Universitat Oberta de Catalunya (UOC)

Máster Universitario en Ciencia de Datos (*Data Science*)

TRABAJO FINAL DE MÁSTER

Área: Deep Reinforcement Learning Applications

Explorando técnicas avanzadas de RL para conducción autónoma

Autor: Víctor Díaz Bustos

Tutor: Raúl Parada Medina

Profesor: Susana Acedo Nadal

Granada, 6 de mayo de 2025

Créditos/Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	Explorando técnicas avanzadas de RL para conducción autónoma
Nombre del autor:	Víctor Díaz Bustos
Nombre del colaborador/a docente:	Raúl Parada Medina
Nombre del PRA:	Susana Acedo Nadal
Fecha de entrega (mm/aaaa):	05/2025
Titulación o programa:	Master Ciencia de Datos
Área del Trabajo Final:	Área 5
Idioma del trabajo:	Español
Palabras clave	Aprendizaje por Refuerzo Profundo, Conducción autónoma, Entornos Simulados

Dedicatoria/Cita

TODO: Breves palabras de dedicatoria y/o una cita.

Agradecimientos

TODO: Si se considera oportuno, mencionar a las personas, empresas o instituciones que hayan contribuido en la realización de este proyecto.

Abstract

Achieving completely safe **autonomous driving** represents one of the greatest technological challenges, with the potential to transform both the automotive industry and our daily lives. An efficient, accident-free mobility system would redefine our perception of transportation, both in urban environments and long-distance travel. Recent advances in telecommunications, sensors, and **Artificial Intelligence** are bringing this vision closer to reality.

This work addresses the challenge of achieving smooth and precise driving within a simulated autonomous driving environment. To this end, **Deep Reinforcement Learning** (DRL) methods will be employed — a discipline that combines **Reinforcement Learning** (RL) with **Deep Learning** (DL).

The simulator chosen for the project's development is **SUMO**, which allows large-scale vehicular traffic simulation and facilitates the integration of custom algorithms for mobility management.

Keywords: Autonomous Driving, Deep Reinforcement Learning, SUMO, Simulated Environments.

Resumen

Lograr una **conducción autónoma** completamente segura representa uno de los mayores desafíos tecnológicos, con el potencial de transformar tanto la industria automovilística como nuestra vida diaria. Un sistema de movilidad eficiente y sin accidentes redefiniría nuestra percepción del transporte, tanto en entornos urbanos como en trayectos de larga distancia. Los recientes avances en telecomunicaciones, sensores e **Inteligencia Artificial** acercan cada vez más esta visión a la realidad.

Este trabajo aborda el reto de conseguir una conducción fluida y precisa dentro de un entorno simulado de conducción autónoma. Para ello, se emplearán métodos de Aprendizaje por Refuerzo Profundo (DRL, **Deep Reinforcement Learning**), una disciplina que combina el Aprendizaje por Refuerzo (RL, **Reinforcement Learning**) con Redes Neuronales Profundas (DL, **Deep Learning**).

El simulador seleccionado para el desarrollo del proyecto es **SUMO**, que permite simular tráfico vehicular a gran escala y facilita la integración de algoritmos personalizados para la gestión de la movilidad.

Palabras clave: Conducción Autónoma, Aprendizaje por Refuerzo Profundo, SUMO, Entornos Simulados.

Índice general

Índice de figuras

Índice de cuadros

1. Introducción

1.1. Contexto y motivación

La **conducción autónoma** es uno de los campos de mayor impacto en la investigación de **sistemas inteligentes** de transporte. La simulación de escenarios urbanos permite evaluar, de manera segura y controlada, estrategias de conducción basadas en algoritmos de **aprendizaje por refuerzo**.

Existen **seis niveles de conducción autónoma**, que van desde el nivel 0 (sin ningún tipo de automatización) hasta el nivel 5, en el que el vehículo opera de manera totalmente autónoma en todas las condiciones sin intervención humana [4].

Actualmente, la mayoría de los vehículos comerciales disponen de sistemas de nivel 2, que ofrecen asistencia parcial en dirección, aceleración y frenado. Algunos prototipos y modelos experimentales ya han empezado a incorporar capacidades de nivel 3, permitiendo cierta automatización en entornos controlados, y en áreas georrestringidas se están probando vehículos de nivel 4. Empresas como **Waymo** y **Cruise** han desplegado servicios de robotaxi en áreas georestringidas (como algunas zonas de Phoenix o San Francisco, respectivamente) [5] [6]. En estos entornos controlados, los vehículos pueden operar de forma completamente autónoma sin intervención humana, siempre y cuando se mantengan dentro de los límites geográficos y de condiciones predefinidas. Sin embargo, alcanzar la plena autonomía (nivel 5) sigue siendo un reto tanto tecnológico como regulatorio.

Sobre estos niveles y las tecnologías de IA comentadas se hablará en detalle en el capítulo de estado del arte.

La elección de la temática para el trabajo fin de máster ha estado motivada por dos razones principales. Por un lado, DL y RL han sido dos de las asignaturas más interesantes cursadas durante el máster y este proyecto me da la oportunidad de ampliar mis conocimientos en estos campos. Por otra parte, actualmente trabajo en el sector de la automoción, por lo que adquirir experiencia en un campo con una proyección como la conducción autónoma puede beneficiarme laboralmente

El presente TFM propone el desarrollo de un entorno en SUMO que integra un vehículo controlable mediante la API TraCI, permitiendo:

- Registrar el estado del vehículo y su entorno (posición, velocidad, información de carril, detección de vehículos cercanos, etc.).
- Penalizar comportamientos no deseados (colisiones, teletransportación, maniobras forzadas) y recompensar conductas seguras.

- Reasignar rutas de forma dinámica cuando el vehículo alcanza el final del tramo, simulando condiciones reales de adaptación en la conducción.

Esta propuesta es relevante tanto desde el punto de vista científico como práctico, ya que aporta un marco de pruebas para algoritmos de RL en entornos complejos y contribuye al avance de la conducción autónoma).

La elección de este TFM surge de mi fuerte interés en explorar el potencial del aprendizaje por refuerzo y del aprendizaje profundo, áreas en las que he profundizado a través de las asignaturas *"Deep Learning"* y *Reinforcement Learning"*. Aunque mi experiencia práctica en simulaciones de conducción autónoma es aún incipiente, estoy motivado por aplicar técnicas avanzadas de RL a problemas reales, especialmente en el ámbito del transporte urbano. Considero que la integración de simuladores como SUMO con métodos de aprendizaje por refuerzo abre la puerta a soluciones innovadoras que pueden mejorar significativamente la seguridad y eficiencia de los sistemas de movilidad, convirtiéndose en una contribución valiosa tanto a nivel académico como práctico.

1.2. Objetivos

Este trabajo final de máster se centra en el diseño, implementación y evaluación de un entorno de simulación para la conducción autónoma utilizando SUMO (Simulation of Urban MObility) y la interfaz TraCI. Se desarrolla un entorno en Python en el que un vehículo controlable es entrenado mediante algoritmos de aprendizaje por refuerzo (RL) para aprender a conducir de forma segura y eficiente.

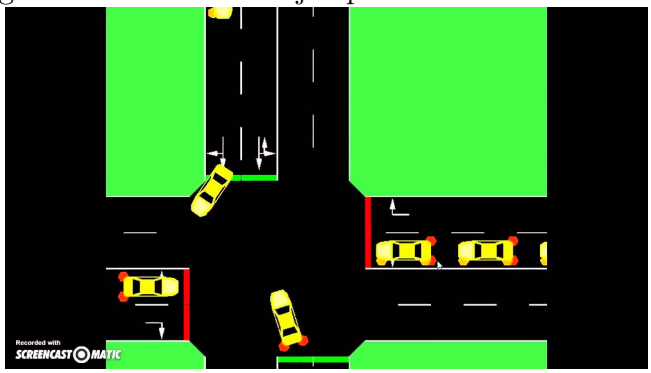
El sistema penaliza eventos adversos, como colisiones, teletransportaciones y maniobras inválidas, y recompensa comportamientos deseables (por ejemplo, mantener velocidad adecuada, recorrer distancias y alcanzar el final de la ruta). Además, se implementa la reasignación dinámica de rutas, de modo que al llegar al final de un tramo, el vehículo se redirige hacia una nueva ruta aleatoria desde su posición actual, imitando situaciones reales en las que el conductor debe adaptarse al entorno.

En la Figura 1 se aprecia una situación de ejemplo del simulador SUMO.

Para alcanzar este objetivo, haremos uso de métodos pertenecientes a dos de las ramas de la IA más prometedoras: el **Aprendizaje Profundo** (DL) y el **Aprendizaje por Refuerzo**(RL).

En DL se intenta, a través de un algoritmo automático jerárquico, emular la forma de aprender del cerebro humano. Los modelos de DL aprenden por sí mismos y sin reglas previamente establecidas mediante una fase de entrenamiento. Debido a esta similitud con el aprendizaje humano, las redes utilizadas en DL se denominan redes neuronales. Estas redes son capaces de obtener conceptos abstractos y complejos "mezclando" otros más simples a lo largo de sus

Figura 1: Situación de ejemplo del simulador SUMO.



capas.

Por otra parte, en RL un agente aprende a tomar decisiones correctas a través de prueba y error, interaccionando con el entorno. Dado un estado, el agente realizará una acción en base a recompensas generadas por dicho entorno. Esta acción dará lugar a un nuevo estado del agente. Escenificamos a continuación esta lógica con un ejemplo simple. El agente es el coche autónomo. Una recompensa podría ser negativa al llegar al borde de la carretera. Al recibir esta recompensa el coche realizará la acción de girar el volante para mantenerse en la carretera.

A lo largo de este trabajo, aplicaremos técnicas de **Aprendizaje por Refuerzo Profundo** (DRL). Estos modelos aúnan las cualidades de las técnicas expuestas anteriormente. Siguiendo la lógica del RL, en DRL la capacidad de aprendizaje del agente viene dada por una red neuronal profunda. De esta forma, el agente podrá tomar decisiones a partir de datos de entrada no estructurados como son las imágenes.

1.2.1. Objetivo principal

Desarrollar e implementar un entorno de simulación basado en SUMO que permita entrenar a un agente de aprendizaje por refuerzo para la conducción autónoma, comparando diferentes algoritmos de RL que buscan maximizar la seguridad y eficiencia en la circulación.

1.2.2. Objetivos secundarios

- Definir y desarrollar el entorno de simulación: Crear una interfaz en Python utilizando TraCI que permita controlar un vehículo manual en SUMO, extrayendo información del entorno y gestionando rutas dinámicas.

- Diseñar la función de recompensa: Establecer penalizaciones para colisiones, teletransportaciones y maniobras inválidas, y recompensas para la velocidad adecuada, distancia recorrida y llegada exitosa al destino.

- Implementar el cambio dinámico de rutas: Permitir que, al finalizar un tramo, el vehículo reciba una nueva ruta aleatoria desde su posición actual.
- Recopilar y procesar el estado del vehículo y su entorno: Obtener, en cada paso, información detallada del vehículo (posición, velocidad, ángulo, carril, etc.) y de los vehículos cercanos, para alimentar al algoritmo de RL.
- Evaluar y comparar distintos algoritmos de RL: Analizar el desempeño del agente en diferentes escenarios y ajustar la función de recompensa para mejorar la conducción.

1.3. Sostenibilidad, diversidad y desafíos ético/sociales

El proyecto se centra en el desarrollo de un entorno de simulación para la conducción autónoma utilizando técnicas de aprendizaje por refuerzo. A continuación se evalúan los posibles impactos en distintas dimensiones [16]:

Sostenibilidad Aunque el desarrollo se realiza en un entorno simulado y, en principio, no implica un consumo directo de recursos a gran escala, el enfoque del proyecto tiene un potencial impacto positivo en la movilidad urbana sostenible. La aplicación de algoritmos de RL en la conducción autónoma podría, en escenarios reales, contribuir a optimizar el flujo de tráfico, reducir los atascos y, por ende, disminuir el consumo de combustible y las emisiones contaminantes. Además, una mejor eficiencia en la conducción se alinea con algunos de los Objetivos de Desarrollo Sostenible (ODS), como el ODS 11 (Ciudades y comunidades sostenibles) y el ODS 9 (Industria, innovación e infraestructura). Se analizará en el desarrollo cómo la optimización de la conducción puede influir en la reducción de la huella ecológica del transporte.

Comportamiento ético y responsabilidad social Al tratarse de un entorno de simulación para la investigación en aprendizaje por refuerzo, el proyecto es en gran medida técnico. Sin embargo, su aplicación en la conducción autónoma tiene implicaciones éticas y sociales importantes. Por un lado, el desarrollo de sistemas de conducción más seguros y eficientes puede reducir el número de accidentes y mejorar la seguridad vial, lo que tiene un impacto positivo en la sociedad. Por otro lado, la integración de estas tecnologías en entornos reales exige cumplir con las normativas relativas a la privacidad, la seguridad de los datos y la responsabilidad profesional. En este sentido, el proyecto se desarrollará siguiendo los principios éticos de la profesión y se tendrá en cuenta la normativa vigente en materia de protección de datos, garantizando un tratamiento adecuado de la información personal si llegara a emplearse en futuros desarrollos o validaciones con datos reales.

Diversidad, género y derechos humanos El carácter técnico del proyecto implica que, en su fase de investigación y desarrollo, el impacto directo en términos de diversidad, género o derechos humanos es limitado. No obstante, es importante considerar que la aplicación de tecnologías de conducción autónoma puede influir en la accesibilidad y en la seguridad de todos los usuarios de la vía, incluyendo personas con discapacidad o en situación de vulnerabilidad. Además, la incorporación de sistemas inteligentes en el transporte debe diseñarse de manera inclusiva, garantizando que la tecnología sea accesible y segura para la mayor diversidad de usuarios. Se analizará, en la fase de conclusiones, cómo estas tecnologías pueden contribuir a la igualdad y a la mejora de la calidad de vida en entornos urbanos.

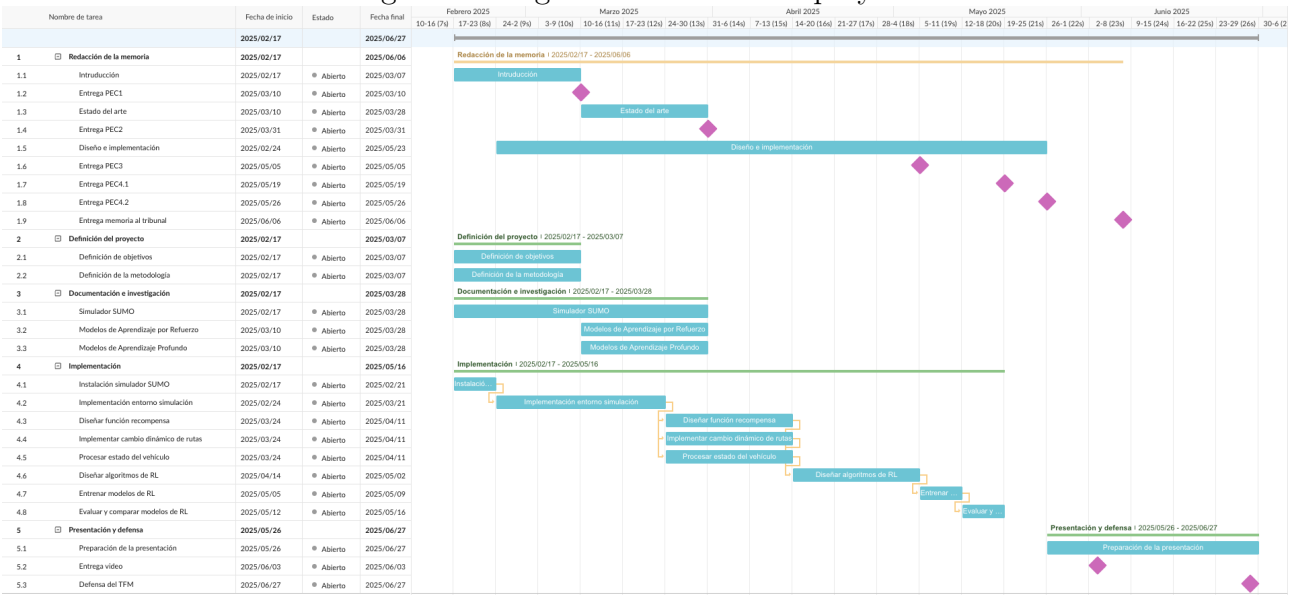
1.4. Enfoque y metodología

El TFM se desarrollará mediante las siguientes estrategias:

- **Investigación bibliográfica**: Revisión de literatura en aprendizaje por refuerzo, simulación de tráfico y conducción autónoma, apoyándose en artículos científicos, tesis y documentación oficial de SUMO y TraCI.
- **Desarrollo del entorno de simulación**:
  - **Implementación en Python**: Se desarrollará un entorno que utilice la API TraCI para controlar un vehículo en SUMO, extrayendo información del estado, aplicando acciones y evaluando recompensas.
  - **Gestión de rutas**: El entorno permitirá asignar rutas válidas y, cuando se alcance el final de la ruta, se forzará una maniobra (incluso si es inválida) para simular una situación de colisión o error, que se penalizará y reiniciará el episodio.
- **Entrenamiento de algoritmos de RL**: Se probarán distintos métodos como ***DDQN*** *(Dueling Deep Q-Network),* ***PPO*** *(Proximal Policy Optimization),* ***A3C*** *(Asynchronous Advantage Actor Critic)* o ***SAC*** *(Soft Actor-Critic)*, en los que se profundizará posteriormente en el apartado de *Implementacion*, para entrenar al agente en el entorno desarrollado.
- **Validación y análisis**: Se analizarán los resultados del entrenamiento, comparando métricas de seguridad y eficiencia, y se realizarán ajustes en la función de recompensa y en la configuración del entorno.

1.5. Planificación

Figura 2: Diagrama de Gantt del proyecto.



Nombre de tarea	Fecha de inicio	Estado	Fecha final
	2025/02/17		2025/06/27
1 Redacción de la memoria	2025/02/17		2025/06/06
1.1 Intruducción	2025/02/17	Abierto	2025/03/07
1.2 Entrega PEC1	2025/03/10	Abierto	2025/03/10
1.3 Estado del arte	2025/03/10	Abierto	2025/03/28
1.4 Entrega PEC2	2025/03/31	Abierto	2025/03/31
1.5 Diseño e implementación	2025/02/24	Abierto	2025/05/23
1.6 Entrega PEC3	2025/05/05	Abierto	2025/05/05
1.7 Entrega PEC4.1	2025/05/19	Abierto	2025/05/19
1.8 Entrega PEC4.2	2025/05/26	Abierto	2025/05/26
1.9 Entrega memoria al tribunal	2025/06/06	Abierto	2025/06/06
2 Definición del proyecto	2025/02/17		2025/03/07
2.1 Definición de objetivos	2025/02/17	Abierto	2025/03/07
2.2 Definición de la metodología	2025/02/17	Abierto	2025/03/07
3 Documentación e investigación	2025/02/17		2025/03/28
3.1 Simulador SUMO	2025/02/17	Abierto	2025/03/28
3.2 Modelos de Aprendizaje por Refuerzo	2025/03/10	Abierto	2025/03/28
3.3 Modelos de Aprendizaje Profundo	2025/03/10	Abierto	2025/03/28
4 Implementación	2025/02/17		2025/05/16
4.1 Instalación simulador SUMO	2025/02/17	Abierto	2025/02/21
4.2 Implementación entorno simulación	2025/02/24	Abierto	2025/03/21
4.3 Diseñar función recompensa	2025/03/24	Abierto	2025/04/11
4.4 Implementar cambio dinámico de rutas	2025/03/24	Abierto	2025/04/11
4.5 Procesar estado del vehículo	2025/03/24	Abierto	2025/04/11
4.6 Diseñar algoritmos de RL	2025/04/14	Abierto	2025/05/02
4.7 Entrenar modelos de RL	2025/05/05	Abierto	2025/05/09
4.8 Evaluar y comparar modelos de RL	2025/05/12	Abierto	2025/05/16
5 Presentación y defensa	2025/05/26		2025/06/27
5.1 Preparación de la presentación	2025/05/26	Abierto	2025/06/27
5.2 Entrega video	2025/06/03	Abierto	2025/06/03
5.3 Defensa del TFM	2025/06/27	Abierto	2025/06/27

En la Figura 2 se representa el diagrama de Gantt del proyecto, donde se pueden distinguir las diferentes partes del trabajo y su planificación temporal, que comprende desde mediados de febrero hasta finales de junio de 2025. Este está compuesto por los siguientes items:

- **Grupos o Fases**: Son bloques grandes que engloban un conjunto de tareas relacionadas (por ejemplo, "Redacción de la memoria", "Implementación", etc.). Permiten organizar el proyecto en secciones o etapas lógicas.
- **Hitos**: Aparecen como puntos o diamantes que representan un suceso clave o entrega importante (por ejemplo, una fecha de revisión o la defensa del TFM). Normalmente no tienen duración, solo una fecha concreta.
- **Tareas**: Se muestran como barras horizontales que indican la duración (fecha de inicio y fin) de cada actividad. Suelen incluir una breve descripción de la acción a realizar (por ejemplo, "Diseñar función recompensa").
- **Estado**: Suele ser una columna adicional en la que se refleja el progreso (p.ej., .ªbierto", "En curso", "Hecho", "Cerrado"), lo que facilita el seguimiento del avance en cada tarea o fase.

Se puede observar como algunas de las tareas se realizan de forma paralela.
La descripción de las etapas es la siguiente:

- **Redacción de la memoria**: La memoria se irá redactando a lo largo de toda la duración del proyecto. Existiendo diferentes hitos que corresponden a los entregables exigidos para superar la asignatura.
- **Definición del proyecto**: Durante esta etapa quedan definidos los objetivos y la metodología a seguir durante el TFM.
- **Documentación e investigación**: Las tareas dentro de este apartado tienen como objetivo documentarse sobre todas las posibilidades del simulador SUMO, así como de la teoría de los modelos que se aplicarán en el proyecto.
- **Implementación**: Tras instalar el simulador, se comenzará a implementar el entorno y los modelos de DRL, así como su entrenamiento, evaluación y comparación. Una parte importante será la optimización de los parámetros para conseguir que los modelos se comporten de la mejor manera y sean lo más generales posible. El objetivo será conseguir una conducción autónoma correcta y fluida.
- **Presentación y defensa**: Una vez implementado el proyecto y redactada la memoria, se procede a preparar la presentación que se utilizará en la defensa del TFM. Con los hitos de la entrega del video y la defensa del proyecto se dará por concluido el trabajo fin de máster.

1.6. Resumen de los productos del proyecto

No es necesario describir cada producto en detalle: esto se hará en los capítulos restantes del proyecto.

1.7. Breve descripción de los demás capítulos del informe

Breve descripción de los contenidos de cada capítulo y su relación con el resto del proyecto.

2. Estado del arte

2.1. Introducción al estado del arte

La conducción autónoma representa uno de los campos de investigación más activos y desafiantes en el ámbito de la **Inteligencia Artificial (IA)** y la **robótica**. Numerosos trabajos abordan esta problemática desde perspectivas diversas, incluyendo la **visión por computador**, la **fusión de sensores** y la **planificación de trayectorias**. En los últimos años, las técnicas de Aprendizaje por Refuerzo Profundo (**DRL**) han cobrado relevancia para entrenar agentes capaces de tomar decisiones en tiempo real [8, 9].

En esta sección, se revisan las investigaciones y soluciones más destacadas relacionadas con:

- La conducción autónoma en simuladores (especialmente SUMO).
- El uso de algoritmos de Aprendizaje por Refuerzo para optimizar la toma de decisiones.
- El estado de la investigación en cuanto a limitaciones, éxitos y retos actuales.

Este estudio bibliográfico constituye la base para la propuesta de este TFM, justificando las decisiones metodológicas y los objetivos definidos.

2.2. Relevancia y motivación de la problemática

El avance hacia una conducción autónoma **segura y eficiente** representa uno de los mayores retos tecnológicos y sociales de la actualidad. La conducción autónoma no sólo implica el dominio de la **física del vehículo**, sino también la capacidad de **interactuar** de forma segura en entornos urbanos complejos, donde la densidad del tráfico, la diversidad de actores (vehículos, peatones, ciclistas) y las condiciones cambiantes hacen que la toma de decisiones en tiempo real sea fundamental. En este contexto, la simulación de escenarios de tráfico realistas se vuelve indispensable para evaluar y perfeccionar las estrategias de control, ya que permite replicar situaciones de riesgo sin exponer a personas o bienes a peligros reales [11, 12].

Numerosos estudios han demostrado el potencial del Aprendizaje por Refuerzo Profundo (DRL) para entrenar agentes autónomos en entornos controlados. Sin embargo, gran parte de la literatura se ha enfocado en simuladores con escenarios relativamente simples o en circuitos cerrados, lo que limita la validez de los resultados cuando se pretende aplicar la solución a contextos urbanos reales. Por ejemplo, trabajos como los presentados por Dosovitskiy et al. [11] y Sallab et al. [12] han logrado avances notables en el entrenamiento de agentes para tareas de maniobra y toma de decisiones en entornos simplificados, pero estos enfoques suelen prescindir de la complejidad inherente a la gestión de flujos de tráfico a gran escala.

La integración de SUMO (Simulation of Urban MObility) con algoritmos de DRL se presenta como una solución innovadora y necesaria para cubrir este vacío. SUMO ofrece la capacidad de **modelar escenarios urbanos a gran escala**, lo que permite incorporar aspectos como la interacción entre múltiples vehículos, la variabilidad en las condiciones del tráfico y la influencia de infraestructuras urbanas (intersecciones, semáforos, carriles exclusivos, etc.). Esta capacidad de simulación realista es crucial para evaluar estrategias de control que optimicen no sólo la maniobra de un vehículo, sino también decisiones relacionadas con la elección de rutas, el control de velocidad y la interacción con el resto de actores del tráfico.

Además, al combinar estas dos áreas (simulación de tráfico y DRL) se abre la posibilidad de desarrollar soluciones que puedan adaptarse dinámicamente a situaciones imprevistas y, en última instancia, contribuir a reducir accidentes y mejorar la eficiencia del transporte urbano. Este enfoque también aborda aspectos éticos y de seguridad, ya que permite probar y validar los algoritmos en un entorno controlado antes de su eventual aplicación en entornos reales.

Por lo tanto, en este TFM se propone explorar y validar la integración de SUMO y técnicas de DRL para desarrollar un agente que no solo realice maniobras de conducción a nivel individual, sino que también tome decisiones complejas en un entorno urbano simulado. Este trabajo pretende contribuir al estado del arte proporcionando un marco de simulación más realista y escalable, que permita una mejor extrapolación de los resultados a escenarios del mundo real.

2.3. Trabajos previos y justificación

2.3.1. Uso de simuladores de tráfico

- **SUMO**: *SUMO (Simulation of Urban MObility)* es un simulador de tráfico de código abierto ampliamente reconocido por su capacidad para modelar escenarios urbanos complejos a gran escala. Diversos estudios han demostrado su flexibilidad para simular semáforos, rutas de autobuses, y variaciones en la densidad de tráfico [14]. Sin embargo, la mayoría de las implementaciones actuales se centran en la optimización de sistemas de control de tráfico (por ejemplo, el ajuste de ciclos de semáforos o la optimización de rutas) sin explorar en profundidad el control individual de vehículos mediante algoritmos de Aprendizaje por Refuerzo. Este TFM se diferencia en que propone integrar SUMO con algoritmos de DRL para entrenar agentes que tomen decisiones de conducción a nivel individual, interactuando con otros vehículos y adaptándose a condiciones dinámicas en entornos urbanos reales. Además, el uso de SUMO permite aprovechar una red vial detallada y realista, lo cual es fundamental para validar modelos de RL en escenarios complejos.

- **Otros simuladores**: Simuladores como *CARLA* se centran en la simulación de la física

del vehículo y en la percepción visual, proporcionando un entorno rico para el procesamiento de imágenes y la detección de objetos [11]. Estos entornos son ideales para entrenar algoritmos basados en visión, pero suelen limitarse a escenarios con una menor escala de tráfico o en entornos controlados (p. ej., circuitos cerrados). Además, la alta demanda de recursos computacionales para renderizar gráficos realistas hace que su uso en estudios a gran escala sea más complejo. En contraste, SUMO se orienta hacia la simulación de flujos de tráfico a nivel macro, permitiendo evaluar el comportamiento de múltiples agentes y la gestión global de rutas, lo que resulta esencial para abordar la conducción autónoma en entornos urbanos densos sin requerir hardware de última generación.

2.3.2. Algoritmos de Aprendizaje por Refuerzo

- **Aprendizaje por Refuerzo Clásico**: Técnicas tradicionales como **Q-Learning** o **SARSA** han sido aplicadas en entornos con **estados y acciones discretos**, en los que la complejidad del entorno se reduce significativamente. Aunque estos métodos son útiles para ilustrar conceptos básicos y se han empleado en escenarios de baja dimensión, su **escalabilidad es limitada** y se vuelven inadecuados para entornos con altos grados de incertidumbre o dimensiones continuas, como es el caso de la conducción autónoma.

- **Aprendizaje por Refuerzo Profundo (DRL)**: Los **métodos de DR**L, que combinan Aprendizaje por Refuerzo con Redes Neuronales Profundas, han permitido superar las limitaciones de los algoritmos clásicos. Técnicas como **Dueling Deep Q-Network (DDQN)**, **Proximal Policy Optimization (PPO)**, **Soft Actor-Critic (SAC)** y **Asynchronous Advantage Actor Critic (A3C)** han demostrado un rendimiento sobresaliente en tareas complejas de toma de decisiones [12, 8]. Estos algoritmos permiten la **aproximación de funciones de valor y políticas en espacios de alta dimensión**, lo que los hace aptos para entornos de conducción donde las variables (como posición, velocidad, estados de tráfico, etc.) son continuas y altamente interrelacionadas. No obstante, la mayoría de los estudios en DRL se centran en entornos simplificados o en simuladores con escasa interacción de tráfico, lo que subraya la necesidad de explorar su aplicación en entornos más realistas y complejos, como el propuesto en este TFM.

2.3.3. Retos actuales

- **Escalabilidad**: La conducción autónoma no depende únicamente de la maniobra de un solo vehículo, sino de la interacción con numerosos agentes en un entorno urbano. La escalabilidad se vuelve crítica al aumentar la densidad de tráfico, ya que el modelo debe procesar y reaccionar ante múltiples estímulos de forma simultánea. Este reto es aún más

evidente cuando se intenta entrenar algoritmos de DRL en entornos complejos, donde el espacio de estados y acciones se expande considerablemente.

- **Generalización**: Uno de los mayores desafíos en la aplicación de algoritmos de DRL es la capacidad del modelo para generalizar a situaciones que no fueron contempladas durante el entrenamiento. Muchos modelos entrenados en escenarios reducidos fallan al enfrentarse a condiciones reales y variables, lo que hace imprescindible diseñar entornos de simulación que capturen la complejidad del tráfico urbano. La generalización es clave para asegurar que el agente pueda operar de forma segura y eficiente en diferentes contextos y condiciones imprevistas.

- **Validación en entornos realistas**: Aunque existen numerosos simuladores, la mayoría se centra en escenarios controlados o simplificados. Es fundamental contar con un simulador que reproduzca la complejidad del tráfico urbano, incluyendo semáforos, transporte público, múltiples carriles y diversas condiciones de tráfico. Esta validación realista no solo facilita el entrenamiento de modelos robustos, sino que también permite evaluar el impacto de los algoritmos en la seguridad vial y en la eficiencia del transporte, aspectos críticos para la futura implementación de vehículos autónomos en entornos reales.

2.4. Vacíos detectados

A partir de la literatura revisada, se observa que:

- **Pocos trabajos combinan SUMO y DRL** para abordar la conducción autónoma a gran escala, centrándose principalmente en la optimización de semáforos o enrutamientos.

- Falta de estudios que evalúen el comportamiento de un agente autónomo en **escenarios urbanos completos**, donde el flujo de tráfico se gestione de forma realista.

2.5. Metodología de búsqueda y fuentes consultadas

Para el desarrollo de este estado del arte se han seguido las **pautas metodológicas de Oates** [13], utilizando principalmente las técnicas de ***"document-based research"*** y ***"survey"***:

- Se han empleado **palabras clave** como *"SUMO autonomous driving", "reinforcement learning traffic management", "deep reinforcement learning conduction urbana"* en Google Scholar y Web of Science.

- Se han consultado **repositorios de la UOC y bibliotecas virtuales de otras universidades** para acceder a artículos y libros especializados.
- Se ha utilizado el gestor de bibliografía de **Latex** para organizar las referencias y extraer citas de forma consistente.

2.6. Conclusiones del estado del arte

La revisión bibliográfica confirma la **importancia de la simulación** como herramienta para acelerar la investigación en conducción autónoma. Además, se evidencia la necesidad de **integrar algoritmos de Aprendizaje por Refuerzo Profundo** con entornos de tráfico a gran escala, un área que no está suficientemente explorada y que presenta un alto potencial de contribución. Por tanto, el presente TFM se centrará en:

1. **Desarrollar** un entorno de simulación en SUMO para la conducción autónoma.
2. **Implementar** algoritmos de DRL que controlen un vehículo individual, pero interactuando con múltiples vehículos y rutas variables.
3. **Evaluar** la robustez del agente ante distintas densidades de tráfico y condiciones de la red vial.

Así, se espera contribuir a la línea de investigación sobre conducción autónoma en escenarios urbanos complejos, ofreciendo una base sólida para futuros trabajos que deseen optimizar el tráfico de manera global.

3. Diseño e implementación del trabajo

En este capítulo se detalla el proceso de diseño e implementación del entorno de simulación para la conducción autónoma basado en SUMO y la integración de algoritmos de Aprendizaje por Refuerzo Profundo (DRL). El objetivo es desarrollar un sistema que permita entrenar a un agente de RL para que aprenda a conducir de forma segura y eficiente en un entorno urbano realista, gestionando tanto maniobras de conducción individuales como decisiones a nivel de enrutamiento. Se describirá la arquitectura del sistema, los componentes clave (como la interfaz con SUMO mediante TraCI, la función de recompensa, y el mecanismo de reasignación dinámica de rutas) y la metodología seguida para la implementación y validación del entorno. Este diseño constituye la base para experimentar con distintos algoritmos de DRL y evaluar su rendimiento en escenarios complejos.

3.1. Instalación del entorno SUMO

SUMO es un simulador de tráfico de código abierto que sigue una arquitectura Cliente-Servidor. Para instalar el entorno SUMO se debe descargar la versión correspondiente desde el repositorio oficial, en este caso, la 1.20.0, lo que garantiza contar con una versión estable y probada. En este proyecto se ha optado por instalar SUMO en la máquina local, de modo que tanto el servidor como el cliente se ejecuten en el mismo equipo, facilitando la integración mediante la API **TraCI**.

La instalación se realiza de la siguiente manera:

1. **Descarga e instalación del servidor:** Se accede a la página oficial de SUMO [14] y se descarga el instalador correspondiente a la versión recomendada. Una vez descargado, se ejecuta el instalador y se sigue el proceso habitual de instalación en el sistema operativo.

2. **Configuración del entorno:** Una vez instalado el servidor, se verifica que SUMO se ejecuta correctamente abriendo SUMO-GUI. En este proyecto se utilizará el cliente en Python, por lo que se recomienda crear un entorno virtual (por ejemplo, con `venv` o `conda`) utilizando Python 3.12, que garantice la compatibilidad con las funciones de la API TraCI.

3. **Integración de TraCI:** La API TraCI, que permite la comunicación en tiempo real entre el simulador y el código Python, se encuentra incluida en el paquete de herramientas de SUMO. Se debe añadir la ruta al directorio `tools` de SUMO en la variable de entorno `PYTHONPATH` o incluirla directamente en el script de Python. De este modo, se podrá importar el módulo `traci` y establecer la conexión entre el cliente y el servidor SUMO.

4. **Verificación de la instalación:** Se recomienda ejecutar un script sencillo que inicie SUMO en modo GUI y realice algunos pasos de simulación para confirmar que la comunicación a través de TraCI funciona correctamente. Las especificaciones hardware del sistema utilizado en este proyecto (procesador *Apple M2* y 16GB de RAM) cumplen con los requisitos mínimos para ejecutar SUMO de forma fluida [14].

Estas características, junto con la amplia documentación y comunidad de usuarios de SUMO, garantizan que el simulador pueda integrarse eficazmente en el proyecto, permitiendo una simulación de tráfico a gran escala y el control personalizado del vehículo a través de algoritmos de aprendizaje por refuerzo.

3.2. Arquitectura del sistema

El sistema se compone de los siguientes módulos principales:

- **Entorno SUMO (sumo_environment.py)**: Interfaz entre el agente de RL y el simulador SUMO. Responsable de:
  1. Iniciar y controlar la simulación SUMO.
  2. Añadir y gestionar el vehículo controlado por el agente.
  3. Obtener el estado del entorno (información del vehículo, entorno, etc.).
  4. Aplicar las acciones del agente al vehículo.
  5. Calcular la recompensa.
  6. Gestionar el fin de los episodios.
  7. Visualización (delegada a SUMO-GUI).
- **Agente de RL (algorithm_agent.py)**: Implementa el algoritmo de aprendizaje por refuerzo profundo correspondiente (DDQN, PPO, A3C y SAC). Responsable de:
  1. Recibir el estado del entorno.
  2. Seleccionar una acción basada en su política.
  3. Almacenar experiencias (transiciones).
  4. Aprender de las experiencias (actualizar su política y/o función de valor).
- **Script Principal (main.py)**: Orquesta el proceso de entrenamiento y/o evaluación. Responsable de:
  1. Crear instancias del entorno y del agente.

2. Ejecutar el bucle principal de entrenamiento (episodios).
3. Guardar/cargar el modelo del agente.
4. Visualizar el progreso del entrenamiento.

3.2.1. Diagrama de clases

TODO

3.3. Dueling Deep Q-Network (DDQN)

Para comprender a fondo DDQN es fundamental analizar los algoritmos base en los que se sustenta, principalmente los métodos basados en Q-learning. A continuación se detallan los algoritmos clave:

3.3.1. Q-Network

En primer lugar cabe aclarar que en el aprendizaje por refuerzo existen dos tipos de algoritmos:

- **off-policy**: Un algoritmo off-policy es aquel que aprende una política óptima diferente de la política utilizada para generar los datos. Es decir, la política que se está evaluando y mejorando (política objetivo) no es necesariamente la misma que la que se usa para explorar el entorno (política de comportamiento).
- **on-policy**: Un algoritmo off-policy es aquel que actualiza su política en función de la misma política con la que interactúan en el entorno.

Q-learning es un algoritmo de aprendizaje por refuerzo **off-policy** que busca aprender la **función de valor-acción** $Q(s, a)$. Esta función representa la **recompensa acumulada esperada** al tomar la acción a en el estado s y seguir la política óptima a partir de ese momento. La actualización se realiza utilizando la **ecuación de Bellman**:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right] \tag{1}$$

donde α es la **tasa de aprendizaje** y γ el **factor de descuento**. Debido a que Q-learning almacena los Q-valores en una tabla, su uso se vuelve impracticable en problemas con espacios de estados muy grandes o continuos [17].

3.3.2. Deep Q-Network (DQN)

DQN extiende el Q-learning utilizando redes neuronales profundas para aproximar la función $Q(s, a)$. En lugar de usar una tabla, se entrena una red que, dada la representación del estado, produce los Q-valores para cada acción posible. Entre sus técnicas clave se encuentran:

- **Experience Replay**: Se almacena un historial de experiencias (s, a, r, s') en un buffer y se entrena la red con muestras aleatorias, lo que rompe la correlación entre experiencias consecutivas y estabiliza el aprendizaje.
- **Red Objetivo (Target Network)**: Se utiliza una copia de la red que se actualiza periódicamente para calcular los valores objetivo, reduciendo la inestabilidad de las actualizaciones.

La **función de pérdida** se define como el error cuadrático medio entre el Q-valor estimado y el objetivo obtenido mediante la ecuación de Bellman.

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim D}\left[\left(y - Q(s, a; \theta)\right)^2\right] \tag{2}$$

Donde $\mathbb{E}_{(s,a,r,s')\sim D}$ es el **valor esperado** para una muestra aleatoria de las experiencias almacenadas en el **buffer de experiencias** D, $Q(s, a; \theta)$ es el **valor Q estimado por la red neuronal**, $Q(s', a'; \theta^-)$ es el **valor Q estimado por la red objetivo**, γ es el **factor de descuento**, y es el **valor objetivo para la actualización de la red**, calculado como la **recompensa inmediata** r más el **valor descontado de las futuras recompensas** $(\gamma \max_{a'} Q(s', a'; \theta^-))$, y θ y θ^- son los **parámetros de la red neuronal actual y la red objetivo**, respectivamente [18].

3.3.3. Dueling Deep Q-Network (DDQN)

En el enfoque clásico de DQN, se utiliza una única red neuronal para aproximar la función de valor–acción $Q(s, a)$, la cual estima la recompensa acumulada esperada de ejecutar la acción a en el estado s y seguir la política actual. Sin embargo, en muchos entornos resulta **redundante** evaluar de forma detallada la contribución de cada acción cuando, en muchos estados, el valor intrínseco del estado domina la dinámica de recompensa. El enfoque Dueling DQN propone separar este problema en dos componentes:

- **Valor del Estado** $V(s)$: Estima la bondad intrínseca del estado sin considerar las acciones.
- **Ventaja** $A(s, a)$: Mide la contribución adicional de cada acción respecto al valor promedio del estado.

La salida de la red se combina mediante la fórmula:

$$Q(s,a) = V(s) + \left(A(s,a) - \frac{1}{|A|} \sum_{a'} A(s,a') \right) \tag{3}$$

Esta separación permite que la red se concentre en aprender con mayor precisión qué estados son valiosos, lo que puede ser fundamental en entornos donde la elección de la acción tiene un impacto marginal en el retorno a largo plazo.

La red neuronal de Dueling DQN se compone de un **bloque de extracción de características compartido** (por ejemplo, capas convolucionales en entornos visuales) que luego se divide en dos ramasïndependientes:

- **Stream de Valor**: Esta rama termina en una neurona única que estima $V(s;\theta,\beta)$.
- **Stream de Ventaja**: Esta rama calcula un vector $A(s,a;\theta,\alpha)$ para cada acción a.

La combinación de ambas ramas se realiza mediante una **función de agregación** diseñada para asegurar la **identificabilidad**, es decir, que la descomposición en V y A sea única. Una formulación común es:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|A|} \sum_{a'} A(s,a';\theta,\alpha) \right) \tag{4}$$

Donde $Q(s,a;\theta,\alpha,\beta)$ es la **función de acción-valor Q** que estima el valor de tomar una acción a en el estado s, θ son los **parámetros** de la red neuronal que se encargan de **modelar** tanto la función de valor de estado como la de ventaja, α son los **parámetros** de la red neuronal asociados con la **función de ventaja**, β son los **parámetros** de la red neuronal asociados con la **función de valor del estado**, $V(s;\theta,\beta)$ es la **función de valor del estado** $V(s)$, que estima la calidad de un estado s sin considerar una acción específica, $A(s,a;\theta,\alpha)$ es la **función de ventaja** $A(s,a)$, que representa la diferencia entre el valor de la acción a y el valor promedio de todas las posibles acciones a' en el estado s y $\frac{1}{|A|}\sum_{a'} A(s,a';\theta,\alpha)$ es el **valor promedio de las ventajas** de todas las acciones posibles a' en el estado s, donde $|A|$ es el número total de acciones posibles. Este término ayuda a **centrar la función de ventaja alrededor de 0**, evitando que el valor de $A(s,a)$ sea arbitrario debido a la escala.

La sustracción del promedio de las ventajas garantiza que la estimación de $V(s)$ represente de forma aislada el valor basal del estado, mientras que $A(s,a)$ refleja únicamente el efecto diferencial de cada acción [19].

3.3.4. Proceso de entrenamiento

El entrenamiento de Dueling DQN sigue, en esencia, la metodología de Deep Q-Learning, haciendo uso de técnicas como:

- **Experience Replay**: Se almacena la experiencia en forma de tuplas (s, a, r, s') para romper la correlación temporal y estabilizar la actualización de los pesos.
- **Red Objetivo (Target Network)**: Se utiliza una red objetivo cuyas actualizaciones son periódicas para proporcionar un objetivo de aprendizaje más estable y reducir la oscilación en las estimaciones de Q.
- **Minimización de la Función de Pérdida**: La función de pérdida se define como el error cuadrático medio entre el valor $Q(s, a)$ predicho y el objetivo de Bellman, es decir:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim D}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right] \tag{5}$$

La estructura dueling se integra en el cómputo de $Q(s, a)$ como se describió anteriormente, permitiendo que el gradiente se propague de manera diferenciada en las ramas de V y A.

3.3.5. Pseudocódigo

Algorithm 1 Dueling Deep Q-Network (DDQN)

Entrada: γ (descuento), α (tasa de aprendizaje), τ (actualización de la red objetivo), buffer_size (tamaño del buffer de experiencias)

Inicializar red $Q(\theta)$ y red objetivo $Q'(\theta' = \theta)$
Inicializar buffer de experiencias D
for cada episodio **do**
    $estado \leftarrow entorno.reset()$
    for cada paso t **do**
        $accion \leftarrow \varepsilon$-greedy$(Q(estado), \varepsilon)$
        $nuevo\_estado, recompensa, done \leftarrow entorno.step(accion)$
        Almacenar $(estado, accion, recompensa, nuevo\_estado, done)$ en D
                                                ▷ Muestrear mini-batch de D
        $batch \leftarrow sample(D, batch\_size)$
                                ▷ Calcular Q-target usando red objetivo y descomposición Dueling
        $Q\_target \leftarrow recompensa + \gamma \cdot \text{máx}(Q'(nuevo\_estado)) \cdot (1 - done)$
                                                ▷ Calcular pérdida y actualizar θ
        $perdida \leftarrow MSE(Q(estado)[accion], Q\_target)$
        descenso_gradiente(θ, $perdida$)
                                                ▷ Actualizar red objetivo
        $\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$
        $estado \leftarrow nuevo\_estado$
        if done **then**
            break
        end if
    end for
end for

3.3.6. Ventajas y consideraciones

Entre las ventajas fundamentales de la arquitectura Dueling DQN destacan:

- **Estabilidad en la Estimación del Valor**: Al separar $V(s)$ de $A(s, a)$, el algoritmo puede aprender de manera más robusta qué estados son intrínsecamente buenos, independientemente de las acciones disponibles.

- **Mejora en la Eficiencia del Aprendizaje**: En situaciones en las que la elección de acción tiene un impacto marginal, la red puede concentrar recursos en estimar con precisión el valor del estado, reduciendo la varianza en la actualización de Q.

- **Identificabilidad**: La estrategia de sustracción (ya sea el promedio o, en algunas variantes, el máximo) permite superar el problema de no poder distinguir entre V y A únicamente a partir de Q.

Es importante notar que, en algunas implementaciones, la técnica Dueling se combina con la estrategia de Double DQN para mitigar la sobreestimación de Q, obteniéndose variantes conocidas como Dueling Double DQN o D3QN. Esta combinación utiliza la red actual para seleccionar la acción óptima y la red objetivo para evaluarla, lo que reduce el sesgo en la estimación [20].

3.4. Asynchronous Advantage Actor Critic (A3C)

Para comprender a fondo A3C es fundamental analizar los algoritmos base en los que se sustenta, principalmente los métodos de gradiente de política y las arquitecturas actor-crítico. A continuación se detallan los algoritmos clave:

3.4.1. REINFORCE (Policy Gradient Monte Carlo)

REINFORCE es uno de los algoritmos pioneros en el campo de los **métodos de gradiente de política**, propuesto por Williams en 1992. Sus características principales son:

- **Basado en Monte Carlo**: Utiliza episodios completos para estimar el retorno acumulado $G_t = \sum_{k=t}^{T} \gamma^{k-t} r_k$ desde un estado s_t y así calcular el gradiente de la política.
- **Actualización directa de la política**: Se actualiza la política mediante la regla $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi(a_t \mid s_t; \theta) G_t$, donde α es la **tasa de aprendizaje**.
- **Alta varianza**: Debido a que utiliza retornos completos de episodios, las estimaciones pueden ser ruidosas y poco eficientes, lo que dificulta la convergencia en entornos complejos.

Este algoritmo sienta las bases conceptuales para el aprendizaje basado en gradientes de política, pero su alta varianza llevó al desarrollo de métodos que integraran estimadores de valor para estabilizar el aprendizaje [21].

3.4.2. Métodos Actor-Crítico

Los métodos actor-crítico combinan dos componentes fundamentales:

- **Actor**: Responsable de parametrizar la **política** $\pi(a \mid s; \theta)$ y determinar qué acción tomar en cada estado.

- **Crítico**: Encargado de estimar la **función de valor** $V(s;\theta_v)$ (o, en algunas variantes, la **función acción-valor** $Q(s,a)$) que sirve como referencia para **evaluar** las decisiones del actor.

Principales ventajas:

- **Reducción de varianza**: Al utilizar un estimador de valor (el crítico) para aproximar el retorno esperado, se **reduce la varianza** en la actualización del gradiente en comparación con REINFORCE.
- **Aprendizaje basado en TD (Temporal Difference)**: El crítico se entrena mediante **métodos de diferencia temporal**, lo que permite realizar actualizaciones de forma **incremental** sin necesidad de esperar a la finalización completa del episodio.

La interacción entre actor y crítico se traduce en una actualización que pondera el gradiente de la política con el error de valoración, de forma similar a:

$$\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi(a_t \mid s_t;\theta)\delta_t, \tag{6}$$

donde $\delta_t = r_t + \gamma V(s_{t+1};\theta_v) - V(s_t;\theta_v)$ es el **error de TD**.

Este marco permite que el actor se oriente a tomar acciones que produzcan retornos mayores, mientras que el crítico mejora progresivamente su estimación del valor de los estados [22].

3.4.3. Advantage Actor-Critic (A2C)

El **Advantage Actor-Critic (A2C)** es una evolución natural de los métodos actor-crítico que introduce el concepto de **ventaja** para mejorar la **eficiencia del aprendizaje**:

La ventaja se calcula como:

$$A(s_t,a_t) = Q(s_t,a_t) - V(s_t), \tag{7}$$

o de forma práctica mediante una **aproximación n-pasos**:

$$A(s_t,a_t) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}) - V(s_t). \tag{8}$$

Este término mide lo "mejor" que es una acción en comparación con el promedio esperado en el estado.

La **política** se actualiza utilizando el **gradiente ponderado por la ventaja**, lo que ayuda a **centrar el aprendizaje** en aquellas acciones que superan la **expectativa**:

$$\nabla_\theta \log \pi(a_t \mid s_t; \theta) A(s_t, a_t). \tag{9}$$

En A2C la actualización se realiza de manera **sincrónica**, es decir, se recopilan las experiencias de múltiples muestras o mini-batches y se actualizan los parámetros de forma coordinada.

La introducción del estimador de ventaja ayuda a **reducir la varianza** sin incurrir en un **sesgo** excesivo, lo cual es esencial para entornos con alta complejidad o donde los retornos pueden variar considerablemente [23].

3.4.4. Asynchronous Advantage Actor Critic (A3C)

El algoritmo **Asynchronous Advantage Actor-Critic (A3C)** se puede ver como una extensión de A2C, donde se abordan algunas limitaciones prácticas:

- **Asincronía**: En lugar de sincronizar todas las actualizaciones (como en A2C), A3C utiliza múltiples workers que interactúan de forma independiente con sus propias copias del entorno. Cada worker actualiza de forma asíncrona una red global, lo que acelera la **convergencia** y reduce la **correlación** entre muestras.
- **Exploración diversificada**: Gracias a la ejecución paralela en diferentes entornos o instancias, el algoritmo explora de manera más amplia el **espacio de estados**, lo que mejora la **generalización** y la **robustez** del aprendizaje.
- **Eficiencia computacional**: La arquitectura asíncrona permite aprovechar de forma más eficiente la capacidad de múltiples núcleos de CPU o combinaciones CPU/GPU, lo que resulta en una aceleración sustancial del proceso de **entrenamiento**.

A3C es un algoritmo de aprendizaje por refuerzo profundo que representa un avance significativo en la combinación de **estrategias de política (actor)** y **evaluación de estados (crítico)** con la ventaja adicional de la paralelización asíncrona. Propuesto inicialmente por DeepMind en 2016, A3C surge para mitigar limitaciones inherentes a métodos anteriores, como la **correlación en las muestras** y la **dependencia de replay buffers** en métodos basados en Q-learning. La idea central es **explotar múltiples agentes** (o "workers") que exploran de forma independiente el entorno, acumulando gradientes que se sincronizan de manera asíncrona con una red global. Esto no solo **acelera el aprendizaje**, también **mejora la robustez** al diversificar las experiencias recogidas en entornos potencialmente heterogéneos.

A3C se enmarca dentro de los métodos actor-crítico, donde:

- **Actor**: Es responsable de tomar decisiones. Define una **política estocástica** $\pi(a \mid s; \theta)$ que mapea estados s a distribuciones sobre acciones a.

- **Crítico**: Estima la **función de valor** $V(s;\theta_v)$ para evaluar la calidad del estado actual y servir de base para calcular la ventaja.

La ventaja se define como:

$$A(s_t, a_t; \theta, \theta_v) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}; \theta_v) - V(s_t; \theta_v) \tag{10}$$

Donde $A(s_t, a_t; \theta, \theta_v)$ es la **función de ventaja**, que mide la diferencia entre el valor real de tomar una acción a_t en un estado s_t y el valor esperado de ese estado y θ y θ_v son los **parámetros de las redes neuronales actor y crítica**, respectivamente. γ es el **factor de descuento**, que controla el peso que tiene el futuro en la toma de decisiones, un valor cercano a **1** indica que se valoran mucho las recompensas futuras, mientras que un valor cercano a **0** indica que solo se valoran las recompensas inmediatas. r_{t+i} es la **recompensa recibida** en el paso de tiempo $t+i$, por lo que $\sum_{i=0}^{k-1} \gamma^i r_{t+i}$ calcula la **recompensa acumulada** durante los siguientes k pasos, descontada por el factor γ. $V(s_{t+k};\theta_v)$ es el **valor del estado** s_{t+k} predicho por la **red crítica**, es decir, una estimación de la recompensa futura que se puede esperar a partir de ese estado y $V(s_t;\theta_v$ es el **valor estimado del estado** s_t según la **red crítica**, representa lo que el agente espera recibir en términos de recompensa desde ese estado, sin importar la acción tomada.

El concepto de **ventaja** permite **reducir la varianza en la estimación de gradientes** al comparar el retorno observado con una estimación de valor de referencia, lo que guía de forma más precisa la actualización de la política. Este término ayuda a identificar qué tan buena fue una acción respecto a la expectativa general en ese estado [24].

3.4.5. Proceso de entrenamiento

Una de las innovaciones más relevantes es el uso de múltiples agentes que interactúan en paralelo con copias independientes del entorno. Cada agente (worker) ejecuta su propio proceso y mantiene una versión local de los parámetros θ' y θ'_v, sincronizándose periódicamente con los parámetros globales θ y θ_v. Este diseño asíncrono permite:

- **Descorrelación de muestras**: Al operar en diferentes entornos o diferentes partes del mismo entorno, se reduce la correlación entre las muestras de experiencia.
- **Mejor utilización de recursos computacionales**: Se aprovechan múltiples núcleos de CPU (e incluso combinaciones híbridas CPU/GPU) para acelerar el proceso de entrenamiento.

El proceso de cada worker se resume en los siguientes pasos:

1. Inicializar parámetros locales con los globales.
2. Recoger una secuencia de transiciones (hasta alcanzar $t_{\max}$ o un estado terminal).
3. Calcular el retorno acumulado R utilizando un esquema n-pasos.
4. Acumular gradientes para la política y para el crítico:
   - Para el actor: $\nabla_{\theta'} \log \pi(a_t \mid s_t; \theta') \left(R - V(s_t; \theta'_v)\right)$
   - Para el crítico: Gradiente del error cuadrático $\nabla_{\theta'_v} \left(R - V(s_t; \theta'_v)\right)^2$
5. Actualizar de forma asíncrona los parámetros globales con los gradientes acumulados.

Este enfoque se asemeja a una versión paralelizada del descenso de gradiente estocástico, donde cada worker contribuye de forma independiente a la optimización global.

3.4.6. Pseudocódigo

Algorithm 2 Asynchronous Advantage Actor-Critic (A3C)

Entrada: γ (descuento), α_{actor}, α_{critic} (tasas de aprendizaje), t_{max} (pasos por worker)

Inicializar red global Actor (θ) y Crítico (θ_v)
Crear N workers (hilos paralelos)
for cada worker en paralelo **do**
    while no converja **do**
        $estado \leftarrow entorno.reset()$
        buffer_estados, buffer_acciones, buffer_recompensas $\leftarrow$ [], [], []
        for $t = 1$ hasta t_{max} **do**
            $accion \sim \pi(accion|estado; \theta)$
            $nuevo\_estado, recompensa, done \leftarrow entorno.step(accion)$
            Almacenar $(estado, accion, recompensa)$
            $estado \leftarrow nuevo\_estado$
            if done **then**
                break
            end if
        end for
        ▷ Calcular ventajas (n-pasos) y valores objetivo
        $R \leftarrow 0$ si *done* sino $V(nuevo\_estado; \theta_v)$
        buffer_ventajas $\leftarrow$ []
        for $i = t - 1$ hasta 0 **do**
            $R \leftarrow$ buffer_recompensas[i] $+\gamma \cdot R$
            $ventaja \leftarrow R - V($buffer_estados[i]$; \theta_v)$
            Insertar $ventaja$ en buffer_ventajas al inicio
        end for
        ▷ Actualizar red global (Actor y Crítico)
        $grad_{actor} \leftarrow \nabla_\theta \log \pi(acciones|estados) \cdot ventajas$
        $grad_{critic} \leftarrow \nabla_{\theta_v}(V(estados) - R)^2$
        actualizar_red_global$(\theta, \theta_v, grad_{actor}, grad_{critic})$
    end while
end for

3.4.7. Ventajas y consideraciones

El algoritmo Asynchronous Advantage Actor-Critic (A3C) presenta diversas ventajas y consideraciones que impactan su desempeño y aplicabilidad en problemas de aprendizaje por refuerzo.

Ventajas de A3C:

- **Paralelismo asíncrono:** A3C emplea múltiples agentes que operan en paralelo y de manera asíncrona, lo que permite una exploración más amplia y diversa del espacio de

estados. Este enfoque reduce la correlación entre las muestras y mejora la eficiencia del aprendizaje.

- **Reducción de la varianza en las estimaciones:** Al integrar las funciones de actor y crítico, A3C utiliza la estimación de ventaja para guiar las actualizaciones de la política, lo que disminuye la varianza en las estimaciones y conduce a un aprendizaje más estable.
- **Eficiencia computacional:** La implementación asíncrona permite que A3C aproveche de manera más efectiva los recursos computacionales, especialmente en arquitecturas con múltiples núcleos de procesamiento, acelerando el proceso de aprendizaje.

Consideraciones y Desventajas:

- **Complejidad en la implementación:** La coordinación entre múltiples agentes y la gestión de sus actualizaciones asíncronas pueden complicar la implementación y el mantenimiento del sistema.
- **Consumo de recursos:** El funcionamiento simultáneo de múltiples agentes puede requerir una cantidad significativa de recursos computacionales, lo que podría ser un desafío en entornos con recursos limitados.
- **Estabilidad en rntornos no estacionarios:** En entornos donde las dinámicas cambian con el tiempo, la política aprendida puede volverse inestable debido a la naturaleza asíncrona de las actualizaciones.

En resumen, A3C ofrece mejoras significativas en términos de eficiencia y estabilidad en el aprendizaje por refuerzo al combinar técnicas de actor-crítico con paralelismo asíncrono. Sin embargo, es crucial evaluar las necesidades específicas del problema y los recursos disponibles antes de su implementación.

3.5. Proximal Policy Optimization (PPO)

El **Proximal Policy Optimization (PPO)** es un algoritmo de aprendizaje por refuerzo que surge como respuesta a las limitaciones de métodos anteriores basados en **gradiente de política**, como el **REINFORCE** y, en particular, el **Trust Region Policy Optimization (TRPO)**. La necesidad de equilibrar la **estabilidad** de las actualizaciones de política con la **simplicidad computacional** llevó a la formulación de PPO, propuesto por Schulman et al. en 2017. Mientras que TRPO introducía restricciones fuertes (mediante una **divergencia KL**) para garantizar mejoras monótonas, su implementación resultaba compleja y costosa en términos computacionales. PPO simplifica este enfoque mediante mecanismos de **clipping** que, de forma implícita, limitan la **magnitud** de las actualizaciones [25, 26].

3.5.1. Trust Region Policy Optimization (TRPO)

El algoritmo **Trust Region Policy Optimization (TRPO)** introdujo la idea de **limitar el cambio de la política** entre iteraciones, garantizando que la nueva política no se alejara demasiado de la anterior. Esto se formaliza al imponer una restricción basada en la **divergencia de Kullback-Leibler (KL)**:

$$\max_{\theta} \mathbb{E}\left[\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{\text{old}}}(a \mid s)} A(s,a)\right] \quad \text{sujeto a} \quad D_{\text{KL}}(\pi_{\theta_{\text{old}}} \parallel \pi_\theta) \leq \delta, \tag{11}$$

donde $A(s,a)$ es la **función de ventaja** y δ un parámetro que controla la "**proximidad**" entre políticas.

Aunque este enfoque ofrece garantías teóricas de **mejora monótona**, su resolución implica un problema de **optimización cuadrática** restringido, aumentando la **complejidad** de implementación y la **carga computacional** [25].

3.5.2. Proximal Policy Optimization (PPO)

El algoritmo **Proximal Policy Optimization (PPO)** se fundamenta en la utilización de un objetivo **surrogate** que compara la **nueva política** π_θ con la **política anterior** $\pi_{\theta_{\text{old}}}$ mediante el **ratio de importancia** [26]:

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}. \tag{12}$$

Este ratio mide **cuánto ha cambiado la probabilidad de seleccionar la acción** a_t en el estado s_t tras la actualización.

La innovación clave de PPO es la introducción de un mecanismo de *clipping* en la **función de pérdida**, definida como:

$$L^{CLIP}(\theta) = \mathbb{E}_t\left[\min\left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right] \tag{13}$$

Donde $\hat{A}_t$ es el **estimador de la ventaja** y ϵ es un **hiperparámetro** pequeño (típicamente entre 0.1 y 0.2).

Este objetivo actúa de la siguiente forma:

- Si $r_t(\theta)$ se mantiene dentro del intervalo $[1-\epsilon, 1+\epsilon]$, la actualización se realiza de manera **libre**, maximizando directamente $r_t(\theta)\hat{A}_t$.
- Si $r_t(\theta)$ excede este rango, el **clipping** evita que la contribución de esa muestra empuje la actualización de forma demasiado agresiva. Esto **limita el riesgo de grandes desviaciones en la política** y promueve actualizaciones más **conservadoras y estables**.

3.5.3. Proceso de entrenamiento

El proceso de entrenamiento de PPO se basa en dos fases principales que se repiten de forma iterativa:

1. **Recolección de Datos (Rollout)**:
    - El agente interactúa con el entorno usando su política actual para recolectar trayectorias (secuencias de estados, acciones, recompensas y, a veces, información adicional como valores estimados).
    - Estas trayectorias se pueden recopilar en **paralelo** mediante entornos **vectorizados**, lo que aumenta la **eficiencia** al obtener muestras de varios episodios simultáneamente.
2. **Actualización de la Política y del Valor**:
    - **Cálculo de ventajas y retornos**: Se utiliza una técnica como la **Estimación de Ventaja Generalizada (GAE)** para calcular, a partir de las recompensas y las **predicciones del valor**, tanto el **retorno** (suma descontada de recompensas) como la **ventaja** (la diferencia entre el retorno observado y el valor estimado).
    - **Optimización del objetivo PPO**: Se define una función objetivo **surrogate** que compara la probabilidad de haber tomado una acción bajo la nueva política frente a la antigua. Mediante un mecanismo de **clipping** se evita que la actualización se aleje demasiado de la política previa.

      La actualización se realiza con varios pasos de descenso de gradiente (usualmente mini-batches y múltiples épocas) sobre este objetivo, lo que permite aprovechar mejor los datos recolectados.
    - **Actualización del crítico**: Se entrena la red de valor **minimizando la diferencia** (por ejemplo, el error cuadrático medio) entre la estimación del valor y el retorno calculado, lo que ayuda a que la ventaja se calcule con **mayor precisión** en futuras iteraciones.

En resumen, PPO alterna entre recopilar experiencia en el entorno y actualizar tanto la política (usando la función de pérdida recortada) como el valor (mediante regresión sobre los retornos), lo que garantiza actualizaciones estables y eficientes en la búsqueda de la política óptima.

3.5.4. Pseudocódigo

Algorithm 3 Proximal Policy Optimization (PPO)

Entrada: γ (descuento), ϵ (clip_range), K (épocas de actualización)

Inicializar política $\pi(\theta)$ y Crítico $V(\theta_v)$
Inicializar buffer de trayectorias D
for cada iteración **do**
    $estado \leftarrow entorno.reset()$
    while no se llene D **do** ▷ Fase de recolección de datos
        $accion \sim \pi(accion|estado;\theta)$
        $nuevo\_estado, recompensa, done \leftarrow entorno.step(accion)$
        Almacenar $(estado, accion, recompensa, V(estado))$ en D
        $estado \leftarrow nuevo\_estado$
    end while
    ▷ Calcular ventajas y retornos (GAE)
    calcular_ventajas(D, γ, λ)
    for $epoca = 1$ hasta K **do** ▷ Fase de optimización
        for cada $batch$ en D **do**
            $ratio \leftarrow \frac{\pi(accion|estado;\theta)}{\pi(accion|estado;\theta_{old})}$
            $ventaja\_normalizada \leftarrow \frac{ventaja - \mu_{ventaja}}{\sigma_{ventaja}}$
            ▷ Pérdida del Actor (con clipping)
            $perdida_{actor} \leftarrow mean(-\min(ratio \cdot ventaja\_normalizada, \text{clip}(ratio, 1-\epsilon, 1+\epsilon) \cdot ventaja\_normalizada))$
            ▷ Pérdida del Crítico
            $perdida_{critic} \leftarrow MSE(V(estado), retorno)$
            ▷ Actualizar θ y θ_v
            descenso_gradiente$(\theta, \theta_v, perdida_{actor} + perdida_{critic})$
        end for
    end for
end for

3.5.5. Ventajas y consideraciones

La creación de PPO respondió a varias necesidades críticas en la investigación en aprendizaje por refuerzo:

- **Estabilidad en las actualizaciones**: Se buscaba evitar cambios bruscos en la política que pudieran llevar a comportamientos inestables o a la pérdida de la convergencia, sin recurrir a costosas optimizaciones.
- **Simplicidad y facilidad de implementación**: Al reemplazar la restricción explícita (como la divergencia KL en TRPO) por un método de clipping, PPO permite una

implementación más directa y menos dependiente de soluciones numéricas complejas.

- **Eficiencia en el uso de datos**: Al garantizar que las actualizaciones sean moderadas, se logra una mayor eficiencia en el aprovechamiento de las muestras recolectadas, mejorando la tasa de convergencia en entornos complejos.
- **Robustez en aplicaciones reales**: La estabilidad y simplicidad de PPO lo han convertido en uno de los algoritmos preferidos para tareas de control, robótica y juegos, donde la eficiencia y la robustez son cruciales.

Además del enfoque basado en clipping, existen variantes de PPO que incorporan directamente una penalización por divergencia KL en la función objetivo. Aunque ambas variantes persiguen la meta de mantener actualizaciones "proximales", la versión con clipping es la más utilizada por su simplicidad y eficacia empírica.

3.6. Soft Actor-Critic (SAC)

El algoritmo **Soft Actor-Critic (SAC)** es un método de aprendizaje por refuerzo que se inscribe dentro del paradigma de la **maximización de entropía**, combinando ideas de los enfoques **actor-crítico** y métodos **off-policy**.

A diferencia de los métodos clásicos que únicamente maximizan la recompensa acumulada, SAC incorpora un término de **entropía** en el objetivo. La **función de recompensa** modificada es:

$$J(\pi) = \mathbb{E}\left[\sum_t \left(r(s_t, a_t) + \alpha \cdot H(\pi(\cdot|s_t))\right)\right] \tag{14}$$

Donde $H(\pi(\cdot|s))$ representa la **entropía de la política** en el estado s y α es un parámetro que regula la importancia de la entropía.

Este enfoque favorece **políticas estocásticas**, promoviendo una **exploración más robusta** y evitando la **convergencia prematura** a soluciones subóptimas.

SAC surge como evolución de métodos basados en actor-crítico, particularmente aquellos diseñados para entornos continuos. Se puede trazar su línea evolutiva a partir de:

- **Deep Deterministic Policy Gradient (DDPG)**: Un método **off-policy** para control continuo, que sin embargo es propenso a **problemas de estabilidad** y **exploración limitada** debido a su naturaleza **determinista** [27].
- **Soft Q-Learning**: Introdujo el concepto de **maximización de entropía** en el aprendizaje por refuerzo, inspirando la incorporación de la entropía en la función de valor [28].

- **Actor-Crítico Estocástico**: Donde la **política** se parametriza de manera **probabilística**, permitiendo la incorporación natural del término entropía.

El enfoque SAC combina estas ideas: utiliza la estructura **actor-crítico**, pero con **políticas estocásticas** y una formulación que incorpora la **maximización de la entropía**, logrando mejorar tanto la **estabilidad** del entrenamiento como la **calidad** de la exploración [29, 30].

3.6.1. Proceso de entrenamiento

El entrenamiento se basa en dos componentes principales

1. **Crítico (Q-function)**: SAC aprende una aproximación de la función de valor acción-estado, **minimizando el residuo de Bellman** modificado:

$$L(\theta) = \mathbb{E}(s, a, s') \sim D \left[\left(Q\theta(s, a) - \left(r(s, a) + \gamma \cdot \left(\min_i \hat{Q}\theta_i(s', a') - \alpha \cdot \log \pi\phi(a'|s') \right) \right) \right)^2 \right] \tag{15}$$

   Donde $r(s, a)$ es la **recompensa inmediata**, γ el **factor de descuento**, α el **coeficiente de entropía**, y π_ϕ la **política parametrizada** por ϕ.

   Se utiliza la técnica de **Double Q-Learning** para **reducir el sesgo positivo** y se emplean **redes objetivo** para **estabilizar** las actualizaciones.

2. **Actor (Política)**: La política se actualiza **minimizando la divergencia Kullback-Leibler (KL)** entre la **política actual** y una **distribución suavizada derivada de la función** Q, lo que ajusta la política hacia acciones que **maximizan** tanto la **recompensa inmediata** como la **entropía**. La reparametrización, mediante una transformación de una **variable latente** (por ejemplo, una gaussiana), facilita la estimación eficiente de gradientes.

Una innovación clave en SAC es la capacidad de **ajustar automáticamente el coeficiente de entropía** α, equilibrando de forma dinámica la **exploración** y la **explotación** según la **señal de recompensa**, lo cual mejora la **robustez** del algoritmo frente a la selección de hiperparámetros.

Al ser un método **off-policy**, SAC reutiliza muestras almacenadas en un **buffer de reproducción**, lo que incrementa la **eficiencia** de la muestra y permite aplicar técnicas de aprendizaje profundo con redes neuronales para **aproximar tanto la política como la función** Q.

3.6.2. Pseudocódigo

Algorithm 4 Soft Q-Learning with Entropy Regularization

Entrada: γ (descuento), α (coef. entropía), τ (suavizado de redes objetivo)

Inicializar redes $Q_1(\theta_1)$, $Q_2(\theta_2)$, política $\pi(\phi)$
Inicializar redes objetivo $Q'_1(\theta'_1)$, $Q'_2(\theta'_2)$
Inicializar buffer de experiencias D
for cada episodio **do**
    $estado \leftarrow entorno.reset()$
    for cada paso t **do** ▷ Exploración con política estocástica
        $accion \sim \pi(\phi) + ruido$
        $nuevo\_estado, recompensa, done \leftarrow entorno.step(accion)$
        Almacenar $(estado, accion, recompensa, nuevo\_estado, done)$ en D
        $batch \leftarrow sample(D, batch\_size)$
        ▷ Actualizar redes Q
        $Q\_target \leftarrow recompensa + \gamma \cdot (min(Q'_1, Q'_2) - \alpha \cdot \log \pi(nuevo\_estado))$
        $perdida_{Q1} \leftarrow MSE(Q_1(estado, accion), Q_{target})$
        $perdida_{Q2} \leftarrow MSE(Q_2(estado, accion), Q_{target})$
        descenso_gradiente$(\theta_1, \theta_2, perdida_{Q1} + perdida_{Q2})$
        ▷ Actualizar política (maximizar entropía)
        $acciones\_nuevas \sim \pi(\phi)$
        $perdida_\pi \leftarrow (\alpha \cdot \log \pi(acciones\_nuevas) - min(Q_1, Q_2)).mean()$
        descenso_gradiente$(\phi, perdida_\pi)$
        ▷ Actualizar redes objetivo y α (opcional)
        actualizar_redes_objetivo$(\theta'_1, \theta_1, \tau)$
        actualizar_redes_objetivo$(\theta'_2, \theta_2, \tau)$
        ajustar α
    end for
end for

3.6.3. Ventajas y consideraciones

Propuesto por Haarnoja et al. en 2018, SAC fue diseñado para abordar las limitaciones de algoritmos anteriores en entornos de control continuo. Los principales motivos fueron:

- **Estabilidad en el entrenamiento**: La incorporación del término de **entropía reduce la varianza** en las actualizaciones y mejora la **estabilidad** del entrenamiento en comparación con métodos deterministas como DDPG.
- **Exploración eficiente**: La maximización de la entropía incentiva la **exploración sistemática del espacio de acciones**, fundamental en entornos con altos grados de incertidumbre o con múltiples óptimos locales.

- **Robustez a la elección de hiperparámetros**: El **ajuste automático** del parámetro α permite que el algoritmo se adapte a diferentes escenarios sin requerir un costoso proceso de afinación manual, facilitando su aplicación en una amplia variedad de tareas.

SAC ha tenido un impacto significativo en el campo del aprendizaje por refuerzo, tanto en aplicaciones de robótica como en simulaciones complejas, al demostrar que un marco basado en la máxima entropía puede superar los desafíos tradicionales de la exploración y la estabilidad [29, 30].

4. Experimentación y análisis de resultados

En este capítulo se detalla el proceso experimental llevado a cabo para evaluar el rendimiento de los diferentes algoritmos de Aprendizaje por Refuerzo Profundo (DRL) aplicados al problema de conducción autónoma en el entorno simulado desarrollado. Se describe la configuración del entorno, los algoritmos implementados, los hiperparámetros utilizados, las métricas registradas y la plataforma sobre la que se realizaron los experimentos.

4.1. Entorno de simulación

Para asegurar la reproducibilidad y permitir una comparación justa entre los algoritmos, se estableció una configuración experimental común en la medida de lo posible. Además, se ha subido el código del proyecto a un repositorio de GitHub [1].

- **Simulador**: Se ha utilizado SUMO (Simulation of Urban MObility) versión *1.20.0*. SUMO es un simulador de tráfico microscópico, multimodal y de código abierto, ampliamente utilizado en investigación.
- **Escenario**: Los experimentos se realizaron sobre un escenario urbano denominado "*scene*", cuya red (osm.net.xml.gz) y elementos visuales (osm.poly.xml.gz) fueron generados o adaptados (posiblemente a partir de datos de OpenStreetMap, como sugiere el prefijo .ºsm"). Este escenario incluye múltiples intersecciones, algunas de ellas reguladas por semáforos, y permite la simulación de tráfico en un entorno urbano representativo aunque de complejidad controlada.
- **Vehículo controlado**: El agente de RL controla un único vehículo con ID *"manualVehicle"* y tipo *"veh_passenger"*. Se asumió que este tipo pertenece a la clase vehicular *"passenger"* para las comprobaciones de permisos en la red.
- **Dinámica de rutas**: Se ha implementado un sistema de rutas dinámicas cortas. Al inicio de cada episodio, se asigna al vehículo un edge de partida aleatorio (válido para su tipo) y un edge de destino aleatorio conectado al de partida. Una vez que el vehículo alcanza el final del edge de destino actual, se le asigna automáticamente un nuevo edge de destino aleatorio partiendo del edge actual, permitiendo episodios potencialmente largos y continuos dentro de la red.
- **Condiciones de episodio**: Cada episodio finaliza si ocurre alguna de las siguientes condiciones:
  - Se alcanza el número máximo de pasos ($max\_steps = 2000$).

- El vehículo sufre una colisión (detectada por SUMO o forzada por una acción inválida).
- El vehículo es teleportado por SUMO (generalmente indica un bloqueo irrecuperable).
- El vehículo alcanza el final de su segmento de ruta actual, pero no se puede asignar un nuevo edge de destino válido (ej., callejón sin salida o error).

- **Espacio de estados**: Se ha definido un vector de características de **56 dimensiones** como entrada para las redes neuronales. Este estado incluye (consultar SUMOEnvironment._get_state() para detalles exactos y normalización):
  - **Información del vehículo**: Posición (x, y normalizada), velocidad (escalar normalizada), ángulo (componentes sin/cos).
  - **Información de carril/edge**: Índice del carril actual (normalizado), número total de carriles en el edge (normalizado), límite de velocidad del carril (normalizado).
  - **Información de intersección**: Indicador booleano de si está en una intersección, indicadores booleanos de posibles maniobras (recto, izquierda, derecha) desde el carril actual.
  - **Vehículos cercanos**: Información de los 8 vehículos más cercanos dentro de un radio de 50 metros. Para cada vehículo cercano: posición relativa (x', y' normalizadas respecto al ego), velocidad relativa (vx', vy' normalizadas), distancia (normalizada). Se utiliza padding con valores específicos ([0, 0, 0, 0, 2.0]) si se detectan menos de 8 vehículos.
  - **Semáforos**: Estado del próximo semáforo (codificación *one-hot* [1] [Verde, Ámbar, Rojo]) y distancia normalizada a él.
- **Espacio de acciones**: Se ha definido un espacio de acciones discreto con **8 acciones** posibles:

  0: Mantener estado.

  1: Acelerar.

  2: Frenar/reversa.

[1] La codificación *one-hot* es una técnica utilizada para representar variables categóricas (como los colores de un semáforo) como vectores binarios. Se crea un vector con tantos elementos como categorías existan. Solo el elemento correspondiente a la categoría activa toma el valor 1 ('hot'), mientras que todos los demás son 0. Para las categorías [Verde, Ámbar, Rojo], Verde sería [1, 0, 0], Ámbar [0, 1, 0] y Rojo [0, 0, 1]. Esto permite a los algoritmos procesar información categórica numéricamente sin asumir un orden inherente entre las categorías.

3: Cambiar a carril izquierdo.

4: Cambiar a carril derecho.

5: Girar a la izquierda en la próxima intersección.

6: Girar a la derecha en la próxima intersección.

7: Seguir recto en la próxima intersección.

- **Función de recompensa**: Se ha diseñado una función de recompensa compuesta para guiar el aprendizaje, cuyos componentes y pesos se definen en *DEFAULT_REWARD_WEIGHTS* dentro de *sumo_environment.py*. Los componentes clave incluyen:
  - **Penalizaciones**: Colisión, teleportación, parada de emergencia (fuera de intersección y no por semáforo), parada larga en intersección, fallo de cambio de carril, error de ruta, ir en dirección contraria, pasar semáforo en rojo.
  - **Recompensas**: Velocidad, progreso hacia el final del edge actual (*approaching_goal*), alcanzar la meta del segmento (*goal_reached*), detenerse correctamente en semáforo rojo, mantener carril. Se han realizado ajustes iterativos a estos pesos durante la fase de experimentación (ver Subsección 6.1).

4.2. Algoritmos Evaluados

Se han implementado y evaluado los siguientes algoritmos DRL, buscando comparar enfoques basados en valor y basados en política/actor-crítico:

- **Dueling Double Deep Q-Network (DDQN)**: Una mejora sobre DQN que utiliza redes separadas para estimar el valor del estado (V) y la ventaja de cada acción (A), y desacopla la selección de la acción de la evaluación de su valor para reducir la sobreestimación (implementado en ddqn_agent.py).
- **Asynchronous Advantage Actor-Critic (A3C)**: Un algoritmo actor-crítico basado en política que utiliza múltiples workers (hilos) para explorar el entorno en paralelo y actualizar una red global de forma asíncrona (implementado en a3c_agent.py).
- **Proximal Policy Optimization (PPO)**: Un algoritmo actor-crítico basado en política on-policy que optimiza un objetivo sustituto (surrogate objective) utilizando un recorte (clipping) para limitar los cambios en la política entre actualizaciones, mejorando la estabilidad (implementado en ppo_agent.py).

- **Soft Actor-Critic (SAC)**: Un algoritmo actor-crítico basado en política off-policy que busca maximizar tanto la recompensa acumulada como la entropía de la política, fomentando una mejor exploración y robustez. Se implementó una versión adaptada para el espacio de acciones discreto, con ajuste automático opcional del coeficiente de entropía alfa (implementado en sac_agent.py).

4.3. Hiperparámetros principales

[**TODO: actualizar tabla si se ajustan parametros al seguir iterando**]

Cuadro 1: Hiperparámetros principales utilizados para cada algoritmo.

Hiperparámetro	DDQN	A3C	PPO	SAC
gamma (γ)	0.99	0.99	0.99	0.99
lr / actor_lr	0.0005	0.0001	0.0001	0.0003
critic_lr	N/A	N/A	N/A	0.0003
epsilon_decay	0.999	N/A	N/A	N/A
epsilon_min	0.05	N/A	N/A	N/A
target_update (pasos)	1000	N/A	N/A	N/A
tau (actualización suave)	N/A	N/A	N/A	0.005
batch_size	64	N/A (rollout)	32	256
memory_size / buffer_size	50000	N/A	N/A (on-policy)	1000000
gae_lambda (λ)	N/A	0.95	0.95	N/A
clip_epsilon (ϵ)	N/A	N/A	0.2	N/A
vf_coef (pérdida valor)	N/A	0.5	0.5	N/A
entropy_coef (pérdida entropía)	N/A	0.01	0.05	N/A
alpha (entropía SAC)	N/A	N/A	N/A	0.2 (inicial, auto)
learn_alpha (SAC)	N/A	N/A	N/A	True
alpha_lr (SAC)	N/A	N/A	N/A	0.0003
n_steps (rollout)	N/A	20	64	N/A
n_epochs (opt PPO)	N/A	N/A	3	N/A
n_workers (A3C)	N/A	4	N/A	N/A
max_grad_norm (clipping)	1.0	0.5	0.5	N/A

Los hiperparámetros clave para cada algoritmo se definieron centralizadamente en el archivo config.py. La tabla resume los valores utilizados en la mayoría de los experimentos finales, aunque algunos (notablemente para PPO) fueron ajustados tras observar resultados iniciales (ver Sección 6). Se habilitó el recorte de gradientes ($clipping = True$) para los algoritmos que lo soportaban en la implementación (DDQN, PPO, A3C).

4.4. Métricas Registradas

Para evaluar y comparar el rendimiento y el proceso de aprendizaje, se han registrado las siguientes métricas:

- **Durante el entrenamiento**:
  - Número de episodio.
  - Recompensa total del episodio.
  - Número de pasos del episodio.
  - Número total de pasos acumulados.
  - Recompensa promedio móvil.
  - Número de metas alcanzadas en el episodio.
  - Metas alcanzadas promedio móvil.
  - Pérdidas específicas del algoritmo (promediadas por episodio): loss (DDQN), policy_loss, value_loss, entropy_loss (PPO), actor_loss, critic_loss, alpha_loss (SAC), valor de epsilon (DDQN), valor de alpha (SAC).
- **Durante la evaluación**:
  - Número de episodio de evaluación.
  - Recompensa total del episodio.
  - Número de pasos del episodio.
  - Número de metas alcanzadas en el episodio.
  - Número de colisiones (forzadas o detectadas).
  - Número de teleportaciones.
  - Número de paradas de emergencia (o paradas largas en intersección).

Todos los datos de métricas se han guardaro en archivos *.csv* y se han generado gráficos *.png* para visualización.

4.5. Plataforma Experimental

Los experimentos se han llevado a cabo principalmente en un ordenador portátil personal con las siguientes especificaciones aproximadas:

- **CPU**: Apple M2 (Arquitectura ARM)
- **RAM**: 16 GB
- **GPU**: GPU integrada del Apple M2 (Aunque PyTorch puede usarla vía MPS, el cuello de botella principal suele ser SUMO en CPU).
- **Sistema operativo**: macOS
- **Software clave**: Python 3.12, PyTorch, SUMO , TraCi, NumPy, Pandas, Matplotlib, Seaborn.

5. Resultados del entrenamiento

En este apartado se presentan y analizan los resultados obtenidos durante la fase de entrenamiento para cada uno de los algoritmos implementados: DDQN, A3C, PPO y SAC. El análisis se centra en las métricas clave registradas (recompensa, pasos, metas alcanzadas, pérdidas) para evaluar la convergencia, estabilidad y eficacia del aprendizaje de cada método en el entorno SUMO configurado.

5.1. Dueling Deep Q-Network (DDQN)

Se ha entrenado el agente DDQN durante 5000 episodios. Los hiperparámetros utilizados se detallan en la Subsección 4.3 y *config.py*.

5.1.1. Curvas de aprendizaje

Figura 3: Recompensa promedio móvil - entrenamiento (DDQN).

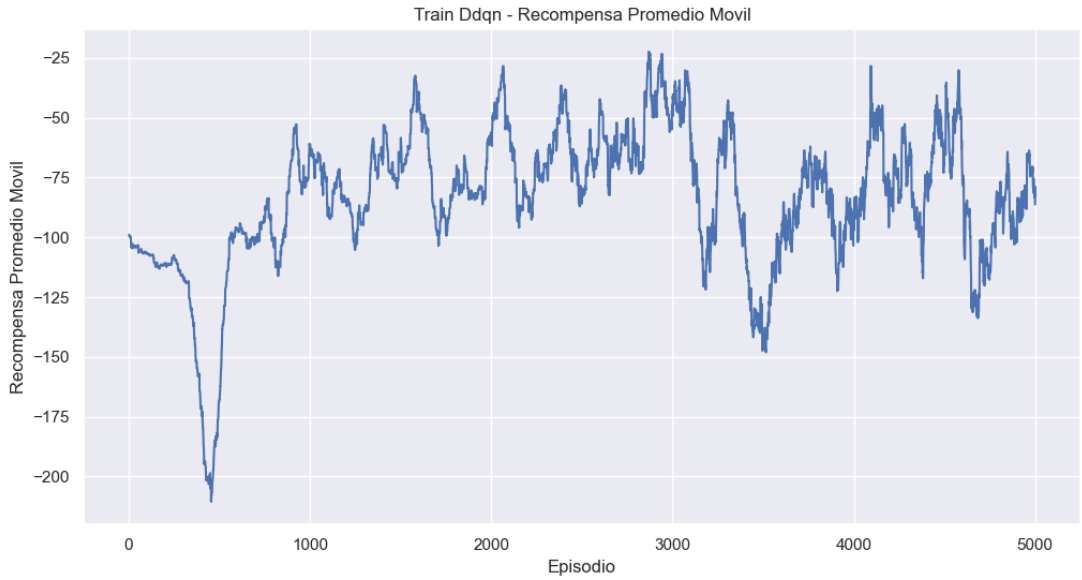


Figura 4: Pasos por episodio - entrenamiento (DDQN).

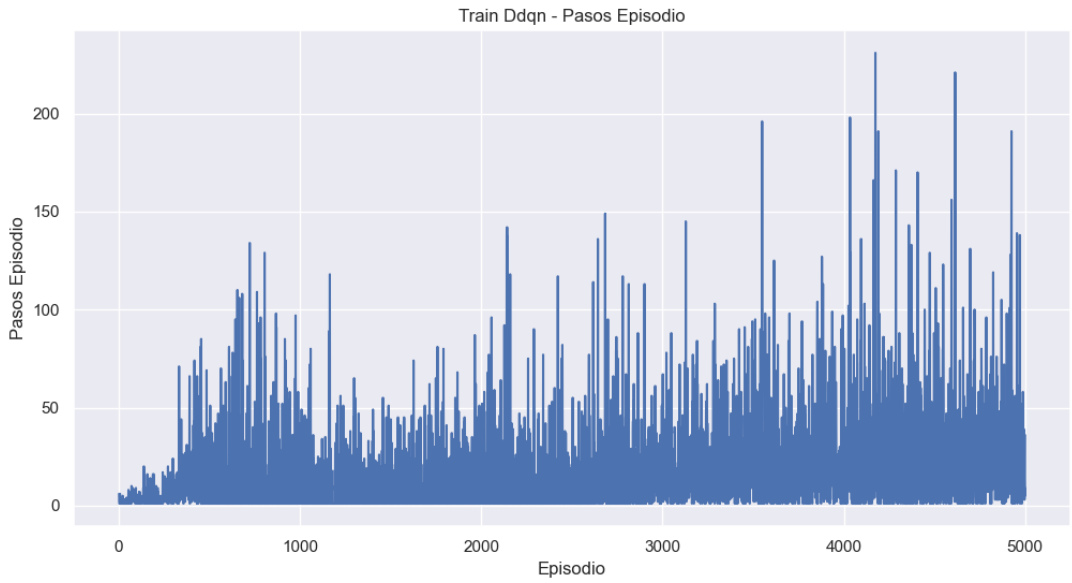


Figura 5: Metas slcanzadas promedio móvil - entrenamiento (DDQN).

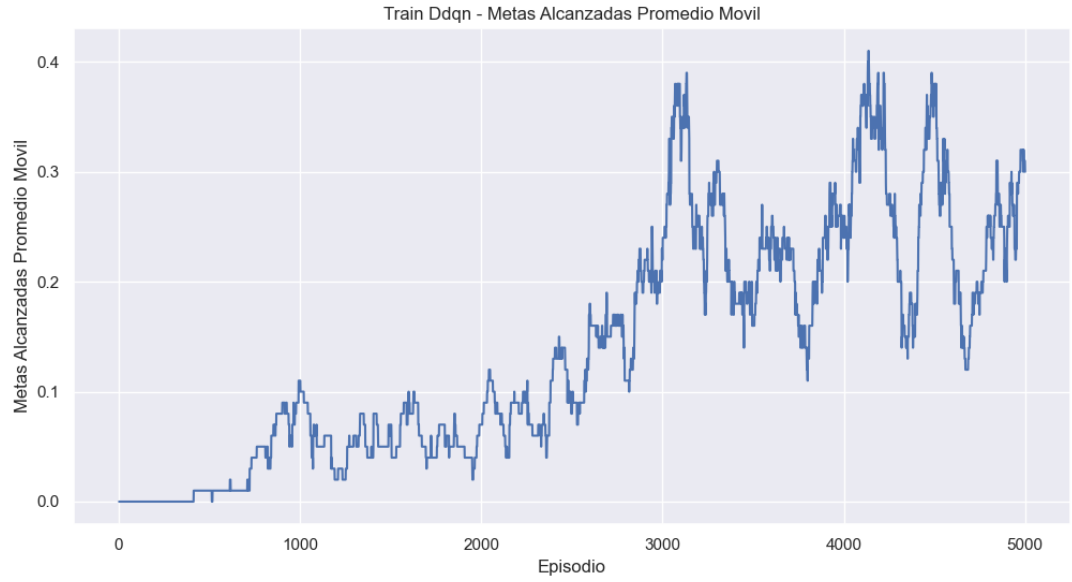
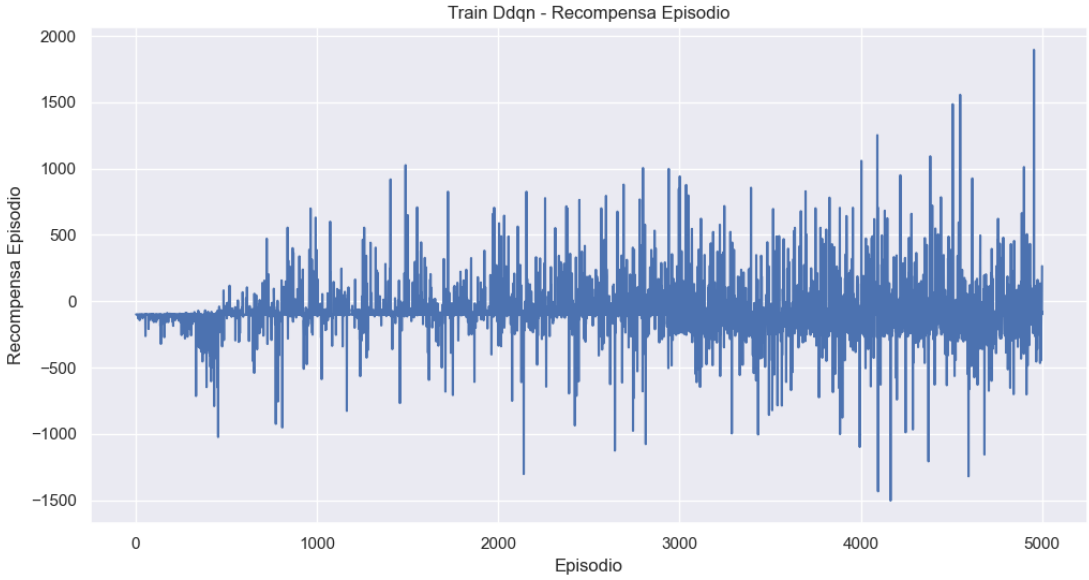


Figura 6: Recompensa por episodio - entrenamiento (DDQN).



5.1.2. Análisis de convergencia y rendimiento

- **Recompensa**: La Figura 3 muestra una clara mejora inicial. La recompensa promedio móvil comienza en valores muy negativos (por debajo de -150) y aumenta significativamente durante los primeros 1000-1500 episodios, alcanzando un máximo local cercano a -25. Sin embargo, después de este punto, el rendimiento no mejora de forma sostenida, sino que entra en una fase de alta oscilación, variando entre aproximadamente -25 y -125. Esto indica que el agente aprende a evitar los peores resultados iniciales, pero lucha por encontrar una política consistentemente buena y estable. La recompensa por episodio (Figura 6) es muy ruidosa, pero confirma la aparición de episodios con recompensas positivas altas (> 500), aunque mezclados con muchos episodios de recompensa negativa.

- **Pasos por episodio**: La Figura 4 corrobora el aprendizaje inicial. La duración de los episodios aumenta drásticamente desde casi cero hasta un promedio de 50-100 pasos, con picos frecuentes por encima de 150 pasos. Esto demuestra que el agente aprende a sobrevivir más tiempo, lo cual es un requisito previo para alcanzar metas. No obstante, la persistencia de episodios muy cortos (< 25 pasos) indica que los fallos prematuros (colisiones, errores) siguen siendo comunes.

- **Metas alcanzadas**: La Figura 5 es la evidencia más fuerte del aprendizaje. La media móvil de metas alcanzadas por episodio pasa de cero a un valor promedio que oscila entre

0.1 y 0.4. El gráfico de metas por episodio individual (no mostrado pero inferido del CSV) indica picos donde se alcanzan múltiples metas (hasta 5-6). Esto demuestra que el agente aprende el valor de la recompensa goal_reached y desarrolla estrategias para completar segmentos de ruta. Sin embargo, la baja media móvil ($< 0{,}5$) sugiere que no logra hacerlo en la mayoría de los episodios.

5.1.3. Conclusión

El agente DDQN muestra una capacidad de aprendizaje significativa en este entorno, superando el comportamiento inicial aleatorio y aprendiendo a completar segmentos de ruta ocasionalmente. Sin embargo, sufre de inestabilidad en el rendimiento y no converge a una política robusta y consistentemente buena dentro de los 5000 episodios. La alta variabilidad en la recompensa y la persistencia de episodios cortos sugieren problemas con la exploración o la estabilidad de la estimación de valor en ciertas situaciones del entorno.

5.2. Asynchronous Advantage Actor Critic (A3C)

La implementación del algoritmo A3C, si bien teóricamente atractiva por su potencial de paralelismo y eficiencia de datos, ha presenado una serie de **obstáculos técnicos significativos e irresolubles** dentro de la configuración experimental de este proyecto, impidiendo la obtención de resultados de entrenamiento concluyentes. Estos desafíos están centrados principalmente en la **gestión concurrente de múltiples simulaciones SUMO** y sus correspondientes **conexiones TraCI desde hilos (threading) de Python**.

5.2.1. Descripción de los desafíos técnicos

Problema principal de Thread-Safety en traci.start()

La raíz del problema reside en que la función estándar *traci.start(sumo_cmd, port=...)*, utilizada para lanzar un proceso SUMO externo y establecer una conexión TraCI con él, no es intrínsecamente segura para hilos (thread-safe) cuando se invoca concurrentemente desde múltiples hilos para iniciar instancias separadas de SUMO. La librería TraCI en Python parece mantener internamente un estado global (posiblemente relacionado con un pool de conexiones o la referencia a la conexión activa). Cuando múltiples hilos llaman a *traci.start()* casi simultáneamente, incluso especificando puertos únicos, se producen condiciones de carrera al intentar modificar este estado global compartido. Un hilo puede sobrescribir el estado interno establecido por otro hilo antes de que este último complete su secuencia de inicialización o envíe su primer comando.

Manifestación del error

Este conflicto interno se manifestó consistentemente como un *'traci.exceptions.FatalTraCIError: Not connected.'* que ocurría después de que la función *traci.connect* (llamada internamente por *traci.start* o explícitamente en el enfoque con subprocess) reportara una conexión aparentemente exitosa en un puerto específico. El error ocurría típicamente en la primera llamada a *traci.simulationStep()* dentro del método *reset()* del entorno del worker. Esto sugiere que, aunque la conexión TCP a nivel de socket se establecía brevemente, el estado interno de la librería TraCI se corrompía por las llamadas concurrentes de otros hilos, haciendo que la librería perdiera la referencia a la conexión correcta o la considerara inválida al intentar usarla inmediatamente después.

Errores de inicio del proceso SUMO

En las fases iniciales de depuración, también se observaron errores *'RuntimeError: No se pudo iniciar la simulación...'* acompañados de *'Connection refused'*. Aunque se solucionaron problemas como puertos duplicados y rutas relativas, la persistencia de estos errores en los workers A3C sugiere que el lanzamiento concurrente de procesos sumo desde hilos también podría estar causando conflictos a nivel del sistema operativo (agotamiento temporal de recursos, bloqueos) o errores internos silenciosos en SUMO que provocan su cierre prematuro antes de que TraCI pueda conectar de forma estable.

Intentos de mitigación y sus limitaciones

- **Puertos únicos**: Asignar un puerto TCP distinto a cada instancia de SUMO/TraCI (ej., 8813, 8815, 8817...) fue un paso necesario para evitar conflictos directos de *binding* [2], pero no resolvió el problema del estado global compartido dentro de la librería TraCI.

- **Rutas absolutas**: Utilizar rutas absolutas para el archivo .sumocfg descartó problemas de directorio de trabajo relativo en los hilos, pero no afectó la condición de carrera interna de TraCI.

- **Retardos entre workers**: Introducir pausas entre el inicio de cada worker redujo la simultaneidad del lanzamiento de procesos sumo, pero no eliminó la posibilidad de que las llamadas a *traci.connect* (o las operaciones internas de *traci.start*) aún ocurrieran de forma concurrente y conflictiva una vez que los hilos estaban activos.

[2]En programación, "**binding**"(vinculación o enlace, en español) se refiere al proceso de **asociar un identificador** (como el nombre de una variable o función) **con un valor, objeto, dirección de memoria o implementación específica**. Esta asociación permite que, cuando usas un identificador en código, el sistema sepa a qué çosareal te estás refiriendo. Por ejemplo, en 'x = 5', se vincula el nombre 'x' al valor entero '5'; al llamar a una función, se vincula el nombre de la función con el bloque de código que la ejecuta; y en el contexto de redes, "binding.ª un puerto significa asociar un socket específico de tu programa a una dirección IP y número de puerto concretos para poder escuchar o enviar datos a través de esa combinación [31].

- **subprocess.Popen + traci.connect**: Se ha intentado desacoplar el inicio del proceso SUMO (usando *subprocess*) de la conexión TraCI (usando *traci.connect*). Aunque esto evitaba usar la parte problemática de *traci.start*, el error *'FatalTraCIError: Not connected.'* persistió, indicando que *traci.connect* en sí misma (o las operaciones subsiguientes inmediatas como *simulationStep*) también sufren de problemas de thread-safety o estado global cuando múltiples hilos la usan para gestionar conexiones distintas. La serialización de *traci.connect* mediante un *threading.Lock* tampoco resolvió completamente el problema, sugiriendo que la interferencia podría ocurrir en otras funciones de TraCI que acceden al estado global.

5.2.2. Resultados

Debido a estos problemas técnicos irresolubles dentro del marco del proyecto, no ha sido posible completar una ejecución de entrenamiento estable y significativa para el agente A3C. Por lo tanto, no se dispone de curvas de aprendizaje ni métricas de rendimiento comparables a las de otros algoritmos.

5.2.3. Conclusión

La dificultad encontrada subraya los retos de implementar algoritmos DRL paralelos que interactúan con simuladores externos complejos como SUMO usando únicamente el módulo threading de Python. La gestión del estado global de TraCI y el lanzamiento de procesos externos parecen ser los principales cuellos de botella. Alternativas como el uso del módulo multiprocessing (que proporciona aislamiento de memoria) o librerías especializadas en RL paralelo (como Ray RLlib, que abstraen gran parte de la gestión de procesos y comunicación) serían líneas de investigación futuras más prometedoras para implementar A3C o algoritmos similares en este contexto [32, 33].

5.3. Proximal Policy Optimization (PPO)

Se ha entrenado el agente PPO durante 5000 episodios. Se realizaron ajustes significativos en los hiperparámetros después de observar una divergencia inicial, reduciendo n_steps a 64, n_epochs a 3, lr a 0.0001 y aumentando entropy_coef a 0.05 (ver config.py).

Figura 7: Recompensa Promedio Móvil - entrenamiento (PPO).

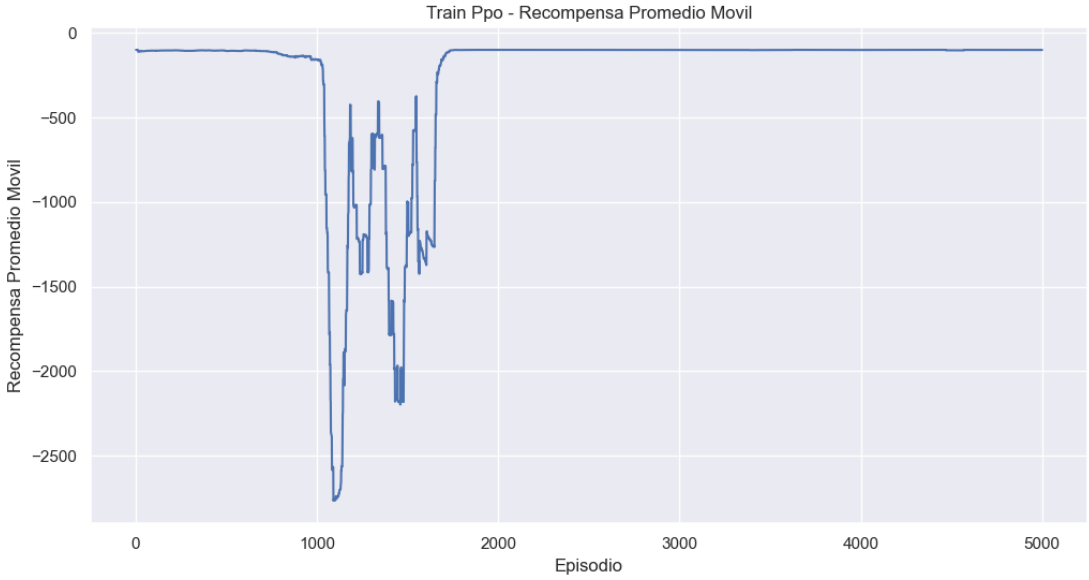


Figura 8: Metas Alcanzadas Promedio Móvil - entrenamiento (PPO).

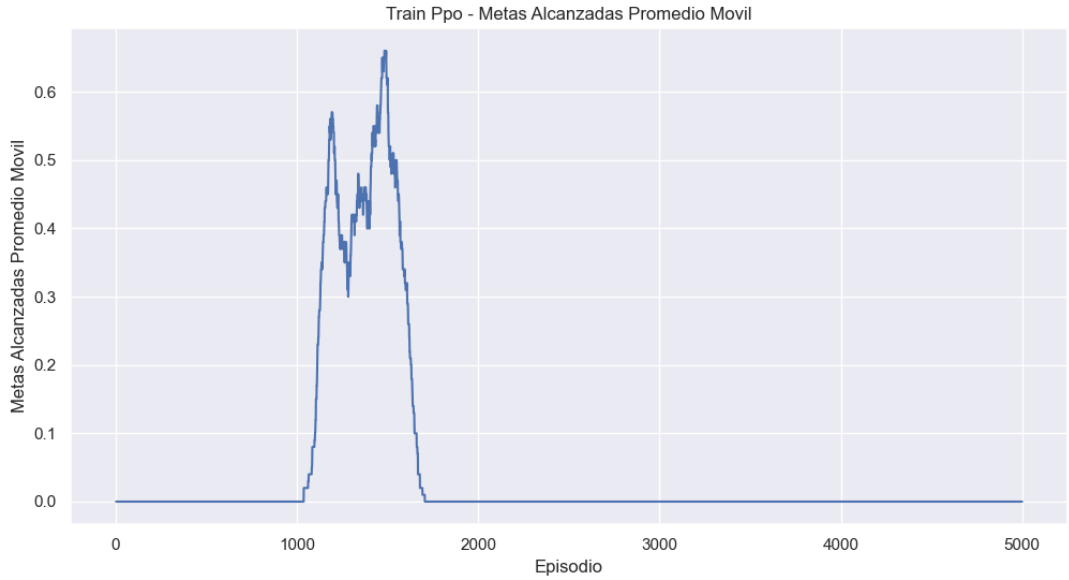


Figura 9: Pasos por Episodio - entrenamiento (PPO).

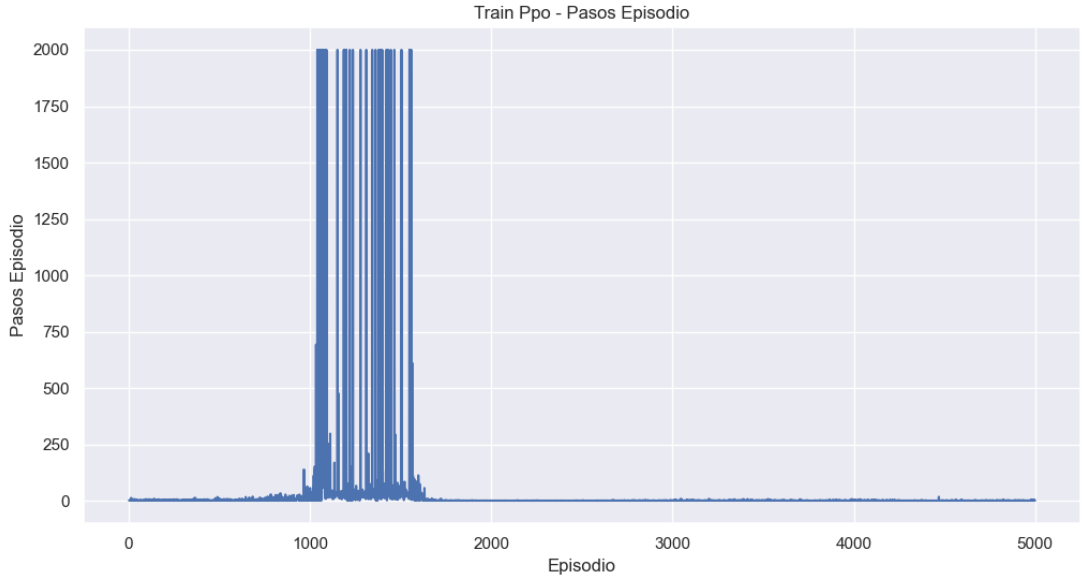
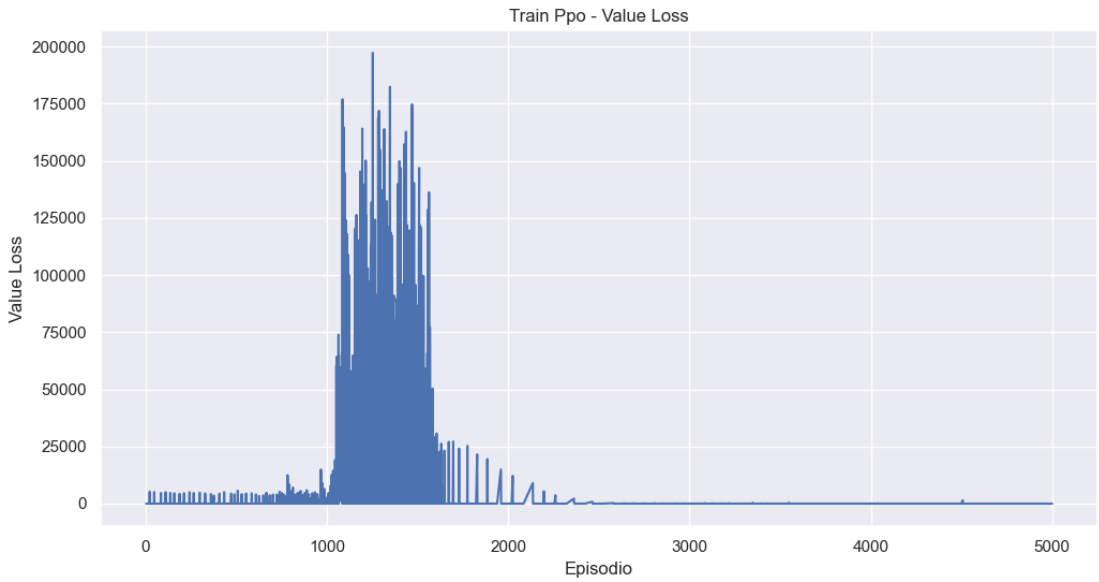


Figura 10: Pérdida de Valor (Value Loss) - entrenamiento (PPO).



5.3.1. Curvas de aprendizaje

5.3.2. Análisis de convergencia y rendimiento

- **Colapso inicial**: La Figura 7 muestra un comportamiento dramático. Tras un inicio cercano a -100, la recompensa promedio se desploma a valores extremadamente bajos

Figura 11: Pérdida de Valor (Value Loss) - entrenamiento (PPO).

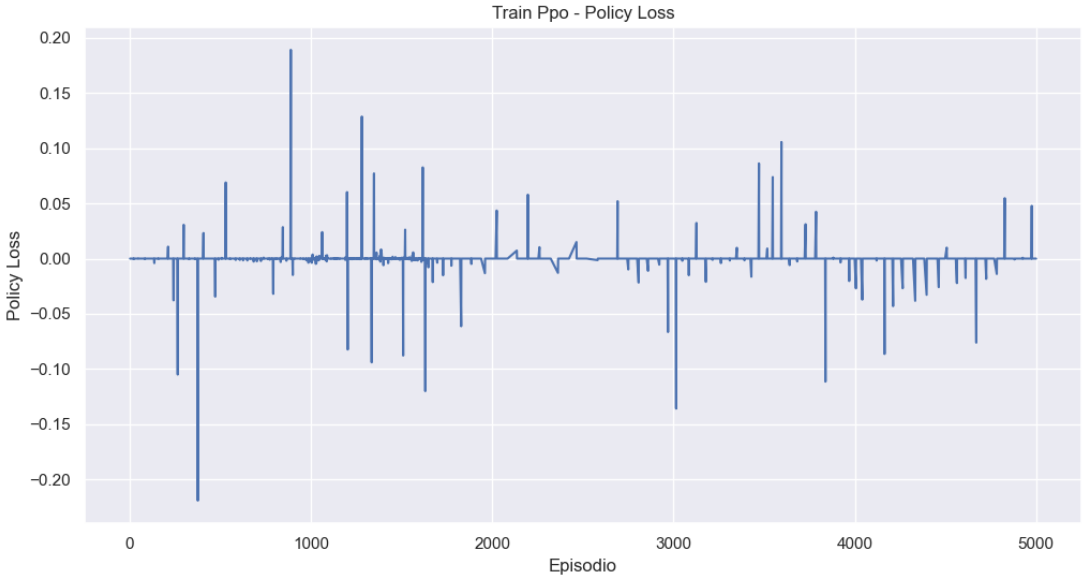
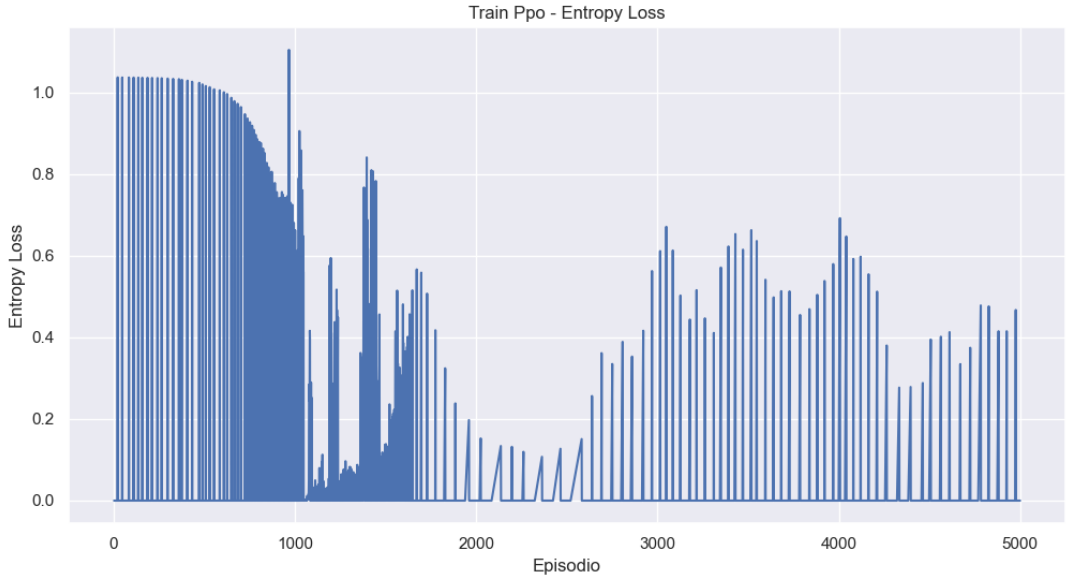


Figura 12: Pérdida de Valor (Value Loss) - entrenamiento (PPO).



(< -2500) entre los episodios 1000 y 1500 aproximadamente. Esto coincide con una explosión en la Value Loss (Figura 10) y un aumento repentino en la duración de los episodios (Figura 9), donde el agente parece quedarse atascado durante los 2000 pasos

máximos. Este comportamiento sugiere una divergencia o colapso de la **función de valor y/o la política** durante esa fase.

- **Recuperación y estancamiento**: Después del colapso, la recompensa promedio se recupera rápidamente y se estabiliza cerca de -5, un valor significativamente mejor que el inicial. Sin embargo, no muestra una tendencia ascendente continua después del episodio 1800. Se queda estancada en ese nivel ligeramente negativo.

- **Metas alcanzadas**: La Figura 8 es reveladora. Muestra un breve período (coincidiendo con la caida de la recompensa) donde el agente aprende a alcanzar algunas metas (promedio móvil $>$ 0,6). En estos episodios el agente alcanza algunas metas, pero luego se queda estancado acumulando una recompensa muy negativa en el episodio pese a haber alcanzado algún objetivo. Posteriormente vuelve a caer abruptamente a cero y permanece allí durante el resto del entrenamiento. Esto indica que la política aprendida durante la fase de recuperación no era robusta y colapsó a una estrategia que ya no busca ni alcanza las metas.

- **Pasos y pérdidas**: Los **pasos por episodio** (Figura 9) reflejan este colapso: después del bloque de episodios largos (1000-1700), vuelven a ser consistentemente bajos ($<$ 20 pasos), indicando fallos rápidos. La **pérdida de valor** (Figura 10), tras explotar, se estabiliza cerca de cero, lo que podría indicar que la red de valor aprendió a predecir un valor bajo y constante, o que las ventajas son muy pequeñas debido a los episodios cortos. La **pérdida de política** (Figura 11) es muy ruidosa, con picos positivos y negativos, sin mostrar una tendencia clara hacia la minimización (recordemos que **PPO minimiza -L_CLIP**), lo que sugiere actualizaciones erráticas. La **pérdida de entropía** (Figura 12) empieza alta (exploración inicial), luego cae drásticamente a cero durante la fase de colapso/recuperación (indicando que la política se volvió muy determinista/segura de sí misma, probablemente de forma prematura y errónea) y, aunque intenta recuperarse ligeramente después, permanece en niveles bajos, lo que es consistente con una política estancada y poco exploratoria. El ajuste del entropy_coef a 0.05 no parece haber sido suficiente para mantener la exploración activa a largo plazo.

5.3.3. Conclusión

A pesar de los ajustes de hiperparámetros, PPO no ha logrado aprender una política útil y estable para este problema. Ha experimentado una divergencia severa inicial, seguida de una recuperación parcial pero finalmente ha colapsado a una política ineficaz que no logra los objetivos. Las causas podrían ser una combinación de: **sensibilidad residual a hiperparámetros**,

la **escala de recompensas** que aún causan inestabilidad numérica, o la propia **dificultad del problema de exploración** para un algoritmo **on-policy** como PPO en este entorno con fallos frecuentes.

5.4. Soft Actor-Critic (SAC)

Se ha entrenado el agente SAC (versión discreta) durante 1000 episodios y se ha utilizado un ajuste automático de alfa.

5.4.1. Curvas de aprendizaje

Figura 13: Recompensa Promedio Móvil - entrenamiento (SAC).

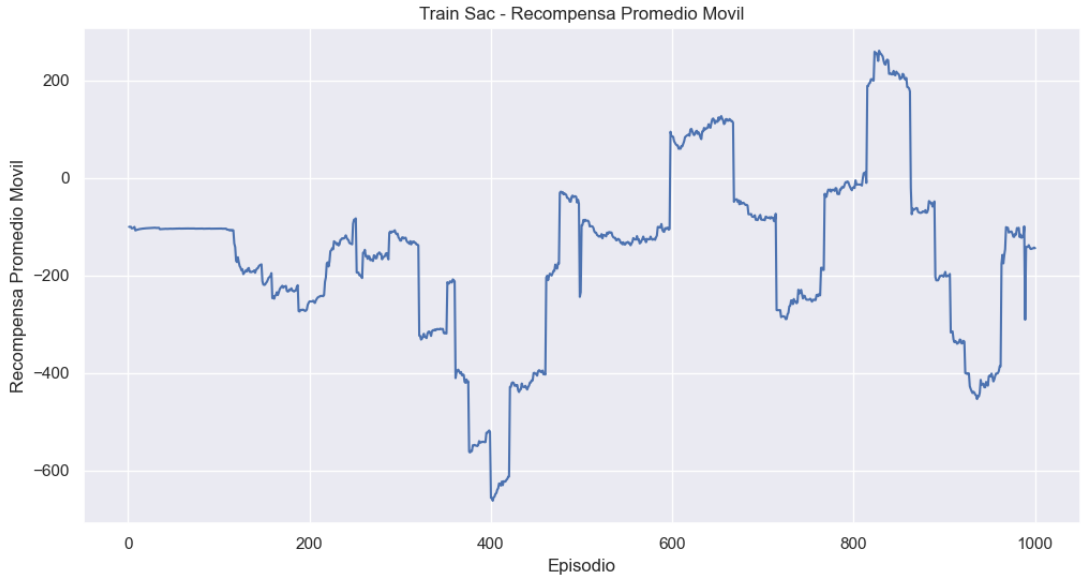


Figura 14: Metas Alcanzadas Promedio Móvil - entrenamiento (SAC).

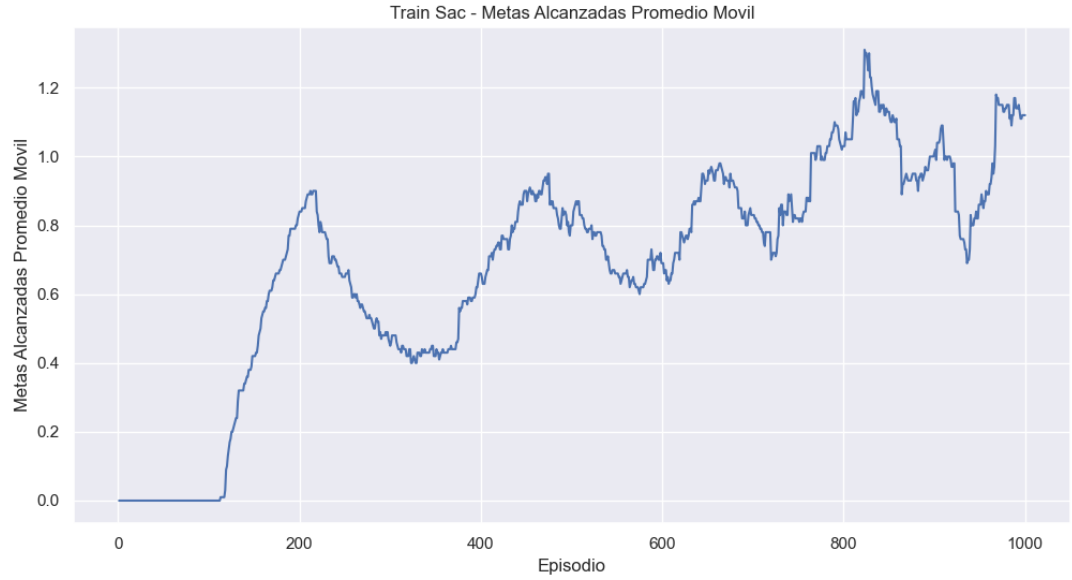


Figura 15: Pasos por Episodio - entrenamiento (SAC).

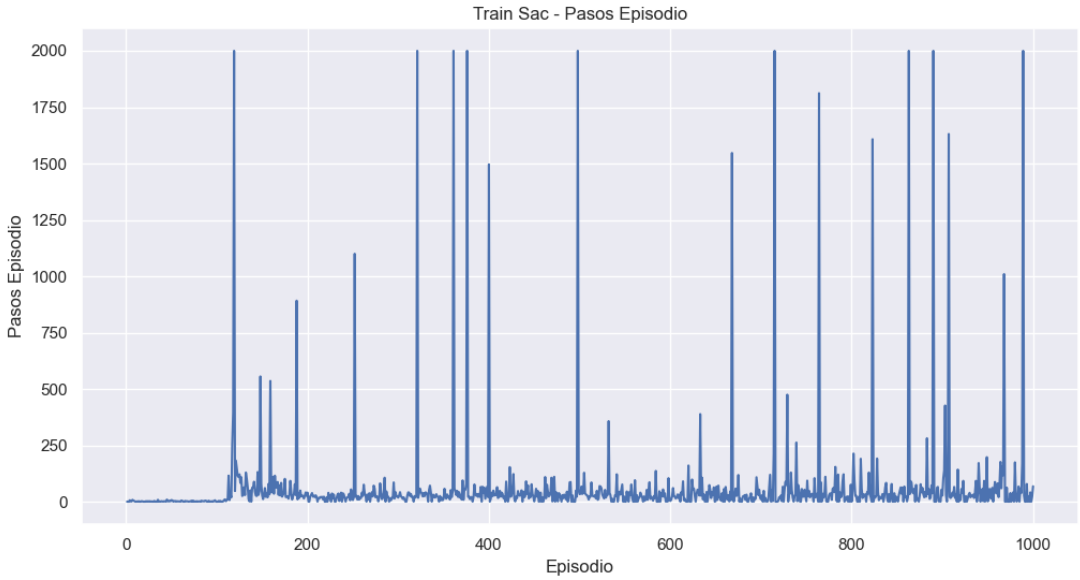
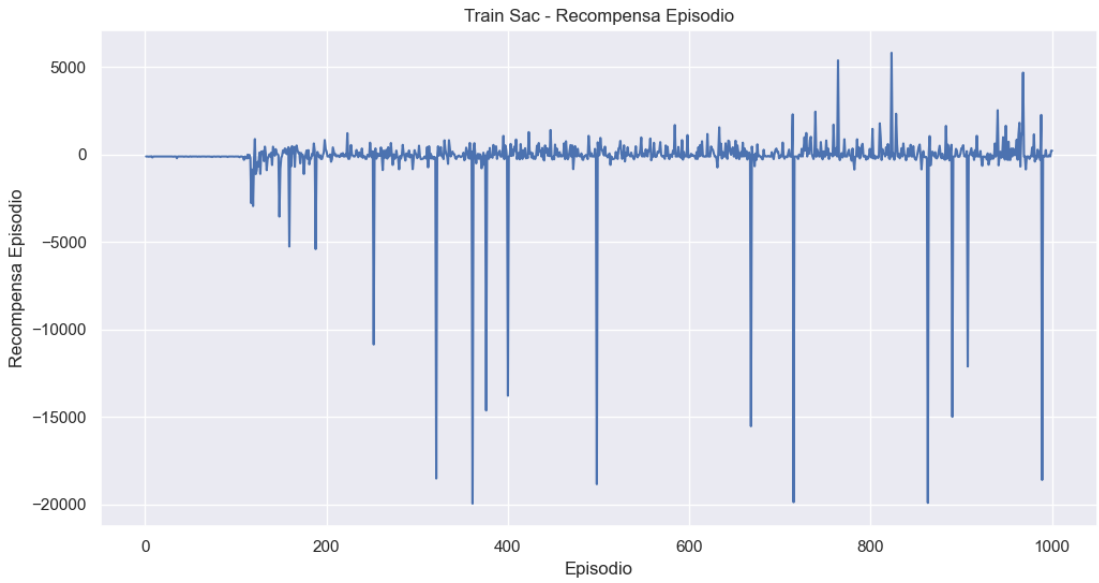


Figura 16: Recompensa por Episodio - entrenamiento (SAC).



5.4.2. Análisis de convergencia y rendimiento

- **Aprendizaje inicial rápido**: SAC muestra el comportamiento de aprendizaje más prometedor inicialmente. La recompensa promedio móvil (Figura 13) aumenta de forma constante y significativa desde valores negativos hasta alcanzar valores positivos (hasta +200).
- **Metas alcanzadas**: El promedio móvil de metas alcanzadas (Figura 14) refleja este aprendizaje, subiendo rápidamente y alcanzando un pico por encima de 1.2 metas por episodio en promedio, indicando que el agente logra completar segmentos de ruta con frecuencia.
- **Pasos por episodio**: La Figura 15 muestra que, coincidiendo con la mejora en recompensas y metas, la duración de los episodios aumenta drásticamente, alcanzando frecuentemente el límite de max_steps (2000). Esto indica que el agente aprende a sobrevivir, aunque esto coincide con episodios de recompensa muy negativa, en los que el agente se queda estancado sin realizar movimientos.

5.4.3. Conclusión

SAC demuestra ser el algoritmo más eficaz en la fase inicial de aprendizaje para este problema, logrando recompensas positivas, alta duración de episodios y múltiples metas alcanzadas.

Su mecanismo de maximización de entropía parece facilitar una exploración más efectiva que DDQN o PPO inicialmente. Sin embargo, muestra signos de inestabilidad a largo plazo o posible colapso/olvido de la política aprendida. Esto podría deberse a la interacción del ajuste automático de alfa, la acumulación de errores en el buffer off-policy, o la necesidad de un ajuste más fino de hiperparámetros como tau o las tasas de aprendizaje.

6. Análisis de problemas comunes y soluciones intentadas

Durante la fase de experimentación con los diferentes algoritmos de DRL, han surgido varios desafíos recurrentes que han afectado significativamente al rendimiento y capacidad de aprendizaje de los agentes. Esta sección analiza estos problemas clave y las distintas estrategias y ajustes que se han implementado en un intento por mitigarlos.

6.1. Diseño de la función de recompensa (iteraciones y efectos)

La definición de una función de recompensa adecuada demostró ser uno de los aspectos más críticos y desafiantes. Una recompensa mal diseñada puede guiar al agente hacia óptimos locales no deseados o impedir completamente el aprendizaje.

- **Desbalance inicial entre penalizaciones y recompensas**: Las primeras configuraciones presentaban penalizaciones muy altas por eventos negativos (colisiones, teleportaciones) y recompensas positivas muy bajas por comportamientos deseados como avanzar en dirección al objetivo (*approaching_goal*). Esto se manifestó en unos resultados iniciales donde los agentes rápidamente aprendían a minimizar la penalización fallando lo antes posible (episodios muy cortos, recompensas consistentemente negativas). El incentivo para explorar y encontrar secuencias de acciones largas y potencialmente recompensantes era prácticamente inexistente.

- **Iteraciones en los pesos**: Se han realizado varios ajustes a los pesos en DEFAULT_REWARD_WEIGHTS:

  - Se incrementó significativamente la recompensa por progreso hacia la meta (approaching_goal), utilizada una fórmula basada en la diferencia entre las distancias al objetivo del paso actual y el paso anterior.
  - Se redujo la penalización por paso (step_reward), creada inicialmente para fomentar llegar a la meta en el menor número de pasos posibles, para no desincentivar episodios largos, buscando alcanzar varias metas.
  - Se redujeron ligeramente algunas penalizaciones como emergency_stop y keep_lane.

- **Efectos observados**: Estos ajustes, particularmente el aumento de la recompensa por progreso/distancia y la eliminación de la penalización por paso, permitieron que agentes como DDQN (ver Figura 5) y SAC (ver Figura 14) mostraran signos claros de aprendizaje,

aumentando la duración de los episodios y el número de metas alcanzadas. Sin embargo, el balance óptimo seguía siendo difícil de encontrar, como lo demuestran las oscilaciones en DDQN y la inestabilidad tardía en SAC. PPO, incluso con los ajustes, siguió mostrando problemas graves (Figura 8).

6.2. Representación del estado

La forma en que se representa el estado del entorno para el agente es fundamental para el aprendizaje de los diferentes agentes.

- **Suficiencia**: El estado de **56** dimensiones implementado captura información local importante (posición, velocidad, vehículos cercanos, próximos giros, semáforos). Sin embargo, carece de información de planificación a largo plazo (ruta completa, distancia al destino final) que podría ser crucial para decisiones estratégicas (se probará en futuras iteraciones) y para una mejor estimación de la función de valor, especialmente para algoritmos actor-crítico como PPO y SAC.

- **Normalización**: Se aplica normalización a la mayoría de las características continuas (velocidad, límite de velocidad, coordenadas relativas, distancias), dividiendo por valores máximos esperados o el radio de detección. Una normalización inadecuada o inconsistente podría haber contribuido a la inestabilidad observada, especialmente en la pérdida de valor (Value Loss) de PPO.

- **Padding/Truncamiento de vehículos cercanos**: El uso de un tamaño fijo (8) para los vehículos cercanos, rellenando con valores placeholder ([0, 0, 0, 0, 2.0]) o truncando si hay más, es una simplificación necesaria para las redes neuronales utilizadas. Sin embargo, introduce limitaciones: pérdida de información si hay más de 8 vehículos relevantes y la necesidad de que la red aprenda a ignorar el padding, lo cual puede ser ineficiente. El valor 2.0 como distancia placeholder [3] podría ser ambiguo. Se consideraron alternativas como usar valores fuera de rango (-1) o máscaras, pero no se implementaron.

6.3. Estabilidad y convergencia algorítmica

Cada algoritmo presenta sus propios desafíos de estabilidad.

[3]Un **placeholder** es un elemento temporal o simbólico que se utiliza para reservar espacio o representar datos que aún no están disponibles o definidos. Es común en programación y diseño de interfaces, como en formularios web, donde un placeholder puede mostrar texto indicativo dentro de un campo de entrada para guiar al usuario sobre qué información introducir.

- **DDQN**: Muestra aprendizaje inicial pero luego entró en fuertes oscilaciones (Figura 3), típico de algoritmos Q-learning donde las estimaciones de valor pueden fluctuar. Los ajustes en *target_update* y *epsilon_decay* intentaron mitigar esto, con éxito parcial.
- **A3C**: La principal barrera es la inestabilidad a nivel de sistema al intentar lanzar múltiples instancias de SUMO/TraCI concurrentemente desde hilos, impidiendo evaluar su convergencia algorítmica.
- **PPO**: Sufrió una gran divergencia inicial, marcada por la explosión de la *Value Loss* (Figura 10). Aunque los ajustes en *n_steps*, *n_epochs*, *lr* y *entropy_coef*, junto con el *clipping* de gradientes, evitaron la divergencia completa en ejecuciones posteriores, la política colapsó rápidamente a un mínimo local subóptimo (entropía cero, cero metas en Figura 8). Esto sugiere una alta sensibilidad a los datos iniciales (posiblemente muy negativos o ruidosos) y/o una dificultad intrínseca del algoritmo on-policy para explorar eficazmente en este entorno con fallos frecuentes.
- **SAC**: Demuestra la mejor capacidad de exploración inicial y convergencia hacia recompensas positivas (Figura 13), probablemente gracias a la maximización de entropía. Sin embargo, mostró inestabilidad a largo plazo, con un declive en el rendimiento y picos negativos extremos en la recompensa por episodio. Esto podría deberse a la deriva de la política off-policy acumulando errores o a una mala sintonización del ajuste automático de alfa.

6.4. Interacción con el entorno SUMO

La propia dinámica del entorno simulado y la interacción con TraCI presentaron desafíos.

- **Rutas dinámicas**: La asignación de una nueva ruta corta ([actual, siguiente]) al alcanzar una meta funcionó, pero la detección del "final"del edge (dist_to_end $<$ 2,0) combinada con la posibilidad de que el siguiente edge fuera muy corto, llevó inicialmente a recompensas goal_reached múltiples e indeseadas en pasos consecutivos. Esto se corrigió rastreando si la meta del segmento ya había sido recompensada.
- **Fallo en asignación de ruta**: En ocasiones, _assign_new_route no encontraba un siguiente edge válido (ej., callejones sin salida). Inicialmente, esto podía dejar al agente atascado; se modificó para que en ese caso se marcara el episodio como terminal (info["terminal"] = "goal_reached_no_new_route").
- **Estado _get_state y Errores TraCI**: Se encontraron y corrigieron errores IndexError al acceder a datos de traci.lane.getLinks y ValueError por inconsistencias en la dimensión del

estado devuelto (especialmente en estados terminales o de error), haciendo más robusta la obtención del estado.

- **Inicio Concurrente (A3C)**: Como se detalló en Subsección 5.2, el inicio concurrente de SUMO vía traci.start desde hilos resultó ser el principal obstáculo técnico irresoluble con esa aproximación.

7. Comparativa de rendimiento en evaluación

Tras la fase de entrenamiento, se ha llevado a cabo una fase de evaluación para medir el rendimiento de los modelos finales de los algoritmos DDQN, PPO y SAC en el entorno SUMO configurado. El objetivo de esta sección es comparar su capacidad para generalizar lo aprendido y desenvolverse en episodios no vistos previamente, centrándose en métricas clave de eficiencia, finalización de tareas y seguridad.

*** Nota:** Para esta evaluación se ha mantenido la aleatoriedad de los episodios, por lo que cada evaluación es diferente. Esto puede ser un problema al no reflejar la realidad, dependiendo de la suerte para obtener un mejor o peor resultado, lo cual se solucionará en futuras iteraciones en las que se creará un escenario de evaluación determinado común a todos los agentes.

7.1. Resultados de evaluación por episodio

En este apartado se presentan las gráficas que muestran el rendimiento de cada algoritmo evaluado (DDQN, PPO, SAC) a lo largo de los 20 episodios de la fase de evaluación. Estas gráficas permiten observar la variabilidad del rendimiento entre episodios individuales, aunque se profundizará más acerca de los resultados obtenidos en los siguientes apartados de esta sección.

7.1.1. DDQN

Figura 17: Colisiones - evaluación (DDQN).

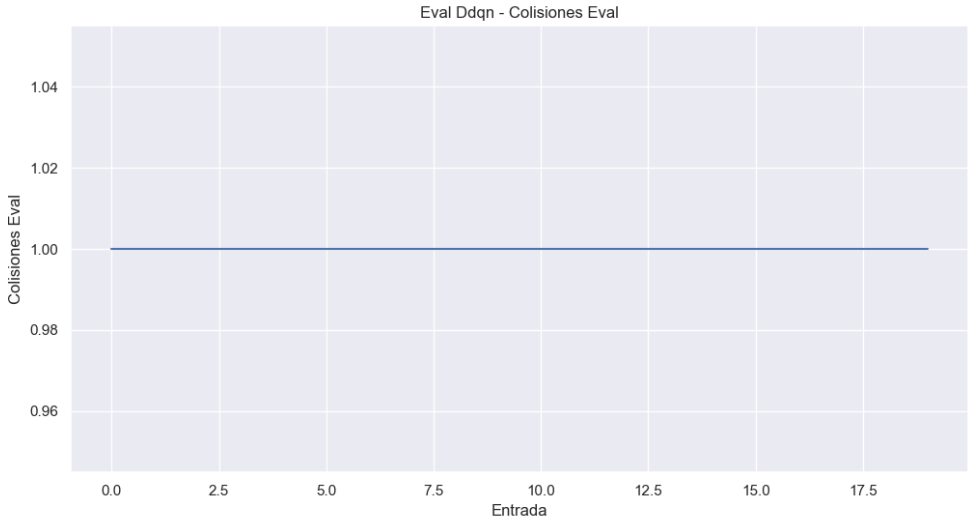


Figura 18: Metas alcanzadas - evaluación (DDQN).

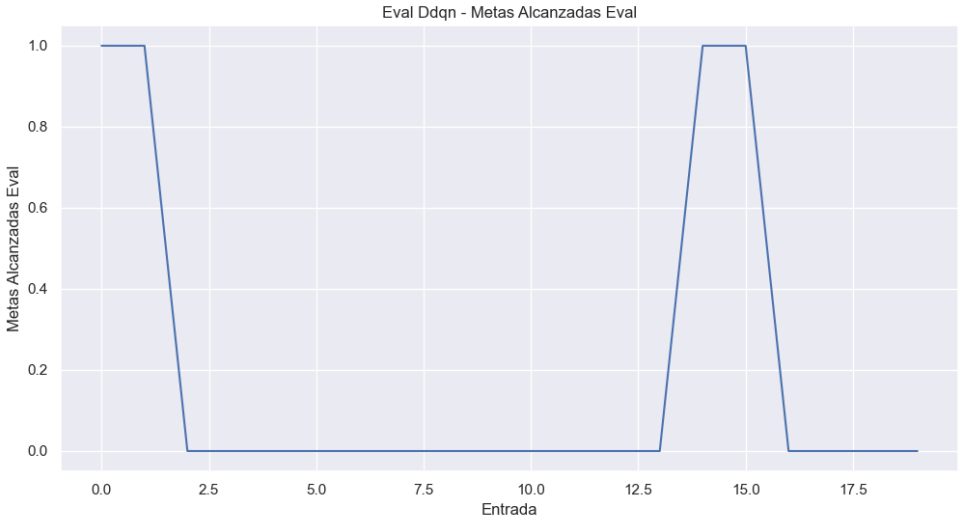


Figura 19: Pasos - evaluación (DDQN).

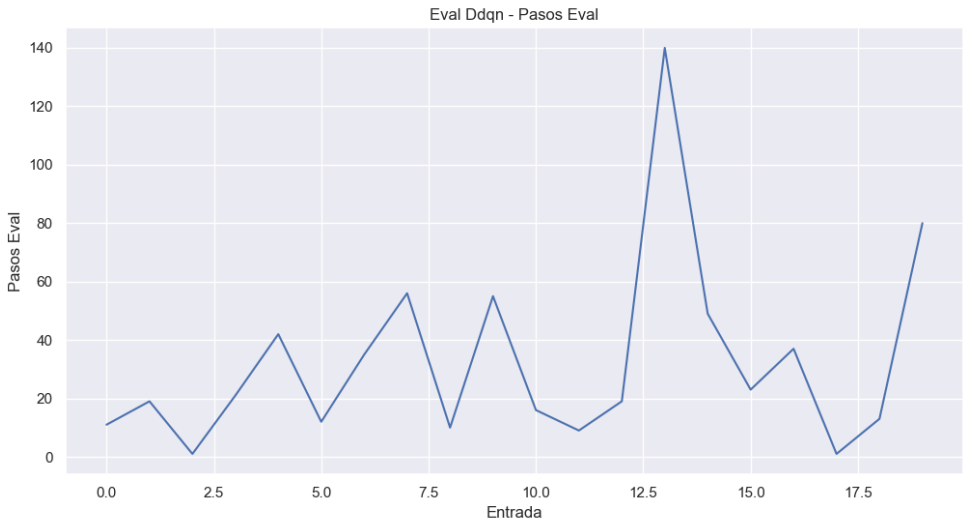


Figura 20: Recompensas - evaluación (DDQN).

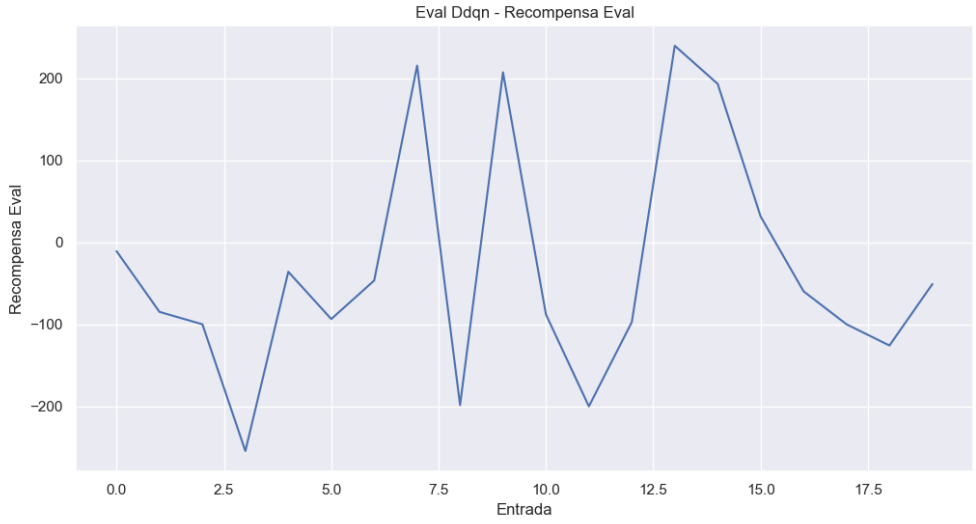
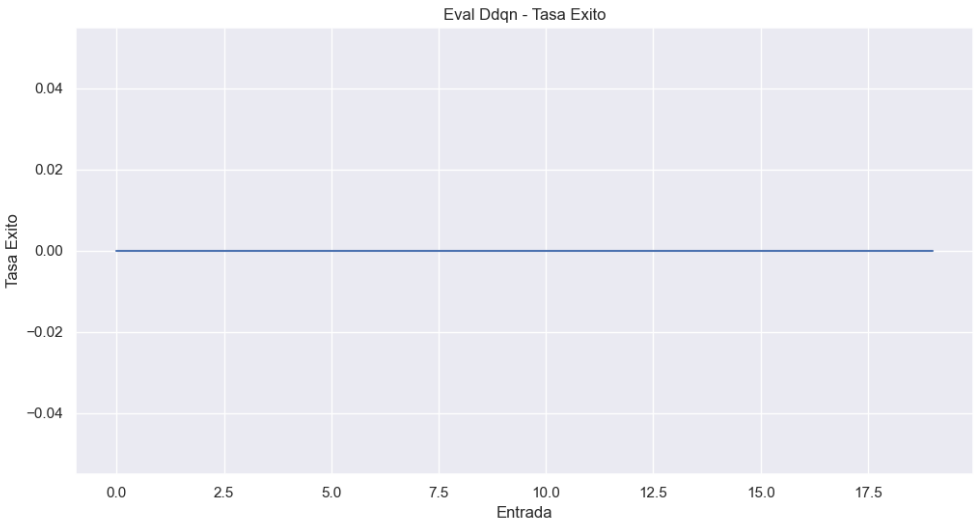


Figura 21: Tasa de éxito - evaluación (DDQN).



Estas gráficas ilustran una alta variabilidad en el rendimiento de DDQN, con algunos episodios logrando recompensas positivas y durando más pasos, mientras que otros fallan rápidamente.

7.1.2. PPO

Figura 22: Colisiones - evaluación (PPO).

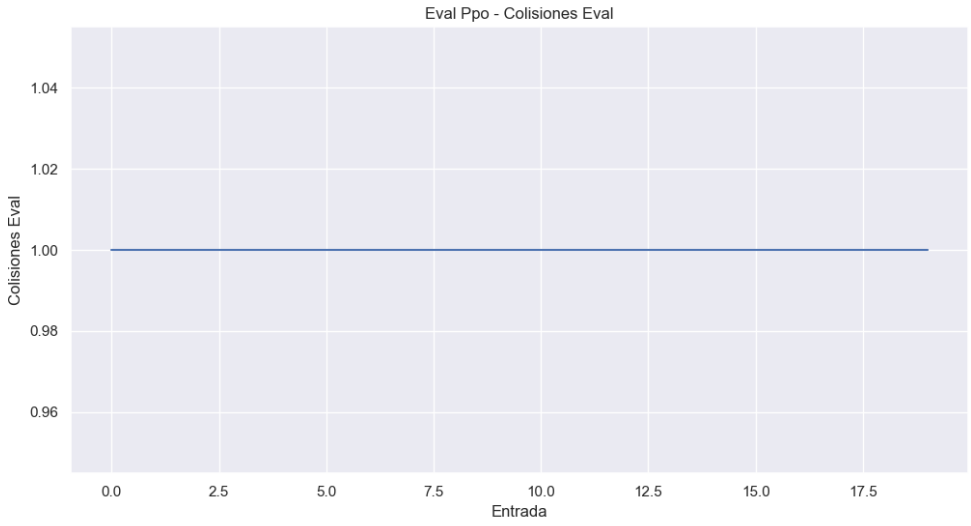


Figura 23: Metas alcanzadas - evaluación (PPO).

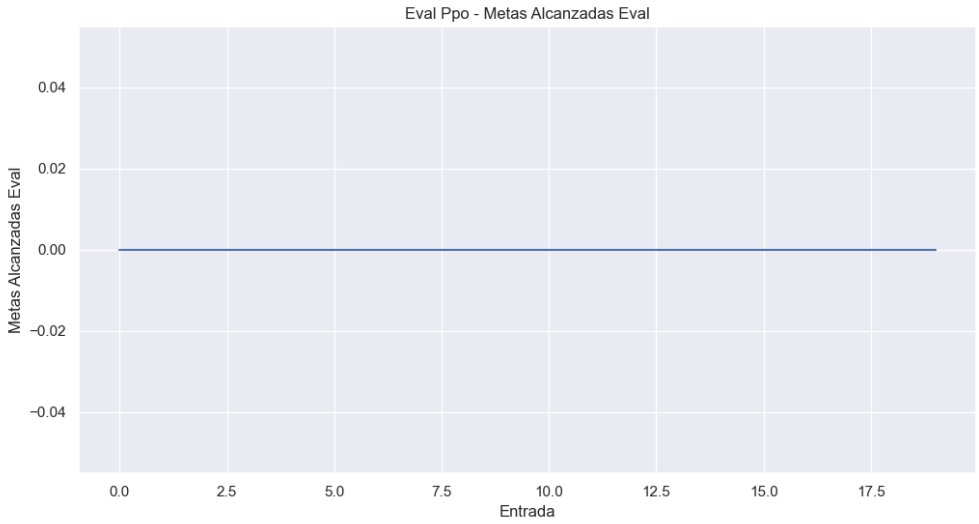


Figura 24: Pasos - evaluación (PPO).

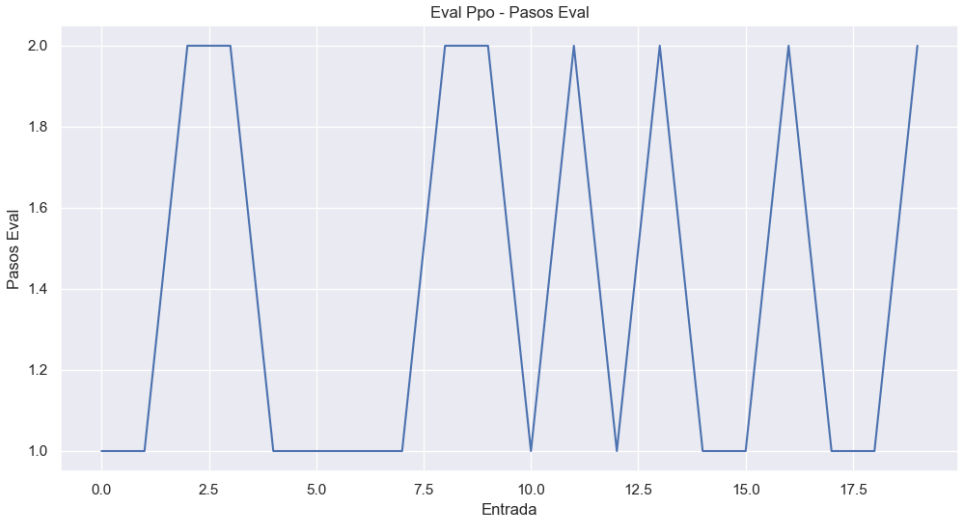


Figura 25: Recompensas - evaluación (PPO).

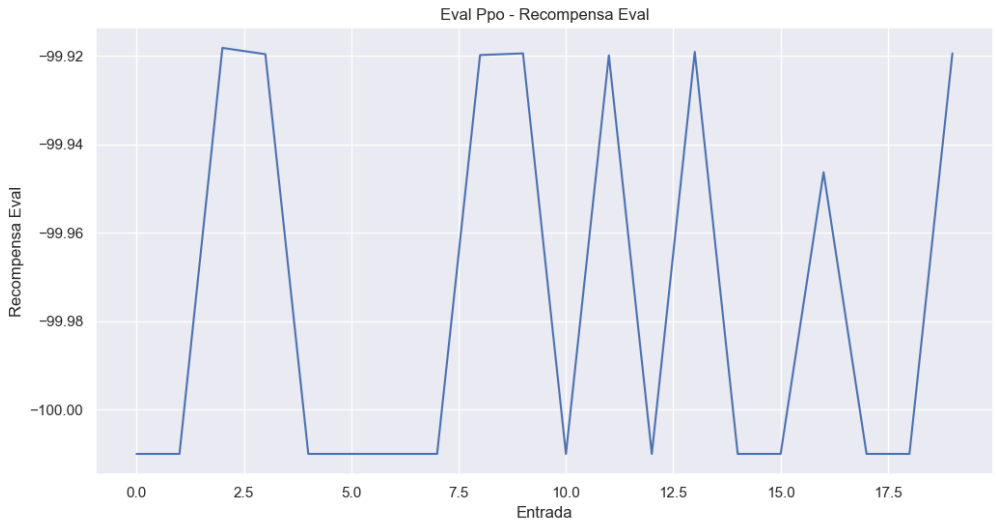
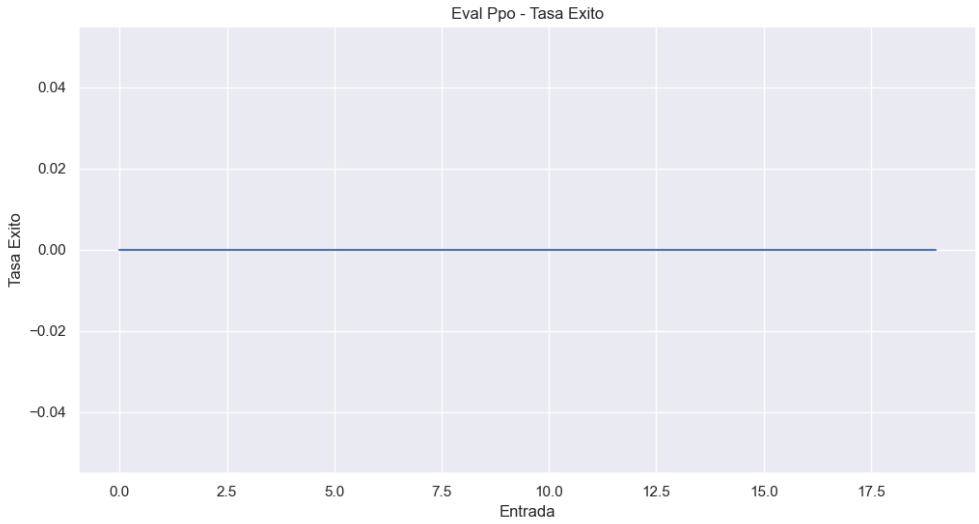


Figura 26: Tasa de éxito - evaluación (PPO).



Como se observa, el rendimiento de PPO en evaluación es consistentemente bajo, confirmando el colapso de la política durante el entrenamiento. Los episodios terminan casi inmediatamente.

7.1.3. SAC

Figura 27: Colisiones - evaluación (SAC).

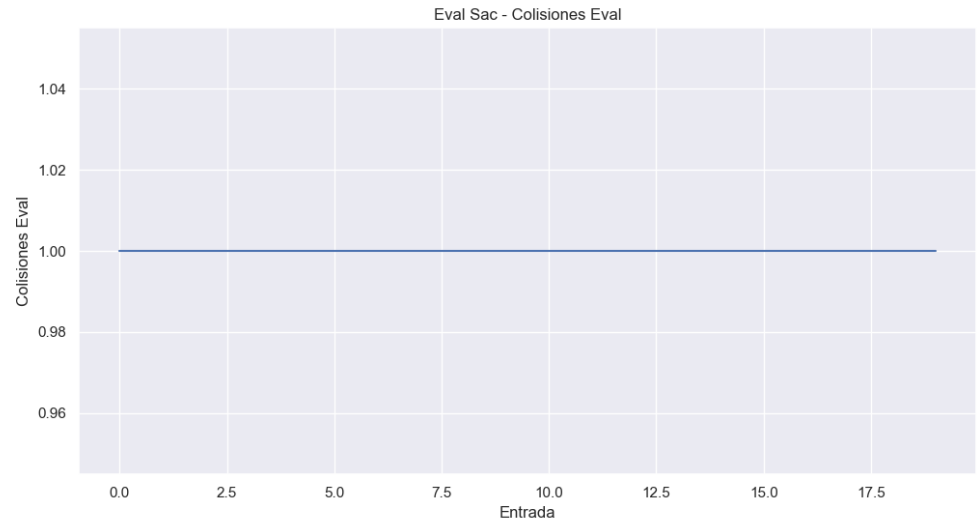


Figura 28: Metas alcanzadas - evaluación (SAC).

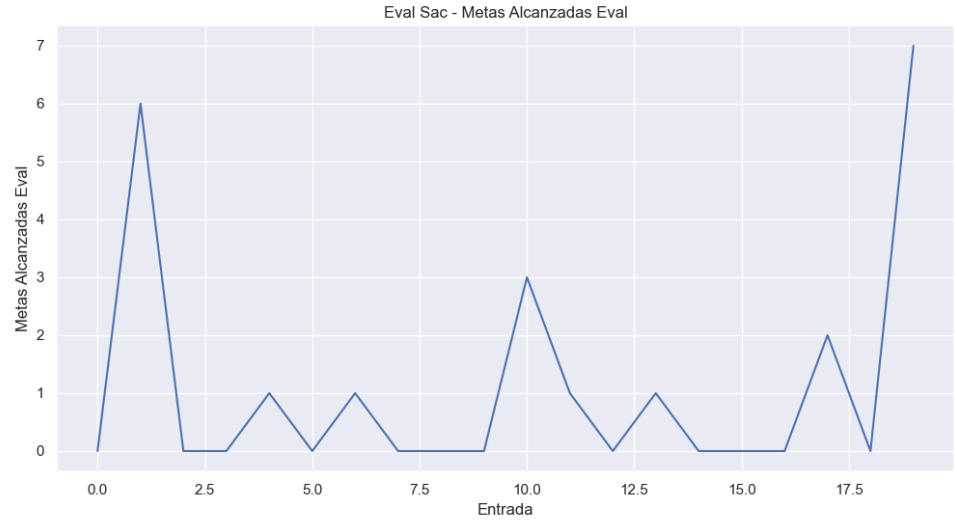


Figura 29: Pasos - evaluación (SAC).

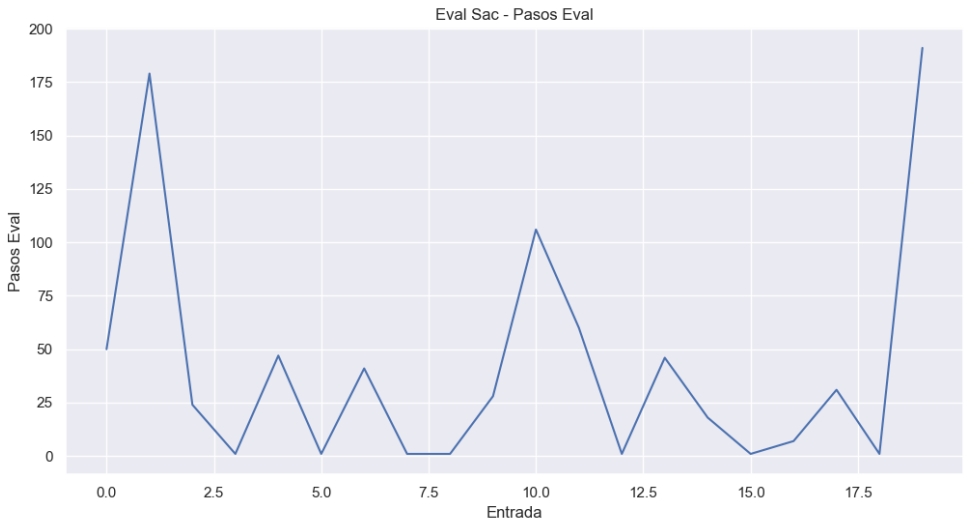


Figura 30: Recompensas - evaluación (SAC).

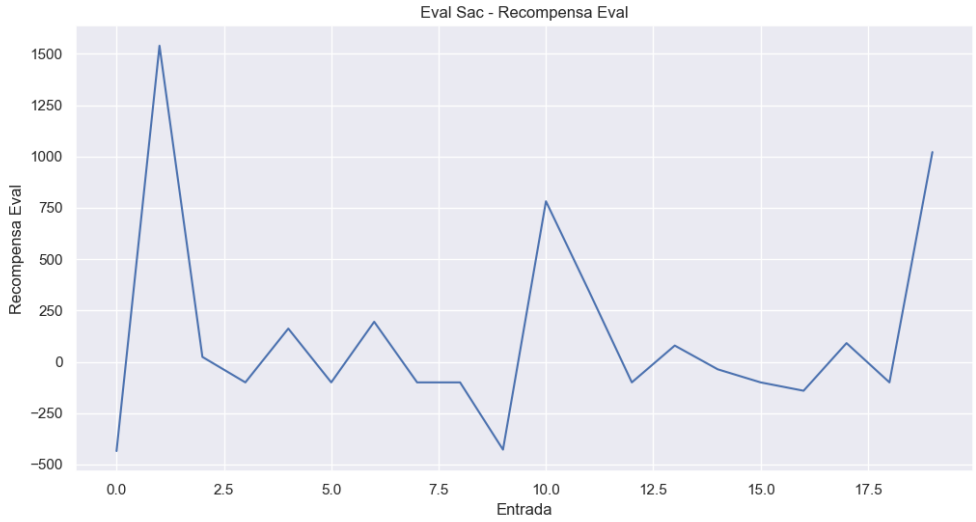
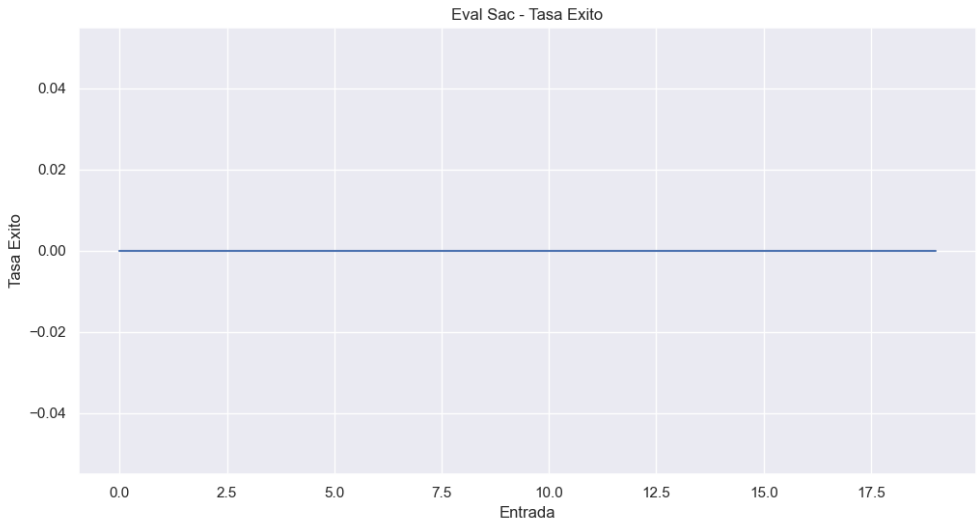


Figura 31: Tasa de éxito - evaluación (SAC).



Las gráficas de SAC muestran episodios con recompensas y metas significativamente altas, pero también una gran dispersión y episodios con resultados muy negativos, evidenciando la inestabilidad y la falta de seguridad consistente discutida previamente.

7.2. Tabla resumen de métricas de evaluación

La siguiente tabla resume las métricas promedio obtenidas durante la fase de evaluación para cada algoritmo.

Cuadro 2: Resumen de métricas promedio durante la evaluación (20 episodios)

Métrica	DDQN	PPO	SAC	A3C
Recompensa Prom.	−45,86	−99,96	149,67	N/A
Desv. Est. Recompensa	138,33	0,05	442,59	N/A
Pasos Prom.	44,15	1,55	61,85	N/A
Metas Prom.	0,15	0,00	1,25	N/A
Tasa Éxito (%)	0,00	0,00	0,00	N/A
Colisiones/Fallos Prom.	1,00	1,00	1,00	N/A

Nota: La Tasa de Éxito del 0 % y Colisiones/Fallos Prom. de 1.0 indican que todos los episodios terminaron en fallo.

La métrica *Colisiones/Fallos Prom.* muestra un valor de 1.0 para todos los algoritmos. Observando los gráficos de colisiones, se ve una línea plana en 1.0. Esto, combinado con una "Tasa Éxito"del 0.0 %, sugiere que todos los episodios de evaluación terminaron prematuramente debido a algún tipo de fallo (colisión detectada por SUMO, colisión forzada por acción inválida, o posiblemente teleportación u otro error no explícitamente separado en esa métrica). Por lo tanto, aunque SAC y DDQN muestran cierta capacidad de alcanzar metas y sobrevivir más pasos en algunos episodios, ninguno demuestra un comportamiento consistentemente seguro o exitoso en esta configuración.

7.3. Análisis comparativo

Basándonos en los resultados de la evaluación (Tabla 2) y recordando el proceso de entrenamiento (Sección 5) podemos sacar las siguientes conclusiones:

- **¿Qué algoritmo aprendió más rápido?**
  Durante el entrenamiento, SAC ha mostrado la curva de aprendizaje más prometedora y rápida, tanto en recompensa promedio móvil como en metas alcanzadas promedio móvil (Figura 13 y Figura 14), superando la fase inicial de recompensas muy negativas mucho antes que DDQN. DDQN también muestra algo de aprendizaje (Figura 3 y Figura 5) pero

más lento y con mayor inestabilidad temprana. PPO por su parte, colapsó y no mostró aprendizaje útil (Figura 7 y Figura 8).

- **¿Cuál alcanzó el mejor rendimiento promedio final (Eval)?**
  En términos de recompensa promedio durante la evaluación, SAC obtuvo el valor más alto (∼150), aunque con una desviación estándar extremadamente alta, indicando una gran variabilidad entre episodios. DDQN tuvo un rendimiento promedio mucho menor (∼-45) y también alta variabilidad. PPO tuvo un rendimiento consistentemente pésimo (∼-100) con varianza casi nula (Tabla 2).

- **¿Cuál fue más estable/consistente en evaluación?**
  Si la consistencia se mide por baja varianza, PPO fue el más consistente (0,05), pero en un estado de fallo inmediato (siempre ∼1.5 pasos, ∼-100 recompensa). Entre los que mostraron alguna capacidad, ninguno fue estable. Tanto DDQN como SAC tuvieron desviaciones estándar muy altas en la recompensa (138.33 y 442.59 respectivamente), reflejando episodios muy buenos mezclados con otros muy malos. Los gráficos de evaluación por episodio (Figura 20 y Figura 30) confirman esta alta variabilidad para DDQN y SAC.

- **¿Cuál logró completar más metas (Eval)?**
  SAC fue claramente superior en este aspecto, logrando un promedio de 1.25 metas por episodio en evaluación (Figura 28), con picos de hasta 7 metas en un solo episodio. DDQN logró muy pocas metas (promedio 0.15, Figura 18). PPO no logró ninguna meta en evaluación (Figura 23).

- **¿Cuál tuvo el comportamiento más seguro (menos colisiones/fallos) (Eval)?**
  Según la interpretación de los datos (Tasa Éxito 0 % y Fallos Promedio 1.0 para todos), ninguno de los algoritmos demostró un comportamiento seguro en la evaluación. Todos los episodios terminaron en algún tipo de fallo. Aunque SAC y DDQN sobrevivieron más pasos en promedio (61.85 y 44.15 respectivamente) que PPO (1.55), esto no se tradujo en episodios completados sin incidentes graves.

7.4. Discusión de las diferencias observadas

Las diferencias significativas en el rendimiento de evaluación entre los algoritmos pueden atribuirse a una combinación de sus características intrínsecas y los desafíos encontrados durante el entrenamiento en este entorno específico:

- **PPO (Fracaso total)**: El colapso observado durante el entrenamiento (Subsección 5.3) resultó en una política final inútil. El agente PPO evaluado simplemente ejecutaba 1

o 2 pasos y fallaba consistentemente. Esto subraya la sensibilidad de PPO a los hiperparámetros, la función de recompensa y potencialmente la **dificultad de los algoritmos on-policy** para recuperarse de fases de exploración muy negativas o divergencias en entornos con fallos frecuentes y recompensas posiblemente sparse o conflictivas.

- **DDQN (Aprendizaje parcial e inestable)**: Como algoritmo off-policy basado en valor, DDQN pudo aprender estrategias iniciales para evitar los peores resultados y ocasionalmente alcanzar metas (aprendizaje visible en Figura 3 y Figura 5). Sin embargo, tiene una gran potencial inestabilidad, especialmente con aproximadores de función (redes neuronales) y espacios de estados complejos. La alta varianza en recompensa y pasos durante la evaluación, junto con la incapacidad de converger a una política estable en el entrenamiento, reflejan estas limitaciones. La separación Dueling y el Deep Q-learning (DDQN) mitigan, pero no eliminan por completo, problemas como la **sobreestimación de valores Q**.

- **SAC (Mejor rendimiento, pero inestable e inseguro)**: SAC, siendo off-policy y maximizando la entropía, demuestra la mejor capacidad de exploración y aprendizaje inicial. La maximización de entropía le ayuda a encontrar políticas más diversas y a evitar quedarse atascado en mínimos locales tan fácilmente como DDQN o PPO en la fase inicial. Esto se tradujo en la mayor recompensa promedio, el mayor número de pasos promedio y la mayor cantidad de metas alcanzadas en evaluación. Sin embargo, la extrema varianza en la evaluación y la degradación del rendimiento observada al final del entrenamiento (Figura 13) indican que tampoco alcanzó una política robusta. Las posibles causas incluyen:

  - **Errores acumulados en el buffer**: Al ser off-policy, puede aprender de datos antiguos que ya no son representativos de la política actual, especialmente si la política cambia drásticamente.
  - **Ajuste de alfa**: Aunque el ajuste automático de alpha (temperatura de entropía) es una ventaja, puede volverse inestable o converger a un valor subóptimo en ciertas fases, afectando el equilibrio exploración/explotación.
  - **Complejidad del entorno**: El entorno en sí, con rutas dinámicas cortas y la posibilidad constante de fallos, puede ser intrínsecamente difícil para lograr estabilidad a largo plazo.

- **Falta universal de seguridad/exito**: El hecho de que ningún algoritmo lograra una tasa de éxito superior al 0 % en evaluación es la conclusión más preocupante. Indica que, a pesar de los esfuerzos en el diseño de la recompensa y la exploración de diferentes

algoritmos, la configuración actual (entorno + estado + acción + recompensa + algoritmos/hiperparámetros) no fue suficiente para producir un agente de conducción autónoma mínimamente fiable en el largo plazo. Los fallos constantes (colisiones, teleportaciones o acciones inválidas) sugieren que los agentes no aprendieron a manejar consistentemente situaciones de riesgo o a planificar de forma segura, incluso en los episodios donde lograban avanzar y cumplir metas parciales. Esto podría deberse a:

- Representación de estado insuficiente (falta de información predictiva o de planificación a más largo plazo).
- Función de recompensa que, aunque mejorada, todavía no penaliza suficientemente los comportamientos sutilmente arriesgados o no recompensa adecuadamente la conducción defensiva y fluida a largo plazo.
- Exploración insuficiente o ineficaz para descubrir políticas verdaderamente seguras.
- Complejidad inherente del entorno SUMO simulado.

En resumen, SAC muestra el mayor potencial, aprendiendo más rápido y logrando los mejores resultados promedio en evaluación, pero sufre de alta inestabilidad y falla en demostrar seguridad. DDQN muestra un aprendizaje limitado y también inestable. PPO no logra aprender una política útil en esta configuración. La falta de éxito generalizado en términos de seguridad resalta los desafíos pendientes en este proyecto.

8. Conclusiones y trabajo futuro

TODO:
Esta sección debe incluir lo siguiente:

- Una descripción de las conclusiones del trabajo.
- Una evaluación crítica del grado de logro de los objetivos iniciales.
- Una evaluación crítica de la planificación y metodología utilizadas en el proyecto.
- Considerando los desafíos de sostenibilidad, diversidad y ético-sociales vinculados al proyecto.
- Una discusión de temas para trabajo futuro potencial que no se hayan explorado en este proyecto.

9. Glosario

TODO:
Definición de los términos y acrónimos más relevantes utilizados en este informe.

Bibliografía

[1] Diaz V. Repositorio GitHub del proyecto. (2025). https://github.com/VictorDiazBustos/TFM_Explorando_tecnicas_avanzadas_de_RL_para_conduccion_autonoma

[2] Sutton R. S. & Barto A. G. (2020). *Reinforcement Learning: An Introduction"* (2ª ed.). MIT Press. http://incompleteideas.net/book/RLbook2020.pdf

[3] Goodfellow I., Bengio Y. & Courville A. (2016). *"Deep Learning"*. MIT Press. https://www.deeplearningbook.org/

[4] Grigorescu M., Trasnea B., Cocias T. & Macesanu, G. (2020). *"A Survey of Deep Learning Techniques for Autonomous Driving"*. IEEE Access, 8. https://arxiv.org/pdf/1910.07738

[5] Schwall M., Daniel T., Victor T., Favarò F. & Hohnhold H.(2020). *"Waymo Public Road Safety Performance Data"*. https://arxiv.org/pdf/2011.00038

[6] *"Cruise (autonomous_vehicle)"*. Wikipedia. https://en.wikipedia.org/wiki/Cruise_(autonomous_vehicle)

[7] Rosenzweig J., Bartl M. (2015). *"A Review and Analysis of Literature on Autonomous Driving"*. https://michaelbartl.com/wp-content/uploads/Lit-Review-AD_MoI.pdf

[8] Kiran B. R., Sobh I., Talpaert V., Mannion P., Al Sallab A. A., Yogamani S. & Pérez P. (2020). *"Deep Reinforcement Learning for Autonomous Driving: A Survey"*. https://arxiv.org/pdf/2002.00444

[9] Kendall A., Hawke J., Janz D., Mazur P., Reda D., Allen J.-M., Lam V.-D., Bewley A. & Shah A.(2018). *"Learning to Drive in a Day"*. https://arxiv.org/pdf/1807.00412

[10] Anisimov Y. O. & Katcai D. A. (2021). *"Deep reinforcement learning approach for autonomous driving"*.
https://www.researchgate.net/publication/351755367_Deep_reinforcement_learning_approach_for_autonomous_driving

[11] Dosovitskiy A., Ros G., Codevilla F., Lopez A. & Koltun, V. (2017). *"CARLA: An open urban driving simulator"*. https://arxiv.org/pdf/1711.03938

[12] Sallab A. A., Abdou M., Perot E. & Yogamani S. (2017). *"Deep Reinforcement Learning Framework for Autonomous Driving"*. https://arxiv.org/pdf/1704.02532

[13] Oates B. J. (2006). *Researching Information Systems and Computing"*. Wiley.
https://dokumen.pub/researching-information-systems-and-computing-978-1-4129-022
html

[14] DLR. (2018). *SUMO - Simulation of Urban MObility*. Último acceso 18 marzo 2025.
https://sumo.dlr.de/docs/

[15] DLR. (2024). *TraCI Documentation*. Último acceso 18 marzo 2025.
https://sumo.dlr.de/docs/TraCI.html

[16] European Commission, High-Level Expert Group on Artificial Intelligence. (2020). *"Ethics Guidelines for Trustworthy AI"*. Último acceso 18 marzo 2025.
https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai

[17] Clifton J. & Laber E. (2020). *"Q-Learning: Theory and Applications"*. Annual Review of Statistics and Its Application, Volume 7, 279-301.
https://www.annualreviews.org/content/journals/10.1146/annurev-statistics-031219-041220

[18] Alavizadeh H., Jang-Jaccard J. (2022). *"Deep Q-Learning Based Reinforcement Learning Approach for Network Intrusion Detection"*. Computers 2022, 11, 41.
https://doi.org/10.3390/computers11030041

[19] Wang Z., Schaul T., Hessel M., van Hasselt H., Lanctot M. & de Freitas N. (2016). *"Dueling Network Architectures for Deep Reinforcement Learning"*. Proceedings of The 33rd International Conference on Machine Learning.
https://doi.org/10.48550/arXiv.1511.06581

[20] Chen G., Sun J., Zeng, Q., Jing G. & Zhang Y. (2023). *"Joint Edge Computing and Caching Based on D3QN for the Internet of Vehicles"*. Electronics, 12(10), 2311. https://doi.org/10.3390/electronics12102311

[21] Noorani E. & Baras J. S. (2021) *Risk-sensitive REINFORCE: A Monte Carlo Policy Gradient Algorithm for Exponential Performance Criteria"*. 60th IEEE Conference on Decision and Control (CDC), Austin, TX, USA. https://doi.org/10.1109/CDC45484.2021.9683645

[22] Grondman I., Busoniu L., Lopes G. A. D. & Babuska R. (2012). *"A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients"*. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 42, nº 6. https://doi.org/10.1109/TSMCC.2012.2218595

[23] Paczolay G. & Harmati I. (2020). *"A New Advantage Actor-Critic Algorithm For Multi-Agent Environments"*. 23rd International Symposium on Measurement and Control in Robotics (ISMCR), Budapest, Hungary. https://doi.org/10.1109/ISMCR51255.2020.9263738

[24] Shen H., Zhang K., Hong M. & Chen T. (2023) *"Towards Understanding Asynchronous Advantage Actor-Critic: Convergence and Linear Speedup"* in IEEE Transactions on Signal Processing, vol. 71. https://doi.org/10.1109/TSP.2023.3268475

[25] Schulman J., Levine S., Abbeel P., Jordan M. & Moritz P. (2015). *"Trust Region Policy Optimization"*. Proceedings of the 32nd International Conference on Machine Learning. https://proceedings.mlr.press/v37/schulman15.html

[26] Schulman J., Wolski F., Dhariwal P., Radford A. & Klimov O. (2017). *"Proximal Policy Optimization Algorithms"*. arXiv:1707.06347 https://doi.org/10.48550/arXiv.1707.06347

[27] Tan H. (2021). *Reinforcement Learning with Deep Deterministic Policy Gradient"*. International Conference on Artificial Intelligence, Big Data and Algorithms (CAIBDA), Xi'an, China. https://doi.org/10.1109/CAIBDA53561.2021.00025

[28] Mohammadpour S., Bengio E., Frejinger E. & Bacon P.L. (2024). *"Maximum entropy GFlowNets with soft Q-learning"*. Proceedings of The 27th International Conference on Artificial Intelligence and Statistics. https://proceedings.mlr.press/v238/mohammadpour24a.html

[29] Haarnoja T., Zhou A., Hartikainen K., Tucker G., Ha S., Tan J., Kumar V., Zhu H., Gupta A., Abbeel P. & Levine S. (2018). *"Soft Actor-Critic Algorithms and Applications"*. arXiv:1812.05905 https://doi.org/10.48550/arXiv.1812.05905

[30] Haarnoja T., Zhou A., Abbeel P. & Levine S. (2018). *"Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor"*. Proceedings of the 35th International Conference on Machine Learning. https://proceedings.mlr.press/v80/haarnoja18b

[31] Binding. Microsoft Learn. https://learn.microsoft.com/es-es/dotnet/api/system.windows.data.binding?view=windowsdesktop-8.0

[32] Multiprocessing. (2025). *Python Documentation*. Último acceso 1 mayo 2025. https://docs.python.org/3/library/multiprocessing.html

[33] Liang E., Liaw R., Moritz P., Nishihara R., Fox R., Goldberg K., Gonzalez J.E., Jordan M.I., Stoica I. (2018). *"RLlib: Abstractions for Distributed Reinforcement Learning"*. arXiv:1712.09381 https://doi.org/10.48550/arXiv.1712.09381