UNIVERSIDAD EAFIT

COMPARACIÓN DE ALGORITMOS DE CONTROL TRADICIONALES CON CONTROL POR MEDIO DE IA EN UN ENTORNO SIMULADO 2D

MATEO FERNANDO PENAGOS RAMIREZ

Proyecto de grado

Asesor, docente

Alejandro Puerta Echandía

UNIVERSIDAD EAFIT

ESCUELA DE CIENCIAS APLICADAS E INGENIERÍA

MAESTRÍA EN CIENCIAS DE LOS DATOS Y LA ANALÍTICA

MEDELLÍN

2025

CONTENIDO

Contenido

Comparación de algoritmos de control tradicionales con control por medio de IA en un entorno simulado 2D

1 Resumen

En la tecnología de automatización y control, la eficiencia de los algoritmos de control es crucial para el desempeño y la seguridad de sistemas complejos. Tradicionalmente, los controladores Proporcional-Integral-Derivativo (PID) han sido la piedra angular en la regulación de estos sistemas debido a su simplicidad y robustez, siendo utilizados en más del 90% de las aplicaciones industriales. Métodos como el PID funcionan muy bien en entornos de fácil modelado matemático y poca variabilidad, este tipo de sistemas se puede encontrar en plantas industriales y equipos de producción, sin embargo, para ambientes dinámicos y no lineales pueden presentarse problemas en su desempeño o dificultades en la sintonización de sus parámetros.

Este proyecto tiene como objetivo comparar el desempeño de los algoritmos de control tradicionales, específicamente los PID, con aquellos basados en redes neuronales entrenadas mediante técnicas de aprendizaje por refuerzo. Para ello, se desarrollará un ambiente simulado en 2D que replicará el comportamiento dinámico de un sistema no lineal, para este caso se eligió un dron en vuelo. Este ambiente no lineal permitirá evaluar ambos tipos de controladores bajo una serie de condiciones y desafíos operativos, incluyendo la estabilización, el seguimiento de trayectorias y la respuesta ante perturbaciones externas.

La investigación se enfocará en medir la precisión, la eficiencia y la capacidad de adaptación de cada algoritmo, proporcionando una base de comparación objetiva que considerará métricas clave de desempeño. Además, se analizarán las ventajas y limitaciones inherentes a cada enfoque, incluyendo la complejidad de implementación, los requisitos computacionales y la escalabilidad.

2 Descripción del Proyecto

2.1 Planteamiento del Problema

Entre los métodos tradicionales utilizados para controlar drones se encuentran el control proporcional-integral-derivativo (PID), el control de realimentación lineal (LQR), el control basado en modelos dinámicos y el control basado en navegación

y planificación de trayectorias(Tahir et al., 2023), sin embargo, más del 90% de los controladores utilizados en la industria moderna son controladores de tipo PID (Åström & Hägglund, 2001). Estos enfoques suelen basarse en algoritmos predefinidos que requieren una modelización precisa del sistema y de su entorno. Si bien estos métodos pueden proporcionar un control preciso y estable en entornos conocidos y predecibles, pueden enfrentar dificultades para adaptarse a entornos dinámicos y desconocidos, así como para lidiar con la alta variabilidad y la incertidumbre inherentes de algunos sistemas no lineales como lo es la navegación autónoma de drones.

En este contexto, el uso de técnicas de inteligencia artificial, como redes neuronales y aprendizaje por refuerzo, ofrece una alternativa prometedora para abordar los desafíos del control. Estos enfoques se adaptan fácilmente a condiciones de no linealidad, en el caso del aprendizaje por refuerzo esto se logra por medio de un agente que aprende de manera autónoma a partir de la interacción con su entorno, adaptándose y mejorando su desempeño a lo largo del tiempo. Sin embargo, estas técnicas requieren de un proceso iterativo en el cual durante muchas épocas de entrenamiento el agente será incapaz de controlarse de forma estable, es por esto por lo que resultaría muy costoso e ineficiente realizar un entrenamiento en un entorno físico real.

Aunque existen estudios que exploran ambos enfoques de control de forma aislada, hay una carencia de investigaciones que realicen una comparación detallada entre los controladores PID y las redes neuronales entrenadas por refuerzo en el contexto de plantas de control no lineales. Identificar cuál de estos métodos ofrece el mejor desempeño en términos de precisión, adaptabilidad y capacidad de respuesta ante condiciones adversas o cambiantes puede ser muy beneficioso para avanzar en el desarrollo de mejores controladores.

2.2 Justificación

En el campo de control de drones, el algoritmo de control más utilizado es el PID (Zulu & John, 2016), el cual es un algoritmo robusto y de pocos parámetros, sin embargo, este algoritmo limita las capacidades de los drones en los que se utiliza debido a que el modelo matemático de un dron es un modelo altamente no-lineal e impreciso: El rendimiento del controlador PID depende en gran medida de la precisión del modelo matemático del sistema que controla. Sin embargo, para los drones, es especialmente desafiante modelar con precisión todos los aspectos de su dinámica de vuelo. Esto se debe a las dificultades inherentes en la captura de todos los factores ambientales (como las ráfagas de viento), los efectos aerodinámicos complejos y las variaciones en la masa y la distribución de masa del vehículo durante el vuelo. Los controladores PID no pueden compensar automáticamente estos modelos imprecisos o incompletos, lo que resulta en un control menos fiable y preciso (Zulu & John, 2016).

Aquí es donde el aprendizaje por refuerzo ofrece un enfoque prometedor. A diferencia de los controladores PID, que requieren un modelo matemático preciso del sistema para funcionar eficazmente, los algoritmos de aprendizaje por refuerzo pueden operar en entornos altamente dinámicos y no lineales sin necesidad de un modelo exacto. Esto se debe a que el RL se basa en la interacción directa con el entorno para aprender la mejor estrategia de control, adaptándose continuamente a través de la experiencia (Li, 2018). Esto permite que el sistema de control de un dron aprenda y refine sus acciones en respuesta a los cambios en el entorno y la dinámica del vuelo, mejorando así la estabilidad y la precisión del control en situaciones impredecibles.

Además, el aprendizaje por refuerzo puede optimizar las políticas de control basándose en una función de recompensa, lo que permite una mejora continua del rendimiento. Esta capacidad de aprender de la experiencia y ajustarse a nuevas condiciones sin depender de modelos matemáticos predefinidos hace que el RL sea particularmente adecuado para manejar las complejidades inherentes de estos sistemas.

Se propone el desarrollo de un sistema de control de drones en un entorno simulado en 2D donde se puede realizan comparaciones entre estos modelos, y tomar mediciones precisas de su desempeño.

Al tener una simulación 2D como punto de partida, es posible expandir y agregar complejidad gradualmente, lo que facilita la transición a simulaciones de entornos más realistas. Esto permitirá la incorporación de variables adicionales, como perturbaciones dinámicas (efectos climáticos, paquetes cargados por el dron, entre otros) que son cruciales para la navegación autónoma en entornos del mundo real.

Otra razón importante para abordar este problema es la reducción del riesgo y los costos asociados con el entrenamiento de estos modelos en entornos físicos. Al utilizar una simulación, podemos realizar experimentos y entrenamientos de manera segura y eficiente, sin el riesgo de daños físicos y con la capacidad de repetir y ajustar los experimentos según sea necesario.

En resumen, el desarrollo de un sistema de control no lineal utilizando redes neuronales y aprendizaje por refuerzo en un entorno simulado en 2D es un proyecto que vale la pena explorar debido a su potencial para superar las limitaciones de los métodos tradicionales, su capacidad de escalabilidad y su capacidad para reducir el riesgo y los costos asociados con el entrenamiento de drones en entornos físicos.

2.3 Objetivos

2.3.1 Objetivo general:

Desarrollar y comparar el desempeño de algoritmos de control tradicionales, específicamente el controlador PID, contra algoritmos basados en redes neuronales

entrenadas por refuerzo para la gestión de drones en un entorno simulado 2D, con el fin de identificar el enfoque más efectivo y eficiente en términos de precisión, adaptabilidad y respuesta a perturbaciones externas.

2.3.2 Objetivos específicos:

- Desarrollar un entorno simulado en 2D que modele la dinámica y el comportamiento de un dron en vuelo, incluyendo variables como la velocidad, la altitud y la respuesta a perturbaciones externas. Este entorno será utilizado como plataforma de prueba para evaluar y comparar los diferentes algoritmos de control.

- Implementar y afinar un controlador PID tradicional para el dron en el ambiente simulado, optimizando sus parámetros para lograr la mejor estabilidad y respuesta frente a las condiciones dinámicas simuladas. Se medirá su desempeño en términos de precisión en el seguimiento de trayectorias y capacidad de manejar perturbaciones externas.

- Desarrollar y entrenar una red neuronal mediante técnicas de aprendizaje por refuerzo para controlar el dron dentro del mismo entorno simulado. Se enfocará en ajustar la red para maximizar su capacidad de adaptación y optimizar la respuesta del dron a condiciones cambiantes y escenarios no previstos durante el vuelo.

- Realizar una evaluación comparativa entre los modelos creados mediante el uso de métricas de desempeño como lo son:

    - la precisión del seguimiento de trayectorias
    - la adaptabilidad a cambios en el entorno
    - la eficiencia en el consumo de energía (uso de los motores)
    - la robustez frente a perturbaciones externas.

    Esta comparación se basará en una serie de pruebas estandarizadas dentro del entorno simulado 2D para garantizar una evaluación objetiva y justa de ambos enfoques. Los resultados serán presentados de manera clara y detallada para facilitar su análisis y la toma de decisiones respecto a la viabilidad y aplicabilidad de cada método de control en situaciones reales.

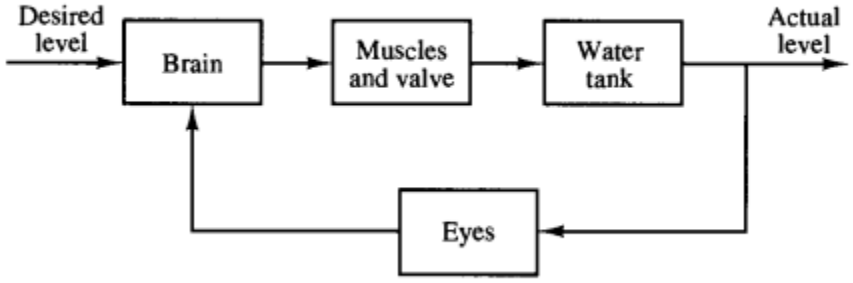
2.4 Estado del arte y Marco teórico

2.4.1 Sistemas de control

Los sistemas de control son fundamentales en la ingeniería y tecnología, ya que permiten gestionar, dirigir y regular el comportamiento de otros sistemas o dispositivos. Su objetivo es hacer que el sistema controlado se comporte de una manera deseada, a pesar de las perturbaciones o cambios en las condiciones de operación. Estos sistemas son esenciales en numerosas aplicaciones industriales, de investigación y cotidianas, abarcando desde el control de procesos químicos hasta la regulación de la temperatura en nuestros hogares.

Existen sistemas de control de lazo abierto y lazo cerrado, en los sistemas de control de lazo abierto no se toma una medición de la salida del sistema como retroalimentación para ajustar la acción de control. Estos sistemas solo se pueden implementar cuando se conoce de antemano la relación entre la entrada al sistema y la salida, y además se sabe que no existen perturbaciones externas; un ejemplo de este sistema es una lavadora, en la cual el comportamiento de los elementos de control y la duración de los ciclos están predefinidos, y se asume que no se tendrán perturbaciones externas. (Katsuhiko Ogata, 2001)

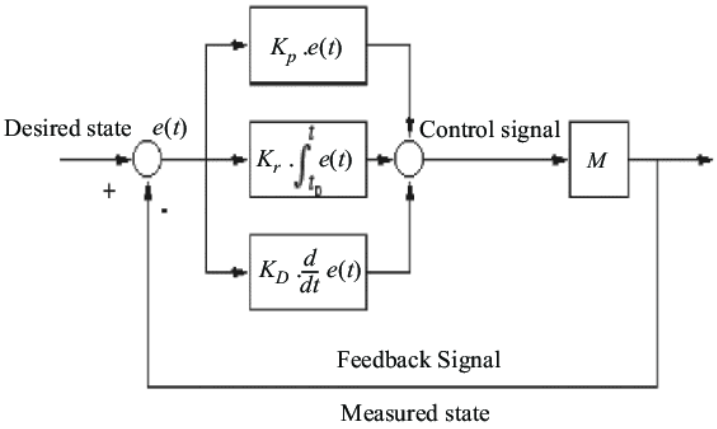
En el caso de los sistemas de control de ciclo cerrado se utiliza una señal de retroalimentación a la salida del sistema para ajustar las acciones de control, y de esta forma poder reaccionar a perturbaciones externas



Uno de los controladores más utilizados en la industria es el controlado Proporcional-Integral-Derivativo (PID). Este controlador se destaca por su baja complejidad y alta robustes, y su funcionamiento depende de los tres parámetros indicados en su nombre:

- P: El parámetro proporcional, encargado de producir un valor de salida proporcional al valor actual del error (la diferencia entre la salida deseada y la salida actual)

- I: El parámetro integral se encarga de reaccionar al error (tanto a su magnitud como a su duración) y tiene en cuenta sus valores acumulados para modificar la acción
- D: El termino derivativo se encarga de predecir la respuesta del sistema lo que puede aumentar los tiempos de respuesta y la estabilidad del sistema(Tim Wescott, 2000)



Para sintonizar estos parámetros se han propuesto diversos métodos para situaciones en las que los modelos matemáticos de la planta no se pueden obtener fácilmente (como lo es el caso de un dron), entre estos uno de los métodos mas utilizado es el de Ziegler-Nichols. Este consiste en dos procedimientos principales:

Método de la curva de reacción: Este método requiere incrementar la ganancia del controlador proporcional hasta que el sistema alcance el límite entre la estabilidad y la inestabilidad, identificando así el punto en el que el sistema comienza a oscilar continuamente. Esta oscilación continua se utiliza para determinar la ganancia crítica (Kc) y el periodo de oscilación crítico (Pc). Con estos valores, se pueden calcular los parámetros del controlador PID usando fórmulas predefinidas.

Método de la oscilación sostenida: Este método implica ajustar el controlador para que opere en modo proporcional solamente, incrementando la ganancia hasta que el sistema muestre una oscilación sostenida (sin aumentar ni disminuir en amplitud). La ganancia y el periodo de esta oscilación se utilizan luego para calcular los parámetros del controlador PID.

Los parámetros del controlador PID ajustados según Ziegler-Nichols están diseñados para ofrecer una respuesta rápida con un sobreimpulso moderado, siendo un punto de partida útil que puede requerir ajustes finos según las necesidades específicas del sistema controlado (Katsuhiko Ogata, 2001).

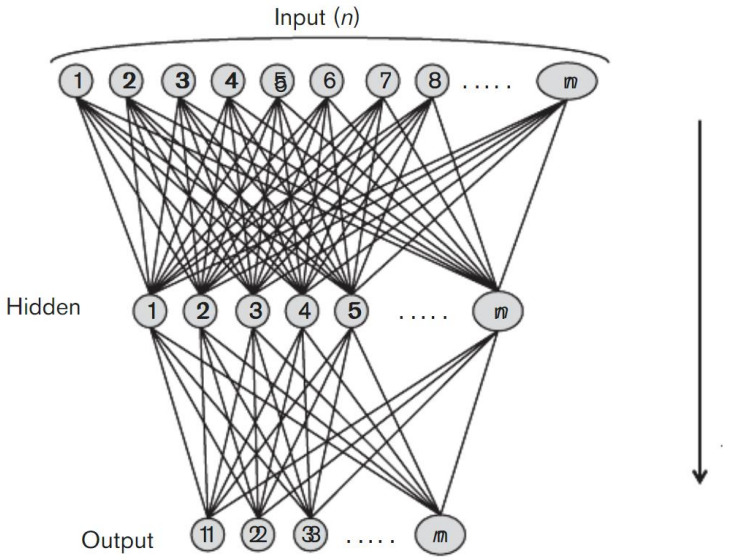
2.4.2 Redes Neuronales

Las redes neuronales son una de las piedras angulares de la inteligencia artificial, inspiradas en la estructura y función del cerebro humano. Son sistemas compuestos por unidades de procesamiento llamadas neuronas, organizadas en capas, que trabajan de manera conjunta para realizar tareas específicas, como reconocimiento de imágenes o audio, procesamiento de lenguaje natural, modelos predictivos, entre otras. (Grossi & Buscema, 2007)

Estas redes se construyen en capas, las cuales se dividen principalmente en 3 tipos:

- Capa de entrada: Recibe las señales (datos) de entrada.
- Capas ocultas: Realizan la mayor parte del procesamiento a través de una combinación de pesos, sesgos y funciones de activación. El número de capas ocultas y el número de neuronas en cada capa pueden variar según la complejidad de la tarea.
- Capa de salida: Produce el resultado final de la red. El número de neuronas en esta capa depende de la naturaleza del problema (por ejemplo, clasificación, regresión).

Estos pesos y sesgos son parámetros ajustables en una red neuronal. Los pesos controlan la importancia de las entradas a cada neurona, y los sesgos permiten ajustar la salida junto con la suma ponderada de las entradas. Además, las funciones de Activación: Ayudan a decidir si una neurona debe activarse o no, introduciendo no linealidades en el proceso de aprendizaje, lo que permite a las redes neuronales aprender patrones complejos.



El aprendizaje en las redes neuronales se realiza mediante un proceso donde la red ajusta sus pesos y sesgos para minimizar la diferencia entre la salida prevista y la salida real (error). Este proceso se lleva a cabo a través de algoritmos de optimización, siendo el más común el “gradient descent” y sus variantes.

Este es un proceso iterativo, donde a cada iteración se le llama una época, y se busca ajustar gradualmente los pesos y sesgos para mejorar la precisión de la red.

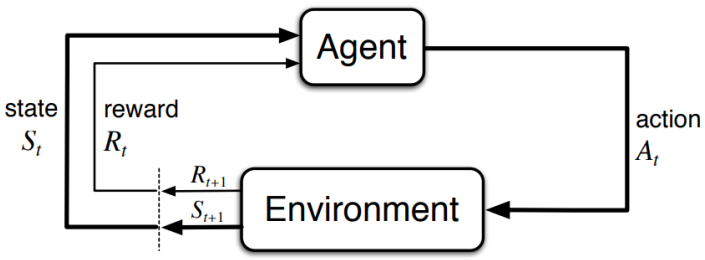
Las redes neuronales son especialmente buenas para resolver problemas de naturaleza no lineal gracias a su capacidad extraer patrones de grandes conjuntos de datos(Grossi & Buscema, 2007). La eficacia de una red neuronal depende en gran medida de su arquitectura, el volumen y la calidad de los datos de entrenamiento, y la precisión del proceso de optimización.

2.4.3 Aprendizaje por refuerzo

A diferencia del aprendizaje supervisado, el aprendizaje por refuerzo no utiliza datos previamente recolectados para el entrenamiento de la red, en lugar de esto se utiliza un entorno con el cual la red neuronal puede interactuar a través de un agente, dependiendo de la aplicación este agente puede ser un vehículo autónomo, una entidad en un video juego, o incluso se pueden tener múltiples agentes interactuando en la misma simulación.

Igualmente, a diferencia del aprendizaje no supervisado el aprendizaje por refuerzo no busca encontrar la estructura oculta en un set de datos, se busca maximizar una señal de recompensa. Para maximizar esta recompensa acumulativa, el agente realiza acciones en el entorno. En cada paso, el agente recibe un estado del entorno, toma una acción y recibe una recompensa y el nuevo estado del entorno como respuesta.

Es importante notar que durante este proceso no se le dan instrucciones al agente de que acciones tomar, en lugar de esto, el agente se inicializa de forma aleatoria, y a partir de este estado explora sus posibles acciones. A medida que este proceso avanza el agente aprovecha las opciones que han dado mejor resultado (mayor valor de la señal de recompensa (Sutton & Barto, n.d.).



2.4.4 Redes Deep Q-Networks (DQN)

Una Red Deep Q-Network (DQN) es un tipo de red neuronal utilizada en el ámbito del aprendizaje por refuerzo para resolver problemas de toma de decisiones en entornos complejos y de alta dimensionalidad. Desarrollada por Google DeepMind, la DQN combina las capacidades de las redes neuronales profundas con los algoritmos de Q-learning para crear un agente que pueda aprender a realizar tareas a través de la interacción directa con su entorno. (Sutton & Barto, n.d.)

Q-Learning es un algoritmo de aprendizaje por refuerzo basado en la idea de aprender una función de valor de acción, conocida como la función Q. Esta función estima la utilidad esperada de tomar una acción en un estado particular, y seguir una política determinada en el futuro.

La fórmula de actualización de Q-Learning se basa en la diferencia temporal (TD), que ajusta las estimaciones de los valores Q en función de la recompensa recibida y las estimaciones futuras.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max a' Q(s',a') - Q(s,a)]$$

En entornos complejos, donde el espacio de estados es muy grande o continuo, es inviable almacenar todos los valores Q en una tabla. En su lugar, las DQNs utilizan una red neuronal profunda para aproximar la función Q.

La red DQN recibe como entrada el estado del entorno y produce como salida una estimación de los valores Q para cada acción posible.

Las DQNs han sido aplicadas exitosamente en una variedad de dominios, como videojuegos, robótica, y control autónomo de vehículos. Un ejemplo famoso es el uso de DQNs para jugar y superar el desempeño humano en varios juegos de Atari, demostrando su capacidad para aprender estrategias complejas a partir de entradas visuales sin conocimiento previo del entorno.

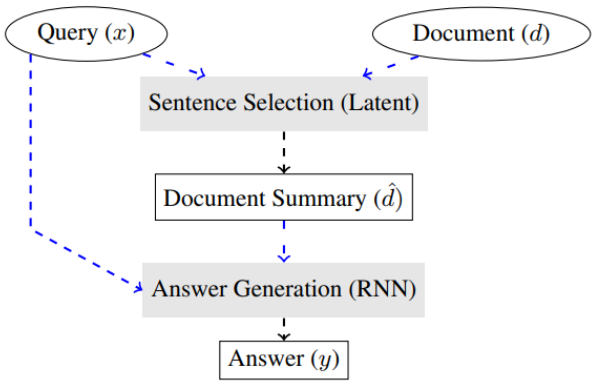
Estos algoritmos utilizan una política de explotación vs exploración, lo que significa que el agente busca balancear la exploración de nuevas acciones que no se han probado, contra acciones que han dado resultados positivos. Para esto se utiliza una política ε-greedy donde el parámetro ε disminuye gradualmente durante el entrenamiento para favorecer la explotación sobre la exploración a medida que el agente aprende.

2.4.5 Aplicaciones industriales de las redes neuronales entrenadas por refuerzo (RL)

Las redes neuronales entrenadas por refuerzo se han utilizado en diversos campos de la industria, como por ejemplo el caso de IBM el cual desarrollo una plataforma

basada en RL para trading (Aishwarya Srinivasan, 2018) en la cual la función de recompensa se mide en base a los retornos de cada transacción.

En el campo del procesamiento de lenguaje natural se han utilizado para optimizar la velocidad de respuesta de redes neuronales recurrentes, esto se logro primero utilizando RL para seleccionar las palabras mas relevantes del texto de entrada y pasando este resultado a la red neuronal recurrente (Choi et al., 2016)



En el campo de la salud se han utilizado para suministrar tratamiento a pacientes a partir de políticas aprendidas por RL, en estos casos se conocen como regímenes de tratamiento dinámicos (DTR por sus siglas en ingles), en este tipo de redes las entradas son observaciones del paciente, y las salidas son recomendaciones de tratamiento (Yu et al., 2019).

Una de las aplicaciones mas famosas de las redes entrenadas por RL es en el área de los videojuegos, se han creado agentes con capacidades muy por encima de las humanas a la hora de controlar videojuegos como lo es por ejemplo AlphaGo Zero, el cual aprendió a jugar el juego Go a la perfección, hasta el punto de derrotar a los mejores maestros de su momento (Silver et al., 2017).

En el área de ingeniería uno de los mayores proyectos fue creado por Facebook y consiste en una plataforma de optimización de productos y servicios a gran escala llamada Horizon (Jason Gauci, 2018), esta ha sido utilizada internamente por Facebook en diversas áreas de su producto como:

- Personalización de sugerencias
- Mejora de calidad de video
- Personalización de notificaciones

Las redes RL también se han utilizado exitosamente en el área de control industrial, como lo es el caso de Google, el cual creo una red llamada Deepmind la cual utiliza para controlar la temperatura de sus centros de datos. Esto logro una reducción del 40% en el consumo energético, esto se debe en parte a que el sistema logra predecir como diferentes configuraciones afectaran la temperatura y el consumo

antes de enviar la configuración sugerida al sistema de control local, este verifica que la configuración sugerida este dentro de los parámetros locales de seguridad antes de ejecutarla. (Chris Gamble, 2018)

2.4.6 Herramientas para implementación de redes neuronales

Actualmente existen diversas librerías que facilitan la implementación de redes neurales de todo tipo, y dado que Python es uno de los lenguajes de programación mas populares para aprendizaje automatizado este cuenta con una gran variedad de estas herramientas.

Entre estas se encuentra TensorFlow, la cual fue desarrollada por Google y es una librería open source que permite construir y entrenar modelos de aprendizaje profundo. Es altamente flexible y puede ser utilizada tanto para investigación como para aplicaciones comerciales. Esta librería permite el uso de hardware como GPUs para el entrenamiento de los modelos, agilizando de gran manera los procesos de entrenamiento que involucran procesamiento de imágenes.

TensorFlow cuenta con un API de alto nivel llamado Keras, el cual simplifica el uso de los modelos permitiendo crearlos y agregar las capas que sean necesarias de una forma sencilla.

Otra de las principales herramientas en el desarrollo de aplicaciones con redes neuronales es Pytorch, esta librería ofrece un acercamiento mas modular dependiendo de la aplicación que se desea desarrollar, algunos de estos módulos son, por ejemplo: TorchVision para procesamiento de imágenes, TorchAudio para procesamiento de audio o TorchText para el procesamiento de texto

Ambas librerías permiten trabajar con el estado del arte en algoritmos de redes neuronales y cuentan con un extenso ecosistema y comunidades para apoyar el desarrollo.

2.4.7 Herramientas para visualización de simulaciones en 2D.

Uno de los objetivos de este proyecto es crear un entorno en el cual un agente pueda experimentar e interactuar de diferentes formas, con el fin de probar las limitaciones y oportunidades de estos sistemas de control y algoritmos de redes neuronales, para esto es necesario utilizar una herramienta de visualización con la cual el usuario pueda ver tanto el proceso de entrenamiento como los resultados de los modelos

En el ámbito de visualización en 2D se encuentran frecuentemente herramientas en C++ debido a su alta velocidad de ejecución, entre estas existen herramientas como Box2d y SFML, sin embargo Python también cuenta con herramientas similares las cuales utilizan C++ de forma interna como lo es PyGame.

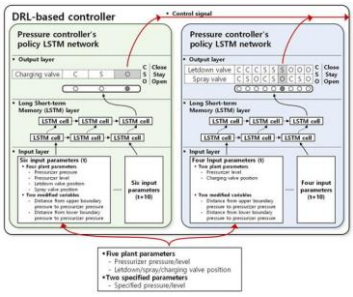
PyGame es una librería originalmente orientada al desarrollo de videojuegos, sin embargo esta presenta todas las funciones necesarias para la visualización de una simulación controlada por modelos de redes neuronales, adicionalmente es posible integrarlo con la librería PyMunk, la cual se especializa en simulaciones de física proporcionando funciones para el cálculo de movimiento con masa, colisiones, gravedad, entre otras.

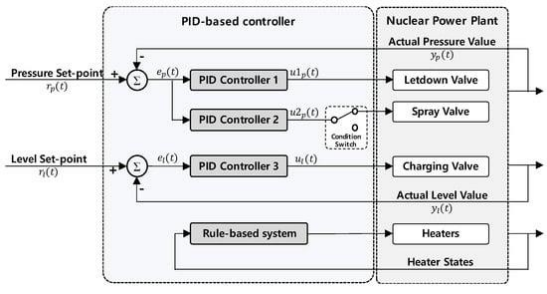
Estas herramientas combinadas permiten la creación de un entorno simulado robusto en el cual se pueden desarrollar y probar los controladores para drones.

2.4.8 Estado del arte en comparaciones entre control tradicional y control por RL

A pesar de que existe una gran cantidad de literatura acerca de estos enfoques por separado, no se encuentran muchos estudios que realicen una comparativa directa entre ellos, algunos ejemplos de estudios similares incluyen una propuesta para utilizar controladores basados en redes neuronales para el control del encendido y apagado en una planta nuclear, además de sintonizar controladores PID utilizando redes neuronales en lugar de los métodos heurísticos clásicos (Lee et al., 2022).

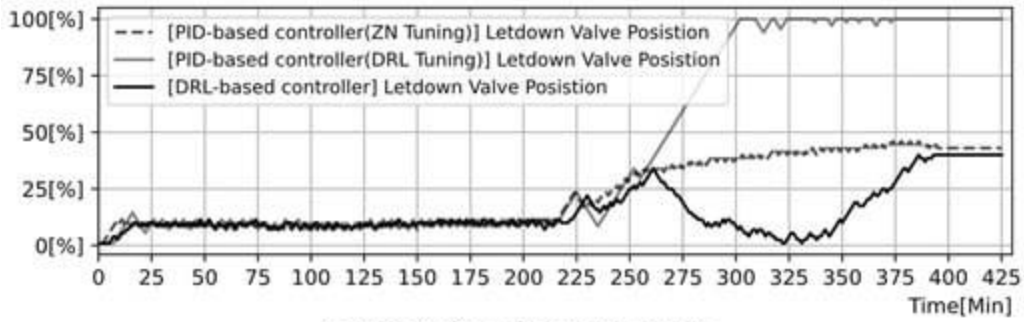
Durante este estudio se prueban diferentes acercamientos a esta problemática, una de ellas es un controlador basado en una red LSTM (Long Short Term Memory) el cual analiza una serie de tiempo de datos sobre la operación de la planta y se utiliza un algoritmo SAC (Soft Actor-Critic) para su entrenamiento con el fin de agilizar el tiempo de entrenamiento y mejorar la estabilidad, este controlador se encarga de accionar válvulas en el sistema del reactor para regular la presión al interior, adicional a esto también se implementaron tres controladores PID para ser utilizados en válvulas diferentes y se sintonizaron utilizando el método heurístico de Ziegler-Nichols, y finalmente implementaron un controlador PID sintonizado por medio de una red neuronal.



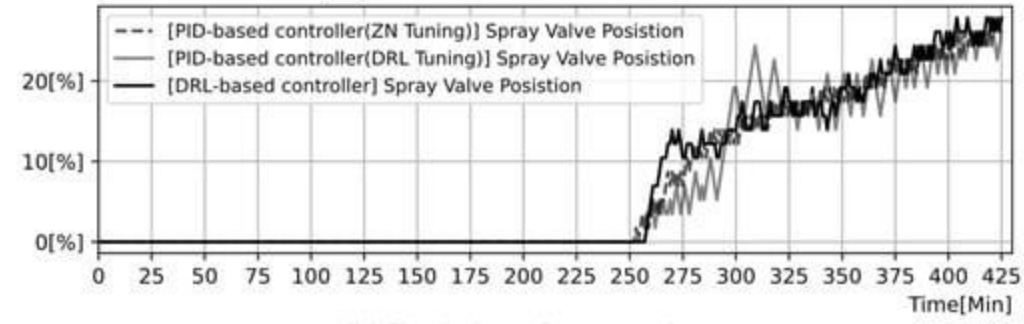


A continuación, presentan resultados de tiempos de respuesta y frecuencia de acción de control de forma comparativa entre estos controladores propuestos, además de incluir resultados interesantes, como la capacidad del controlador

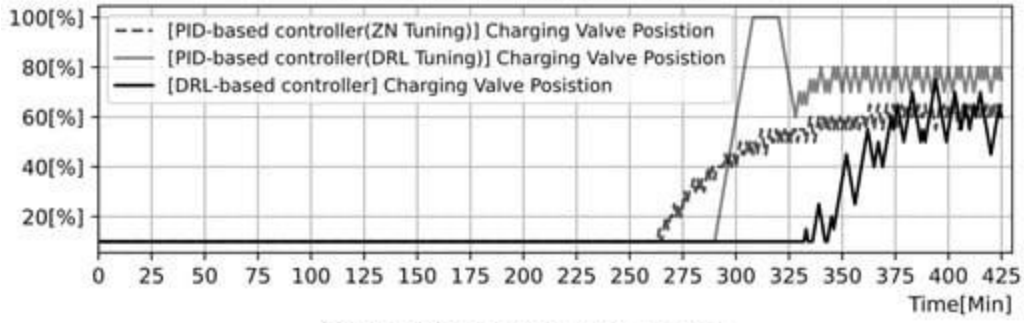
basado en redes neuronales de controlar simultáneamente múltiples actuadores del sistema, mientras que los controladores PID estaban dedicados a un solo componente.



(a) Posistion of letdown valve



(b) Posistion of spray valve



(c) Posistion of charging valve

Uno de los grandes desafíos presentado por los autores de este estudio es la legalidad de utilizar controladores basados en redes neuronales para manejar sistemas críticos como lo es el apagado de una planta nuclear, ya que estos algoritmos no han sido aplicados a control industrial de una forma tan extensa como lo han sido los controladores PID, los cuales ya son reconocidos como una tecnología segura para sistemas críticos.

Sin embargo, los controladores basados en redes neuronales implementados durante estos experimentos probaron responder de forma satisfactoria a los

requerimientos del sistema, y en muchas áreas logra superar el desempeño de los controladores PID creados para esta aplicación particular.

2.5 Metodología

2.5.1 Desarrollo del Entorno Simulado:

Debido a que el proyecto será escrito principalmente en Python, se elige la herramienta de visualización Pygame y Pymunk para crear un entorno 2D que simule la dinámica de vuelo de un dron.

Implementar características como la detección de colisiones, la simulación de fuerzas externas (como el viento), y la capacidad de modificar parámetros de vuelo en tiempo real.

2.5.2 Implementación del Controlador PID:

Desarrollar un controlador PID e integrarlo con el agente en el entorno simulado, recibiendo señales generadas por este agente y emitiendo una señal de control calculada en base a estas señales.

Ajustar los parámetros del PID utilizando el método de Ziegler-Nichols para optimizar su desempeño en diferentes escenarios de vuelo.

2.5.3 Desarrollo y Entrenamiento de la Red Neuronal:

Para la creación y entrenamiento de la red neuronal se elige Pytorch debido a que no es un proyecto que requiera ser escalado a grandes volúmenes de datos, además Pytorch cuenta con una sintaxis más intuitiva.

Utilizando Pytorch se crea una red neuronal de tipo DQN. La red DQN consistirá en una serie de capas completamente conectadas. La entrada de la red será el estado del entorno simulado (por ejemplo, la posición y velocidad del dron, y las fuerzas externas), y la salida serán las acciones posibles que el dron puede tomar (por ejemplo, ajustar la velocidad o cambiar de dirección).

Función de Activación: Se utilizarán funciones de activación ReLU en las capas ocultas para introducir no linealidades y permitir que la red aprenda patrones complejos.

Función de Pérdida: Se empleará una función de pérdida basada en la diferencia temporal para actualizar los valores Q, lo que ayudará a la red a aprender las mejores acciones a tomar en cada estado para maximizar la recompensa acumulativa.

Optimización: Se utilizará el algoritmo Adam para optimizar los pesos de la red neuronal, ya que es conocido por su eficiencia y capacidad de adaptación durante el entrenamiento.

Durante el entrenamiento el agente interactuará con el entorno simulado, recibiendo estados y recompensas, y tomando acciones en consecuencia.

Política de Exploración/Explotación: Se implementará una política ε-greedy para balancear la exploración de nuevas acciones y la explotación de acciones conocidas para maximizar la recompensa y finalmente se utilizará una memoria de repetición para almacenar experiencias pasadas (transiciones de estado, acción, recompensa, siguiente estado) y utilizarlas para entrenar la red, permitiendo un aprendizaje más estable y eficiente.

2.5.4 Evaluación Comparativa:

Realizar una serie de pruebas estandarizadas para evaluar el desempeño del controlador PID y la red neuronal como el seguimiento de trayectorias idénticas, y con esto tomar métricas clave como la precisión en el seguimiento de trayectorias, la eficiencia en el uso de energía, la robustez frente a perturbaciones externas, y el tiempo de respuesta.

Analizar los resultados y realizar una comparación detallada para identificar las ventajas y limitaciones de cada enfoque.

2.6 Plan de Gestión de Datos

Los datos utilizados durante esta investigación serán generados por medio de una simulación de la física de un entorno para el vuelo del dron, por lo cual no conllevan riesgos de privacidad ni confidencialidad.

3 Desarrollo del proyecto

3.1 Entorno de simulación.

Para realizar una evaluación controlada de los algoritmos de meachine learning y de control, se desarrolló un entorno simulado en 2D que modela la dinámica y el comportamiento de un dron en vuelo. Este entorno, implementado en Python utilizando las bibliotecas Pymunk y Pygame, permitió simular aspectos críticos como, inercia, gravedad, fuerzas externas, colisiones, y variaciones en el comportamiento dinámico del dron.

Para estructurar el entorno, se implementó una clase Dron, que encapsula todas las propiedades y comportamientos asociados al dron. Esta clase permitió gestionar de manera eficiente las interacciones físicas, el control de los motores, y la respuesta a perturbaciones externas, además de asegurar el mismo comportamiento del dron entre los distintos algoritmos implementados

3.1.1 Clase Dron

La clase Dron fue diseñada para modelar el comportamiento del dron como un cuerpo rígido dentro del espacio de simulación. A continuación, se describen sus principales características:

3.1.1.1 Estandarización de parámetros iniciales:

Para asegurar un mejor entrenamiento de las redes neuronales, y crear un entorno adecuado para pruebas comparativas, se aseguraron parámetros inciales estandarizados para todos los modelos

Para esto, durante la inicialización de una entidad dron se incluye la selección de parámetros físicos como las dimensiones, masa, momento de inercia, forma, posición, entre otros. Mediante el uso de instancias de una clase se puede garantizar que estos parámetros sean iguales para todas las iteraciones de entrenamiento o pruebas

```
class Drone:
    def __init__(self, space, x, y):
        drone_width = 60
        drone_height = 20
        mass = 1
```

```
        inertia = pymunk.moment_for_box(mass, (drone_width, drone_height))
        self.body = pymunk.Body(mass, inertia)
        self.body.position = x, y
        self.shape = pymunk.Poly.create_box(self.body, (drone_width,
drone_height))
        self.shape.collision_type = 1
        self.shape.elasticity = 0.5
        self.shape.friction = 1
        space.add(self.body, self.shape)
```

Además de establecer los parámetros físicos del dron, también se establecen parámetros operacionales como lo son la potencia inicial aplicada a cada uno de los motores, el tiempo de vuelo, y las coordenadas del objetivo.

```
    def set_target(self, target_x, target_y):
        self.target_x = target_x
        self.target_y = target_y
        self.calculate_distance_to_target()
```

3.1.1.2 Aplicación de fuerzas

El dron, al estar en un espacio bidimensional, utiliza dos motores con fuerzas independientes aplicadas en puntos específicos del cuerpo. Esto permite controlar tanto la dirección como la velocidad. La clase dron se encarga de aplicar las fuerzas resultantes del empuje de los motores e interacciones con el entorno mientras mantiene el rango de empuje de los motores entre los límites establecidos.

```
    def apply_thrust(self, left_thrust, right_thrust):
        self.left_thrust = np.clip(left_thrust, 0, 1000)
        self.right_thrust = np.clip(right_thrust, 0, 1000)

        self.angle = (math.degrees(self.body.angle) +180) % 360 - 180
        # self.store_data()

        left_force = (0, -1*left_thrust )
        right_force = (0, -1*right_thrust )
        self.body.apply_force_at_local_point(left_force,
self.left_propeller_pos)
        self.body.apply_force_at_local_point(right_force,
self.right_propeller_pos)
```

```
        self.calculate_distance_to_target()
        self.calculate_angle()
```

3.1.1.3 Interacciones con el entorno

La clase dron también se encarga de los cálculos resultantes de interacciones con el entorno, como lo es la gravedad y el viento.

```
    def wind(self, direction=None):
        self.wind_strength = 100
        self.wind_direction = direction if direction is not None else
np.random.uniform(0, 360)

        wind_angle_rad = math.radians(self.wind_direction)
        wind_force_x = self.wind_strength * math.cos(wind_angle_rad)
        wind_force_y = self.wind_strength * math.sin(wind_angle_rad)

        self.body.apply_force_at_world_point((wind_force_x, wind_force_y),
self.body.position)
```

3.1.1.4 Calculo de distancias y ángulos

El dron continuamente calcula las distancias y ángulos después de los movimientos resultantes de las fuerzas aplicadas, los valores que se capturan son:

- Empuje actual de ambos motores
- Distancia (lineal) al objetivo
- Componente en X de la distancia al objetivo
- Componente en Y de la distancia al objetivo
- Velocidad actual
- Seno del ángulo actual
- Coseno del ángulo actual
- Seno del ángulo actual respecto al objetivo
- Coseno del ángulo actual respecto al objetivo
- Seno del ángulo actual del vector de velocidad
- Coseno del ángulo actual del vector de velocidad
- Diferencia entre el ángulo de velocidad y el ángulo al objetivo

```
-       def calculate_angle(self):
-           dx = self.target_x - self.body.position.x
```

```
-           dy = self.target_y - self.body.position.y
-           raw_angle_to_target = math.atan2(dy, dx)
-
-           # Normalize velocity angle to radians
-           raw_velocity_angle = math.atan2(self.body.velocity.y,
    self.body.velocity.x)
-
-           self.angle_sin = math.sin(self.body.angle)
-           self.angle_cos = math.cos(self.body.angle)
-
-           self.angle_to_target_sin = math.sin(raw_angle_to_target)
-           self.angle_to_target_cos = math.cos(raw_angle_to_target)
-
-           self.velocity_angle_sin = math.sin(raw_velocity_angle)
-           self.velocity_angle_cos = math.cos(raw_velocity_angle)
-
-           raw_delta_angle = raw_velocity_angle - raw_angle_to_target
-           self.delta_angle_sin = math.sin(raw_delta_angle)
-           self.delta_angle_cos = math.cos(raw_delta_angle)
-           delta_angle = math.atan2(self.delta_angle_sin,
    self.delta_angle_cos)
-           self.absolute_delta_angle_normalized = abs(delta_angle) /
    math.pi
```

```
-   def calculate_distance_to_target(self):
-           self.distance_to_target =
    np.rint(np.linalg.norm(self.body.position - [self.target_x,
    self.target_y]))
-
-           x_distance = self.body.position[0] - self.target_x
-           y_distance = self.body.position[1] - self.target_y
-
-           # Optional: Store x and y distances if needed
-           self.x_distance_to_target = np.rint(x_distance)
-           self.y_distance_to_target = np.rint(y_distance)
```

La información calculada se emite para ser utilizada en el entrenamiento de modelos o presentación de la información.

```
-       def get_info(self):
-           return {
-               'left_thrust': self.left_thrust,
-               'right_thrust': self.right_thrust,
-               'distance_to_target': self.distance_to_target,
```

```
-                'x_distance_to_target': self.x_distance_to_target,
-                'y_distance_to_target': self.y_distance_to_target,
-                'speed': np.rint(self.body.velocity.length),
-                'y_speed': np.rint(self.body.velocity.y),
-                'angle_sin': self.angle_sin,
-                'angle_cos': self.angle_cos,
-                'angle_to_target_sin': self.angle_to_target_sin,
-                'angle_to_target_cos': self.angle_to_target_cos,
-                'velocity_angle_sin': self.velocity_angle_sin,
-                'velocity_angle_cos': self.velocity_angle_cos,
-                'delta_angle_sin': self.delta_angle_sin,
-                'delta_angle_cos': self.delta_angle_cos,
-                'delta_angle': self.absolute_delta_angle_normalized
```

Los ángulos inicialmente se plantearon en grados en un rango de -180 grados a 180 grados, sin embargo, se optó por usar funciones trigonométricas ya que al ser funciones continuas dieron mejores resultados en el entrenamiento de los modelos.

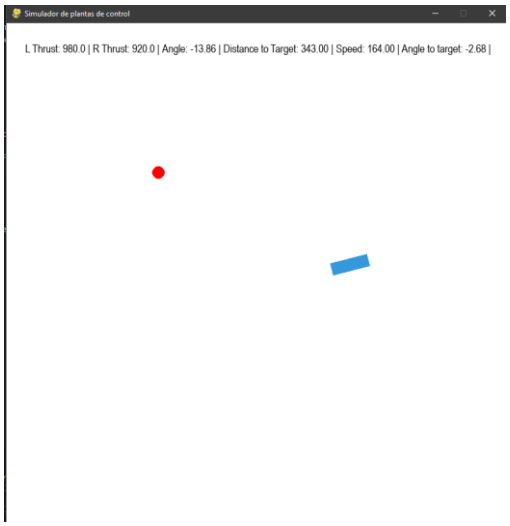
3.1.2 Espacio de vuelo.

Estas entidades Dron se instanciaron dentro de un entorno controlado que consta de una aceleración constante por gravedad, objetivos posicionados aleatoriamente y un espacio que permitiera al dron movimiento libre. Para la creación de este se utilizó pymunk para las interacciones físicas y pygame para dar una simple representación visual, sin embargo, pygame no se utilizó durante el entrenamiento de los modelos ya que no es necesario supervisar visualmente cada iteración, además de agregar una gran carga computacional a la simulación, por lo tanto esta visualización se mantuvo simple utilizando formas sencillas para representar el dron, los objetivos y el viento.

```
    clock = pygame.time.Clock()
    fps = 60
    dt = 1 / fps
    space = pymunk.Space()
    space.gravity = (0, 981)
    draw_options = pymunk.pygame_util.DrawOptions(screen)
    font = pygame.font.SysFont("Arial", 18)
```

```
def draw(space, screen, draw_options, information, font, score,
graph_controller, targetX, targetY, wind_active, wind_direction):

    screen.fill("white")
    if graph_controller['show_graphs']:
        draw_charts(information, graph_controller['selected_graph'])
    space.debug_draw(draw_options)
    output_information(information, font, score)
    pygame.draw.circle(screen, (255, 0, 0), (targetX, targetY), 10)
```

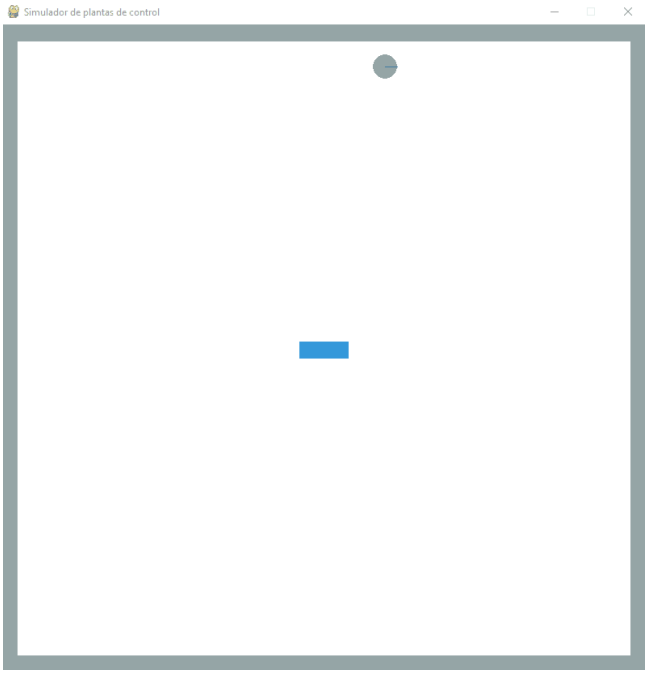


3.2 Control manual

Antes de implementar controladores automatizados para el dron se probó el sistema con controles manuales utilizando 5 estados predeterminados para los motores del dron y cambiando estos estados con una entrada manual por medio de un teclado, esto se hizo con el fin de probar el funcionamiento correcto del entorno.

Los estados predeterminados son:

- Ambos motores al 50% de potencia (estado de reposo)
- Ambos motores al 100% de potencia (estado utilizado para ascender)
- Motor izquierdo al 40% de potencia (estado utilizado para girar a la izquierda)
- Motor derecho al 40% de potencia (estado utilizado para girar a la derecha)
- Ambos motores apagados



Enlace para ver la imagen animada

3.3 Controlador PID

Se implementaron varios controladores PID que utilizan los datos recolectados por el dron y accionan los motores con el fin de servir como un punto de comparación con los controladores tradicionales.

La implementación de estos PID se realizó con una clase para la cual el constructor recibe como argumentos los parámetros proporcional, integral y derivativo y realiza las operaciones matemáticas necesarias siguiendo la formula:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt}$$

Además también recibe dos parámetros para establecer los limites superior e inferior que puede tomar la señal de control.

```
import numpy as np
class PID:
    def __init__(self, kp, ki, kd, lower_signal_limit, upper_signal_limit,
FPS = 60):
        self.kp = kp
        self.ki = ki
        self.kd = kd
```

```
        self.integral = 0
        self.previous_error = 0
        self.lower_signal_limit = lower_signal_limit
        self.upper_signal_limit = upper_signal_limit
        self.delta = 1/FPS

    def calculate(self, measurement, target):
        error = target - measurement
        self.integral += error*self.delta
        derivative = (error - self.previous_error)/self.delta
        self.previous_error = error

        return np.clip((self.kp * error + self.ki * self.integral + self.kd
* derivative), self.lower_signal_limit, self.upper_signal_limit )
```

Para estos controladores se utilizó un acercamiento de controladores en cascada, creando múltiples instancias de esta clase donde la salida de un controlador PID es la entrada de otro. Además, se utilizaron dos grupos de controladores PID en cascada para el control del dron.

Estos dos grupos constan de un grupo de control de ángulo, y un grupo de control de amplitud.

3.3.1 Control de ángulo

Para este grupo se tomo como entrada la posición en la coordenada X del dron, comparándola con la coordenada en X del objetivo y se genera una referencia angular.

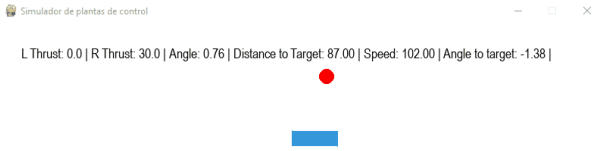
Este valor se utiliza como entrada en un segundo controlador el cual compara el ángulo actual del dron con la referencia de ángulo para dar una corrección al ángulo, esta corrección genera una diferencia de empuje entre los motores del dron.

3.3.2 Control de amplitud

En este grupo se toma como entrada la posición en la coordenada Y del dron y se compara con la posición en Y del objetivo para generar una referencia de empuje total.

Esta referencia de empuje se toma como entrada del segundo controlador el cual lo compara con la velocidad en Y actual del dron, con esto se obtiene un empuje deseado y se aplica de forma uniforme a los dos motores.

Este cambio de empuje uniforme, en combinación con el cambio de empuje diferencial dado por el grupo de control de ángulo logran controlar exitosamente el dron y moverlo al objetivo de una forma estable.



[Enlace para ver la imagen animada]

3.4 Recopilación de datos

Para los experimentos realizados con redes neuronales se almacenaron los datos en la nube utilizando la herramienta Weights and Biases, la cual permite visualizar de forma ordena graficas de desempeño de los modelos como la recompensa obtenida por el dron o la función de perdida, además de mostrar datos del costo computacional como memoria alocada para el entrenamiento y temperaturas de la GPU utilizada para realizarlo.

A través de los diferentes modelos utilizados se realizaron mas de 60 entrenamientos con variaciones en los parámetros (función de recompensa, numero de iteraciones, propiedades del entorno de simulación, propiedades de los entornos de gymnasium), algunos con duraciones de aproximadamente 20 horas.

3.5 Red neuronal DQN

3.5.1 Entorno de entrenamiento de Gymnasium

Para entrenar la red DQN y permitirle controlar el dron, se utilizó la biblioteca Stable Baselines 3 en conjunto con una clase personalizada llamada DroneEnvRL. Esta clase se definió basada en la API de Gymnasium, la cual permite una fácil integración con SB3.

Para definir esta clase se crearon métodos requeridos por SB3 que se mencionan a continuación:

3.5.1.1 definición del espacio de observación.

El espacio de observación consta de todas las entradas a la red neuronal, las cuales se obtienen de las observaciones tomadas por el dron al interactuar con su entorno, este espacio utiliza las variables mencionadas anteriormente que emite la clase Dron como lo son el ángulo del dron, velocidad, distancia al objetivo, entre otras.

```
        low = np.array([-180,0,0,-180,-180,-180])
        high = np.array([180,2000,1000,180,180,180])
        self.observation_space = gym.spaces.Box(low=low, high=high,
dtype=np.float32, shape=(6,))
```

3.5.1.2 Definición del espacio de acción.

El espacio de acción define las posibles decisiones que puede tomar el dron con el fin de modificar su estado actual, en este caso el espacio de acción consta de los valores de empuje de los motores, para cada una de las redes DQN creadas se utilizo un espacio de acción diferente los cuales se definirán en sus respectivas secciones.

3.5.1.3 Método Step y función de recompensa

Este método aplica las acciones seleccionadas por la red neuronal y evalúa el resultado en términos de recompensa y estado del dron.

Se incluyen penalizaciones por desviaciones del ángulo ideal, distancia al objetivo, y salidas del área de simulación.

Este es el método que se utiliza para avanzar la simulación y medir el desempeño del agente para el aprendizaje de la red.

```
def step(self, action):
        action_limiter = range(3)
        # Apply thrust based on action
        action = int(action)
        self.reward = 0
        for _ in action_limiter:
            left_thrust = 490
            right_thrust = 490
            self.time += 1/60
            if action == 0:  # no action
                pass
            elif action == 1:  # full thrust
                left_thrust = 800
                right_thrust = 800
            elif action == 2:  # reduced left thrust
                left_thrust *= 0.8
            elif action == 3:  # reduced right thrust
                right_thrust *= 0.8
            elif action == 4:  # no thrust
                left_thrust = 0
                right_thrust = 0

            self.drone.apply_thrust(left_thrust, right_thrust)
            self.space.step(1/60)


            done = False

            self.reward += 1/60

            scaling_factor = 1000

            self.reward -= self.drone.distance_to_target / scaling_factor

            if abs(self.drone.angle) > 50:
                self.reward -= 5
```

```
            if self.time > self.time_limit:
                done = True
            if self.drone.distance_to_target < 50:
                self.reward += 300
                self.targetX = np.random.randint(30, 770)
                self.targetY = np.random.randint(30, 770)
                self.drone.set_target(self.targetX, self.targetY)
                self.targets_hit += 1
                print(f'targets hit: {self.targets_hit}')
            if self.drone.distance_to_target > 1000:
                self.reward -= 500
                done = True

            if(self.enable_visualization):
                self.render()

        return self.get_obs(), float(self.reward), done, False, {}
```

3.5.1.4 Visualización

Para la visualización del entrenamiento en tiempo real se creo un método render, el cual contiene el código mostrado anteriormente de pygame y se llama en cada paso de iteración, sin embargo este método se utilizo solo durante las primeras iteración para verificar el funcionamiento de esta clase en conjunto con SB3, ya que durante los experimentos se omitió la visualización para no consumir recursos adicionales de forma innecesaria.

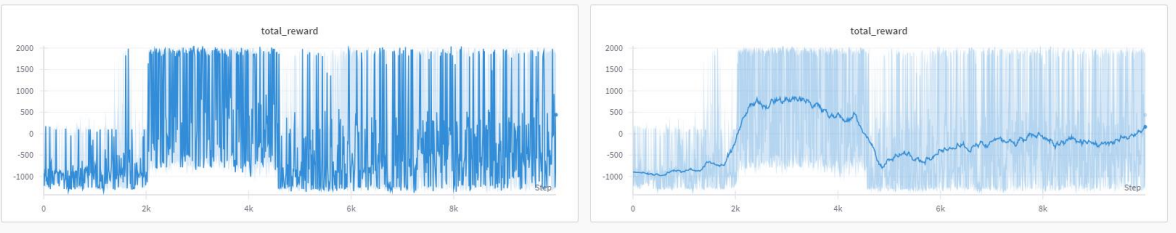
```
    def render(self):
        if self.enable_visualization:
            self.screen.fill("white")
            self.space.debug_draw(self.draw_options)
            pygame.draw.circle(self.screen, (255,0,0), (self.targetX,
self.targetY), 10)
            pygame.display.update()
            self.clock.tick(60)
```

3.5.2 Red neuronal DQN con espacio de acción continuo.

El primer diseño de la red DQN empleó un espacio de acción “continuo”, en el que cada motor del dron podía seleccionar un valor de empuje entre 0 y 1000

(asumiendo solo valores enteros). Esto permitiría una mayor flexibilidad en el control del dron, pero presentó serias dificultades computacionales ya que el modelo requería evaluar una cantidad muy grande de combinaciones posibles de empuje para los motores (10e6 combinaciones). Esto aumentó significativamente el tiempo de entrenamiento y la memoria requerida.

Debido a esto la red tuvo dificultades para converger hacia políticas estables, y los resultados obtenidos tenían una alta varianza con picos y valles muy altos en la recompensa.



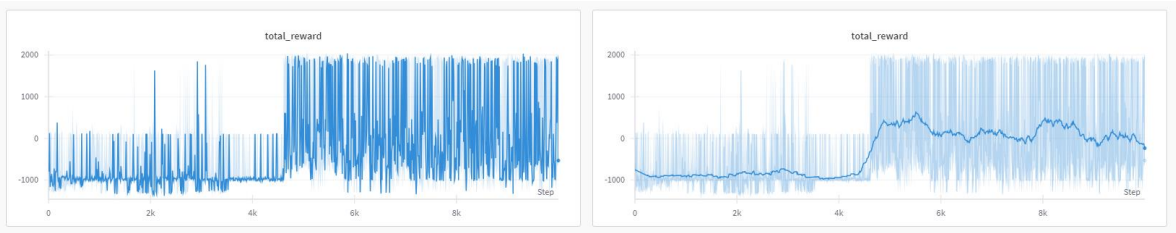
En el entorno de simulación, el dron no era estable, además de que debido al costo computacional que causaba el espacio de acción seleccionado la simulación tenia un mal rendimiento de alrededor de 20 pasos de simulación por segundo.

3.5.3 Red neuronal DQN con espacio de acción discretizado

En un segundo acercamiento, el espacio de acción se redujo a 5 estados discretos para cada motor, utilizando los mismos estados propuestos para el control manual:

- Reposo: Sin cambio en el empuje, con ambos motores al 50% de potencia.
- Subir: Ambos motores al 100% de potencia.
- Izquierda: 40% de potencia en el motor izquierdo, 50% en el derecho.
- Derecha: 50% de potencia en el motor izquierdo, 40% en el derecho.
- Bajar: Ambos motores al 0% de potencia.

Esto redujo significativamente el costo computacional de la red y permitió obtener resultados mas estables en el entrenamiento y desempeño del dron



Sin embargo, estos no llegaron a ser resultados aceptables ya que el dron aun no lograba obtener consistentemente los objetivos, como se puede ver en la gráfica de recompensa, la cual en promedio se mantuvo alrededor de 0.

3.6 Red neuronal SAC

Buscando un mejor desempeño en el espacio continuo que se planteó, se implementó también una red neuronal Soft-actor-critic (SAC), la cual mostró resultados significativamente mejores en comparación con la red DQN, tanto en términos de costo computacional como en la calidad del control.

Al igual que para la red DQN para esta red también se implemento un ambiente de gymnasium con los métodos necesarios para ser integrado con Stable Baselines.

3.6.1 Entorno de entrenamiento de Gymnasium

El entorno de gymnasium creado para esta red comparte los métodos mencionados para la red DQN, pero tiene cambios importantes en algunos de ellos.

3.6.1.1 Definición del espacio de observación

Para este modelo se realizo el cambio en las medidas angulares mencionado anteriormente, por lo cual los valores observados están en un rango de -1 a 1 para todos los ángulos, y tiene una dimensionalidad mayor que en la red DQN

```
        low = np.array([-1, -1, -1, -1, -1, -1, 0, 0, 0])
        high = np.array([1, 1, 1, 1, 1, 1, 1000, 2000, 1])
        self.observation_space = gym.spaces.Box(low=low, high=high,
dtype=np.float32, shape=(9,))
```

3.6.1.2 Definición del espacio de acción

El espacio de acción utilizado en este modelo se planteo de una forma similar la señal de control emitida por los controladores PID, en la cual se tiene una señal que controla la amplitud del empuje de los motores, y una señal que controla la diferencia de amplitud entre los dos motores.

Se definió entonces como un espacio de acción continuo con dos salidas que tienen valores entre -1 y 1.

Estos valores de salida se utilizan en el método step para obtener un valor de empuje total de la siguiente forma

$$Empuje\ final\ motor1 = E_b + E_b * a_1 + a_2 * k$$

$$Empuje\ final\ motor2 = E_b + E_b * a_1 - a_2 * k$$

Donde Eb es el empuje base, el cual se estableció como el valor que mantiene el dron en vuelo sin ganar o perder altura, y K es una constante que se puede ajustar para obtener movimientos mas o menos agresivos del dron, con un valor de 10 se obtuvieron movimientos que eran veloces sin llegar a ser demasiado agresivos.

3.6.1.3 Método Step y función de recompensa

En este método se reciben las acciones seleccionadas y se realizan las operaciones mencionadas anteriormente para obtener los valores de empuje finales.

```
    def step(self, action):
        # Apply thrust based on action
        self.reward = 0
        amplitude_change, phase_change = action
        action_limiter = range(3)
        base_thust = 475
        left_thrust = base_thust
        right_thrust = base_thust
        left_thrust += amplitude_change * base_thust
        right_thrust += amplitude_change * base_thust
        left_thrust += phase_change * 10
        right_thrust -= phase_change * 10
```

Con esto se calcula la recompensa obtenida por el dron en este paso de simulación, este cálculo consta de:

- una penalidad por distancia al objetivo (mientras mas lejos mayor la penalidad)
- una recompensa por el tiempo de vida del dron (mientras mas tiempo viva sin salir del área definida mayor la recompensa)
- Una penalización adicional grande por salir del área designada
- Una recompensa adicional grande por llegar a las coordenadas del objetivo

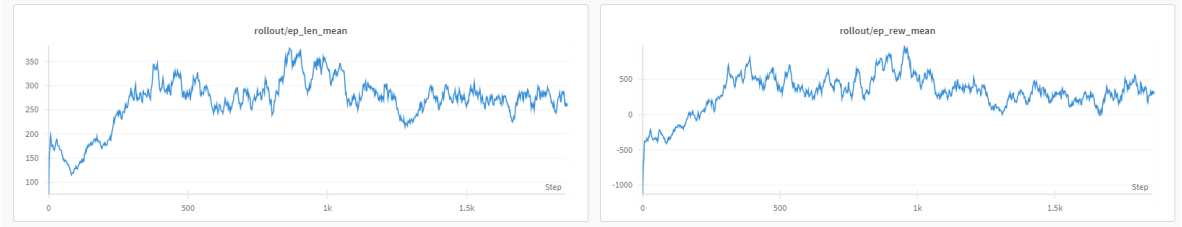
```
-               self.reward += 1/60
-               self.reward -= self.drone.distance_to_target / 6000
-
-               if self.time > self.time_limit:
-                   done = True
-
-               if self.drone.distance_to_target < 50 and not
    self.mouse_target:
-                   self.reward += 500
-                   self.targetX = np.random.randint(30, self.width -30)
```

```
-                   self.targetY = np.random.randint(30, self.height -30)
-                   self.drone.set_target(self.targetX, self.targetY)
-                   self.targets_hit += 1
-                   print(f'targets hit: {self.targets_hit}')
-               if self.drone.distance_to_target > 1000:
-                   self.reward -= 500
-                   done = True
```

3.6.2 Resultados del entrenamiento

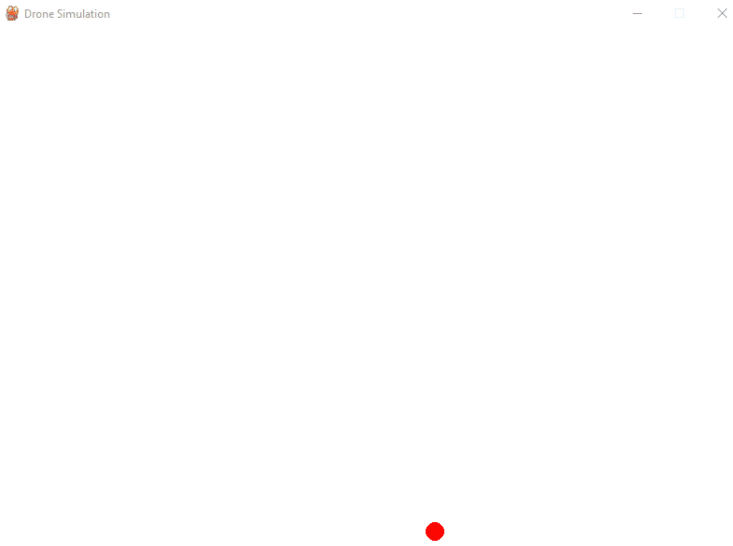
Con este modelo inmediatamente se obtuvieron mejores resultados que con la red neuronal DQN, mostrando tendencias positivas en la ganancia de recompensa, y decrecimientos en la función de perdida

Dado esto se realizaron entrenamientos de larga duración con el fin de obtener una estabilización en la función de recompensa, estos experimentos llegaron a tener alrededor de 5 millones de pasos de simulación, y duraciones de alrededor de 22 horas, las graficas resultantes de algunos de estos experimentos son las siguientes:



Se observa después del crecimiento inicial una oscilación en un valor positivo de recompensa.

Se obtuvo entonces un aprendizaje exitoso del dron para ir a las coordenadas del objetivo con movimientos mucho mas agresivos que los que presenta el controlador PID, sin embargo, estos movimientos agresivos resultaron en el dron saliendo del área designada algunas veces.

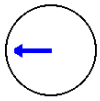


Enlace para ver la imagen animada

3.7 Perturbaciones aleatorias

Como perturbaciones aleatorias en el sistema se implemento viento al entorno de vuelo, el cual ejerce una fuerza adicional en el dron en una dirección que se puede establecer como aleatoria, o seleccionar una dirección horizontal constante.

Para representar estas perturbaciones se añadió un indicador a la interfaz que muestra la dirección del viento.



3.8 Repositorio de código

El control de versiones del proyecto se realizo utilizando GitLab en el repositorio que se encuentra en el siguiente Link

Adicional a esto, como se menciono anteriormente para el almacenamiento de datos se utilizó la herramienta Weights and Biases en el espacio de trabajo que se encuentra en el siguiente Link

4 Resultados

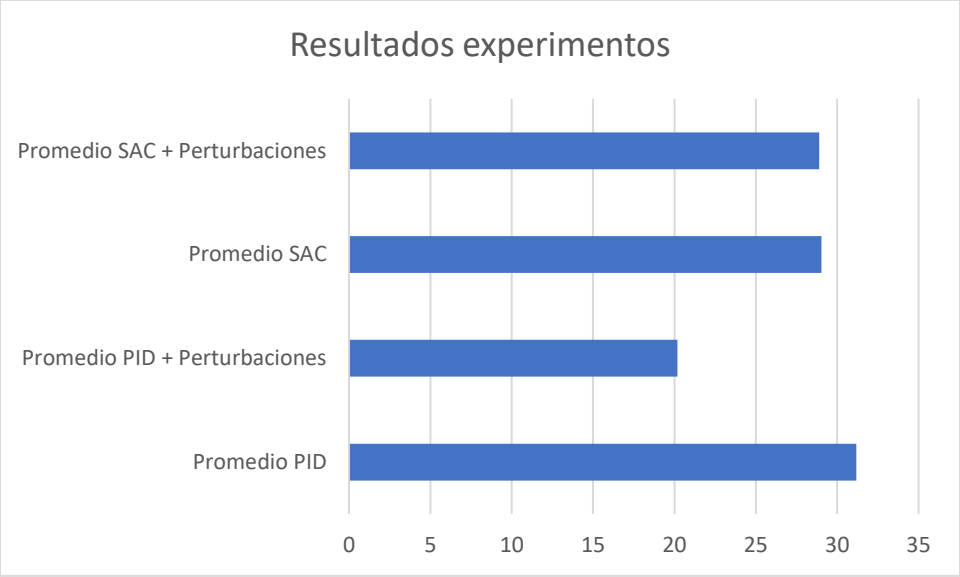
Después de tener el modelo final de la red SAC y el controlador PID con la mejor sintonización obtenida se realizaron 400 experimentos distribuidos de la siguiente manera:

Cada método se sometió a 200 experimentos divididos en:

- 100 experimentos sin perturbaciones aleatorias
- 100 experimentos con perturbaciones aleatorias,

cada experimento tuvo una duración de 100 segundos y se obtiene al final del experimento el numero de objetivos alcanzado por el dron en este tiempo, con esto se obtuvieron los siguientes resultados:

Modelo	Promedio de objetivos (sin PA)	Promedio de objetivos (con PA)	Reducción de desempeño a causa de las perturbaciones aleatorias
SAC	29.04	28.9	0.48%
PID	31.17	20.18	35.25%



5 Conclusiones

El aprendizaje por refuerzo utilizando una red SAC es capaz de aprender exitosamente el control de un dron simulado, además de ser muy resistente a perturbaciones inesperadas y adaptarse a ellas, sin embargo, este desempeño solo se empezó a ver después de sesiones de entrenamiento de varios millones de iteraciones, y acompañado de un alto costo computacional. Adicional a esto, el modelo muestra un overshoot que algunas veces causa que se aleje de su área designada, lo cual en un entorno real podría llevar a colisiones.

El PID por otro lado, tuvo una gran reducción en su rendimiento al aplicar perturbaciones aleatorias, pero el vuelo del dron fue muy estable incluso al aplicar fuertes vientos, lo cual se considera un comportamiento mas adecuado para aplicaciones reales donde un solo error en el comportamiento del dron podría ser catastrófico. Además, la implementación de los controladores PID es muy rápida ya que no se necesita de un proceso de entrenamiento, sino solamente una sintonización de parámetros, que con métodos ya existentes como lo es el método de Ziegler-Nichols, se puede realizar muy rápidamente.

Es por estas razones que se concluyó que para usos generales es mas apropiado un controlador tradicional. Los controladores por inteligencia artificial podrían tener lugar en el ámbito de los drones en aplicaciones especializadas donde se requieran movimientos agiles como por ejemplo competencias de drones, o posibles aplicaciones militares.

6 Referencias bibliográficas

Aishwarya Srinivasan. (2018, June 26). *Reinforcement Learning: The Business Use Case*.

Åström, K. J., & Hägglund, T. (2001). The future of PID control. *Control Engineering Practice*, *9*(11), 1163–1175. https://doi.org/10.1016/S0967-0661(01)00062-4

Choi, E., Hewlett, D., Lacoste, A., Polosukhin, I., Uszkoreit, J., & Berant, J. (2016). *Hierarchical Question Answering for Long Documents*.

Chris Gamble, J. G. (2018, August 17). *Safety-first AI for autonomous data centre cooling and industrial control*.

Grossi, E., & Buscema, M. (2007). Introduction to artificial neural networks. *European Journal of Gastroenterology & Hepatology*, *19*(12), 1046–1054. https://doi.org/10.1097/MEG.0b013e3282f198a0

Jason Gauci, E. C. K. V. (2018, November 1). *Horizon: The first open source reinforcement learning platform for large-scale products and services*.

Katsuhiko Ogata. (2001). *Modern Control Engineering*. Prentice Hall PTR.

Lee, D., Koo, S., Jang, I., & Kim, J. (2022). Comparison of Deep Reinforcement Learning and PID Controllers for Automatic Cold Shutdown Operation. *Energies*, *15*(8), 2834. https://doi.org/10.3390/en15082834

Li, Y. (2018). *Deep Reinforcement Learning*.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., & Hassabis, D. (2017). Mastering the game of Go without human knowledge. *Nature*, *550*(7676), 354–359. https://doi.org/10.1038/nature24270

Sutton, R. S., & Barto, A. G. (n.d.). *Reinforcement Learning An Introduction second edition*.

Tahir, M. A., Mir, I., & Islam, T. U. (2023). Control Algorithms, Kalman Estimation and Near Actual Simulation for UAVs: State of Art Perspective. *Drones*, *7*(6). https://doi.org/10.3390/drones7060339

Tim Wescott. (2000). PID without a PhD. *FLIR Systems*.

Yu, C., Liu, J., & Nemati, S. (2019). *Reinforcement Learning in Healthcare: A Survey*.

Zulu, A., & John, S. (2016). *A Review of Control Algorithms for Autonomous Quadrotors*. https://doi.org/10.4236/ojapps.2014.414053

Coordinador Maestría en Ciencias de los Datos y Analítica

La Universidad

Estimado coordinador,

A continuación, presento el proyecto titulado: "***Comparación de algoritmos de control tradicionales con control por medio de IA en un entorno simulado 2D***" para que este sea considerado como trabajo de maestría en la modalidad de Profundización.

Este proyecto ha sido revisado por el director **Alejandro Puerta Echandía** y recibe su aval para ser presentado al comité. De antemano agradezco su colaboración y trámite respectivo ante el comité.

	Alejandro Puerta E.
Mateo Fernando Penagos Ramírez	**Alejandro Puerta Echandía**
1037629615	1040181938