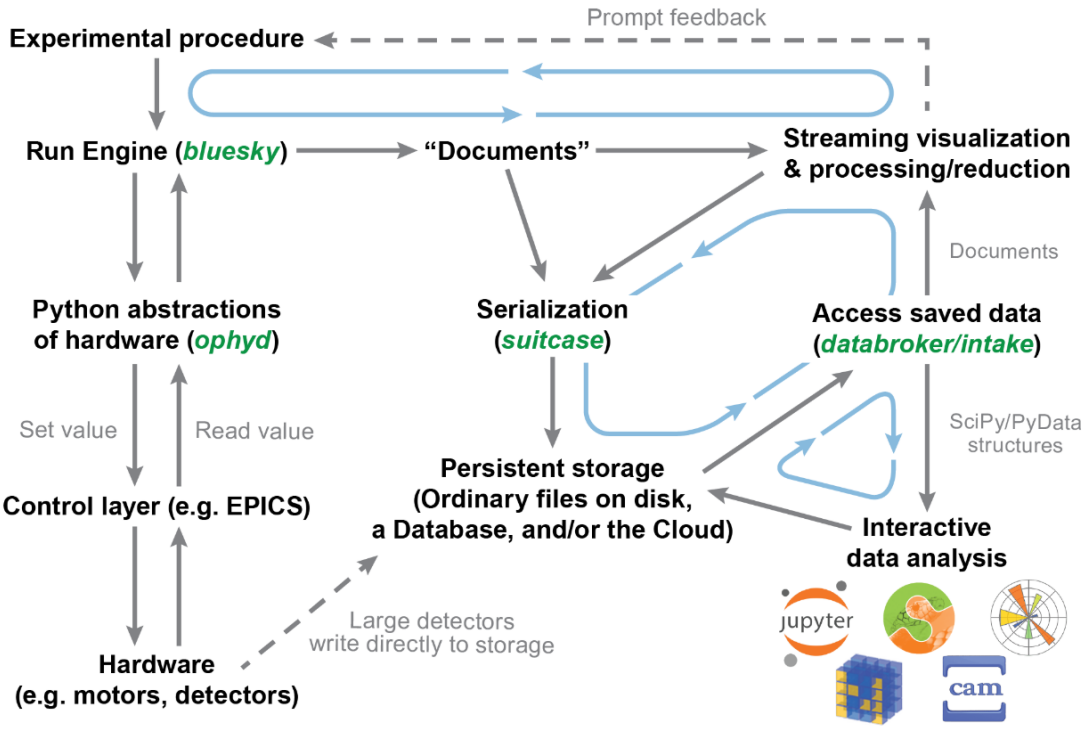


Figura 5: Flujo de trabajo de BlueSky para la adquisición, procesamiento y almacenamiento de datos experimentales. Incluye interacción con hardware, abstracción en Python, serialización, almacenamiento persistente y análisis de datos en entornos interactivos.

1.3. Motivación

A medida que avanzaba en mis estudios en el grado en Gestión Aeronáutica, me ha interesado mucho la evolución de las tecnologías especialmente de las AI y de su aplicación al ATC y de cómo estas pueden mejorar la seguridad y eficiencia en la aviación. En un mundo donde el número de vuelos sigue en aumento y donde la automatización desempeña un papel cada vez más relevante, considero que el RL representa una oportunidad clave para optimizar la gestión del espacio aéreo y la toma de decisiones.

El tráfico aéreo es un sistema altamente complejo, donde cada decisión puede afectar la seguridad. Tradicionalmente, las estrategias de control se basan en reglas predefinidas y en la experiencia de los controladores. Sin embargo, con el crecimiento del tráfico aéreo y la incorporación de drones y aeronaves autónomas, surgen desafíos que requieren soluciones más dinámicas y adaptativas.

El uso de AI permite desarrollar sistemas que pueden aprender y mejorar continuamente en la gestión del tráfico aéreo. Con BlueSky-Gym, se pueden entrenar modelos de RL en entornos simulados sin poner en riesgo operaciones reales.

Como estudiante, creo que este proyecto es una oportunidad única para combinar conocimientos de gestión del tráfico aéreo con herramientas tecnológicas avanzadas. Explorar cómo los algoritmos de AI pueden integrarse en ATC no solo es un tema innovador, sino que también tiene aplicaciones concretas en el sector aeronáutico, desde la automatización del control de tráfico hasta la optimización de la capacidad aeroportuaria.

En conclusión, este trabajo no solo representa una oportunidad para profundizar en un tema innovador dentro de la gestión aeronáutica, sino que también contribuye al desarrollo de nuevas estrategias para la optimización del tráfico aéreo mediante inteligencia artificial. A través de esta investigación, espero no solo aprender sobre BlueSky-Gym y el aprendizaje por refuerzo, sino también aportar soluciones prácticas que puedan optimizar el tráfico aéreo del futuro.

1.4. Objetivos

El objetivo general del trabajo es explorar el uso del aprendizaje por refuerzo en el control aéreo, con un enfoque específico en el estudio de las funcionalidades y aplicaciones potenciales del paquete BlueSky-Gym. Para alcanzar este objetivo general, se han definido los siguientes objetivos específicos, que guiarán el desarrollo del trabajo:

1. Revisar la literatura existente acerca de aprendizaje por refuerzo y el simulador BlueSky.
2. Estudiar el paquete BlueSky-Gym: características y aplicaciones.
3. Escribir un tutorial BlueSky-Gym para no expertos.
4. Llevar a cabo simulaciones o experimentos computacionales con varios escenarios y analizar resultados.
5. Estudiar limitaciones y posibles desarrollos del paquete.

1.5. Metodología

Para abordar los objetivos planteados en el trabajo, se pueden combinar diversas metodologías de investigación, en función de la naturaleza de cada objetivo. En primer lugar, para revisar la literatura existente sobre aprendizaje por refuerzo y el simulador BlueSky, se realizará una búsqueda sistemática en bases de datos académicos como IEEE Xplore y Google Scholar. Esta revisión permitirá identificar las aplicaciones más comunes del aprendizaje para reforzar el control de tráfico aéreo y analizar las tendencias, limitaciones y avances en el uso de BlueSky.

Posteriormente, el estudio del paquete BlueSky-Gym se llevará a cabo mediante un análisis de datos y una verificación técnica. Se examinará la documentación oficial, el código fuente y los ejemplos disponibles, con el fin de identificar sus características principales, las aplicaciones disponibles y prácticas reportadas en la literatura o en proyectos anteriores. Para ello, se utilizarán recursos como la documentación de BlueSky-Gym.

Una vez comprendido el funcionamiento del paquete, se diseñará un tutorial dirigido a usuarios sin experiencia en BlueSky-Gym. Este documento incluye instrucciones paso a paso para la instalación, configuración y uso del simulador, complementadas con ejemplos prácticos y capturas de pantalla. Para la elaboración del tutorial, se emplearán herramientas de documentación, así como grabadoras de pantalla.

En la siguiente fase del trabajo, se llevarán a cabo simulaciones y experimentos computacionales con diferentes escenarios dentro de BlueSky-Gym. Se evaluará el algoritmo de aprendizaje por refuerzo, en tareas específicas como resolución de conflictos y fusión de tráfico. Los datos recopilados incluyen métricas de rendimiento como la recompensa acumulada, el tiempo de convergencia y su eficiencia. Para el análisis de los resultados, se utilizarán herramientas como Python [9] y bibliotecas de aprendizaje.

Finalmente, se realizará un análisis crítico de BlueSky-Gym para identificar sus limitaciones y proponer posibles mejoras. Se evaluará la falta de soporte para entornos multiagente, la complejidad de los escenarios y la integración con otras herramientas. A partir de este diagnóstico, se sugerirán desarrollos futuros, como la ampliación a entornos más complejos o la mejora de la documentación. Este proceso se complementará con una revisión de la literatura existente.

A través de esta metodología estructurada, se espera lograr un análisis completo del paquete BlueSky-Gym, generar recursos accesibles para nuevos usuarios y aportar conocimientos valiosos para su mejora y aplicación en el control de tráfico aéreo.

1.6. Análisis de viabilidad

El proyecto presenta una viabilidad técnica sólida, respaldada por los recursos disponibles y las herramientas de software adecuadas. En primer lugar, se cuenta con BlueSky, un simulador de tráfico aéreo de código abierto que permite personalizaciones y es ampliamente utilizado en investigaciones académicas. Además, BlueSky-Gym, una extensión basada en Gymnasium, facilita la integración con algoritmos de RL, lo que agiliza el desarrollo de modelos avanzados. En cuanto al hardware, no se requiere un equipo de alto rendimiento, aunque contar con una GPU podría acelerar el entrenamiento de los modelos. Por otro lado, la documentación disponible en GitHub, junto con estudios previos y referencias académicas, proporciona una amplia base para el desarrollo del proyecto.

En cuanto a la viabilidad legal y ética, el uso de software de código abierto como BlueSky y BlueSky-Gym no presenta restricciones significativas, ya que ambos están disponibles bajo licencias que permiten su uso en investigación y desarrollo. En el caso de utilizar datos de tráfico aéreo real, como los proporcionados por Flightradar24[10] o eAIP [11], es fundamental verificar sus términos de empleo. Desde una perspectiva ética, el impacto en la seguridad aérea es un aspecto crítico. Un modelo mal entrenado podría generar recomendaciones erróneas.

La viabilidad económica del proyecto también es favorable. Los costos de software son bajos, ya que BlueSky y BlueSky-Gym son gratuitos y de código abierto. En cuanto al hardware, los costos pueden variar desde un nivel medio hasta uno más elevado, dependiendo de la capacidad de cómputo requerida.

1.7. Recursos necesarios

Recursos computacionales y de software

1. PC o portátil con capacidad suficiente para correr simulaciones.
2. Python 3.11 (lenguaje principal).
3. BlueSky-Gym (simulador)
4. Bibliotecas adicionales: NumPy 1.24, Matplotlib, Gym, etc.

Recursos Bibliográficos y Teóricos

1. Revisiones recientes sobre aprendizaje por refuerzo en simulaciones.
2. Aplicaciones del aprendizaje para refuerzo en tráfico aéreo o simulaciones similares.

1.8. Cronograma

Actividad	3-17 de febrero		18-4 de marzo		5-19 de marzo		20-3 de abril		4-18 de abril		19-3 de mayo		4-18 de mayo		19-2 de junio			3-17 de junio		18-2 de julio	
Revisar la literatura existente acerca de aprendizaje por refuerzo y el simulador Bluesky.																					
Estudiar el paquete Bluesky-gym: características y aplicaciones.																					
Llevar a cabo simulaciones o experimentos computacionales con escenarios básicos.																					
Iniciar la redacción del tutorial Bluesky-gym para no expertos.																					
Analizar resultados preliminares y realizar ajustes en las simulaciones.																					
Estudiar limitaciones y posibles desarrollos del paquete.																					
Refinar el tutorial Bluesky-gym con base en retroalimentación.																					
Ampliar simulaciones con escenarios más complejos.																					
Analizar resultados avanzados y documentar hallazgos.																					
Finalizar el análisis de limitaciones y propuestas de mejora del paquete.																					
Redacción del informe y ajustes finales en experimentos.																					
Revisión y edición del trabajo completo.																					

Tabla 1: Cronograma de actividades.

1.9. Riesgos

La siguiente tabla muestra las principales fuentes de riesgos identificadas, así como las estrategias a seguir para prevenir, controlar o responder a los posibles problemas antes de que afecten significativamente los objetivos.

Riesgo	**Impacto**	**Estrategia de mitigación**
Falta de experiencia en BlueSky-Gym	Medio	Realizar tutoriales y pruebas con ejemplos básicos antes de los experimentos principales.
Problemas de compatibilidad entre BlueSky-Gym, Python y bibliotecas	Medio	Asegurarse que se usa la versión compatible de los paquetes y revisar documentación y foros de usuarios.
Tiempo de entrenamiento excesivo	Alto	Realizar pruebas con modelos más simples antes de ejecutar experimentos complejos.
Falta de métricas claras para evaluar el rendimiento del modelo	Medio-Alto	Definir métricas clave como eficiencia del tráfico, número de conflictos resueltos y tiempos de convergencia.
Dificultad en la validación de los resultados con datos reales	Bajo	Comparar los resultados de la simulación con estudios previos o modelos heurísticos de control de tráfico aéreo.
Limitaciones de tiempo para cumplir con el cronograma	Alto	Establecer hitos intermedios y aplicar metodologías ágiles para ajustar el plan de trabajo según los avances.

Tabla 2: Tabla de riesgos.

Estructura de la memoria

Capítulo 1: Introducción

- Presentación del tema: Aprendizaje por refuerzo (RL) y su aplicación en la gestión del tráfico aéreo mediante BlueSky-Gym.

Capítulo 2: El aprendizaje por refuerzo (RL)

- Profundizar en el RL: Aprendizaje acumulado, generaciones, convergencia, ventajas y desventajas, con ejemplos e imágenes.

Capítulo 3: Estudio de BlueSky y BlueSky-Gym

- Estudio del paquete BlueSky y BlueSky-Gym: Características principales, funcionalidades y aplicaciones existentes.
- Descripción de los escenarios básicos seleccionados para las simulaciones iniciales.
- Análisis de resultados preliminares y ajustes en los parámetros de las simulaciones.

Capítulo 4: Tutorial de BlueSky-Gym

- Instalación y configuración del entorno. Descripción de los módulos principales: Simulador, entorno de RL y herramientas de visualización.
- Creación de un escenario simple: Definición de aeronaves, rutas y condiciones iniciales. Parámetros clave: Velocidad, altitud, separación entre aeronaves, etc.
- Ejecución de una simulación básica y visualización de resultados.

Capítulo 5: Experimentos computacionales con BlueSky-Gym

- Diseño de escenarios más complejos para evaluar el rendimiento de BlueSky-Gym en condiciones desafiantes.
- Ejecución de simulaciones avanzadas y recopilación de datos. Análisis de resultados avanzados: Eficiencia, limitaciones y posibles mejoras en el paquete.

Capítulo 6: Limitaciones y propuestas de mejora

- Identificación de las limitaciones actuales de BlueSky-Gym: Compatibilidad, escalabilidad y usabilidad.
- Propuestas de desarrollo futuro: Mejoras en la documentación, optimización del código y ampliación de funcionalidades.
- Discusión sobre cómo estas mejoras podrían impactar en la investigación y aplicaciones prácticas.

Capítulo 7: Conclusión

- Resumen de los hallazgos principales del trabajo.
- Reflexiones sobre la importancia de las AI y a herramientas como BlueSky-Gym.
- Perspectivas futuras: Posibles líneas de investigación y desarrollo.

2. El aprendizaje por refuerzo

2.1 Introducción

El aprendizaje por refuerzo es un tipo de aprendizaje automático en el que un agente aprende a tomar decisiones en un entorno interactivo mediante prueba y error. El agente recibe recompensas o castigos por sus acciones, y su objetivo es maximizar la recompensa total a lo largo del tiempo. A diferencia de otros enfoques de aprendizaje automático, como el aprendizaje supervisado, donde el agente aprende a partir de ejemplos etiquetados, o el aprendizaje no supervisado, que busca patrones en datos no estructurados, el RL se basa en la exploración y explotación de acciones para maximizar una recompensa acumulada a lo largo del tiempo [13]. Este enfoque lo convierte en una herramienta poderosa para resolver problemas secuenciales y de toma de decisiones en entornos dinámicos y complejos.

El aprendizaje por refuerzo tiene sus raíces en la psicología conductista, donde en 1855 Alexander Bain y posteriormente en 1898 Edward Thorndike estudiaron cómo los organismos aprenden a través de la interacción con su entorno, finalmente sentando las bases del aprendizaje por ensayo y error, e posteriormente en la década de 1927 Iván Pávlov contribuyó con sus estudios sobre el condicionamiento clásico donde explora cómo los animales pueden asociar estímulos para generar respuestas automáticas. En el ámbito de la inteligencia artificial, el RL se formalizó en la década de 1980, con contribuciones clave de investigadores como Richard Sutton y Andrew Barto, quienes desarrollaron las bases teóricas y prácticas del campo. Durante este periodo, se crearon algoritmos como el Q-learning, que marcaron un avance significativo en el sector de las AIs, esta técnica de aprendizaje por refuerzo utilizada en el aprendizaje automático ha permitido hoy en día realizar diagnósticos basado en imágenes en medicina igualando el nivel de los profesionales del sector [14]. Otro ejemplo, fue la combinación del RL con redes neuronales profundas, ejemplificada por el éxito de DeepMind en dominar videojuegos de Atari y el juego de Go [13].

Figura 6: Richard Sutton y Andrew Barto autores "Introducción al aprendizaje por refuerzo".

Uno de los aspectos más destacados del RL es su capacidad para aprender de manera autónoma, sin necesidad de supervisión explícita. El agente interactúa con el entorno, observa los resultados de sus acciones y ajusta su comportamiento para maximizar las recompensas futuras [13]. Este proceso de aprendizaje acumulado, donde el agente no solo busca recompensas inmediatas sino también a largo plazo, es fundamental en problemas como juegos, control de robots y optimización de recursos [15].

En los últimos años, el RL ha alcanzado hitos significativos gracias a la combinación con técnicas de aprendizaje profundo. Por ejemplo, el algoritmo Deep Q-Network (DQN) permitió a un agente alcanzar un rendimiento a nivel humano en juegos de Atari, utilizando redes neuronales profundas para aproximar la función de valor [15]. Otro avance notable fue AlphaGo Zero, que aprendió a jugar Go a un nivel superhumano sin datos de entrenamiento humanos, utilizando solo RL y búsqueda de árboles Monte Carlo [16]. Además, el algoritmo Deep Deterministic Policy Gradient (DDPG) extendió el RL a entornos con acciones continuas, como el control de robots [17].

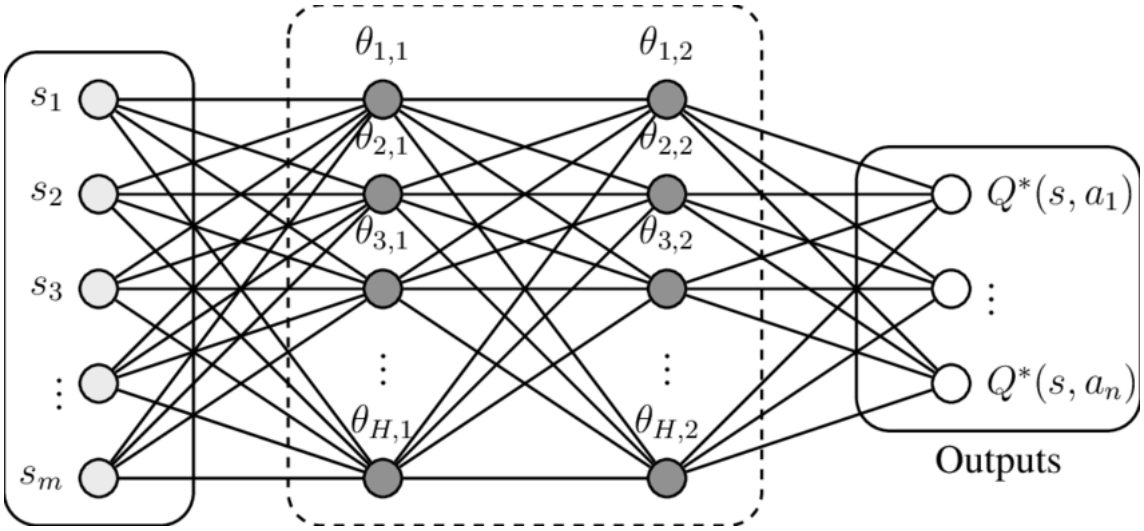


Figura 7: Estructura de la red neuronal utilizada para la Red de Aprendizaje por Refuerzo Profundo (Deep Q-learning Network).

2.2 Fundamentos Teóricos del Aprendizaje por Refuerzo

El aprendizaje por refuerzo (RL) se basa en un marco teórico bien definido que permite a un agente aprender a tomar decisiones óptimas mediante la interacción con un entorno. Este marco se compone de varios elementos clave y conceptos fundamentales que guían el proceso de aprendizaje.

2.2.1. Marco de Trabajo: Agente, Entorno, Estado, Acción y Recompensa

El RL se estructura en torno a la interacción entre un **agente** y un **entorno**. El agente es la entidad que toma decisiones, mientras que el entorno representa el mundo externo con el que el agente interactúa. En cada paso de tiempo, el agente observa el **estado** actual del entorno, selecciona una **acción** y recibe una **recompensa** como retroalimentación. El objetivo del agente es aprender una política que maximice la recompensa acumulada a lo largo del tiempo [13].

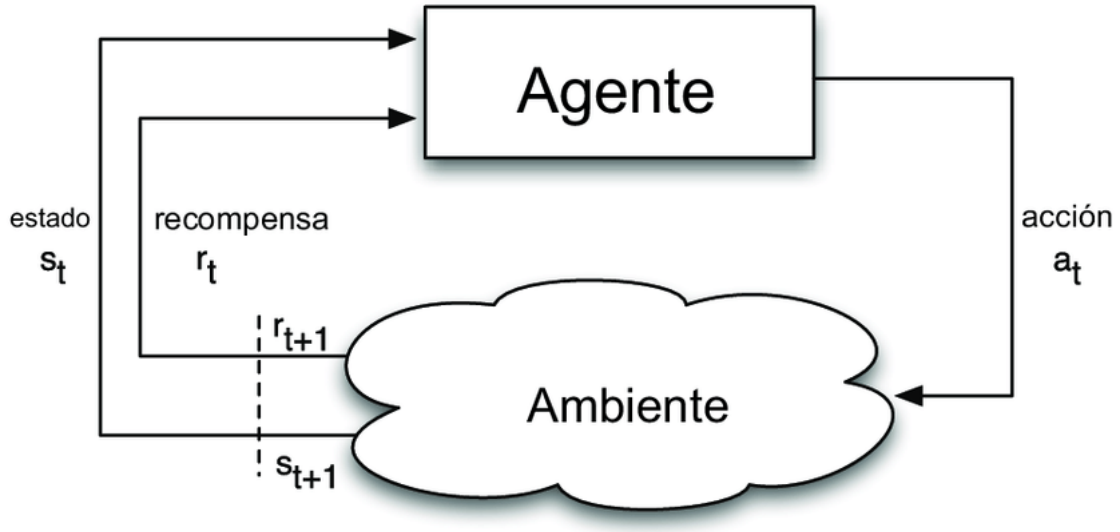


Figura 8: Diagrama simplificado de agente y entorno.

2.2.2. Conceptos Clave

- Política: La política es la estrategia que el agente utiliza para seleccionar acciones en función del estado actual. Puede ser determinista (siempre selecciona la misma acción para un estado dado) o estocástica (asigna probabilidades a las acciones posibles) [13].
- Función de Valor: La función de valor estima la recompensa acumulada que el agente puede esperar obtener a partir de un estado o acción. Existen dos tipos principales: la función de valor de estado (que evalúa cuán bueno es estar en un estado) y la función de valor de acción (que evalúa cuán buena es una acción en un estado dado) [13].
- Aprendizaje Acumulado y Recompensa Descontada: El RL se enfoca en maximizar la recompensa acumulada a largo plazo, no solo la recompensa inmediata. Para ello, se introduce un factor de descuento que pondera la importancia de las recompensas futuras. Este enfoque permite al agente priorizar acciones que generen beneficios sostenidos en el tiempo [2,13].

2.2.3. Exploración vs. Explotación

Uno de los dilemas centrales en el RL es el balance entre exploración y explotación. La exploración implica probar nuevas acciones para descubrir sus efectos, mientras que la explotación se refiere a utilizar acciones conocidas que han generado buenas recompensas. Un agente debe equilibrar ambos aspectos para aprender eficientemente: demasiada explotación puede llevar a soluciones subóptimas, mientras que demasiada exploración puede retrasar la convergencia a una política óptima [2,13].

2.3. Algoritmos Clásicos de Aprendizaje por Refuerzo

El aprendizaje por refuerzo (RL) cuenta con una variedad de algoritmos que se clasifican en tres categorías principales: métodos basados en valor, métodos basados en políticas y métodos basados en modelos. Cada uno de estos enfoques tiene sus propias características y aplicaciones, lo que los hace adecuados para diferentes tipos de problemas.

2.3.1. Métodos Basados en Valor

Estos métodos buscan calcular qué tan bueno es un estado o una acción en un problema de toma de decisiones. Para hacerlo, intentan predecir la recompensa total que se podría obtener en el futuro si se sigue una estrategia determinada. Dos de los algoritmos más representativos son:

Q-learning: Este algoritmo aprende a estimar qué tan buena es cada acción en un determinado estado. Para ello, utiliza una función llamada "función Q", que asigna un valor a cada acción según las recompensas futuras esperadas. Q-learning es un método off-policy, lo que significa que aprende “el camino” más optimo independientemente de las acciones que el agente esté tomando durante el aprendizaje [13].

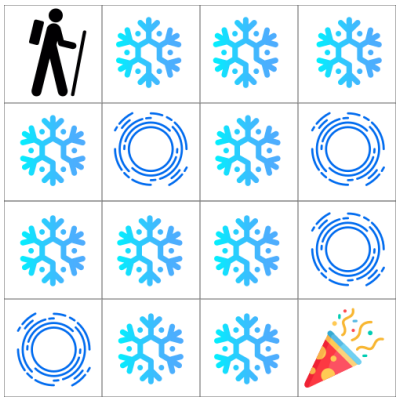


Figura 9: Ejemplo visual de una Q-Table, donde el algoritmo debería alcanzar el objetivo utilizando el camino más corto.

Deep Q-Networks (DQN): DQN es una mejora del algoritmo Q-learning que usa redes neuronales para tomar decisiones en entornos complejos, como videojuegos. En lugar de almacenar todos los valores de las acciones en una tabla, la red neuronal aprende a estimarlos, lo que permite manejar situaciones con muchas posibilidades. Además, usa trucos como guardar experiencias pasadas y tener una red de referencia para hacer el aprendizaje más estable y efectivo [15]. DQN se ha utilizado en juegos de estrategia en tiempo real, como StarCraft II, para entrenar agentes que aprenden a tomar decisiones complejas en entornos dinámicos.

2.3.2. Métodos Basados en Políticas

Estos métodos enseñan directamente al agente cómo tomar decisiones. En lugar de calcular el valor de cada acción, ajustan la forma en que el agente elige sus acciones para obtener la mayor recompensa posible. Dos algoritmos destacados son:

Policy Gradient: Es un método que enseña al agente a mejorar sus decisiones ajustando directamente la forma en que elige sus acciones. En lugar de calcular valores para cada acción, el agente aprende probando diferentes estrategias y ajustándolas para maximizar la recompensa. Aunque este método es flexible y funciona bien en situaciones donde las acciones son aleatorias (estocásticas), puede necesitar muchas pruebas para aprender correctamente [13]. Por ejemplo, un robot debe aprender a lanzar una pelota a un objetivo. En lugar de calcular cuál es el mejor ángulo y fuerza exactos en cada intento, el robot prueba diferentes formas de lanzar la pelota, recibe una puntuación por cada intento y ajusta su estrategia hasta mejorar su precisión.

Deep Deterministic Policy Gradient (DDPG): DDPG es una extensión de Policy Gradient que combina redes neuronales profundas con métodos basados en políticas para manejar entornos con acciones continuas. Este algoritmo es especialmente útil en aplicaciones de control, como la robótica, donde las acciones suelen ser continuas y de alta dimensionalidad. DDPG utiliza un enfoque actor-crítico, donde el actor decide que acción tomar y el crítico que evalúa las acciones tomadas [17]. En este caso imagina un dron que debe aprender a volar suavemente para aterrizar en una plataforma. DDPG permite que el dron ajuste gradualmente la potencia de sus motores en lugar de elegir entre opciones fijas como "subir" o "bajar". Así, puede realizar movimientos más precisos y eficientes.

2.3.3. Métodos Basados en Modelos

Finalmente, estos métodos utilizan una representación del entorno para planificar antes de actuar. En lugar de aprender solo a partir de la experiencia directa, el agente simula diferentes escenarios para predecir los resultados de sus acciones y tomar mejores decisiones. Un ejemplo destacado es:

Búsqueda de Árboles Monte Carlo (MCTS): MCTS es un método que ayuda a tomar decisiones explorando diferentes opciones de manera inteligente. En lugar de analizar todas las posibilidades (lo que puede ser imposible en juegos complejos), prueba algunas opciones al azar, construye un árbol con los mejores movimientos y se enfoca en los más prometedores [16].

2.4. Software y hardware que se utiliza actualmente en las aplicaciones RL

El desarrollo de aplicaciones de aprendizaje por refuerzo (RL) como BlueSky-Gym se ve impulsado por la sinergia software-hardware necesario para realizar simulaciones eficientes y precisas. En el lado del software, herramientas como TensorFlow o PyTorch constituyen la base sobre la que implementar las redes neuronales profundas que resulta esencial en algoritmos como DQN o DDPG que el propio BlueSky-Gym aplica para permitir entrenar a los agentes de gestión del tráfico aéreo. Junto a estas bibliotecas concretas, también se encuentran bibliotecas más especializadas (Stable-Baselines3 o Ray RLlib) que simplifican la experimentación con algoritmos clásicos (PPO o SAC) y que permiten escalar el entrenamiento a entornos distribuidos, lo que resulta fundamental para escenificar entornos complejos y realistas (con aeropuertos que tienen cientos de vuelos simultáneos). Por otra parte, entornos como OpenAI Gym son los que permitieron estandarizar la evaluación de modelos, mientras que simuladores físicos (MuJoCo) o 3D (Unity ML-Agents) inspiraron el diseño de los entornos realistas en BlueSky-Gym y su utilización en variables extra como la meteorológica o la existencia de rutas en función de la situación.

Con respecto al hardware, las exigencias computacionales de BlueSky-Gym requieren unos recursos potentes. Por ejemplo, las GPUs (NVIDIA Tesla, RTX) aceleran el entrenamiento de modelos profundos mediante la ejecución en paralelo de operaciones matriciales, lo que permite reducir el tiempo requerido para simular miles de episodios. Para proyectos extensos, como el optimizador de tráfico aéreo global, se pueden aprovechar las TPUs de Google, que destacan a nivel de rendimiento. Asimismo, la computación distribuida (clusters y herramientas como Ray) permite dividir el trabajo entre varios nodos, lo que resulta un enfoque determinante para el paso de BlueSky-Gym hacia un simulador profesional. Además, el hardware específico (placas NVIDIA Jetson) se podría incorporar en las fases de producción para realizar inferencias en sistemas físicos (torres de control automatizadas).

Para un mayor entendimiento se pasará a explicar los dos mayores componentes que intervienen en una simulación:

2.4.1. Software (Las "Herramientas Digitales")

Sin el software, BlueSky-Gym no tendría cerebro: no podría aprender ni simular.

- Frameworks de Aprendizaje Profundo y RL:
    - TensorFlow y PyTorch: Son como los cuadernos de notas de la AI, como los cuadros de una novela, permitiendo programar redes neuronales (modelos matemáticos inspirados en el cerebro humano) para que la máquina aprenda de sus errores. Es decir, si BlueSky-Gym hace un error dando las órdenes para dirigir un avión, estos programas se hacen cargo de realizar correcciones en sus cálculos para hacerlo mejor la próxima vez.

    - Stable-Baselines3: Es como el libro de recetas con algoritmos predefinidos y que si quieres enseñar a BlueSky-Gym a tomar decisiones, aquí encuentras métodos que han sido probados, como el PPO (simula un entrenador que da premios por buenas acciones) o el DQN (que aprende por prueba y error).
    - OpenAI Gym: Un entorno de desarrollo que ofrece una variedad de entornos simulados para probar y evaluar algoritmos de RL, desde juegos clásicos hasta problemas de control, pero los de BlueSky-Gym se usan para simular aviones, rutas y emergencias.
    - Ray RLlib: Hay que pensar en esto como en un equipo de trabajadores, de tal forma que si BlueSky-Gym necesita ejecutar varios aviones, entonces el RLlib divide el trabajo en varias máquinas para así hacerlo lo más rápido posible.
- Herramientas de Simulación:
    - Unity ML-Agents: Es como los estudios de cine que crean mundos virtuales. Unity se usa para simulaciones en 3D (por ejemplo, un aeropuerto con gráficos realistas), y MuJoCo es para simular las leyes físicas (como el viento afectando a un avión).

2.4.2. Hardware (Los "Motores" que Hacen el Trabajo Pesado)

Sin el hardware, sería como un cerebro sin cuerpo: sabría qué hacer, pero no podría hacerlo rápido.

- Unidades de Procesamiento Gráfico (GPUs): Son tarjetas que ejecutan cálculos de forma muy rápida. En BlueSky-Gym, una mejor GPU ayudará a acelerar el entrenamiento de modo que simular cientos de aterrizajes lleve horas en vez de días.
- Unidades de Procesamiento Tensor (TPUs): Desarrolladas por Google, las TPUs están optimizadas para operaciones de aprendizaje profundo y son utilizadas en aplicaciones de RL que requieren un alto rendimiento computacional, como el entrenamiento de modelos a gran escala. Podríamos pensar en ellos como motores de Fórmula 1. No son para uso doméstico, pero en el caso de proyectos masivos (como el que trata de simular el tráfico aéreo de todo un país) son extremadamente rápidos.
- Computación Distribuida: Sería como si quisiéramos resolver un enorme rompecabezas al contratar a un pool enorme de personas que lo hicieran. En vez de usar una sola computadora instalada en una oficina, BlueSky-Gym podría usar decenas conectadas en red (usando herramientas como Ray) para hacer el trabajo repartiendo las tareas
- Hardware Especializado: Es hardware que tiene una potencia enorme en formatos muy pequeños: computadoras que usan drones o robots (en el caso que BlueSky-Gym quisiera controlar aviones). Esta opción permitiría poder ejecutar el modelo entrenado en tiempo real fuera de la oficina, por ejemplo, a un avión que vuela por la costa incluso sin internet disponible.

Figura 10: Ejemplo de elementos de hardware y de software.

2.5. Aplicaciones del Aprendizaje por Refuerzo

2.5.1. Juegos

Los juegos han sido un campo de pruebas ideal para el RL debido a su naturaleza estructurada y la posibilidad de definir recompensas claras. Dos ejemplos emblemáticos son:

Aplicación de DQN en juegos de Atari: El algoritmo Deep Q-Network (DQN) revolucionó el campo del RL al combinar redes neuronales profundas con Q-learning. Este enfoque permitió a un agente aprender a jugar directamente a partir de imágenes de pantalla, alcanzando un rendimiento a nivel humano en varios juegos clásicos de Atari, como Breakout y Space Invaders. DQN demostró que el RL puede manejar entornos de alta dimensionalidad y aprender políticas efectivas sin supervisión explícita [15].

AlphaGo Zero y su dominio del juego de Go: AlphaGo Zero es un hito en la historia del RL. A diferencia de su predecesor, AlphaGo, que utilizaba datos de partidas humanas, AlphaGo Zero aprendió a jugar Go desde cero, utilizando únicamente RL y búsqueda de árboles Monte Carlo (MCTS). Este sistema no solo superó a los mejores jugadores humanos, sino que también descubrió estrategias novedosas que revolucionaron la comprensión del juego [16].

2.5.2. Robótica

El RL ha encontrado aplicaciones significativas en robótica, donde la capacidad de aprender en entornos dinámicos y complejos es crucial. Un ejemplo destacado es:

Control de robots con DDPG: El algoritmo Deep Deterministic Policy Gradient (DDPG) ha sido utilizado para controlar robots en tareas que requieren precisión y coordinación, como la manipulación de objetos y el movimiento autónomo. DDPG es especialmente adecuado para entornos con acciones continuas, lo que lo convierte en una opción ideal para aplicaciones de control en robótica. Este enfoque ha permitido a los robots aprender tareas complejas de manera autónoma, sin necesidad de programación explícita [17].

2.5.3. Otras Aplicaciones

Investigaciones en el ámbito del control del tráfico aéreo han explorado el uso de algoritmos de Reinforcement Learning (RL) para optimizar la gestión del espacio aéreo en tiempo real. En particular, el proyecto Dynamic Airspace Configuration (DAC) de la NASA ha demostrado mediante simulaciones sobre el espacio aéreo de Kansas City que ciertas técnicas de reconfiguración dinámica pueden reducir los retrasos promedio por vuelo en escenarios de alta congestión en hasta un 15–20 % [18]. Estas simulaciones consideraron múltiples variables como restricciones operacionales, condiciones meteorológicas y distribución del tráfico aéreo. Aunque en los documentos públicos del proyecto DAC no se mencionan algoritmos específicos de aprendizaje profundo, otras investigaciones han demostrado el potencial del enfoque Deep Deterministic Policy Gradient (DDPG) en entornos aeronáuticos. Este algoritmo de aprendizaje por refuerzo profundo desarrollado por Google DeepMind en 2015. Está diseñado para resolver tareas de control continuo, donde las acciones no son discretas (como girar a la izquierda o a la derecha), sino continuas (como ajustar gradualmente el ángulo de vuelo, velocidad o altitud). Gracias a este algoritmo, una tesis de la Universidad de Cranfield desarrolló un sistema de control de vuelo autónomo basado en DDPG, capaz de operar en condiciones de alta complejidad dinámica con seis grados de libertad [19].

Microsoft Flight Simulator incorpora avanzadas técnicas de inteligencia artificial generativa, apoyadas en tecnologías de Azure y Blackshark.ai, que permiten reconstruir de forma precisa y detallada el entorno global mediante el uso de datos satelitales y fotogrametría [20,21]. Este sistema crea más de 1,500 millones de edificaciones, carreteras, vegetación y otras estructuras, incluso rellenando automáticamente zonas con datos escasos, como aeropuertos secundarios o extremos de paisaje, y reproduce efectos meteorológicos en tiempo real [20]. Además, integra tráfico aéreo real basado en datos “live” de transpondedores, lo que contribuye a una simulación altamente inmersiva y realista [22]. Aunque no se ha confirmado el uso de algoritmos de aprendizaje por refuerzo puro como Proximal Policy Optimization (PPO) en el sistema de tráfico, investigaciones académicas han aplicado PPO para el control autónomo de tráfico aéreo simulado, demostrando que estas técnicas permiten a las aeronaves tomar decisiones autónomas, como cambiar rutas ante tormentas o mantener la separación entre aviones [1].

Investigadores del MIT Lincoln Laboratory han desarrollado avanzados sistemas de evitación de colisiones basados en aprendizaje profundo, tales como ACAS-X y su variante para drones ACAS Xu, diseñados para operar en entornos mixtos de tráfico tripulado y no tripulado [23,24]. Estos sistemas han sido evaluados con millones de simulaciones en entornos de encuentro entre vehículos aéreos, utilizando técnicas de Monte Carlo en simulaciones rápidas, evaluando maniobras evasivas como cambios de altitud y trayectoria para aumentar la seguridad [25,26]. Los desarrollos de ACAS Xu han sido distinguidos por su innovación y están en proceso de ser integrados en sistemas certificados para drones, lo que representa un paso clave hacia su eventual uso conjunto con agencias como la FAA [24,27].

2.6. Desafíos y Limitaciones del Aprendizaje por Refuerzo

A pesar de su potencial y versatilidad, el RL enfrenta varios desafíos y limitaciones que dificultan su aplicación en problemas complejos y del mundo real. Estos desafíos están relacionados con aspectos computacionales, de exploración y de estabilidad en el aprendizaje.

2.6.1. Alto Costo Computacional

Uno de los principales obstáculos del RL es su alto costo computacional. Los algoritmos de RL requieren una gran cantidad de interacciones con el entorno para aprender políticas efectivas, lo que puede ser extremadamente costoso en términos de tiempo y recursos. Por ejemplo, en el caso de Deep Q-Networks (DQN), se necesitaron millones de pasos de entrenamiento para alcanzar un rendimiento a nivel humano en juegos de Atari. Además, muchos algoritmos de RL, especialmente aquellos que combinan RL con redes neuronales profundas, requieren grandes cantidades de datos y potencia de cálculo, lo que limita su aplicabilidad en entornos donde los recursos son escasos [2].

2.6.2. Dificultad en la Exploración en Entornos Grandes

La exploración eficiente es un desafío fundamental en el RL, especialmente en entornos grandes o con recompensas escasas. En estos casos, el agente puede tener dificultades para encontrar acciones que generen recompensas significativas, lo que retrasa el aprendizaje. El dilema de exploración vs. explotación es particularmente crítico: si el agente explota demasiado acciones conocidas, puede quedar atrapado en soluciones subóptimas; si explora demasiado, puede tardar mucho tiempo en converger a una política efectiva [2,13]. Este problema se agrava en entornos con espacios de estado y acción de alta dimensionalidad, donde la exploración aleatoria no es viable.

2.6.3. Inestabilidad en la Convergencia de Algoritmos

La inestabilidad en la convergencia es otro desafío importante en el RL. Muchos algoritmos, especialmente aquellos que combinan RL con redes neuronales profundas, pueden ser inestables durante el entrenamiento. Por ejemplo, en el caso de Deep Deterministic Policy Gradient (DDPG), pequeños cambios en los hiperparámetros o en la inicialización de los pesos de la red pueden llevar a resultados muy diferentes, lo que dificulta la reproducibilidad y la confiabilidad del aprendizaje [17]. Además, la convergencia a una política óptima no está garantizada en todos los casos, especialmente en entornos no estacionarios o con recompensas ruidosas [2].

2.7. Ventajas y Desventajas del Aprendizaje por Refuerzo

El RL es un enfoque poderoso y versátil en el campo de la inteligencia artificial, pero como cualquier técnica, tiene sus ventajas y desventajas. A continuación, se describen los aspectos más destacados de ambas.

2.7.1. Ventajas

Aprendizaje Autónomo y Adaptabilidad a Entornos Dinámicos: Una de las principales ventajas del RL es su capacidad para aprender de manera autónoma, sin necesidad de supervisión explícita. El agente interactúa con el entorno y ajusta su comportamiento en función de las recompensas obtenidas, lo que lo hace ideal para entornos dinámicos y cambiantes donde las reglas no están predefinidas [2,13].

Optimización a Largo Plazo en Problemas Secuenciales: El RL está diseñado para maximizar la recompensa acumulada a lo largo del tiempo, lo que lo hace especialmente útil en problemas secuenciales y de toma de decisiones a largo plazo. Esto contrasta con otros enfoques que se centran en optimizar recompensas inmediatas. Ejemplos notables incluyen el dominio de juegos como Go y Atari, donde el agente debe planificar varias jugadas adelante para alcanzar el éxito [14,15].

2.7.2. Desventajas

Dificultad para Escalar a Problemas de Alta Dimensionalidad: Aunque el RL ha demostrado su eficacia en problemas con espacios de estado y acción discretos o de baja dimensionalidad, escalar a entornos de alta dimensionalidad sigue siendo un desafío. Por ejemplo, en aplicaciones de control de robots o procesamiento de imágenes, el alto costo computacional y la complejidad del espacio de búsqueda pueden dificultar el aprendizaje [2,16].

Dependencia de la Calidad de las Recompensas y la Exploración: El rendimiento del RL depende en gran medida de la definición adecuada de las recompensas y de una exploración eficiente. Si las recompensas no están bien diseñadas, el agente puede aprender políticas subóptimas o incluso contraproducentes. Además, el dilema de exploración vs. explotación puede dificultar el aprendizaje en entornos donde las recompensas son escasas o difíciles de obtener [2,13].

3. Estudio de BlueSky-Gym y BlueSky

3.1. Introducción

El análisis de herramientas de simulación en el campo aeronáutico es esencial para impulsar el desarrollo de ingeniería de sistemas que se utilizan en el control de tráfico aéreo y la gestión de flotas. En este sentido, BlueSky y su extensión BlueSky-Gym se presentan como unas plataformas de simulación de referencia en el campo de la simulación para entornos aeronáuticos, así como para la automatización y la optimización de tareas complejas, mediante la integración de algoritmos de aprendizaje por refuerzo (RL). Esta finalidad de este capítulo es presentar las principales características, las funcionalidades y las aplicaciones que tienen estas herramientas, así como dar a conocer los resultados preliminares obtenidos en simulaciones básicas con escenarios escogidos.

BlueSky es un simulador específico de tráfico aéreo de código abierto que ofrece un entorno realista y altamente configurable para simular operaciones aéreas; en cambio, BlueSky-Gym es una interfaz de software que permite la integración de BlueSky con el ecosistema OpenAI Gym; lo que permite implementar y evaluar algoritmos RL en entornos aeronáuticos. Estas herramientas han sido empleadas en trabajos de investigación recientes para abordar problemas como la gestión del tráfico aéreo, la optimización de rutas y la minimización de colisiones, demostrando así su versatilidad y su propio potencial en el ámbito de la inteligencia artificial en la aviación.

En este apartado, se hará una presentación detallada de las características y funcionalidades de BlueSky y BlueSky-Gym, de los escenarios básicos escogidos para las simulaciones iniciales, así como de los resultados preliminares analizados. Del mismo modo, se discutirán ajustes confeccionados en los parámetros de las simulaciones para su optimización. Este estudio puede considerarse como una base para los estudios futuros que empleen estas herramientas en problemas más complejos y escalables.

3.2. Estudio del Paquete BlueSky-Gym y BlueSky

En este apartado, se lleva a cabo un estudio en profundidad de las herramientas BlueSky y BlueSky-Gym donde se presentan sus principales características, funciones y aplicaciones concretas dentro del ámbito de la investigación y de la simulación aeronáutica. Estas herramientas han sido diseñadas para proporcionar un entorno de simulación adecuado y muy configurable que haga posible la conexión con determinados algoritmos de aprendizaje por refuerzo (RL) así como la simulación de situaciones complejas de tráfico aéreo.

3.2.1. Principales características

BlueSky es un simulador de tráfico aéreo Open Source que destaca por su flexibilidad y su capacidad para simular operaciones aéreas en tiempo real. Las principales características de BlueSky son:

- Simulación en tiempo real: BlueSky es capaz de simular el comportamiento de aeronaves, controladores aéreos y otros elementos del espacio aéreo con un elevado grado de precisión.
- Interfaz gráfica y API: Proporciona una GUI intuitiva para la visualización de simulaciones y una API robusta para la automatización de tareas y la integración con otras herramientas.
- Escalabilidad: Puede gestionar desde escenarios simples con pocas aeronaves hasta entornos complejos con cientos de vuelos simultáneos.

Por otro lado, BlueSky-Gym es una extensión de BlueSky que conecta el simulador con el ecosistema OpenAI Gym, facilitando así la implementación y evaluación de algoritmos de RL. Sus principales características son:

- Compatibilidad con OpenAI Gym: Proporciona una interfaz estándar para definir los entornos de RL con soporte para que los investigadores aplicen algoritmos ya implementados y comparen resultados de forma consistente (Sun et al., 2020).
- Configuración flexible: Se pueden definir escenarios, recompensas y métricas de evaluación que se adapten a los objetivos de la investigación.

3.2.2. Funcionalidades

Las funcionalidades de BlueSky y BlueSky-Gym las hacen herramientas muy adecuadas para la investigación en control del tráfico aéreo y la gestión de flotas. Las funcionalidades más destacadas son las siguientes:

- Modelado de aeronaves: Incluir modelos de todo tipo de aeronaves comerciales o privadas, definiendo parámetros tales como la velocidad, la altitud o la ruta.
- Gestión del espacio aéreo: Permitir simular la interacción entre aeronaves y controladores aéreos, incluyendo gestión de colisiones, desvíos, retrasos, etc.
- Integración con RL: BlueSky-Gym proporciona funciones para definir recompensas, estados y acciones, lo que facilita el entrenamiento de agentes para RL en escenarios de aviones.

3.2.3. Algoritmos

El paquete BlueSky-Gym integra cuatro distintos algoritmos que podemos utilizar al momento de realizar los distintos entrenamientos, siendo estos DDPG, TD3, PPO y SAC.

- DDPG (Deep Deterministic Policy Gradient): Es el más antiguo de los algoritmos disponibles y predecesor de estos, se puede considerar una extensión del DQN (Deep Q-Networks) pero aplicado a espacios de acciones continuos. Entre sus principales características se pueden considerar la utilización de dos redes neuronales (Actor-Critico), es decir el actor toma una acción y el critico determina que tan buena ha sido esa acción, mientras que es un algoritmo Off-Policy que utiliza un replay buffer, una memoria que almacena transiciones pasadas y de esta forma el algoritmo puede aprender experiencias antiguas. Finalmente, su política es determinista donde el actor produce una única acción para un estado dado, en lugar de una distribución de probabilidad sobre las acciones.
- TD3 (Twin Delayed Deep Deterministic Policy Gradient): TD3 es el sucesor directo de DDPG y fue diseñado para solucionar uno de sus mayores problemas relacionados con la sobreestimación del valor Q. Es decir, el algoritmo DDPG suele ser “optimista” sobre las recompensas futuras, lo que ralentiza el aprendizaje y da pie a soluciones subóptimas, para paliar esto el TD3 añadió tres innovaciones claves:
    - Críticos Gemelos (Twin Critics): Se entrenan dos redes de críticos en lugar de una.
    - Actualizaciones de Política Retrasadas (Delayed Policy Updates): El actor y las redes "objetivo" (target networks) se actualizan con menos frecuencia que el crítico.
    - Suavizado del Ruido en el Objetivo (Target Noise Smoothing): Suaviza el aprendizaje lo que evita picos y hace el aprendizaje más robusto.
- PPO (Proximal Policy Optimization): PPO es un algoritmo on-policy de tipo actor-crítico. Es conocido por su robustez, debido a que a diferencia de los demás algoritmos realiza actualizaciones de sus políticas de manera conservadora limitando los cambios posibles entre entrenamientos, a su vez elimina el replay buffer, lo que significa que después de cada interacción descarta las experiencias y usa unas nuevas. Finalmente, su política es estocástica, lo que significa que, para un estado dado, produce una distribución de probabilidad sobre las acciones.
- SAC (Soft Actor-Critic): SAC es un algoritmo off-policy y actor-crítico que introduce el concepto de maximización de la entropía. Es considerado uno de los algoritmos más eficientes y potentes para control continuo. El objetivo de SAC no es solo maximizar la recompensa acumulada, sino hacerlo mientras se mantiene la política lo más aleatoria posible (alta entropía). Al igual que el algoritmo PPO utiliza una política estocástica.

Característica	DDPG	TD3	PPO	SAC
Tipo de política	Off-policy	Off-policy	On-policy	Off-policy
Eficiencia	Alta	Alta	Baja	Muy alta
Política	Determinista	Determinista	Estocástica	Estocástica
Exploración	Ruido añadido	Ruido añadido	Natural (estocástica)	Maximización de entropía
Estabilidad	Baja	Media-Alta	Muy alta	Alta
Complejidad	Media	Media-Alta	Baja	Alta
Característica clave	Actor-Crítico para acciones continuas	Críticos Gemelos y actualizaciones retrasadas	Objetivo "recortado" para actualizaciones seguras	Maximización de la entropía

Tabla 3: Tabla comparativa de los distintos algoritmos de BlueSky.

3.2.1. Escenarios disponibles

BlueSky-Gym viene con siete entornos diseñados para facilitar la investigación del aprendizaje por refuerzo en la gestión del trafico aéreo, estos entornos se pueden dividir en tres categorías principales:

- Control vertical: Estos escenarios se centran en el control vertical de la aeronave. Los entornos dentro de esta categoría son:
  - DescentEnv-v0: Este entorno está preparado para enseñar al agente a realizar un descenso eficiente, donde el agente debe mantener el mayor tiempo posible la velocidad crucero antes de antes de iniciar el descenso a pista en el momento óptimo.
  - VerticalCREnv-v0: En este caso el agente debe descender de manera segura y controlada mientras evita a los intrusos y obstáculos que aparecen en el escenario.
- Resolución de Conflictos Horizontales: Estos escenarios se enfocan en evitar colisiones modificando la ruta en el plano horizontal, es decir equivalente al rumbo.
  - HorizontalCREnv-v0: En este escenario el agente debe evitar las colisiones con otras aeronaves manteniendo una distancia mínima de seguridad.
  - SectorCREnv-v0: Este caso es similar pero centrado en un sector del espacio aéreo. El agente controla una única aeronave que debe cruzar un sector mientras otras aeronaves (no controladas) también lo atraviesan.
  - MergeEnv-v0: En este escenario el agente controla el rumbo y la velocidad de las aeronaves en una de las corrientes de tráfico para que se incorporen a un punto de fusión (merge-point). Debe evitar en todo momento la colisión y además conseguir que crucen todas las aeronaves en el menor tiempo posible.

- Control Horizontal: En esta categoría se incluyen aquellos escenarios horizontales que se centran en otras tareas y no tanto en la resolución de conflictos.
    - ObstacleAvoidanceEnv-v0: En este caso el agente debe llevar el avión de un punto de inicio al otro evitando obstáculos fijos, por lo tanto, el agente entrena para interpretar la ruta más corta hasta su objetivo.
    - MultiGoalEnv-v0: Aquí el agente debe pasar por unos waypoints (puntos de ruta) de la manera más eficiente, este escenario carece de obstáculos, por lo tanto, el agente solo debe centrarse en encontrar el camino más corto que pase por todos los puntos de ruta.

3.2.2. Aplicaciones existentes

BlueSky y BlueSky-Gym son utilizados en muchos ámbitos de la investigación y formación de aeronáutica. Algunos ejemplos donde se podrían aplicar son:

- Control de tráfico aéreo: Todo tipo de investigaciones que buscan optimizar las rutas, minimizar el riesgo de colisiones y maximizar la gestión del espacio aéreo.
- Formación de controladores: Los simuladores son usados para entrenar controladores aéreos en escenarios que son lo más realistas y dinámicos posible.
- Desarrollo de algoritmos de RL: Investigaciones que emplean BlueSky-Gym para entrenar y evaluar agentes de RL en tareas como la gestión de flotas o la planificación de rutas.

3.3. Descripción de los Escenarios Básicos Seleccionados

3.3.1. Criterios de Selección de Escenarios

PlanWaypointEnv-v0: Es un escenario de dificultad media, en él el agente navegar tiene que ir navegando por una serie de waypoints sucesivamente, es un escenario en el que planificar la ruta, y realizar parámetros de rumbo. No incluye el control en conflictos con otras aeronaves, lo que hace que sea un buen punto de partida para empezar a aprender.

Representatividad: Representa un caso típico de un tipo de tarea de la navegación aérea: navegar por una ruta determinada. Es algo habitual en la navegación de vuelos, dadas las operaciones de vuelo, tanto en vuelo para control aéreo, como en vuelo libre.

Objetivos de aprendizaje:

- Desarrollar destrezas de navegación básica.
- Aprender a adaptar el rumbo a waypoints determinados.
- Familiarizarse con la dinámica del entorno de simulación.

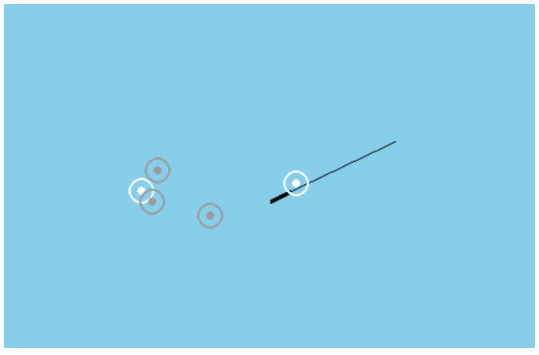


Figura 11: Escenario PlanWaypointEnv-v0.

VerticalCREnv-v0: Este escenario cuenta con un nivel de complejidad superior a la anterior, ya que se requiere que el agente gestione conflictos verticales con otras aeronaves. Además de mantener una altitud objetivo, el agente debe localizar otras aeronaves para evitar posibles colisiones.
Representatividad: Representa una circunstancia que se puede dar en coordinación con otras aeronaves (la coordinación de altitud y de velocidad vertical), pero que es especialmente propio de situaciones en espacios aéreos congestionados en los que la separación por el vertical es prioritaria.
Objetivos de aprendizaje:

- Desarrollar habilidades para mantener una altitud objetivo.
- Aprender a evitar conflictos verticales con otras aeronaves.
- Mejorar la toma de decisiones en situaciones no estacionarias y potencialmente conflictivas.

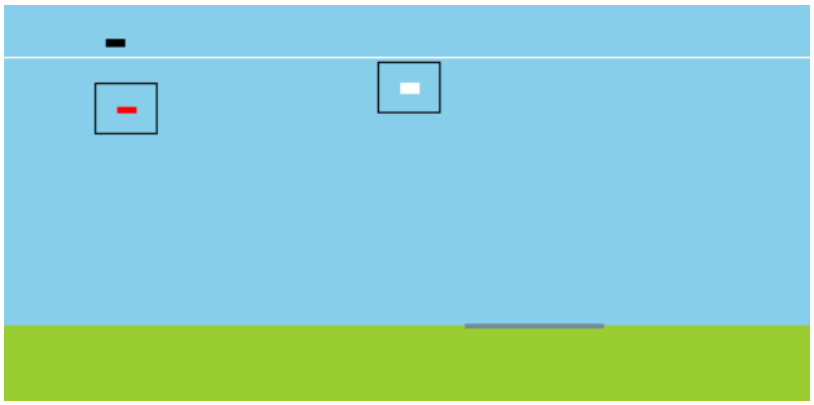


Figura 12: Escenario VerticalCREnv-v0.

HorizontalCREnv-v0: Este entorno es de la misma complejidad que VerticalCREnv-v0, pero aquí se necesita controlar los conflictos horizontales. El agente controlará su rumbo a partir del conflicto que debe evitar en el camino para navegar hacia un waypoint específico, lo que requerirá cautela en la planificación y la realización de las tareas.
Representatividad: Representa una situación habitual en el caso de la navegación aérea en la que se encuentra el agente evitando aeronaves en conflicto mientras navega hacia un destino concreto. Se hace hincapié en este objetivo también en los corredores aéreos congestionados o en los alrededores de los aeropuertos.
Objetivos de aprendizaje:

- Desarrollar habilidades para ajustar el rumbo y evitar conflictos horizontales.
- Aprender a navegar hacia waypoints en presencia de otras aeronaves.
- Mejorar la capacidad de anticipación y reacción ante situaciones dinámicas.

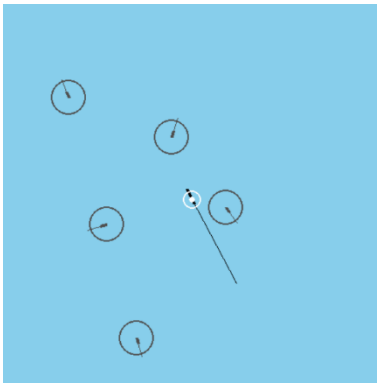


Figura 13: Escenario HorizontalCREnv-v0.

3.3.2. Descripción de los Escenarios

- Escenario 1: [PlanWaypointEnv-v0].

Es un escenario de ejemplo diseñado para demostrar la lógica de control horizontal. El objetivo del agente es aprender a planificar de manera eficiente una trayectoria que visite una serie de waypoints (puntos de ruta) generados aleatoriamente. El agente controla el rumbo de la aeronave para cumplir esta tarea.

n_updates	50000	100000	200000	300000	400000	500000	1000000	1500000	2000000
rollout									
ep_len_mean	300	300	300	300	300	300	292	297	271
ep_rew_mean	1.25	2.46	2	1.35	1	1.7	2.78	2.57	4.13
time									
episodes	4	168	164	164	4	164	1668	1668	1732
time_elapsed	74	3018	2791	2804	72	3464	30148	28252	30212
total_timesteps	1200	49942	48934	49187	1200	49133	498896	499316	499651
train									
actor_loss	-3.36	-1.72	-5.52	-3.27	-5.93	-15.9	0.2	0.182	-0.217
critic_loss	0.0241	0.0223	0.261	0.0964	0.148	2.48	0.00316	0.00656	0.0211
ent_coef	0.00715	0.00894	0.0265	0.0186	0.0202	0.0507	0.000797	0.00116	0.00214
ent_coef_loss	0.529	0.144	0.0857	0.00477	0.746	0.0938	-0.968	-0.35	-0.696
learning_rate	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003

Tabla 4: Resultados entrenamiento escenario PlanWaypointEnv-v0.

- Escenario 2: [VerticalCREnv-v0].

Este es un escenario de control vertical para la resolución de conflictos. El agente debe mantener una altitud de crucero objetivo y descender a una pista, evitando colisiones con otras aeronaves que se encuentran en trayectorias de crucero conflictivas. El agente controla la velocidad vertical de la aeronave para lograrlo.

n_updates	50000	100000	200000	300000	400000	500000	1000000	1500000	2000000
rollout									
ep_len_mean	39.8	39.9	40	40	40	40	40	40.2	40.1
ep_rew_mean	-182	-97.9	-91.5	-77.3	-71.6	-65.4	-56.4	-49.8	-51.2
time									
episodes	8	1256	3756	6256	8756	11256	11260	11244	11232
time_elapsed	40	5387	15728	25844	35994	46548	47349	45942	51088
total_timesteps	318	50151	150095	250074	350151	450187	450291	450392	450428
train									
actor_loss	73	62.7	59.4	56.2	52.4	48.3	47.4	43.6	42.8
critic_loss	33.9	13.4	14.9	18.2	11.2	39.7	7.92	5.47	4.83
ent_coef	0.114	0.0991	0.0888	0.0859	0.078	0.0765	0.073	0.0759	0.0686
ent_coef_loss	1.37	-0.157	-0.216	-0.25	0.178	-0.533	-0.298	-0.103	-0.537
learning_rate	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003

Tabla 5: Resultados entrenamiento escenario VerticalCREnv-v0.

- Escenario 3: [HorizontalCREnv-v0].

En este entorno de resolución de conflictos horizontales, el agente aprende a navegar hacia un destino final mientras evita a otras aeronaves mediante cambios de rumbo. Las otras aeronaves se generan inicialmente en trayectorias que entran en conflicto con la del agente.

n_updates	50000	100000	200000	300000	400000	500000	1000000	1500000	2000000
rollout									
ep_len_mean	150	150	150	150	150	150	149	148	146
ep_rew_mean	-97.8	-103.1	-92	-86.3	-70.8	-68.9	61.6	-54.2	-53.1
time									
episodes	7	1235	3693	6127	8533	10872	11009	11131	11126
time_elapsed	46	5387	15634	25823	35894	45936	46562	45629	46957
total_timesteps	1376	6389	195490	311757	397263	501745	501381	501847	502008
train									
actor_loss	94.4	87.7	76.7	71.3	64.9	61.5	58.7	54.6	52.9
critic_loss	49.8	41.3	29.7	28.9	24.3	27.1	21.4	15.9	13.4
ent_coef	0.271	0.141	0.0978	0.0917	0.0872	0.0811	0.0765	0.074	0.0691
ent_coef_loss	3.52	1.98	0.043	-0.092	-0.173	-0.297	-0.166	-0.209	-0.132
learning_rate	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003

Tabla 6: Resultados entrenamiento escenario HorizontalCREnv-v0.

3.4. Análisis de Resultados Preliminares

3.4.1. Metodología de Evaluación

Explicación de los parámetros:

- Recompensa media por episodio (ep_rew_mean): Indica cómo de bien está desempeñando el agente su tarea.
- Longitud media del episodio (ep_len_mean): Muestra si el agente está realizando los episodios de forma eficiente.
- Pérdidas del actor y crítico (actor_loss; critic_loss): Reflejan cómo está aprendiendo el agente respecto a maximizar la recompensa (actor_loss) y predecir los valores de los estados (critic_loss). Estas métricas tienden a 0, siendo que 0 indica una política clara tomada por el agente y una predicción perfecta en los valores de estado.
- Coeficiente de entropía (ent_coef): Muestra el nivel de exploración que tiene el agente. Cuanto más cercano a 0 es el coeficiente de entropía más explotación realiza el agente y menor es la exploración. Siendo la máxima exploración el 1 y la máxima explotación el 0.
- Tiempo y pasos de entrenamiento (time_elapsed; total_timesteps): Muestra la evolución del entrenamiento.

Métricas clave:

Para proceder con la valoración de los resultados del entrenamiento, se ha realizado un análisis de estos basado en diferentes técnicas. En primer lugar, se ha llevado a cabo un análisis de tendencias en el que se han observado las evoluciones de las métricas clave que nos interesan, es decir, la recompensa media, la longitud del episodio, las pérdidas, el coeficiente de entropía, etc. En segundo lugar, se ha ejecutado un análisis por etapas, comparando las métricas en los momentos iniciales, intermedios y finales del entrenamiento, con el propósito de ver cambios significativos y patrones claros. También, se ha realizado un análisis del comportamiento de las métricas mencionadas, entendiendo que las fluctuaciones son parte del proceso de exploración y aprendizaje en el marco del aprendizaje por refuerzo. Por último, se ha realizado una evaluación de la convergencia, es decir, si las métricas clave, en especial las pérdidas del actor y del crítico, se estabilizan en valores similares, lo que implica que el agente ha sido capaz de generar cierta política más o menos eficaz. Este análisis en profundidad permite entender lo que ha ido logrando el agente, pero también permite entender la eficacia del propio proceso de entrenamiento.

3.4.2. Resultados Obtenidos

El estudio de los datos que se han generado a lo largo de las etapas de entrenamiento del agente pone de manifiesto una progresión notoria en el aprendizaje que ha adquirido durante la formación del agente, ya que se ha producido también una evolución bastante evidente en lo que respeta de su funcionamiento. En primer lugar, la media de la recompensa del episodio concluye con una tendencia positiva (siendo la primera de 1.25 y , al final del entrenamiento , de 4.13), lo que quiere decir que ha mejorado su capacidad para maximizar el aprendizaje del agente. Sin embargo, en el trascurso del entrenamiento han aparecido oscilaciones, como la caída a 1.0 en etapas intermedias, lo que es normal por la exploración activa que estaba realizando el agente. Por otro lado, la longitud media del episodio se ha mantenido constante en 300 pasos en la mayoría del entrenamiento, aunque en el final ha disminuido levemente a 271 pasos, lo que sugiere que el agente también ha optimizado su eficiencia.

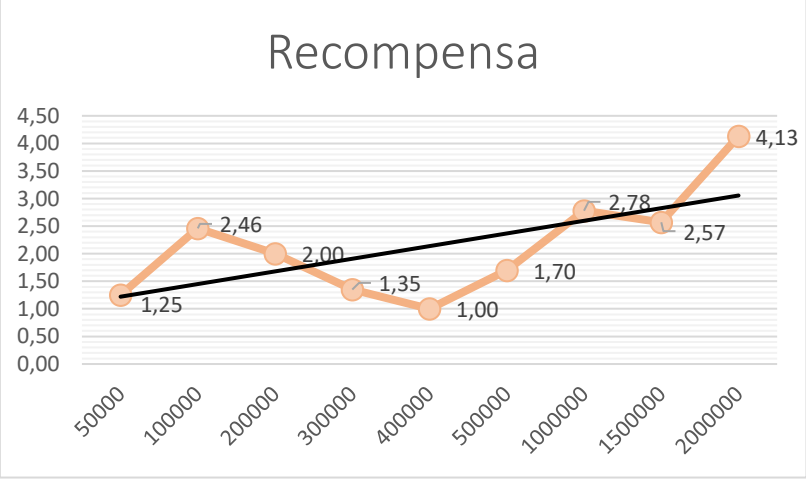


Figura 14: Recompensa escenario PlanWaypointEnv-v0 con línea de tendencia.

La pérdida del actor y la pérdida del crítico que se han podido observar tendencia a converger a valores más estables en las etapas finales. La pérdida del actor, que inicialmente fluctúa entre -3.36 y -15.9, alcanzan unos valores iguales a cero en las últimas fases, es decir que da conformidad a una política ya fijada. De forma similar, la pérdida del crítico, que arranca 0.0223 alcanzando picos de 2.48, se reduce hasta valores de 0.00316, evidenciando una mejoría en la predicción de los valores de los estados. Por otra parte, el coeficiente de entropía que comienza con un valor de 0.00715 está en 0.00214 al final del episodio, lo que también muestra una disminución, que señala que el agente está pasando de una fase de exploración a una fase de explotación de lo que ha aprendido.

Finalmente cabe resaltar que pese a que el aprendizaje del agente ha sido evidente ha necesitado casi dos millones de actualizaciones para empezar aumentar la eficiencia y reducir el tiempo que tarda en realizar la tarea lo cual indica que la eficiencia del entrenamiento es reducida y podría significar que los hiperparámetros no están del todo optimizados.

3.4.3. Ajustes en los Parámetros de las Simulaciones

Con el fin de mejorar la velocidad y la eficiencia del entrenamiento del agente en el código, se han modificado ciertos parámetros de la configuración del algoritmo y del entorno.

Primero, se ha aumentado la tasa de aprendizaje (learning_rate=1e-3) para acelerar las actualizaciones del modelo, de este modo, el agente puede aprender más rápidamente de cada una de las experiencias. Se ha incrementado el tamaño del lote (batch_size=256), lo que permite que el entrenamiento sea más estable, dado que se obtendrán más muestras para calcular los gradientes en cada paso de optimización. Por último, se ha ampliado el tamaño del buffer de reproducción (buffer_size=1_000_000); en este caso se permite contar con un mayor número de experiencias pasadas, favoreciendo un entrenamiento más estable y diverso.

Adicionalmente, se ha ajustado el coeficiente de entropía (ent_coef=0.1), para equilibrar la exploración y la explotación durante el aprendizaje del agente. Igualmente se ha incrementado el parámetro tau (tau=0.01), para acelerar la actualización de los valores objetivo en los algoritmos de actores-críticos, permitiendo que el modelo se adapte más rápidamente.

En lo que respecta a la configuración del entorno, se ha realizado paralelización del entorno de entrenamiento mediante la función make_vec_env con varios entornos a la vez (es decir, n_envs=4). Esta estrategia permite recolectar de forma más rápida experiencias reformulando distintas copias del entorno simultáneamente, siendo el método de entrenamiento más efectivo en su funcionamiento.

Además, se ha añadido la normalización de observaciones y recompensas mediante el uso de VecNormalize, que resulta en un entrenamiento más estable puesto que no se producirán entradas o salidas desbalanceadas o variables en el modelo.

Para salvaguardar el avance en el proceso del entrenamiento de posibles problemas, se ha incluido una devolución de llamada de checkpoints (CheckpointCallback) que guarda regulamente el estado del modelo, lo que permitirá continuar el entrenamiento desde el último punto guardado sin perder información relevante.

Por último, se ha adicionado una fase de evaluación del agente justo después del entrenamiento, en donde se medirá la recompensa total obtenida en distintos episodios, haciendo una salida de los resultados en formato CSV y graficando el rendimiento, aportando en cierta forma una imagen clara y cuantificada del rendimiento del agente que aminora el análisis y comparación de diferentes versiones del modelo.

3.4.4. Resultados iniciales tras ajustes

Gracias a las modificaciones en el código ahora disponemos de una carpeta de resultados donde se guardan los históricos del entrenamiento, en este caso se puede apreciar la media en intervalos de 40000 actualizaciones, lo que permite ver mejor la evolución del agente, y evitar datos aislados que podrían falsear experimentos futuros. Se ha decidido volver a entrenar al agente desde 0 en el escenario 1 (PlanWaypointEnv-v0) dado que es el escenario menos complejo y se podrán notar mayores diferencias iniciales.

n_updates	episodes	ep_len_mean	ep_rew_mean	time_elapsed	total_timesteps	actor_loss	critic_loss	ent_coef	ent_coef_loss	learning_rate	fps	timestamp
40000	100	300	0,949999988	-1,74416E+18	160000	-7,006715775	0,001744943	0,050295554	-4,91553688	0,0003	-2,29337E-14	2025-04-09 01:51:16
80000	100	300	1,5	-1,74416E+18	320000	-4,427482605	0,004698405	0,002755786	-6,277266502	0,0003	-4,58675E-14	2025-04-09 02:12:37
120000	100	298,24	1,629999995	-1,74416E+18	480000	-2,533821344	0,002091667	0,000454962	0,671323717	0,0003	-6,88012E-14	2025-04-09 02:32:40
160000	100	299,16	1,309999943	-1,74416E+18	640000	-1,603900433	0,009544099	0,000452502	1,154852986	0,0003	-9,1735E-14	2025-04-09 02:53:06
200000	100	298,32	1,460000038	-1,74416E+18	800000	-0,94804585	0,002141542	0,000449906	0,067853563	0,0003	-1,14669E-13	2025-04-09 03:13:29
240000	100	297,91	2,190000057	-1,74416E+18	960000	-0,570666909	0,004232383	0,000477243	0,766058564	0,0003	-1,37602E-13	2025-04-09 03:33:35
280000	100	299,71	2,230000019	-1,74416E+18	1120000	-0,328757197	0,003918371	0,000511667	-0,578528166	0,0003	-1,60536E-13	2025-04-09 03:53:53
320000	100	299,85	1,690000057	-1,74416E+18	1280000	-0,174185723	0,001835785	0,000406696	-1,118625164	0,0003	-1,8347E-13	2025-04-09 04:13:52
360000	100	298,81	1,629999995	-1,74416E+18	1440000	-0,124477297	0,007152225	0,00038733	0,187580675	0,0003	-2,06404E-13	2025-04-09 04:33:59
400000	100	297,45	1,789999962	-1,74416E+18	1600000	-0,032563914	0,00566678	0,00035912	0,943505049	0,0003	-2,29337E-13	2025-04-09 04:54:26
440000	100	300	1,870000005	-1,74416E+18	1760000	-0,00595837	0,00800099	0,000295775	0,48614943	0,0003	-2,52271E-13	2025-04-09 05:14:52
480000	100	296,65	1,730000019	-1,74416E+18	1920000	0,03705686	0,001997148	0,000298926	1,028913379	0,0003	-2,75205E-13	2025-04-09 05:35:29

Tabla 7: Resultados entrenamiento escenario PlanWaypointEnv-v0 tras modificaciones iniciales.

Como se puede ver a partir de la tabla de aprendizaje del agente esta segunda vez también genera unas primeras estimaciones que nos indican una mejoría en la recompensa media por episodio (ep_rew_mean), la cual pasa de 0'95 a ~2'23, aunque en un segundo momento se presentaron unas oscilaciones que nos pueden sugerir inestabilidad o una exploración activa. La duración media de los episodios (ep_len_mean) se estabilizó en torno a 300 pasos siendo el límite de tiempo establecido por episodio, pero se puede notar una pequeña reducción en la etapa final. Las pérdidas del actor (actor_loss) evolucionan de -7.0 a ~0.037, con lo que nos indicarían un proceso de mejora de la política, mientras que la del crítico (critic_loss) se mantenía en cifras bajas (0.001–0.008), si bien el bajo valor que nos ofrece podría sugerir incluso una subestimación de las recompensas. El coeficiente de entropía (ent_coef) pasó de 0.050 a 0.0003, con lo que nos podría indicar que la exploración se iría reduciendo con el paso del tiempo. Sin embargo, hemos podido detectar algunas anomalías críticas, como valores imposibles desde el punto de vista de la temporalidad, como el que nos arroja el time_elapsed, que tiene el valor -1.7E+18, en el apartado de fps podemos ver que dicho valor era de -2.29E-13, lo que nos puede indicar que existían fallas de registro.

En comparación se puede apreciar como las primeras tablas muestran un comportamiento más volátil en relación con la realizada tras los cambios en el código, asociado a caídas profundas en las recompensas (ep_rew_mean), oscilando entre 1.0 y 2.46, y sin que se pueda identificar una tendencia clara de mejora, a diferencia de lo observado en el segundo experimento, donde el aumento progresivo (aunque inestable) de la recompensa se dejaba ver en la media de las recompensas. Para esta segunda tabla, el ep_len_mean se mantiene fijado en 300 (límite fijado por el entorno), al igual que en la primera tabla. Pero las pérdidas del actor (actor_loss) son muy profundas (de -3.36 a -15.9), lo que hace suponer inestabilidad, a diferencia de lo observado en la primera tabla, donde convergían hacia cero; el crítico (critic_loss) también presenta valores peores, hasta 2.48 (vs. 0.008 en la primera tabla) lo que sugiere que es complejo estimar el valor de los estados, y el coeficiente de entropía (ent_coef) es más alto en la segunda tabla (0.007–0.0507 vs. 0.0003–0.050) lo que podría significar una mayor exploración pero menos eficiente.

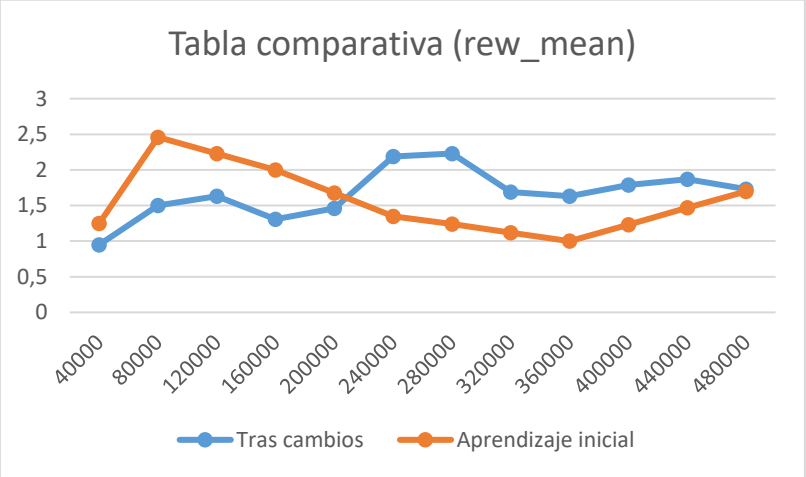


Figura 15: Gráfica comparativa PlanWaypointEnv-v0.

3.5. Conclusiones del Capítulo

Los resultados del entrenamiento constituyen una demostración de la capacidad del aprendizaje por refuerzo (RL) para resolver el dominio del tráfico aéreo, aunque con divergencias importantes entre los distintos escenarios. Los experimentos evidencian una clara evolución en la consistencia del Escenario 1 (PlanWaypointEnv-v0) con una mejoría en la recompensa media (de 1.25 a 4.13) y, la estabilización progresiva de la pérdida del actor y la pérdida del crítico (significativas de cierta forma) sugiere una política de navegación satisfactoria y una estimación adecuada del valor de los estados. Sin embargo, esta mejora requiere elevados tiempos de entrenamiento (cercanos a los dos millones de actualizaciones), y además se han evidenciado la volatilidad entre unos resultados y otros, pasando de máximos a mínimos, lo que conlleva la necesidad de una optimización de hiperparámetros como el learning rate o el tamaño del batch con el objetivo de acelerar la convergencia y adicionalmente mejorar la forma en que se muestra los resultados para poder hacer mejores análisis en el futuro.

El Escenario 2 (VerticalCREnv-v0), por su parte, supone un entorno más complejo, en el que se aprecian recompensas negativas, métricas inestables y con grandes oscilaciones, si bien existieron pequeñas progresiones. En este caso, la oscilación de las pérdidas del crítico, además del incremento de las pérdidas (hasta 2.48) no sólo indican que existieron mejoras, pero, por otro lado, también ilustra el hecho de que, incluso en entornos complejos, se pueden alcanzar dificultades de generalización. Este hecho pone de manifiesto la importancia de crear un entorno específico, del tipo:

- ✓ Mayor exploración controlada (ajustar ent_coef).
- ✓ Normalización robusta de observaciones y recompensas
- ✓ Paralelización de entornos (con make_vec_env) para incrementar la variedad de experiencias.

Las mejoras implementadas para el modelo, como el incremento del buffer de experiencias, la aplicación de checkpoints y la evaluación sistemática, fueron cruciales para un aprendizaje de RL que fuera estable y recuperable, si bien el coste computacional y la sensibilidad a los metaparámetros subrayan la necesidad de mejorar el modelo, sobre todo para entornos dinámicos con restricciones de operación reales.

Por tanto, este trabajo muestra que el RL se puede aplicar a la gestión del tráfico aéreo, pero también señala los retos que le quedan por afrontar: la eficiencia del entrenamiento en entornos complejos y la delicadeza de encontrar un equilibrio entre la exploración y la explotación. Los resultados apuntan hacia futuras líneas de investigación para reducir el coste computacional y aumentar la transferibilidad a entornos reales, donde la robustez y la adaptación son esenciales.

4. Tutorial de BlueSky-Gym

4.1. Instalación y configuración del entorno

El dispositivo utilizado es un portátil matebookD15 con Windows 10, tiene un procesador AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx 2.10 GHz con una RAM de 8,00 GB (6,94 GB usable) y un sistema operativo de 64 bits.

1. Descargar e instalar Python 3.11
2. Agregar la Ruta de pip a las Variables de Entorno
    - Agregar Python y pip al PATH
        - Presiona Win + R, escribe sysdm.cply presiona Enter.
        - Ve a la pestaña "Opciones avanzadas" y haz clic en "Variables de entorno".
        - En la sección "Variables del sistema", busca y selecciona la variable Path y haz clic en Editar.
        - Haz clic en Nuevo y agrega las siguientes rutas: C:\Users\franm\AppData\Local\Programs\Python\Python311\ C:\Users\franm\AppData\Local\Programs\Python\Python311\Scripts\
        - Haz clic en Aceptar para guardar los cambios.
        - Compruebe si tiene pip instalado usando: py -m pip –version o python -m pip –version
3. Instalar las dependencias del sistema (compiladores)
        - Ve a este enlace y descarga el instalador de Build Tools for Visual Studio. (https://visualstudio.microsoft.com/es/downloads/)
        - Durante la instalación, seleccione la opción "Herramientas de compilación C++".
        - Asegúrate de marcar la casilla de "Windows 10 SDK".
        - Haga clic en Instalar y espere a que termine el proceso.
4. A partir de Python 3.10, el paquete distutils fue descontinuado y no se incluye por defecto en algunas distribuciones de Python. Este, es necesario para la instalación de muchas bibliotecas que requieren compilación (como numpy).
        - python -m pip install setuptools o py -m pip install setuptools
5. Para el programa Bluesky-gym es necesario descargar e instalar numpy 1.24
        - pip install numpy==1.24.0
6. Con todo ello ya debería ser posible instalar el programa Bluesky-gym
        - pip install bluesky-gym

4.2. Descripción de los módulos principales: Simulador, entorno de RL y herramientas de visualización.

Posteriormente a la instalación podremos ver una carpeta denominada bluesky-gym-main donde se divide en otros sub ficheros. Inicialmente solo contará con las carpeta de bluesky, bluesky_gym, docs/media y scrips, y después del primer entrenamiento se generará la carpeta models. A continuación analizaremos las principales carpetas y su contenido.

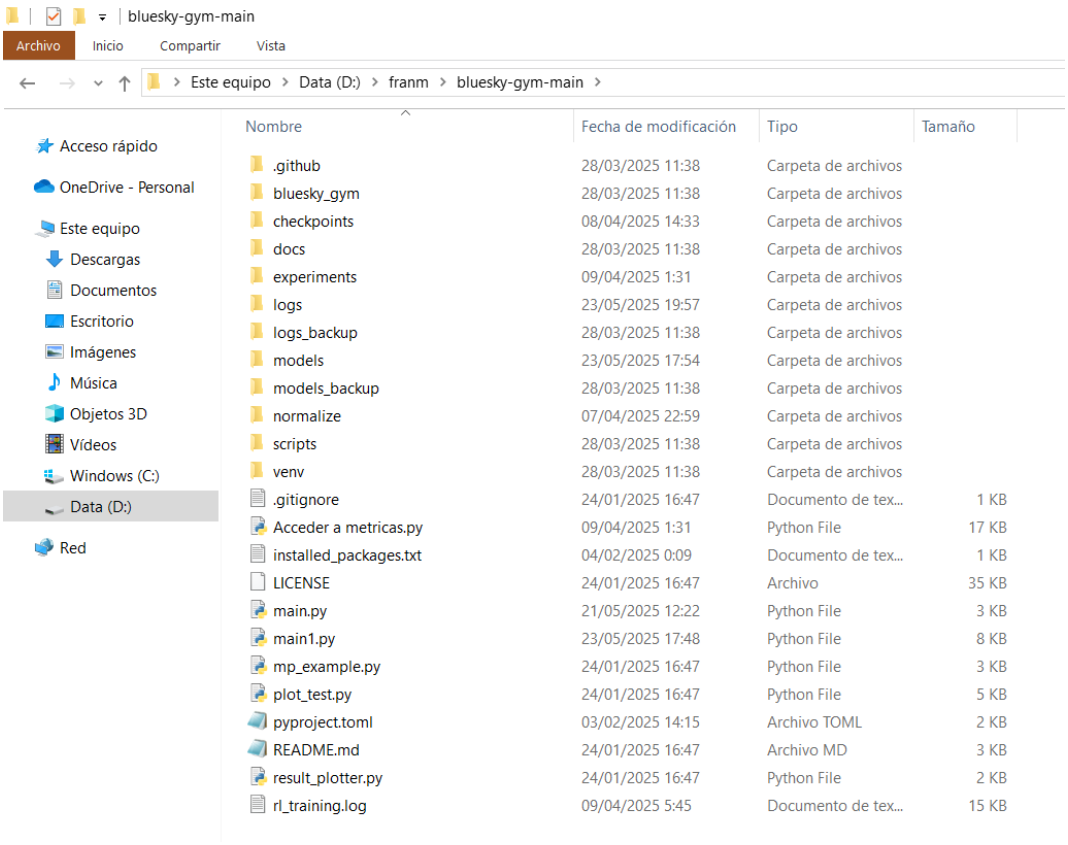


Figura 16: Contenido de la carpeta principal BlueSky-Gym.

4.2.1. El corazón del programa

El archivo de Python denominado “main” es el núcleo del programa, comienza definiendo y llamando las variables del entorno. Es necesario definir el nombre del entorno entre los siete distintos entornos existentes, además podemos elegir entre cuatro distintos algoritmos descritos anteriormente en el punto 3 “estudio del paquete BlueSky-Gym”. También se encarga de buscar un modelo existente pudiendo cambiar entre distintos modelos para un mismo escenario, en caso de que no se haya entrenado previamente ningún modelo lo creará. Otro apartado importante es que podemos definir el número de pasos que deseamos entrenar y una vez finalizado el entrenamiento guardará el modelo y mostrará una simulación de este.

```python
import numpy as np
import gymnasium as gym
from stable_baselines3 import PPO, SAC, TD3, DDPG
import numpy as np
import bluesky_gym
import bluesky_gym.envs
from bluesky_gym.utils import logger

# Registra los entornos de Bluesky
bluesky_gym.register_envs()

# Define el nombre del entorno y el algoritmo
env_name = 'HorizontalCREnv-v0'
algorithm = SAC

# Inicializar el logger
log_dir = f'./logs/{env_name}/'
file_name = f'{env_name}_{str(algorithm.__name__)}.csv'
csv_logger_callback = logger.CSVLoggerCallback(log_dir, file_name)

# Configuración de entrenamiento y evaluación
TRAIN = True # True comienza entrenamiento, False pasa a visualización de resultados
EVAL_EPISODES = 10
```

Figura 17: Parte inicial del código main.

4.2.2. Carpeta bluesky_gym

En esta carpeta se encuentra la carpeta “envs” donde se sitúan los 7 siete escenarios que podemos utilizar para las simulaciones.

__pycache__	28/03/2025 11:38	Carpeta de archivos	
envs	20/05/2025 15:54	Carpeta de archivos	
utils	28/03/2025 11:38	Carpeta de archivos	
__init__.py	24/01/2025 16:47	Python File	2 KB
README.md	24/01/2025 16:47	Archivo MD	1 KB

En el interior de la carpeta “envs” además de los distintos escenarios encontramos la carpeta “common”, en esta hay tres archivos de Python llamados “functions” (realiza las operaciones necesarias para situar los waypoints y otros objetos de varios escenarios), polygon_generator (se encarga de generar las formas de los obstáculos y otros objetos) y finalmente screen_dummy (que permite la visualización de la simulación).

__pycache__	23/05/2025 19:57	Carpeta de archivos	
common	28/03/2025 11:38	Carpeta de archivos	
__init__.py	24/01/2025 16:47	Python File	1 KB
aircraft_data.csv	15/05/2025 19:47	Archivo de valores separados por comas de Microsoft Excel	1 KB
aircraft_data.xlsx	15/05/2025 19:45	Hoja de cálculo de Microsoft Excel	9 KB
descent_env.py	24/01/2025 16:47	Python File	9 KB
horizontal_cr_env real.py	22/05/2025 11:28	Python File	14 KB
horizontal_cr_env.py	23/05/2025 19:57	Python File	19 KB
merge_env.py	24/01/2025 16:47	Python File	17 KB
NuevoEscenario.py	30/04/2025 14:48	Python File	15 KB
plan_waypoint_env.py	24/01/2025 16:47	Python File	10 KB
README.md	24/01/2025 16:47	Archivo MD	6 KB
sector_cr_env.py	24/01/2025 16:47	Python File	17 KB
static_obstacle_env real.py	24/01/2025 16:47	Python File	17 KB
static_obstacle_env.py	17/05/2025 11:00	Python File	14 KB
vertical_cr_env.py	24/01/2025 16:47	Python File	16 KB

4.2.3. Carpeta bluesky

Aquí se encuentra el simulador, es decir el motor que permite simular distintos escenarios y mover todo para que el agente pueda aprender y nosotros podamos visualizarlo. Una vez instalado en mi caso apareció la carpeta venv que posee la librería y el conjunto de recursos que utiliza el simulador.

Include	14/02/2025 12:50	Carpeta de archivos	
Lib	20/05/2025 16:14	Carpeta de archivos	
Scripts	28/03/2025 11:38	Carpeta de archivos	
.gitignore	03/02/2025 12:25	Documento de tex...	1 KB
pyvenv.cfg	03/02/2025 12:25	Archivo CFG	1 KB

4.2.4. Docs/media

Esta carpeta posee gif que permiten visualizar como se comporta el agente en una política establecida en los escenarios DescentEnv-v0, HorizontalCREnv-v0 y PlanWaypointEnv-v0 de un agente con una política aún por definir.

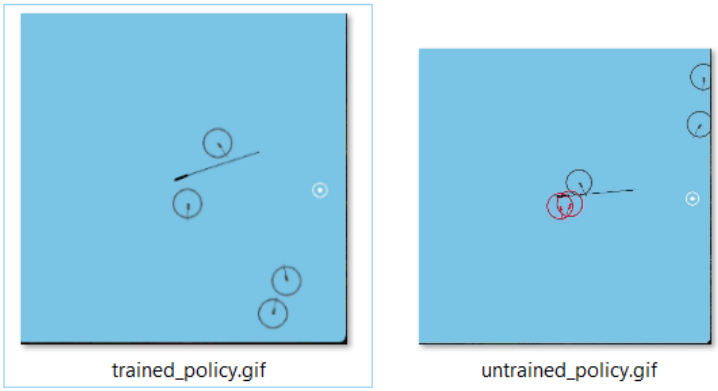


Figura 18: Ejemplo del contenido de la carpeta media.

4.2.5. Scripts

Esta carpeta contiene bucles de entrenamiento de ejemplo para los diferentes escenarios en bluesky_gym/envs creados por los autores del programa para facilitar su entendimiento y uso.

common	28/03/2025 11:38	Carpeta de archivos	
descent_env_sac.py	24/01/2025 16:47	Python File	1 KB
README.md	24/01/2025 16:47	Archivo MD	1 KB
vertical_cr_env_sac.py	24/01/2025 16:47	Python File	1 KB

4.2.6. Models

En esta carpeta se guardan los modelos por escenario, los modelos se guardan en un archivo comprimido, podemos apreciar que además del escenario también genera una carpeta distinta si el archivo tiene un algoritmo distinto a SAC, en este caso se ha guardado el modelo de una prueba con el algoritmo PPO.

DescentEnv-v0	28/03/2025 11:38	Carpeta de archivos
HorizontalCREnv-v0	23/05/2025 17:25	Carpeta de archivos
HorizontalCREnv-v0_PPO	23/05/2025 17:54	Carpeta de archivos
MergeEnv-v0	28/03/2025 11:38	Carpeta de archivos
PlanWaypointEnv-v0	08/04/2025 10:16	Carpeta de archivos
SectorCREnv-v0	28/03/2025 11:38	Carpeta de archivos
StaticObstacleEnv-v0	28/03/2025 11:38	Carpeta de archivos
VerticalCREnv-v0	28/03/2025 11:38	Carpeta de archivos

Al revisar los modelos creados podemos ver que guarda información como la política, el actor, crítico y el coeficiente de entropía además de datos adicionales, como las variables o el motor del simulador.

..			Carpeta de archivos		
data	14.890	14.890	Archivo	10/03/2025 9:10	201BBACB
_stable_baselines3_version	5	5	Archivo	10/03/2025 9:10	2018FC1E
system_info.txt	186	186	Documento de texto	10/03/2025 9:10	545CB486
pytorch_variables.pth	1.180	1.180	Archivo PTH	01/01/1980 0:00	25F8EB37
policy.pth	1.363.574	1.363.574	Archivo PTH	01/01/1980 0:00	96AEEC4B
actor.optimizer.pth	547.406	547.406	Archivo PTH	01/01/1980 0:00	1BE01D9F
critic.optimizer.pth	1.090.986	1.090.986	Archivo PTH	01/01/1980 0:00	6C2A2C49
ent_coef_optimizer.pth	1.940	1.940	Archivo PTH	01/01/1980 0:00	108A6B20

4.2.7. Otras carpetas y añadidos

Finalmente, encontramos otras carpetas como por ejemplo de log y algunas creadas al realizar las pruebas para un mayor entendimiento y poder disponer de datos para analizar los entrenamientos. En este caso la carpeta más importante es la de experimentos donde guarda el modelo (siendo redundante en este caso), los datos del entrenamiento en formato Excel para poder generar graficas de aprendizaje, checkpoints para evitar la pérdida del entrenamiento completo en caso de fallo durante el entrenamiento y otros datos de interés.

config.json	09/04/2025 1:31	Archivo JSON
results	11/04/2025 1:12	Carpeta de archivos
model	09/04/2025 5:45	Carpeta de archivos
checkpoints	09/04/2025 5:35	Carpeta de archivos
logs	09/04/2025 1:31	Carpeta de archivos
tensorboard	09/04/2025 1:31	Carpeta de archivos

4.3. Ejecución de simulaciones básicas y visualización de resultados y evolución en video.

En la finalización de los entrenamientos del agente, por defecto se muestra una simulación de este para que el espectador pueda hacer un seguimiento de los avances, hemos decidido realizar un video en el escenario PlanWaypointEnv-v0 que ejemplifique las mejoras de manera visual que ha ido obteniendo durante el aprendizaje hasta un límite de dos millones de pasos.

En este video: Video entrenamiento agente.mp4, se puede ver como el agente va mejorando lentamente formando una política adecuada, si bien al principio realiza una fase de exploración, va pasando a una fase de explotación a medida que el agente va obteniendo una política más definida, lo que permite que finalice la mayoría de escenarios. Sin embargo, después de dos millones de pasos, aún es incapaz de encontrar la ruta más rápida para conseguir sus objetivos y además los avances se vuelven más lentos debido a que el agente realiza una mayor explotación que exploración.

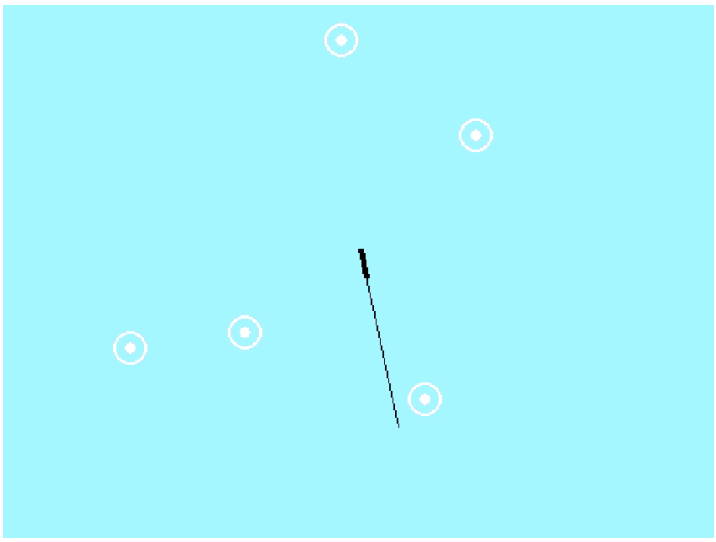


Figura 19: Captura de uno de los videos durante la simulación.

5. Experimentos computacionales con BlueSky-Gym

El objetivo de este apartado es ver hasta qué punto podemos utilizar el simulador BlueSky-Gym para crear entornos más realistas y de esta forma ver sus virtudes y limitaciones.

5.1. Creación de un escenario simple: Definición de aeronaves, rutas y condiciones iniciales. Parámetros clave: Velocidad, altitud, separación entre aeronaves, etc.

Para crear un escenario primero debemos situarnos en la carpeta de escenarios denominada en el archivo como "envs" mencionada anteriormente y seleccionar el escenario que queremos modificar.

5.1.1. Configuración y parámetros globales

Para empezar, debes llamar a las bibliotecas necesarias:

```
import numpy as np
import pygame

import bluesky as bs
from bluesky_gym.envs.common.screen_dummy import ScreenDummy
import bluesky_gym.envs.common.functions as fn

import gymnasium as gym
from gymnasium import spaces
```

Posteriormente se establecen las constantes o variables globales. Aquí es donde nos permite modificar en numero de aeronaves intrusas que aparecen, también el numero de puntos objetivos que debe pasar el agente, así como velocidades y otras variables. También se definen las recompensas y penalizaciones que ayudarán al agente a conseguir una heurística adecuada.

```
DISTANCE_MARGIN = 5 # km
REACH_REWARD = 1

DRIFT_PENALTY = -0.1
INTRUSION_PENALTY = -1

NUM_INTRUDERS = 5
NUM_WAYPOINTS = 1
INTRUSION_DISTANCE = 5 # NM

WAYPOINT_DISTANCE_MIN = 100
WAYPOINT_DISTANCE_MAX = 150

D_HEADING = 45

AC_SPD = 150

NM2KM = 1.852

ACTION_FREQUENCY = 10
```

5.1.2. Función __init__(self, ...)

Este es el constructor, su principal función es crear el entorno de la simulación, destaca el “bs.init(...):” pues se encarga de inicia el motor de simulación de BlueSky y también “self.observation_space:” donde se define todo lo que la AI puede "ver" o percibir del mundo. En este caso, es un diccionario que contiene el estado de su propia aeronave (rumbo y velocidad), el estado de los intrusos (distancia y rumbo relativo) y la ubicación de los waypoints. Finalmente tenemos “self.action_space:” que define lo que la AI puede "hacer". En este caso, su única acción es decidir cuánto cambiar su rumbo (entre -30 y +30 grados). Cada vez que un episodio termina se inicializan variables internas como “self.intrusion = False” o “self.reward = 0” para empezar cada episodio desde cero.

```
class HorizontalCREnv(gym.Env):
    """
    Horizontal Conflict Resolution Environment

    TODO:
    - look at adding waypoints instead of staying straight
    """
    metadata = {"render_modes": ["rgb_array","human"], "render_fps": 120}

    def __init__(self, render_mode=None):
        self.window_width = 512
        self.window_height = 512
        self.window_size = (self.window_width, self.window_height) # Size of the rendered environment


        # Observation space should include ownship and intruder info
        # Maybe later also have an option for CNN based intruder info, could be interesting
        self.observation_space = spaces.Dict(
            {
                "intruder_distance": spaces.Box(-np.inf, np.inf, shape = (NUM_INTRUDERS,), dtype=np.float64),
                "cos_difference_pos": spaces.Box(-np.inf, np.inf, shape = (NUM_INTRUDERS,), dtype=np.float64),
                "sin_difference_pos": spaces.Box(-np.inf, np.inf, shape = (NUM_INTRUDERS,), dtype=np.float64),
                "x_difference_speed": spaces.Box(-np.inf, np.inf, shape = (NUM_INTRUDERS,), dtype=np.float64),
                "y_difference_speed": spaces.Box(-np.inf, np.inf, shape = (NUM_INTRUDERS,), dtype=np.float64),
                "waypoint_distance": spaces.Box(-np.inf, np.inf, shape = (NUM_WAYPOINTS,), dtype=np.float64),
                "cos_drift": spaces.Box(-np.inf, np.inf, shape = (NUM_WAYPOINTS,), dtype=np.float64),
                "sin_drift": spaces.Box(-np.inf, np.inf, shape = (NUM_WAYPOINTS,), dtype=np.float64)

        self.action_space = spaces.Box(-1, 1, shape=(1,), dtype=np.float64)

        assert render_mode is None or render_mode in self.metadata["render_modes"]
        self.render_mode = render_mode

        # initialize bluesky as non-networked simulation node
        bs.init(mode='sim', detached=True)

        # initialize dummy screen and set correct sim speed
        bs.scr = ScreenDummy()
        bs.stack.stack('DT 5;FF')

        # initialize values used for logging -> input in _get_info
        self.total_reward = 0
        self.total_intrusions = 0
        self.average_drift = np.array([])

        self.window = None
        self.clock = None
```

5.1.3. Función reset(self, ...)

Esta función tiene la tarea de reiniciar la simulación, es decir limpia el escenario utilizando bs.traf.reset() que elimina los componentes de la simulación anterior, además restablece las puntuaciones a 0 para posteriormente generar la aeronave principal controlada por el agente, las naves intrusas y finalmente los waypoint.

```
def reset(self, seed=None, options=None):
    super().reset(seed=seed)

    bs.traf.reset()

    self.total_reward = 0
    self.total_intrusions = 0
    self.average_drift = np.array([])

    bs.traf.cre('KL001',actype="A320",acspd=AC_SPD)

    self._generate_conflicts()
    self._generate_waypoint()
    observation = self._get_obs()
    info = self._get_info()

    if self.render_mode == "human":
        self._render_frame()

    return observation, info
```

5.1.4. Función step(self, action)

Esta parte se dedica a hacer avanzar la simulación, recibe la acción del agente para posteriormente aplicar la decisión en el mundo y calcular una puntuación obtenida en ese paso. Devuelve el nuevo estado de la simulación, la recompensa y “terminated” que indica si el episodio ha finalizado.

```
def step(self, action):

    self._get_action(action)

    action_frequency = ACTION_FREQUENCY
    for i in range(action_frequency):
        bs.sim.step()
        if self.render_mode == "human":
            observation = self._get_obs()
            self._render_frame()

    observation = self._get_obs()
    reward, terminated = self._get_reward()

    info = self._get_info()

    # bluesky reset?? bs.sim.reset()
    if terminated:
        for acid in bs.traf.id:
            idx = bs.traf.id2idx(acid)
            bs.traf.delete(idx)

    return observation, reward, terminated, False, info
```

5.1.5. Función generate_conflicts y función generate_waypoint (self, acid = 'KL001'):

Este segmento es uno de los más interesantes porque la primera parte se encarga de generar las naves intrusas u obstáculos y la segunda parte se encarga de generar los waypoint que son los objetivos que debe alcanzar el agente. Permite añadir dificultad a al escenario como generar nuevos objetivos y normas al momento de realizar la simulación y por lo tanto será una de las partes modificadas más adelante para intentar hacer que se asemeje más a una situación real.

```
def _generate_conflicts(self, acid = 'KL001'):
    target_idx = bs.traf.id2idx(acid)
    for i in range(NUM_INTRUDERS):
        dpsi = np.random.randint(45,315)
        cpa = np.random.randint(0,INTRUSION_DISTANCE)
        tlosh = np.random.randint(100,1000)
        bs.traf.creconfs(acid=f'{i}',actype="A320",targetidx=target_idx,dpsi=dpsi,dcpa=cpa,tlosh=tlosh)

def _generate_waypoint(self, acid = 'KL001'):
    self.wpt_lat = []
    self.wpt_lon = []
    self.wpt_reach = []
    for i in range(NUM_WAYPOINTS):
        wpt_dis_init = np.random.randint(WAYPOINT_DISTANCE_MIN, WAYPOINT_DISTANCE_MAX)
        wpt_hdg_init = 0

        ac_idx = bs.traf.id2idx(acid)

        wpt_lat, wpt_lon = fn.get_point_at_distance(bs.traf.lat[ac_idx], bs.traf.lon[ac_idx], wpt_dis_init, wpt_hdg_init)
        self.wpt_lat.append(wpt_lat)
        self.wpt_lon.append(wpt_lon)
        self.wpt_reach.append(0)
```

5.1.6. Función get_obs(self):

Se podría considerar los “sentidos” del agente, pues su función es traducir el estado en "crudo" de la simulación a la "percepción" que la AI entiende. Para cada intruso y waypoint, calcula la distancia y los ángulos, posteriormente transforma estos datos al formato definido en observation_space y normaliza los valores dividiendo las distancias y velocidades por valores máximos (WAYPOINT_DISTANCE_MAX, AC_SPD). Esto mantiene todos los datos de entrada en un rango similar entre 0 y 1 fundamental para que el algoritmo de aprendizaje funcione.

```
def _get_obs(self):
    ac_idx = bs.traf.id2idx('KL001')

    self.intruder_distance = []
    self.cos_bearing = []
    self.sin_bearing = []
    self.x_difference_speed = []
    self.y_difference_speed = []

    self.waypoint_distance = []
    self.wpt_qdr = []
    self.cos_drift = []
    self.sin_drift = []
    self.drift = []

    self.ac_hdg = bs.traf.hdg[ac_idx]
```

5.1.7. Función get_info, get_reward, check_waypoint, check_drift y check_intrusion (self):

Estas funciones sirven como el sistema de puntuación, calculando aspectos como la desviación respecto al objetivo, si ha alcanzado un waypoint o si ha invadido el margen de seguridad de una aeronave intrusa todas estas puntuaciones se suman dentro de "get_reward" para obtener la puntuación que ha obtenido en ese paso.

```
def _get_info(self):
    # Here you implement any additional info that you want to return after a step,
    # but that should not be used by the agent for decision making, so used for logging and debugging purposes
    # for now just have 10, because it crashed if I gave none for some reason.
    return {
        'total_reward': self.total_reward,
        'total_intrusions': self.total_intrusions,
        'average_drift': self.average_drift.mean()
    }

def _get_reward(self):

    # Always return done as false, as this is a non-ending scenario with
    # new waypoints spawning continously

    reach_reward = self._check_waypoint()
    drift_reward = self._check_drift()
    intrusion_reward = self._check_intrusion()

    total_reward = reach_reward + drift_reward + intrusion_reward
    self.total_reward += total_reward

    if 0 in self.wpt_reach:
        return total_reward, 0
    else:
        return total_reward, 1
```

5.1.8. Función get_action(self,action):

Esta parte actúa como un traductor entre la decisión abstracta de la AI y el comando concreto que entiende el simulador de vuelo. Para ello lo hace de la siguiente manera, toma la decisión normalizada de la AI (un valor entre -1 y 1), la convierte en un cambio de rumbo específico en grados, calcula el nuevo rumbo absoluto y envía esta orden final al simulador para que la aeronave la ejecute.

```
def _get_action(self,action):
    action = self.ac_hdg + action * D_HEADING


    bs.stack.stack(f"HDG KL001 {action[0]}")
```

5.1.9. Función render_frame(self):

Render_frame se encarga de dibuja el estado actual de la simulación en una ventana para que podamos observarla, toma como referencia la aeronave controlada por el agente situándola en el centro y moviéndose según se mueva la aeronave.

5.2. Creación de un escenario complejo

Si bien podemos modificar las variables de entorno de todos los escenarios para nuestras simulaciones, también podemos crear escenarios complejos, o modificar las bases de uno existente. En este caso se ha modificado el escenario "horizontal_cr_env" anteriormente analizado, con el objetivo de asemejarlo más al radar de un controlador aéreo, en este caso no solo se ha modificado el apartado visual, sino que también se han creado caminos "airway" que conectan los distintos puntos "waypoints", al crear los caminos también se debe crear un valor y unas condiciones, estos caminos no son aleatorios sino que siempre deben estar conectados y hacer de enlace con los distintos puntos, para calcular y crear estos caminos se ha tenido que crear una nueva función en el archivo "functions" para que los "airway" no se crearan de manera aleatoria como si fueran "waypoints", además también se le debe asignar una recompensa para que el agente identifique que debe ir por estas rutas establecidas, al mismo tiempo están las aeronaves intrusas que debe evitar, para evitar que el agente prefiera mantenerse por la ruta por encima de evitar una colisión con una aeronave ajena se ha decidido aumentar la reducción de puntos en caso de acercamiento y también en caso de colisión, también se ha decidido comenzar la simulación fuera de una vía aérea para que el agente intente aproximarse lo más rápido posible a una. Para un mayor entendimiento vamos a pasar a la explicación del código paso por paso.

5.2.1. Explicación de los nuevos parámetros globales y recompensas/penalizaciones

Después de llamar las bibliotecas necesarias debemos definir los parámetros principales o también denominados parámetros de entorno, están conformado por constantes que actúan como reglas de la simulación, en nuestro caso hemos utilizado 14 parámetros de entorno, pero las principales son:

- NUM_INTRUDERS: Define cuántas otras aeronaves habrá en el cielo junto a la nuestra.
- NUM_WAYPOINTS: El número de puntos de referencia que conforman la ruta a seguir.
- NUM_CONNECTIONS_PER_WAYPOINT: Indica cuántas aerovías (conexiones) debe tener cada waypoint, creando una red de rutas.
- AC_SPD y INTRUDER_SPD_RANGE: Definen las velocidades de nuestra aeronave y de las demás.
- MIN_SEP_DISTANCE y COLLISION_DISTANCE: Son las distancias de seguridad. Si un intruso se acerca más que MIN_SEP_DISTANCE, es una infracción. Si se acerca más que COLLISION_DISTANCE, se considera una colisión.

A su vez debemos definir las recompensas y penalizaciones siendo esta una de las partes más importantes debido a que afectan al aprendizaje de la AI diciéndole lo que hace mal o bien. Como el objetivo es que el agente pase por los distintos waypoints en un tiempo

de 150 segundo he decidido otorgar al agente una recompensa de 10 puntos cada vez que pase por algún nodo, siendo esta la recompensa más alta, sin embargo es recomendable que llegue a los nodos siguiendo las aerovías generadas, debido a que en caso de no hacerlo tiene una resta de su puntuación de -0,1 por cada paso que efectúe fuera de una aerovía, además para facilitar que el agente entienda que debe continuar sobre estas vías se le ha puesto una penalización al momento de salirse de una de -0,5, por lo tanto el agente pasaría de sumar 0,5 puntos por cada paso que se mantenga dentro de la aerovía a perder inmediatamente esa cantidad y posteriormente perder -0,1 por cada paso que realice fuera de la misma. Finalmente hemos querido recalcar y priorizar la seguridad, por lo tanto, si el agente no mantiene una distancia de seguridad respecto al resto de aeronaves se le penalizará con -20 puntos y si llega a colisionar perderá -100 durante la simulación, lo que facilitará que el agente priorice evitar colisiones y romper la distancia de seguridad incluso si para ello debe salirse de la ruta o debe esperar para pasar por uno de los nodos.

```
# --- Environment Parameters ---
WINDOW_SIZE = 512 # Size of the display window
NUM_INTRUDERS = 4 # Number of intruder aircraft
NUM_WAYPOINTS = 6 # Number of waypoints defining the main airway
MIN_SEP_DISTANCE = 5 # Minimum separation distance (NM) for intrusion
INTRUSION_DISTANCE = 5 # NM
ACTION_FREQUENCY = 10 # Frequency of action updates (Bluesky simulation steps per agent step)
AC_SPD = 500 # Aircraft speed (kt) for ownship (AUMENTADO)
INTRUDER_SPD_RANGE = (400, 600) # Speed range for intruders (AUMENTADO)
HEADING_CHANGE_MAX = 30 # Maximum heading change allowed (degrees)
AIRWAY_SEGMENT_LENGTH_MIN = 15 # Minimum length of an airway segment (NM)
AIRWAY_SEGMENT_LENGTH_MAX = 50 # Maximum length of an airway segment (NM)
COLLISION_DISTANCE = 2 # Distance for a "crash" (NM)
NUM_CONNECTIONS_PER_WAYPOINT = 3 #Controla cuántas aerovías tiene cada waypoint.

NM2KM = 1.852 # Nautical miles to kilometers conversion

# --- Rewards and Penalties ---
REWARD_REACH_WAYPOINT = 10.0
PENALTY_INTRUSION = -20.0 # Increased penalty for penalty_intrusion
PENALTY_DRIFT = -0.1 # Penalty for drifting off the airway (se mantiene)
PENALTY_CRASH = -100.0 # High penalty for collision
REWARD_ON_AIRWAY = 0.5 # Recompensa por mantenerse en la aerovía
PENALTY_OFF_AIRWAY = -0.5 # Penalización por salirse de la aerovía
AIRWAY_WIDTH_NM = 5 # Ancho de la aerovía para considerar si el avión está "en ella"
```

5.2.2. Primer prototipo

Para este primer prototipo aparte de modificar y añadir valores de entorno vamos a definir el concepto de aerovía, en este caso hemos modificado la función “generate_waypoint”, para que cree líneas que serán airway, la forma más fácil de generar estas vías es guardando las coordenadas del último waypoint generado para posteriormente compararla con el nuevo waypoint y de esta forma crear una línea recta de punto a punto, de esta manera nos aseguramos que cada nodo este conectado con otro y permite formar una aerovía que conecta todos los puntos que el agente debe intentar seguir.

```
# Guardar el segmento de aerovía (del punto anterior al actual)
if _ > 0:
    self.airway_segments.append(
        ((self.waypoint_coords[_ - 1][0], self.waypoint_coords[_ - 1][1]), # lat, lon del punto anterior
         (next_lat, next_lon)) # lat, lon del punto actual
    )

last_lat, last_lon = next_lat, next_lon
```

También debemos modificar el sistema de recompensas y penalizadores, en mi caso he decidido centralizar todo en la función “calculate_reward” y generar mediante condicionantes si el agente recibe recompensa o penalización.

```
def _calculate_reward(self):
    reward = 0
    ownship_lat = bs.traf.lat[self.ownship_idx]
    ownship_lon = bs.traf.lon[self.ownship_idx]

    # Recompensa/Penalización por mantenerse en la aerovía
    if self.airway_segments:
        min_dist_to_airway = float('inf')
        for segment_start, segment_end in self.airway_segments:
            dist_km_to_segment = fn.distance_point_to_segment(
                np.array([ownship_lat, ownship_lon]),
                np.array(segment_start),
                np.array(segment_end)
            )
            min_dist_to_airway = min(min_dist_to_airway, dist_km_to_segment / NM2KM) # Convertir a NM

        if min_dist_to_airway <= AIRWAY_WIDTH_NM / 2: # Dentro de la mitad del ancho de la aerovía
            reward += REWARD_ON_AIRWAY
        else:
            reward += PENALTY_OFF_AIRWAY * (min_dist_to_airway - AIRWAY_WIDTH_NM / 2) # Penalización creciente
```

Sin embargo, este método no nos sirve al momento de escalar a más de una aerovía, debido a que la forma de crearlas es en base al punto anterior y al actuar y por lo tanto no tiene en cuenta otros puntos ya creados.

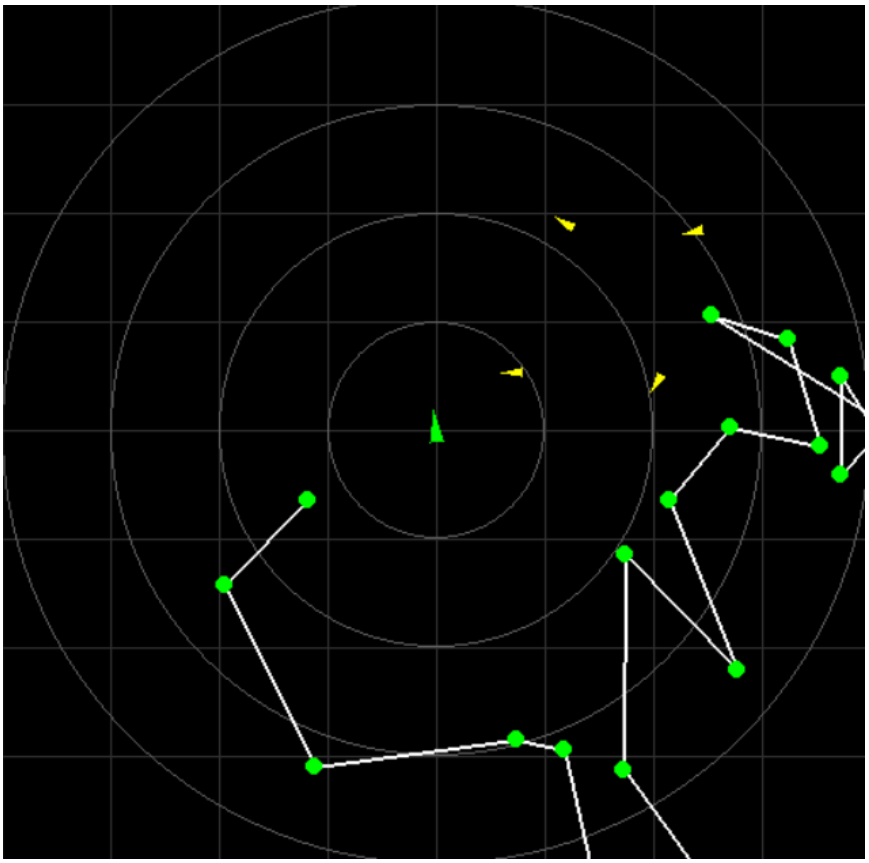


Figura 20: Captura de la simulación del primer prototipo.

5.2.3. Segundo prototipo (Más complejo)

En este segundo escenario se ha vuelto a modificar totalmente la función "generate_waypoint" y la hemos renombrado, esta vez la función crea primero una nube de puntos, es decir, primero genera las coordenadas de todos los waypoints de la simulación.

```
def _generate_waypoints_and_airways(self):
    self.waypoint_coords = []
    self.airway_segments = []

    if self.ownship_idx != -1:
        last_lat, last_lon = bs.traf.lat[self.ownship_idx], bs.traf.lon[self.ownship_idx]
    else:
        last_lat, last_lon = 0.0, 0.0

    # 1. Generar todas las coordenadas de los waypoints primero
    current_lat, current_lon = last_lat, last_lon
    for _ in range(NUM_WAYPOINTS):
        distance_nm = random.uniform(AIRWAY_SEGMENT_LENGTH_MIN, AIRWAY_SEGMENT_LENGTH_MAX)
        bearing_deg = random.uniform(0, 360)

        next_lat, next_lon = fn.get_point_at_distance(current_lat, current_lon, distance_nm * NM2KM, bearing_deg)
        self.waypoint_coords.append((next_lat, next_lon))


        current_lat, current_lon = next_lat, next_lon

    # 2. Generar la red de aerovías usando la nueva función externa

    if self.waypoint_coords:
        self.airway_segments = fn.generate_airway_network(
            self.waypoint_coords,
            self.num_connections
```

Una vez tiene esas coordenadas las pasa a una función externa que he creado dentro del archivo "functions", esta función toma todos los waypoints generados y los conecta para formar una red. Lo hace en dos fases, la primera calcula todas las posibles conexiones, toma un punto y comienza a crear caminos con los puntos más cercanos hasta alcanzar el limite de caminos previamente indicado, momento que pasará al siguiente punto y así hasta finalizar con todos.

```
def generate_airway_network(waypoints: list, num_connections_per_waypoint: int) -> list:
    """
    Genera una red de aerovías conectando los waypoints más cercanos.

    Args:
        waypoints (list): Una lista de tuplas, donde cada tupla es la coordenada (lat, lon) de un waypoint.
        num_connections_per_waypoint (int): El número deseado de conexiones para cada waypoint.

    Returns:
        list: Una lista de segmentos de aerovía. Cada segmento es una tupla con las coordenadas
              de inicio y fin, ej: [((lat1, lon1), (lat2, lon2)), ...].
    """
    num_waypoints = len(waypoints)
    if num_connections_per_waypoint == 0 or num_waypoints < 2:
        return []

    # 1. Calcular todas las posibles aristas (conexiones) y sus distancias
    possible_edges = []
    for i in range(num_waypoints):
        for j in range(i + 1, num_waypoints):
            point_i = waypoints[i]
            point_j = waypoints[j]
            distance = haversine_distance(point_i[0], point_i[1], point_j[0], point_j[1])
            possible_edges.append((i, j, distance))

    # Ordenar las aristas de la más corta a la más larga
    possible_edges.sort(key=lambda x: x[2])

    # 2. Construir las aerovías
    airway_segments = []
    airway_indices = set() # Usamos un set de tuplas ordenadas para evitar duplicados (i, j)
    connection_counts = np.zeros(num_waypoints, dtype=int)

    # --- Fase 1: Conectar los waypoints más cercanos hasta alcanzar el límite deseado ---
    for i, j, distance in possible_edges:
        if connection_counts[i] < num_connections_per_waypoint and \
           connection_counts[j] < num_connections_per_waypoint:

            connection_counts[i] += 1
            connection_counts[j] += 1
            segment = (waypoints[i], waypoints[j])
            airway_segments.append(segment)
            airway_indices.add(tuple(sorted((i, j))))
```

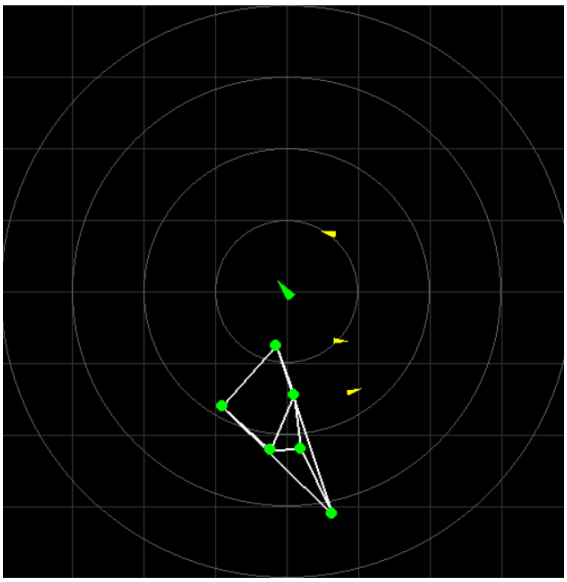
La segunda fase de creación de la red de caminos se basa en buscar si existen puntos aislados, si localiza algún punto sin conexiones creará una conexión con el punto más cercano a este, durante este proceso elimina el camino más lejano que tuviera el punto y añade el nuevo camino, una vez resuelta la incidencia buscará más puntos aislados hasta que no haya ninguno y devolverá el mapa de puntos y aerovías. Cabe destacar que el código intenta que siempre se cumpla la condición de caminos que se haya especificado, sin embargo, hay situaciones donde la combinación de puntos y la exigencia de aerovías hace que esta tarea sea imposible, por ejemplo 3 carriles por punto y una generación de 5 puntos, en estos casos el programa intentará cumplir con el máximo de puntos posibles y aquellos que no sea posible intentará acercarse a lo estipulado.

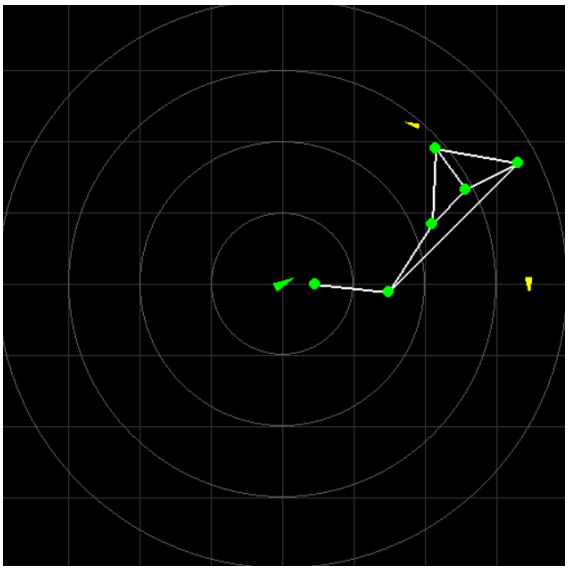
```
# --- Fase 2: Asegurar que no queden waypoints aislados ---
# Un waypoint podría quedar aislado si todos sus vecinos ya alcanzaron su límite de conexiones.
isolated_waypoints_indices = np.where(connection_counts == 0)[0]

for idx in isolated_waypoints_indices:
    # Encontrar el vecino más cercano para este waypoint aislado, sin importar si el vecino ya está lleno
    closest_neighbor_found = False
    for i_neighbor, j_neighbor, dist in possible_edges:
        if i_neighbor == idx:
            # Comprobar si esta conexión ya existe
            if tuple(sorted((idx, j_neighbor))) not in airway_indices:
                connection_counts[idx] += 1
                connection_counts[j_neighbor] += 1
                segment = (waypoints[idx], waypoints[j_neighbor])
                airway_segments.append(segment)
                airway_indices.add(tuple(sorted((idx, j_neighbor))))
                closest_neighbor_found = True
                break # Conectar y pasar al siguiente aislado
        elif j_neighbor == idx:
             if tuple(sorted((idx, i_neighbor))) not in airway_indices:
                connection_counts[idx] += 1
                connection_counts[i_neighbor] += 1
                segment = (waypoints[idx], waypoints[i_neighbor])
                airway_segments.append(segment)
                airway_indices.add(tuple(sorted((idx, i_neighbor))))
                closest_neighbor_found = True
                break # Conectar y pasar al siguiente aislado

return airway_segments
```

En estas imágenes de ejemplo se puede apreciar el resultado, situandose la nave que controla el agente en el centro con forma de flecha verde, la de intrusos con forma de flecha amarilla, los puntos como circulos verdes y las aerovias como lineas blencas.





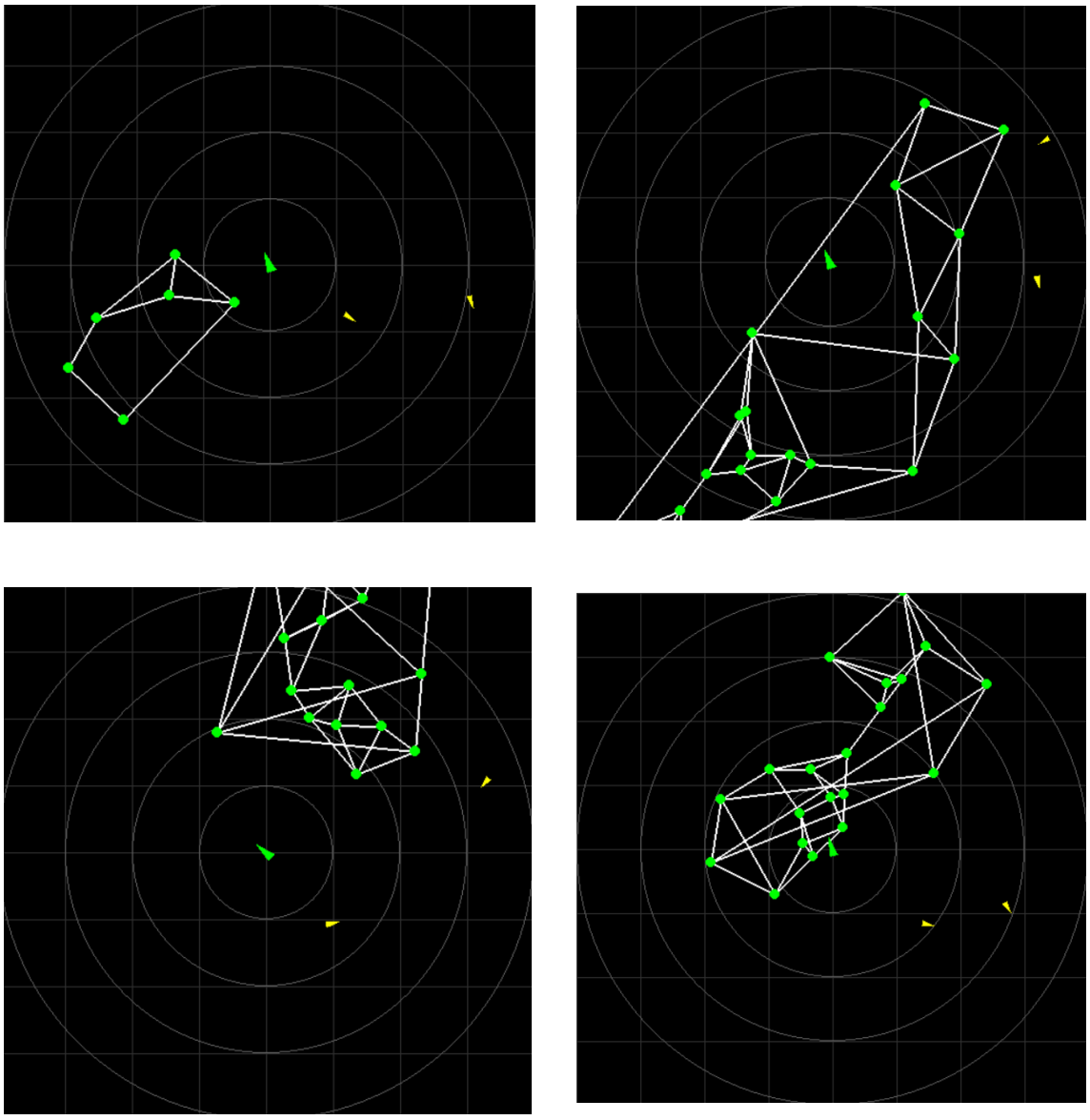


Figura 21: Capturas de distintas simulaciones del segundo prototipo.

6. Limitaciones y propuestas de mejora

Si bien este proyecto demuestra el enorme potencial de la inteligencia artificial en el ámbito aeronáutico, su aplicación práctica en el control de tráfico aéreo (ATC) no está exenta de limitaciones críticas que, hoy en día, comprometen la viabilidad de un despliegue autónomo.

Es fundamental matizar que el siguiente análisis se centra específicamente en la metodología de Aprendizaje por Refuerzo (RL), utilizada en el entorno de simulación BlueSky-Gym, sin profundizar en otros paradigmas de AI que podrían ofrecer soluciones diferentes.

A continuación, se detallan las limitaciones observadas y se expande el análisis a otros desafíos inherentes a esta tecnología.

1. Generalización ante Escenarios Inéditos (Falta de Improvisación)

Un agente entrenado con RL se vuelve extraordinariamente eficiente en la gestión de configuraciones de tráfico para las que ha sido entrenado. Sin embargo, su rendimiento se degrada drásticamente ante eventos imprevistos o no incluidos en su set de entrenamiento (lo que se conoce como out-of-distribution data). Esto puede incluir:

- Un tipo de aeronave con un rendimiento de ascenso/descenso no visto antes.
- Una condición meteorológica adversa y localizada que aparece súbitamente.
- El comportamiento inesperado de un piloto (ej. una desviación no comunicada).

El agente no "improvisa", simplemente carece del modelo aprendido para gestionar una situación que no ha experimentado, lo que en un entorno real es inaceptable.

Propuestas de Mejora:

Domain Randomization (Aleatorización del Dominio): Entrenar al agente no en un único escenario estático, sino en miles de simulaciones donde los parámetros (meteorología, rendimiento de las aeronaves, densidad del tráfico, etc.) varían constantemente y de forma aleatoria. Esto fuerza al agente a aprender políticas de decisión más robustas y generalizables [28].

Curriculum Learning (Aprendizaje Curricular): Exponer al agente a escenarios de complejidad creciente. Se empieza con problemas sencillos (dos aviones, sin viento) y, a medida que el agente los domina, se introduce gradualmente más dificultad y variabilidad [28].

Transfer Learning: Pre-entrenar modelos en un vasto conjunto de datos de tráfico aéreo general (incluso de otros sectores o aeropuertos) y luego afinarlos (fine-tuning) para el sector específico en el que operarán [29].

2. El Equilibrio entre Exploración y Explotación

Un agente que rápidamente encuentra una estrategia "suficientemente buena" (explotación) dejará de buscar alternativas que podrían ser óptimas a largo plazo

(exploración). En el control aéreo, esto es peligroso. Una solución que parece eficiente puede ser frágil y fallar ante una pequeña perturbación que no fue explorada durante el entrenamiento. Forzar una exploración exhaustiva en un espacio de estados tan vasto como el del tráfico aéreo puede llevar a tiempos de entrenamiento computacionalmente prohibitivos.

Propuestas de Mejora:

Algoritmos de Exploración Avanzados: Utilizar técnicas como Intrinsic Curiosity Motivation, donde el agente recibe una recompensa adicional no solo por cumplir el objetivo, sino por visitar estados nuevos y desconocidos. Esto incentiva una exploración más sistemática [30].

Ensembles de Agentes: Entrenar a múltiples agentes de forma independiente. Durante la operación, sus decisiones pueden ser promediadas o sometidas a un sistema de votación, reduciendo el riesgo de que una única política subóptima tome el control [30].

3. El Problema de la "Caja Negra" (Explainability) y la Certificación de Seguridad

Las redes neuronales profundas, que son el cerebro de los agentes de RL modernos, operan como "cajas negras". Pueden tomar una decisión óptima, pero es extremadamente difícil (a veces imposible) trazar el razonamiento exacto que los llevó a ella. En aviación, toda decisión crítica debe ser auditable y explicable. ¿Por qué el agente decidió desviar el avión A en lugar del B? Sin una respuesta clara, es imposible certificar el sistema bajo los estrictos estándares de la aviación (como DO-178C).

Propuestas de Mejora:

AI Explicable (XAI - Explainable AI): Desarrollar e integrar herramientas que "traduzcan" las decisiones del agente a un formato comprensible para un humano. Por ejemplo, mediante mapas de atención que resalten qué aviones o datos fueron más influyentes para una decisión concreta [31].

Modelos Híbridos: Combinar el RL con sistemas basados en reglas o lógicas simbólicas. El agente de RL puede proponer una estrategia, pero esta debe ser validada por un "guardián" basado en reglas (por ejemplo, "nunca violar la separación mínima de 5 millas náuticas"). Esto crea una red de seguridad verificable [31].

Verificación Formal: Utilizar métodos matemáticos para probar formalmente que, bajo cualquier circunstancia dentro de un conjunto definido, el agente nunca tomará una acción que lleve a un estado inseguro [31]

4. Escalabilidad y Complejidad del Espacio Aéreo Real

Un simulador como BlueSky-Gym, aunque avanzado, simplifica la realidad. El espacio aéreo real es un sistema multi-agente masivo. No se trata solo de gestionar un sector, sino de coordinar traspasos fluidos con docenas de sectores adyacentes, cada uno con su propio controlador (o agente). La complejidad computacional (el "curse of dimensionality") crece exponencialmente con cada avión y cada agente añadido.

Propuestas de Mejora:

Aprendizaje por Refuerzo Jerárquico (HRL): Diseñar una estructura de agentes de varios niveles. Un "meta-agente" de alto nivel podría tomar decisiones estratégicas (ej. gestionar el flujo general de un corredor aéreo), mientras que agentes de bajo nivel se encargarían de tareas tácticas (ej. mantener la separación entre un par de aviones) [33].

Aprendizaje Multi-Agente (MARL): En lugar de un único agente omnisciente, entrenar a múltiples agentes que aprendan a cooperar y comunicarse entre sí, imitando la estructura de los centros de control del mundo real [33].

5. Diseño de la Función de Recompensa (Reward Hacking)

Definir qué es una "buena" gestión del tráfico aéreo en una fórmula matemática (la función de recompensa) es increíblemente difícil. Un agente de RL es un optimizador implacable y podría encontrar lagunas o atajos para maximizar su recompensa de maneras no deseadas (reward hacking). Por ejemplo, si la recompensa se basa únicamente en la eficiencia del combustible, el agente podría guiar a los aviones por rutas muy juntas, justo en el límite legal de separación, aumentando el riesgo para maximizar su puntuación.

Propuestas de Mejora:

Aprendizaje por Refuerzo Inverso (IRL - Inverse Reinforcement Learning): En lugar de definir manualmente la recompensa, el agente la aprende observando a controladores aéreos humanos expertos. Intenta deducir cuál es la función de recompensa implícita que guía las decisiones humanas [28].

Funciones de Recompensa Multiobjetivo: Crear una función de recompensa que equilibre múltiples objetivos, a menudo contrapuestos: seguridad (máxima separación), eficiencia (rutas directas), puntualidad, y confort del pasajero (evitar virajes bruscos). Asignar penalizaciones severas por cualquier acción que se acerque a un límite de seguridad [28].

7. Conclusión

A lo largo de este trabajo, se ha profundizado en los fundamentos teóricos del Aprendizaje por Refuerzo (RL), sentando las bases para su aplicación práctica en la gestión del tráfico aéreo. La revisión de la literatura y el estudio de algoritmos clave como PPO o SAC fueron pasos indispensables para abordar el núcleo del proyecto: la exploración del simulador BlueSky-Gym. Para facilitar su adopción por parte de otros investigadores y estudiantes, se desarrolló un pequeño tutorial, cumpliendo con uno de los objetivos clave de este trabajo y contribuyendo a hacer esta herramienta un poquito más accesible.

La fase experimental no solo se limitó a evaluar los escenarios predefinidos, sino que se extendió al diseño y creación de un entorno de simulación propio y más complejo, que modela una red de aerovías para reflejar condiciones más realistas. Los resultados de los experimentos muestran tanto las oportunidades como los retos que presenta el aprendizaje por refuerzo (RL) en este campo. En situaciones simples como PlanWaypointEnv-v0, se evidenció que el agente puede aprender políticas de navegación efectivas, aunque esto viene con un alto costo computacional y una notable sensibilidad a los hiperparámetros. La complejidad aumentó en los escenarios de resolución de conflictos y en nuestro entorno personalizado, donde la inestabilidad de las métricas destacó la dificultad del agente para generalizar su aprendizaje en condiciones cambiantes. Estos hallazgos confirman que, aunque el RL es funcional, su eficiencia depende en gran medida de una configuración cuidadosa y de la complejidad del entorno.

De cara al futuro, la aplicación de inteligencias artificiales como las exploradas en este proyecto se pueden considerar más una herramienta de asistencia más que como un sustituto autónomo del controlador humano. Las limitaciones observadas, especialmente la falta de capacidad para improvisar ante escenarios no previstos y la naturaleza de "caja negra" de las redes neuronales, representan barreras significativas para su certificación en un entorno donde la seguridad es innegociable. La viabilidad de una implementación real dependerá de superar estos escollos, posiblemente a través de arquitecturas híbridas que combinen el RL con sistemas basados en reglas y el desarrollo de la IA Explicable (XAI). Sin embargo, su aplicación se podría aplicar como un asistente para el controlador, de forma que facilite y agilice las operaciones aéreas. Por tanto, aunque la visión de un ATC totalmente automatizado sigue siendo un horizonte lejano, este trabajo confirma que BlueSky-Gym es un paso más y una plataforma de investigación necesaria para instar a los estudiantes a forjar los sistemas inteligentes que harán de la aviación del mañana un espacio más seguro y eficiente.

Anexo

Código inicial:

```
import gymnasium as gym
from stable_baselines3 import PPO, SAC, TD3, DDPG
import numpy as np
import bluesky_gym
import bluesky_gym.envs
from bluesky_gym.utils import logger
bluesky_gym.register_envs()
env_name = 'StaticObstacleEnv-v0'
algorithm = SAC
# Initialize logger
log_dir = f'./logs/{env_name}/'
file_name = f'{env_name}_{str(algorithm.__name__)}.csv'
csv_logger_callback = logger.CSVLoggerCallback(log_dir, file_name)
TRAIN = True
EVAL_EPISODES = 10
if __name__ == "__main__":
    env = gym.make(env_name, render_mode=None)
    obs, info = env.reset()
    model = algorithm("MultiInputPolicy", env, verbose=1,learning_rate=3e-4)
    if TRAIN:
        model.learn(total_timesteps=2e6, callback=csv_logger_callback)

model.save(f"models/{env_name}/{env_name}_{str(algorithm.__name__)}/model")
        del model
    env.close()
```

```
# Test the trained model
    model = algorithm.load(f"models/{env_name}/{env_name}_{str(algorithm.__name__)}/model", env=env)
    env = gym.make(env_name, render_mode="human")
    for i in range(EVAL_EPISODES):
        done = truncated = False
        obs, info = env.reset()
        tot_rew = 0
        while not (done or truncated):
            # action = np.array(np.random.randint(-100,100,size=(2))/100)
            # action = np.array([0,-1])
            action, _states = model.predict(obs, deterministic=True)
            obs, reward, done, truncated, info = env.step(action[()])
            tot_rew += reward
        print(tot_rew)
    env.close()
```

Código tras primera revisión:

```
import gymnasium as gym
from stable_baselines3 import SAC
from stable_baselines3.common.env_util import make_vec_env
from stable_baselines3.common.vec_env import VecNormalize
from stable_baselines3.common.callbacks import CheckpointCallback
import numpy as np
import bluesky_gym
import bluesky_gym.envs
from bluesky_gym.utils import logger
# Registrar los entornos de Bluesky
bluesky_gym.register_envs()
# Definir el nombre del entorno y el algoritmo
env_name = 'PlanWaypointEnv-v0'
algorithm = SAC
# Crear múltiples entornos en paralelo
n_envs = 4
env = make_vec_env(env_name, n_envs=n_envs, seed=0)
# Normalizar las observaciones y recompensas
env = VecNormalize(env, norm_obs=True, norm_reward=True)
# Inicializar el logger
log_dir = f'./logs/{env_name}/'
file_name = f'{env_name}_{str(algorithm.__name__)}.csv'
csv_logger_callback = logger.CSVLoggerCallback(log_dir, file_name)
# Configuración de entrenamiento y evaluación
TRAIN = True  # Cambia a False si no deseas entrenar
EVAL_EPISODES = 10
if __name__ == "__main__":
    # Cargar o crear el modelo
```

```
    model_path = f"models/{env_name}/{env_name}_{str(algorithm.__name__)}/model"
    if TRAIN:
        try:
            # Intentar cargar el modelo previamente entrenado
            model = algorithm.load(model_path, env=env)
            print("Modelo cargado exitosamente.")
        except:
            # Si no existe, crear un nuevo modelo
            model = algorithm(
                "MultiInputPolicy",
                env,
                verbose=1,
                learning_rate=1e-3,  # Aumentar la tasa de aprendizaje
                buffer_size=1_000_000,  # Aumentar el tamaño del buffer
                batch_size=256,  # Aumentar el tamaño del batch
                ent_coef=0.1,  # Ajustar el coeficiente de entropía
                tau=0.01)  # Aumentar tau para actualizar el target network más rápido
            print("Nuevo modelo creado.")
        # Callback para guardar el modelo periódicamente
        checkpoint_callback = CheckpointCallback(
            save_freq=10_000,
            save_path="./checkpoints/",
            name_prefix=f"{env_name}_{str(algorithm.__name__)}")
        # Entrenar el modelo
        model.learn(total_timesteps=500000, callback=[csv_logger_callback, checkpoint_callback], log_interval=10)
        # Guardar el modelo entrenado
        model.save(model_path)
        print("Modelo guardado exitosamente.")
```

```
# Cerrar el entorno de entrenamiento
env.close()
# Evaluar el modelo entrenado
env = gym.make(env_name, render_mode="human")
model = algorithm.load(model_path, env=env)  # Cargar el modelo para evaluación
for i in range(EVAL_EPISODES):
    done = truncated = False
    obs, info = env.reset()
    tot_rew = 0
    while not (done or truncated):
        action, _states = model.predict(obs, deterministic=True)
        obs, reward, done, truncated, info = env.step(action[()])
        tot_rew += reward
    print(f"Episodio {i+1}: Recompensa total = {tot_rew}")
# Cerrar el entorno de evaluación
env.close()
```

Fichero de BlueSky-Gym utilizado durante los experimentos

bluesky-gym-main

Bibliografía

[1] M. Brittain, and P. Wei, “Autonomous Air Traffic Controller: A Deep Multi-Agent Reinforcement Learning Approach”, arXiv, 10.48550/arXiv.1905.01303, 2019.

[2] L. Lascorz, “Aprendizaje por Refuerzo: Elementos básicos y algoritmos”, Universidad de Zaragoza, TAZ-TFG-2018-2390, 2018.

[3] J. Groot, G. Leto, A. Vlaskin, A. Moec, and J. Ellerbroek, “BlueSky-gym: Reinforcement learning environments for air traffic applications”, SESAR, 10.61009/SID.2024.1.10, 2024.

[4] BlueSky, “User Documentation”, BlueSky Data Collection Framework, https://blueskyproject.io/, consultado el 19 de febrero de 2025.

[5] M. A. S. B. Affridi, and S. M. Abdul Rahman, “BlueSky simulator for air traffic control training platform”, Universiti Teknologi MARA, eISSN 2773-5494, 2020.

[6] Farama Foundation, “Documentación del gimnasio”, Gymnasium, https://gymnasium.farama.org/, consultado el 04 de marzo de 2025.

[7] W. Wu, H. Wu, and H. Zhao, “Self-Directed Turing Test for Large Language Models”, arXiv, 10.48550/arXiv.2408.09853, 2024.

[8] AENA, “Fechas e Informes Estadísticos”, Estadísticas de tráfico aéreo, https://www.aena.es/es/estadisticas/inicio.html, consultado el 17 de febrero de 2025.

[9] Python Software Foundation, “Python documentation”, https://www.python.org/, consultado el 19 de febrero de 2025.

[10] Flightradar24.com, “Aviation data”, https://www.flightradar24.com, consultado el 04 de marzo de 2025.

[11] ENAIRE, “Servicio de Información Aeronáutica”, AIP, https://aip.enaire.es/AIP/, consultado el 17 de febrero de 2025.

[12] H. Shum, X. He, and D. Li, “From Eliza to XiaoIce: Challenges and opportunities with social chatbots”, Frontiers of Information Technology & Electronic Engineering, vol. 19, pp. 10–26, 2018, doi: 10.1631/FITEE.1700826.

[13] K. H. Yu, A. L. Beam, and I. S. Kohane, “Artificial Intelligence in Healthcare”, Nature Biomedical Engineering, vol. 2, pp. 719–731, 2018, doi: 10.1038/s41551-018-0305-z.

[14] R. S. Sutton, and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed., MIT Press, 2018.

[15] V. Mnih, et al., “Human-level control through deep reinforcement learning”, Nature, vol. 518, no. 7540, pp. 529–533, 2015.

[16] D. Silver, et al., “Mastering the game of Go without human knowledge”, Nature, vol. 550, no. 7676, pp. 354–359, 2017.

[17] T. P. Lillicrap, et al., “Continuous control with deep reinforcement learning”, arXiv, 10.48550/arXiv.1509.02971, 2015.

[18] S. Zelinski, “NextGen Simulation Technologies”, NASA, https://aviationsystems.arc.nasa.gov/publications/2011/DASC2011_Zelinski.pdf, consultado el 25 de junio de 2025.

[19] A. T. Budiarti, “Development of model-free flight control system using Deep Deterministic Policy Gradient (DDPG)”, Cranfield University, tesis de maestría, 2019. Disponible en: https://dspace.lib.cranfield.ac.uk/handle/1826/14798

[20] Microsoft, “How Microsoft Flight Simulator uses AI to create its world”, Microsoft Flight Simulator Official Blog, 2020. https://news.xbox.com/en-us/2020/08/19/how-microsoft-flight-simulator-uses-ai-to-create-its-world/

[21] Blackshark.ai, “AI at the core of Microsoft Flight Simulator”, 2020. https://www.blackshark.ai/microsoft-flight-simulator

[22] Reddit.com, “Discusión sobre tráfico aéreo y simulación”, Comunidad de usuarios Microsoft Flight Simulator, 2023. https://www.reddit.com/r/MicrosoftFlightSim/

[23] MIT Lincoln Laboratory, “ACAS Xu for Unmanned Aircraft Systems”, 2020. https://www.ll.mit.edu/r-d/projects/acas-xu-unmanned-aircraft-systems

[24] NASA, “UAS Traffic Management (UTM) Project”, Aeronautics Research Mission Directorate, 2020. https://www.nasa.gov/ames/utm

[25] E. R. Mueller, and M. J. Kochenderfer, “Challenges in aircraft collision avoidance”, IEEE Transactions on Intelligent Transportation Systems, vol. 18, no. 6, pp. 1551–1559, 2016. https://doi.org/10.1109/TITS.2016.2603007

[26] MIT Lincoln Laboratory, “Fast-time Monte Carlo Simulations for Collision Avoidance Systems”, 2021. https://www.ll.mit.edu/publications/fast-time-monte-carlo-simulations

[27] FAA, “FAA Exploring Collision Avoidance Algorithms for UAS Integration”, Federal Aviation Administration, 2022. https://www.faa.gov/uas/research_development/traffic_management

[28] S. Ghosh, et al., “Deep Ensemble Multi-Agent Reinforcement Learning for Air Traffic Control”, arXiv, 10.48550/arXiv.2004.01387, 2020.

[29] L. D. Ávila, D. Aguirre, J. V. Martel, and I. Pérez, “Air Traffic Control Using Deep Reinforcement Learning: A Review”, Expert Systems with Applications, vol. 240, 2024, https://doi.org/10.1016/j.eswa.2024.122776

[30] A. Vouros, et al., “Automating the Resolution of Flight Conflicts: Deep Reinforcement Learning in Service of Air Traffic Controllers”, ResearchGate, 2022. https://www.researchgate.net/publication/362184328

[31] H. Wang, et al., “Safe and Explainable Reinforcement Learning for Autonomous Air Mobility”, arXiv, 10.48550/arXiv.2211.13474, 2022.