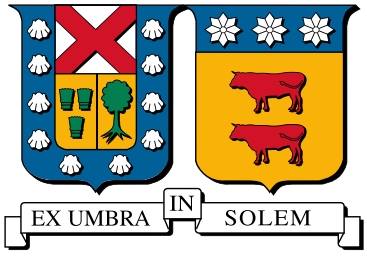
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA

DEPARTAMENTO DE ELECTRÓNICA

VALPARAÍSO - CHILE



"CONDUCCIÓN AUTÓNOMA BASADA EN VISIÓN CON APRENDIZAJE POR REFUERZO DISTRIBUIDO EN UN SIMULADOR DE ALTA FIDELIDAD"

SAMUEL ANDRÉS ORTIZ VELÁSQUEZ

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELECTRÓNICO.

PROFESOR GUIA: DR. WERNER CREIXELL FUENTES

PROFESOR CORREFERENTE: DR. MARCOS ZÚÑIGA BARRAZA

OCTUBRE 2025

CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: Conducción autónoma basada en visión con aprendizaje por refuerzo distribuido en un simulador de alta fidelidad

Nombre del candidato(a): Samuel Andrés Ortiz Velásquez

Carrera / Grado: Ingeniería Civil Electrónica

Campus: Casa Central **Departamento:** Electrónica

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Werner Creixell, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☒ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por **(marcar una opción)**:

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

_____

_____

_____

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 28/10/2025 **Firma:** Wcreixell

Estudiante o Candidato(a):

Fecha: 28/10/2025 **Firma:** 

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

DEPARTAMENTO DE ELECTRONICA
UNIVERSIDAD TECNICA FEDERICO SANTA MARIA

DEPARTAMENTO DE ELECTRONICA

Conducción autónoma basada en visión con aprendizaje por refuerzo distribuido en un simulador de alta fidelidad

Samuel Andrés Ortiz Velásquez

Memoria para optar al título de Ingeniero Civil Electrónico, mención Computadores, submención Gestión.

Universidad Técnica Federico Santa María

Profesor Guía: Dr. Werner Creixell Fuentes

Profesor Correferente: Dr. Marcos Zúñiga Barraza

OCTUBRE 2025

Resumen

Este trabajo presenta el desarrollo de un marco para el entrenamiento y la evaluación de agentes de conducción autónoma basados en visión en el simulador de carreras de alta fidelidad *Assetto Corsa*, integrado con una arquitectura distribuida de aprendizaje reforzado. El sistema permite la interacción en tiempo real y la recolección eficiente de datos, soportando experimentos con el algoritmo Soft Actor-Critic mediante secuencias temporales de imágenes, variables auxiliares de telemetría y demostraciones supervisadas. Los resultados muestran que el marco permite que los agentes aprendan a completar circuitos y que la asistencia humana inicial acelera el aprendizaje en escenarios complejos. El estudio resalta desafíos como la degradación del rendimiento y la sensibilidad a los hiperparámetros, al mismo tiempo que establece un banco de pruebas reproducible y adaptable para futuras investigaciones en conducción autónoma basada en visión.

Palabras Clave: Conducción autónoma; Aprendizaje reforzado; Soft Actor-Critic; Control basado en visión; Simulador de alta fidelidad; Aprendizaje profundo; Interacción en tiempo real; Aprendizaje asistido por humano; Marco de entrenamiento; Banco de pruebas.

Vision-Based Autonomous Driving with Distributed Deep Reinforcement Learning in a High-Fidelity Simulator

Samuel Andrés Ortiz Velásquez

Thesis for the fulfillment of the B.S. in Electronic Engineering, major in Computer Electronics, minor in Management (6 year program).

Universidad Técnica Federico Santa María

Advisor: Dr. Werner Creixell Fuentes

Co-Advisor: Dr. Marcos Zúñiga Barraza

OCTUBRE 2025

Abstract

This thesis presents the development of a framework for training and evaluating vision-based autonomous driving agents in the high-fidelity racing simulator *Assetto Corsa*, integrated with a distributed reinforcement learning architecture. The system enables real-time interaction and efficient data collection, supporting experiments with the Soft Actor-Critic algorithm using temporal image sequences, auxiliary telemetry, and supervised demonstrations. Results show that the framework allows agents to learn track completion and that human-assisted initialization accelerates learning in complex scenarios. The study highlights challenges such as performance degradation and hyperparameter sensitivity, while establishing a reproducible and adaptable testbed for future research in vision-based autonomous driving.

Keywords: Autonomous driving; Reinforcement Learning; Soft Actor-Critic; Vision-based control; High-fidelity simulator; Deep learning; Real-time interaction; Human-in-the-loop; Training framework; Testbed.

Glosario

API Interfaz de Programación de Aplicaciones (*Application Programming Interface*). Conjunto de funciones, protocolos y herramientas que permiten la comunicación e interacción entre diferentes programas o componentes de software..

CNN Red neuronal convolucional, una arquitectura de red neuronal profunda especialmente eficaz para procesar datos con estructura de cuadrícula, como imágenes. Utiliza capas convolucionales para extraer características espaciales jerárquicas, lo que la hace ampliamente utilizada en visión por computador. (Convolutional Neural Network).

Exploración Probar acciones desconocidas en busca del óptimo. Demasiada exploración podría hacer que no se aproveche lo ya conocido.

Explotación Tomar acciones que ya se sabe que funcionan, aprovechar al máximo lo que ya se conoce. Demasiada explotación podría dejar atrapado al agente en un máximo local, quizás el óptimo está en otro lado.

Función de Recompensas Es utilizada por el algoritmo RL para determinar qué tan bien está realizando una tarea. En RL tradicional debe ser definida por el programador.

Highlands Circuito ficticio incluido en Assetto Corsa. Presenta un trazado amplio, con secciones rápidas y variedad de curvas, diseñado para pruebas y eventos dentro del simulador..

Hotlap Start Modo de inicio en simuladores de conducción en el que el vehículo comienza desde una posición en pista con el motor encendido y listo para registrar tiempos de vuelta cronometrados, eliminando la fase de salida desde boxes..

MLP Perceptrón multicapa, una arquitectura de red neuronal artificial compuesta por múltiples capas de neuronas completamente conectadas. Cada capa transforma linealmente sus entradas y aplica una función de activación no lineal, permitiendo modelar relaciones complejas entre variables. Es ampliamente utilizada como clasificador o módulo de integración dentro de redes profundas. (Multi-Layer Perceptron).

Política Estrategia o conjunto de reglas que rigen la acción que tomará el agente en cada estado posible.

Reward Hacking Problema que suele surgir con funciones de recompensas para tareas complejas. El agente optimiza la función de recompensas, pero realiza un comportamiento distinto al esperado porque había un comportamiento no considerado que le entrega una mayor recompensa que el deseado.

RL Aprendizaje Reforzado, rama del aprendizaje de máquinas que busca optimizar una función de recompensas para completar una tarea de forma óptima. (Reinforcement Learning).

Silverstone 1967 Versión histórica del circuito de Silverstone, utilizada en simuladores de conducción. Se caracteriza por un trazado más simple, con curvas amplias y sin elementos visuales complejos que distraigan la conducción..

TLS *Transport Layer Security*, protocolo criptográfico que proporciona comunicaciones seguras en redes de computadoras, asegurando la confidencialidad e integridad de los datos transmitidos..

Índice de contenidos

Índice de tablas

Índice de figuras

1. Introducción

La conducción autónoma ha emergido como una de las áreas más activas de la ingeniería y la inteligencia artificial, impulsada por los avances en aprendizaje profundo, la disponibilidad de datos y el incremento del poder de cómputo. Su objetivo es desarrollar sistemas capaces de percibir el entorno, tomar decisiones y controlar un vehículo sin intervención humana. No obstante, su implementación enfrenta desafíos técnicos y de validación que requieren entornos de prueba seguros y flexibles.

Los simuladores de conducción de alta fiabilidad se han consolidado como una herramienta clave para abordar estos retos, permitiendo experimentar en condiciones controladas y reproducibles sin los riesgos y costos de la conducción real. En este trabajo se emplea el simulador *Assetto Corsa* [2] (véase Anexo A.1), caracterizado por un motor físico preciso y un realismo visual comparable al utilizado en el entrenamiento profesional, lo que lo convierte en un banco de pruebas idóneo para evaluar algoritmos de control vehicular.

El estudio se enfoca en la implementación de un agente de conducción autónoma basado en visión y entrenado mediante aprendizaje por refuerzo profundo, utilizando el algoritmo *Soft Actor-Critic* [3] (véase Anexo A.3) por su estabilidad y eficiencia en entornos con espacios de acción continuos. Para ello, se desarrolló una interfaz que conecta el simulador con una arquitectura distribuida de entrenamiento basada en TrackMania Reinforcement Learning [4] (véase Anexo A.4), permitiendo la recolección de experiencias en tiempo real. Además, se evaluó la incorporación de asistencia humana inicial para acelerar la convergencia del aprendizaje.

El alcance de este trabajo se restringe a escenarios de conducción controlados, sin tráfico ni variaciones climáticas. Cabe destacar que el simulador utilizado permite modificar condiciones de pista y clima, por lo que la metodología y las herramientas desarrolladas en este estudio pueden extenderse a contextos más complejos y realistas en futuras investigaciones.

1.1. Estado del arte

Para contextualizar adecuadamente el presente trabajo, resulta conveniente revisar tres aspectos fundamentales. En primer lugar, el estado actual de la conducción autónoma, sus niveles de automatización y los enfoques predominantes en la investigación reciente. En segundo lugar,

los fundamentos del aprendizaje por refuerzo, paradigma que permite a los agentes aprender políticas de control mediante interacción con el entorno. Finalmente, el análisis de trabajos similares que emplean enfoques de extremo a extremo proporciona un marco de referencia para situar las contribuciones del presente estudio.

1.1.1. Conducción autónoma

La conducción autónoma se refiere al uso de sistemas computacionales capaces de percibir el entorno, tomar decisiones y controlar un vehículo sin intervención humana. Su desarrollo se clasifica según la escala propuesta por la *Society of Automotive Engineers* (SAE) [5], que va desde el nivel 0, sin automatización, hasta el nivel 5, de autonomía total. El objetivo principal de esta tecnología es mejorar la seguridad vial, optimizar el flujo de tráfico y ofrecer nuevas formas de movilidad, con aplicaciones que abarcan desde transporte urbano hasta logística industrial.

En la actualidad, uno de los sistemas más avanzados en operación corresponde a *Waymo*, cuya flota comercial opera bajo un nivel 4 de autonomía en zonas geográficas delimitadas. Un estudio reciente que analiza 7,1 millones de millas recorridas por su servicio *rider-only* —sin conductor de seguridad a bordo— reporta tasas de siniestralidad inferiores a las de conductores humanos en condiciones comparables [6], lo que constituye una validación sólida del potencial de esta tecnología en entornos controlados.

Históricamente, los primeros sistemas de conducción autónoma se basaban en métodos de control clásico y visión por computador tradicional, empleando reglas predefinidas y algoritmos de detección de objetos que utilizaban características diseñadas manualmente, como bordes, esquinas o colores específicos. Un hito temprano en el uso de redes neuronales para esta tarea fue el proyecto ALVINN [7], desarrollado a finales de la década de 1980, que entrenaba un modelo para predecir la dirección de giro a partir de imágenes captadas por una cámara frontal.

En la última década, el incremento del poder de cómputo y los avances en técnicas de aprendizaje automático —especialmente en aprendizaje profundo— han permitido llevar la conducción autónoma a un nuevo nivel. Estos progresos han impulsado el desarrollo de enfoques de ex-

tremo a extremo, en los que redes neuronales procesan directamente datos sensoriales para generar acciones de control. Dichos métodos han demostrado una mayor capacidad de adaptación a entornos complejos y no estructurados, superando muchas de las limitaciones inherentes a los sistemas basados exclusivamente en programación explícita [8].

El éxito de un sistema de conducción autónoma depende de su capacidad para integrar y procesar información proveniente de múltiples sensores, como cámaras, LIDAR, radar, GPS e inerciales. Esta fusión sensorial permite estimar con precisión la posición del vehículo, detectar obstáculos y anticipar cambios en el entorno. La coordinación de estos módulos debe ser robusta frente a fallos y latencias, garantizando la fiabilidad en situaciones críticas [9].

A pesar de los avances, persisten desafíos significativos en la conducción autónoma, como la gestión de las variaciones climáticas, la seguridad en la toma de decisiones bajo incertidumbre y la validación previa al despliegue en entornos reales [10]. Estos retos hacen que la simulación se convierta en una herramienta indispensable: permite experimentar en entornos controlados, replicar escenarios críticos de forma segura y acelerar el ciclo de desarrollo a costos mucho menores [11]. Ejemplos notables de simuladores incluyen CARLA [12], TORCS [13] o AirSim [14], que ofrecen distintos grados de realismo y flexibilidad para la investigación.

1.1.2. Aprendizaje por refuerzo

En contextos complejos como juegos, robótica o conducción autónoma, la complejidad del entorno y la naturaleza secuencial de las decisiones hacen que el aprendizaje por refuerzo (RL, por sus siglas en inglés) sea uno de los enfoques más utilizados para abordar este tipo de problemas [15]. Este paradigma se fundamenta en que un agente autónomo aprende a optimizar su comportamiento mediante la interacción directa con el entorno, ajustando su estrategia a partir de la retroalimentación recibida en forma de recompensas. Dichas recompensas, que pueden ser positivas o negativas, guían la toma de decisiones sin requerir supervisión explícita ni datos previamente etiquetados.

El proceso de aprendizaje se modela comúnmente como un problema de decisión de Markov (MDP), en el que el agente observa un estado del entorno, ejecuta una acción y recibe una recompensa, lo que da lugar a un nuevo estado [16] (véase Figura 1.1). Mediante la exploración

de distintas estrategias, el agente estima una política que determina la mejor acción a realizar en cada estado. Para ello, puede valerse de funciones de valor o de aproximadores como redes neuronales profundas, especialmente en entornos complejos o de alta dimensionalidad [17].

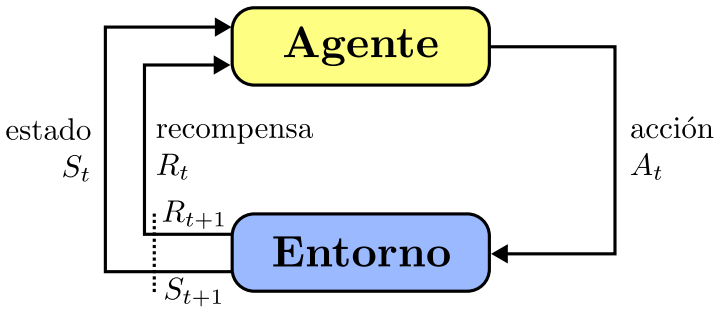


Figura 1.1: Funcionamiento de aprendizaje por refuerzo.

No obstante, el aprendizaje reforzado presenta desafíos significativos, entre ellos la alta varianza en los resultados, la exploración eficiente y la estabilidad del entrenamiento. Para abordar estos problemas, se han desarrollado algoritmos avanzados como Deep Q-Networks [17] (DQN), Proximal Policy Optimization [18] (PPO) o Soft Actor-Critic [3] (SAC), los cuales combinan técnicas de optimización y aprendizaje profundo para mejorar el rendimiento del agente.

1.1.3. Trabajos similares

Dentro de los enfoques de conducción autónoma se pueden distinguir principalmente dos paradigmas: el modular, basado en la secuencia percepción–planificación–acción, y el enfoque de extremo a extremo (*end-to-end*), en el que una red neuronal procesa directamente la entrada sensorial para generar acciones de control. Si bien el primero permite una mayor interpretabilidad al descomponer la tarea en etapas claramente definidas, también introduce puntos de fallo adicionales y dependencia de componentes intermedios. Por el contrario, el enfoque *end-to-end* optimiza la tarea de forma global, permitiendo que el sistema aprenda representaciones internas adaptadas al objetivo final, aunque con menor transparencia en la toma de decisiones. En este trabajo se revisan y analizan estudios recientes que adoptan este último enfoque.

Un ejemplo destacado es el de Lai y Bräunl [19], quienes proponen un modelo con memoria (CNN+LSTM y CNN3D) para manejar maniobras complejas en entornos interiores, como seguir un camino entre muros o realizar giros cerrados. El entrenamiento inicial se llevó a

cabo en un simulador, y posteriormente los modelos fueron implementados en un vehículo a escala denominado *ModCar*[1], lo que permitió validar su desempeño y realizar ajustes finos, demostrando su capacidad de transferencia del dominio simulado al real.

En Peng et al. [20] se desarrolla una arquitectura basada en Dueling Double DQN que integra imágenes de cámara y velocidad del vehículo. Evaluada en el simulador TORCS, supera a conductores humanos en tareas de mantenimiento de carril y aporta interpretabilidad mediante mapas de saliencia que muestran atención a las líneas del camino.

En un trabajo reciente, Lu [21] propone una metodología híbrida que combina aprendizaje por imitación, iniciado a partir de datos humanos, con entrenamiento de RL en simulación (AirSim). Este preprint reporta una reducción aproximada del 80 % en el tiempo de entrenamiento, manteniendo e incluso mejorando el rendimiento final respecto al uso exclusivo de RL, lo que pone de relieve el potencial de la asistencia inicial de un experto para acelerar la convergencia y mejorar la calidad del comportamiento aprendido.

1.2. Alcances y contribuciones

En el presente trabajo se desarrolla un entorno de experimentación para el entrenamiento y evaluación de agentes de conducción autónoma basados en visión, empleando un simulador de carreras con un alto grado de realismo visual y físico.

A diferencia de simuladores como *CARLA* o *AirSim*, orientados a entornos urbanos y con un modelo físico más simplificado, la plataforma utilizada en este trabajo ofrece una simulación vehicular precisa, comparable a la utilizada en entrenamiento profesional de pilotos, junto con una fidelidad visual propia de un título comercial. Esta combinación permite reproducir condiciones de conducción de alta dinámica más cercanas a las que enfrentarían sistemas reales en contextos competitivos, superando las limitaciones visuales y de modelado presentes en opciones como *TORCS*.

Las contribuciones de este trabajo se pueden resumir en los siguientes puntos:

[1]El *ModCar* es un coche a escala desarrollado en 2017 en la Universidad de Western Australia, equipado con Lidar y cámara de amplio rango, con modos de conducción manual, por Lidar y mediante red neuronal. Fue utilizado para validar en entornos interiores reales el modelo entrenado en simulación.

- Implementación de una interfaz que conecta el simulador de carreras con un marco de entrenamiento de aprendizaje por refuerzo distribuido, posibilitando la interacción en tiempo real y la recolección eficiente de datos de experiencia en un entorno no diseñado originalmente para RL.
- Adaptación del flujo de entrenamiento para evaluar distintas configuraciones de observaciones y estrategias de exploración, incluyendo secuencias de imágenes, variables auxiliares y variaciones en el historial de acciones.
- Incorporación y evaluación de un módulo de actuación humana (*human actor*) que permite guiar el comportamiento inicial del agente, facilitando la exploración y acelerando la convergencia del entrenamiento.
- Establecimiento de un banco de pruebas reproducible para futuros estudios de conducción autónoma basados en visión, en entornos de alta fidelidad visual y dinámica realista.
- Análisis comparativo del desempeño de distintos enfoques de entrenamiento, incluyendo aprendizaje por refuerzo puro y estrategias híbridas de imitación–refuerzo.

Este entorno y metodología pueden ser reutilizados y extendidos en investigaciones futuras, tanto para el desarrollo de nuevos algoritmos como para la evaluación de arquitecturas de percepción y control en tareas de conducción autónoma.

2. Propuesta

2.1. Arquitectura de entrenamiento remoto

Una de las ventajas clave de emplear un algoritmo *off-policy* como Soft Actor-Critic (SAC) es la posibilidad de desacoplar la recolección de experiencias del proceso de entrenamiento. Esto permite implementar arquitecturas distribuidas, donde múltiples procesos —incluso en distintas máquinas— pueden cooperar de forma asincrónica. Con este propósito, se utilizó la biblioteca `TMRL` [4], diseñada para facilitar el entrenamiento remoto de agentes en entornos de tiempo real.

La arquitectura de `TMRL` se basa en un esquema cliente-servidor, compuesto por tres entidades principales: *Trainer*, *RolloutWorker(s)* y *Server*. Estas entidades se comunican mediante sockets TCP, lo que habilita su despliegue tanto en una única máquina como en múltiples nodos interconectados.

Servidor (`Server`). Actúa como punto central de comunicación. Recibe muestras de experiencia generadas por los *RolloutWorkers* y distribuye periódicamente los pesos actualizados del modelo desde el *Trainer*. Es la primera entidad que debe inicializarse y permanece a la espera de conexiones entrantes.

Trabajadores de ejecución (`RolloutWorker`). Cada *RolloutWorker* ejecuta una instancia del entorno y genera trayectorias ejecutando la política actual. Al finalizar uno o más episodios, comprime las muestras recolectadas y las transmite al *Server*. Además, se encarga de recibir y aplicar los pesos actualizados enviados por el *Trainer*. Estos trabajadores pueden correr en paralelo sobre distintas instancias del entorno, acelerando significativamente la recolección de datos.

Entrenador (`Trainer`). Esta entidad centraliza el proceso de optimización de la política. Recibe muestras comprimidas desde el *Server* y las almacena en una memoria de repetición local. Desde allí, se extraen lotes para realizar pasos de entrenamiento mediante el algoritmo SAC. De forma periódica, el *Trainer* envía los pesos actualizados de la política al *Server*, los cuales son luego retransmitidos a todos los *RolloutWorkers* conectados.

Sincronización y rendimiento. Gracias a la naturaleza *off-policy* de SAC, la arquitectura

permite que los *RolloutWorkers* operen de manera completamente asincrónica respecto al *Trainer*, recolectando experiencias sin necesidad de esperar confirmaciones tras cada paso. Esta característica es particularmente útil en entornos de simulación no deterministas o con restricciones de tiempo real, como el utilizado en este trabajo.

Modularidad y extensibilidad. Cada componente de `TMRL` ha sido diseñado de forma modular. Por ejemplo, el *RolloutWorker* acepta un compresor de muestras personalizado, lo que permite reducir el tráfico de red eliminando redundancias (e.g., buffers de acciones). Del mismo modo, el *Trainer* puede usar una memoria adaptada al formato comprimido y realizar la descompresión al momento de usar los datos. Esta separación de responsabilidades permite optimizar el pipeline completo según las características específicas del entorno y del algoritmo.

La Figura 2.1[2] resume esta arquitectura distribuida. La comunicación entre componentes puede realizarse tanto en redes locales como remotas, y el uso de contraseñas o cifrado TLS es posible si se requiere seguridad adicional.

Adaptaciones específicas. Para el desarrollo de este trabajo, fue necesario extender la funcionalidad de la clase *RolloutWorker* a fin de incorporar mecanismos de seguimiento y depuración. En primer lugar, se integró la generación de registros compatibles con *TensorBoard*, lo cual permite visualizar en tiempo real métricas relevantes como la recompensa o duración de los episodios. Además, se implementó una lógica de guardado automático del modelo: cada vez que se detecta un nuevo valor récord de recompensa acumulada en episodios de prueba, los pesos de la política correspondiente se almacenan de forma persistente.

Adicionalmente, se añadió una interfaz visual simplificada para observar la interacción entre el agente y el entorno durante la ejecución. Esta funcionalidad, orientada al *debugging*, permite validar visualmente el comportamiento del agente, identificar errores en la recolección de datos y facilitar el análisis de trayectorias.

Estas modificaciones mejoran significativamente la trazabilidad y control del proceso de entrenamiento, especialmente en fases tempranas de desarrollo y evaluación experimental.

[2]Los términos en la figura se mantienen en inglés para preservar la nomenclatura original del software y las clases utilizadas en la implementación.

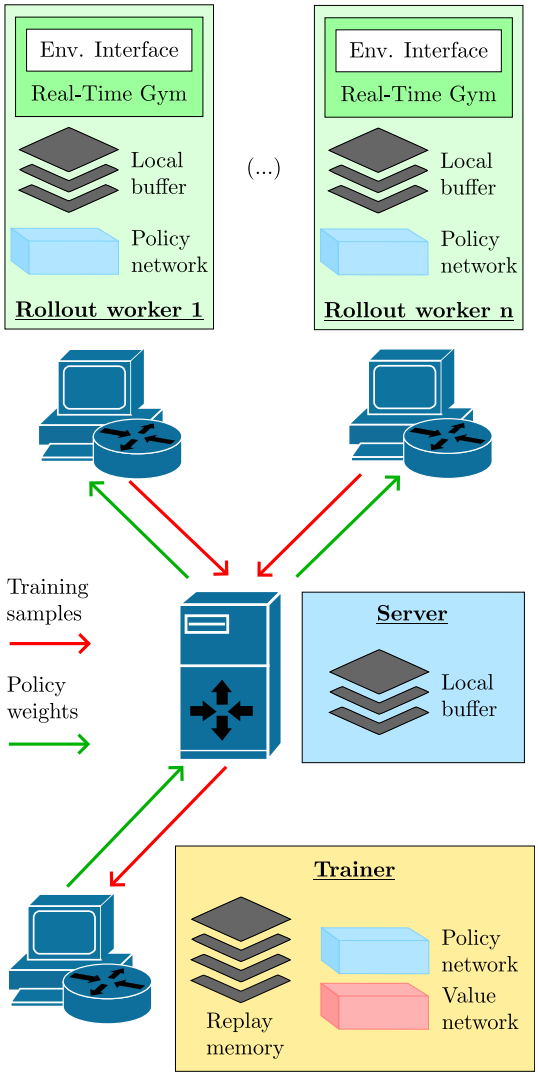


Figura 2.1: Arquitectura distribuida empleada.

2.2. Componente temporal en observaciones

En tareas como la conducción autónoma basada en visión, una imagen aislada no contiene suficiente información para inferir de forma precisa el estado dinámico del vehículo [22]. Por ejemplo, al observar un solo cuadro del entorno, es difícil determinar si el automóvil está en movimiento, acelerando, frenando o girando. Esta ambigüedad afecta directamente la capacidad de tomar decisiones adecuadas en tiempo real.

Los humanos utilizan una memoria a corto plazo para interpretar el flujo visual y anticipar las consecuencias de sus acciones; de forma análoga, el agente requiere un contexto temporal que

le permita comprender la evolución del entorno. Por esta razón, en este trabajo se representa el estado observado mediante una secuencia de imágenes, un historial de acciones recientes y un conjunto reducido de variables auxiliares que describen explícitamente el movimiento del vehículo.

Secuencias de imágenes. Una única imagen como en la Figura 2.2, incluso si presenta una vista clara del entorno, no permite inferir con precisión magnitudes como la velocidad o la aceleración. Esta limitación se vuelve evidente en situaciones donde se debe anticipar una curva, evaluar una pendiente o realizar una maniobra evasiva. Para superar esta deficiencia, se emplea una secuencia de n imágenes igualmente espaciadas en el tiempo como entrada de la red neuronal. Esta secuencia actúa como un fragmento de video que permite capturar patrones de movimiento —traslación, rotación, frenado— que son codificados implícitamente mediante el flujo visual entre cuadros consecutivos [23].

Figura 2.2: Ejemplo de observación visual aislada. No es posible determinar magnitudes de movimiento.

Historial de acciones. Además de las observaciones visuales, se incorpora un historial de acciones ejecutadas recientemente. Esta información proporciona a la red el contexto necesario para modelar el efecto acumulado de las decisiones pasadas, por ejemplo, si una acción ha sido sostenida o si se ha producido un cambio repentino. Esta representación es especialmente útil en contextos de control en tiempo real, donde existen retrasos inevitables entre la percepción del entorno, la inferencia de la acción y su aplicación efectiva. Incluir las acciones previas ayuda

a compensar este desfase, funcionando como una forma de memoria explícita que reduce la incertidumbre sobre el estado actual [24].

Variables auxiliares. Lee et al. [25] demostraron que la inclusión de variables auxiliares pasadas de forma explícita permite una convergencia más rápida y alcanzar recompensas acumuladas más altas en comparación con modelos basados únicamente en información visual. Siguiendo esta evidencia, en este trabajo se incorporan variables de telemetría —velocidad, cambio y revoluciones por minuto (RPM)— que, al igual que en el tablero de un vehículo real, complementan las observaciones visuales aportando señales cinemáticas explícitas.

En conjunto, esta representación de observación enriquecida, como se aprecia en la Figura 2.3, dota al agente de un contexto temporal y de movimiento suficiente para tomar decisiones informadas. La secuencia de imágenes, dispuestas desde la más antigua a la izquierda hasta la más reciente a la derecha, evidencia que el vehículo se encuentra trazando una curva hacia la derecha: puede observarse cómo, a medida que avanza, se aproxima progresivamente al borde derecho del circuito. Este tipo de información visual permite inferir no solo la orientación del vehículo, sino también la trayectoria esperada y posibles riesgos de colisión.

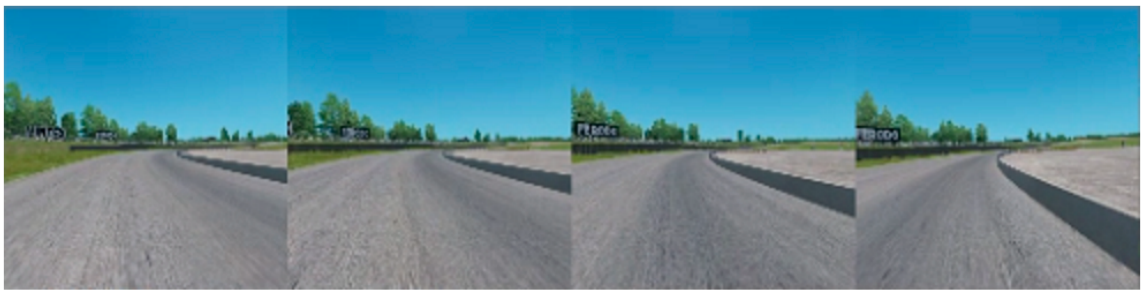
Complementariamente, el historial de acciones revela un ajuste progresivo en el comportamiento del agente. Siguiendo la convención

$$\mathbf{a}_{t-k} = [a_{t-k}^{\text{long}},\, a_{t-k}^{\text{lat}}], \quad k \in \{1, 2, 3, 4\},$$

el historial de acciones previas se representa como

$$\text{PrevActs}_t = \{\mathbf{a}_{t-4},\, \mathbf{a}_{t-3},\, \mathbf{a}_{t-2},\, \mathbf{a}_{t-1}\},$$

donde a_t^{long} corresponde a la aceleración o frenado, y a_t^{lat} al ángulo de dirección del volante. En los primeros pasos se observa una aceleración activa acompañada de un giro marcado hacia la derecha. Posteriormente, el agente disminuye la aceleración y aplica un leve giro hacia la izquierda, presumiblemente como respuesta anticipada a un posible impacto con el borde del circuito.



Speed: 71, Gear: 3, RPM: 4088, Prev Acts: [[0.72, 0.51], [0.35, 0.27], [0.23, -0.01], [0.18, -0.21]]

Figura 2.3: Ejemplo de observación estructurada. Se representa una secuencia de cuatro imágenes de 128 x 128 consecutivas en el tiempo, acompañada de cuatro acciones anteriores y variables auxiliares provenientes de la telemetría del vehículo.

2.3. Interfaz

Para establecer la conexión entre el agente y el entorno, se desarrolló una interfaz personalizada basada en la clase `RealTimeGymInterface` de la biblioteca *rtgym [1]*. Esta interfaz permite enviar acciones, recibir observaciones, calcular recompensas y reiniciar episodios, todo ello bajo un esquema de ejecución en tiempo real.

Dado que Assetto Corsa no dispone de una API oficial, fue necesario implementar un sistema de comunicación propio. El envío de acciones se realiza mediante la biblioteca *Virtual Gamepad* [26], que permite simular un joystick virtual desde Python. La captura de imágenes, por su parte, se realiza con la ayuda de la biblioteca *MSS* [27], que obtiene cuadros del entorno desde la ventana del simulador.

La adquisición de datos de telemetría es un proceso más complejo. Para ello, se desarrolló una aplicación auxiliar basada en el tutorial oficial de Assetto Corsa. Este viene incluido en el juego para asistir en la creación de aplicaciones personalizadas. En este caso, se utilizó el mecanismo de *shared memory* que expone internamente el simulador. La aplicación accede a múltiples variables del estado del vehículo y las transmite externamente mediante el protocolo UDP durante el transcurso de cada episodio. La Figura 2.4 ilustra la estructura general de la interfaz propuesta, que gestiona la comunicación bidireccional entre el simulador y el agente de aprendizaje.

En cuanto al diseño de la interfaz, se contemplaron mecanismos para:

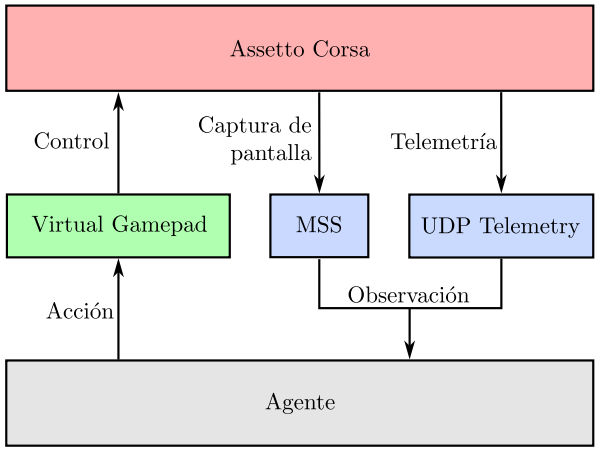


Figura 2.4: Interfaz de comunicación simulador-agente.

- Inicializar y sincronizar correctamente el entorno, incluyendo el reinicio de episodios y el uso de acciones por defecto.
- Capturar una secuencia de imágenes con historial configurable y opcionalmente reescaladas o convertidas a escala de grises.
- Sobrescribir acciones del control virtual en casos de sobrepaso del límite de velocidad.
- Leer datos de telemetría y combinarlos con las imágenes para formar la observación del agente.
- Calcular la recompensa y la señal de finalización a partir de los datos recibidos, utilizando una función configurable.
- Monitorear estadísticas del episodio y conservar registros útiles para depuración y análisis.

El espacio de observación devuelto por esta interfaz incluye, en cada paso, una secuencia de imágenes, la velocidad, el cambio y las revoluciones por minuto del vehículo. Por su parte, el espacio de acción está compuesto por dos valores continuos que controlan la aceleración/frenado y la dirección del volante.

Este diseño modular permite adaptar fácilmente la interfaz a nuevos escenarios y facilita el análisis del comportamiento del agente tanto en tiempo real como en registros históricos.

2.4. Modelo

El modelo de aprendizaje utilizado en este trabajo fue diseñado bajo una arquitectura modular, adaptada a los requerimientos del algoritmo *Soft Actor-Critic* (SAC). El enfoque se basa en una red convolucional 2D sencilla para el procesamiento de imágenes, combinada con un perceptrón multicapa (*MLP*) que integra tanto las características visuales extraídas como información adicional del entorno. Este diseño prioriza la eficiencia computacional y la simplicidad estructural, manteniendo suficiente capacidad expresiva para resolver la tarea de control autónomo en el entorno considerado. La estructura general del modelo se muestra en la Figura 2.5.

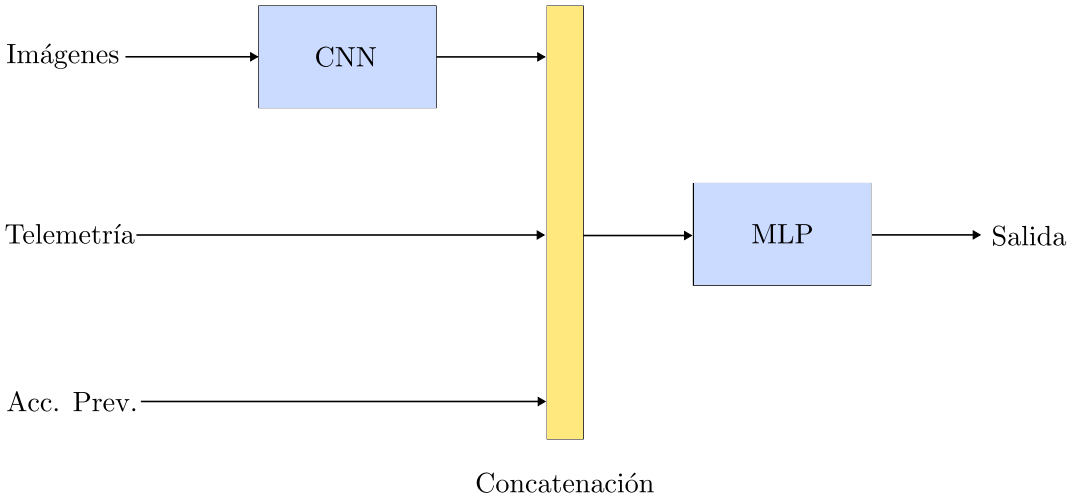


Figura 2.5: Arquitectura general del modelo de aprendizaje propuesto.

Diseño modular. El modelo está compuesto por los siguientes bloques funcionales:

- **CNN (Convolutional Neural Network).** Se define una red convolucional básica, compuesta por cuatro capas `Conv2D` con funciones de activación ReLU. Esta red recibe como entrada una secuencia de imágenes históricas apiladas en la dimensión de canales. Por ejemplo, si se utilizan n imágenes RGB, la entrada resulta en una imagen de $3n$ canales. Esta técnica permite capturar información temporal sin recurrir a arquitecturas recurrentes, reduciendo significativamente la complejidad del entrenamiento.

- **Integración de observaciones.** Las salidas de la CNN son aplanadas y concatenadas con el resto de los elementos de la observación: velocidad, cambio, revoluciones por

minuto (RPM) y el historial de acciones recientes. Esta concatenación se utiliza como entrada para un módulo `MLP`, encargado de proyectar esta representación conjunta hacia el espacio de acciones o hacia una estimación de valor, según se trate del actor o del crítico.

- **Actor estocástico.** El módulo actor sigue el diseño propuesto por SAC, produciendo tanto la media μ como la desviación estándar σ de una distribución normal multivariada. La acción es obtenida mediante una muestra de esta distribución, seguida de una transformación `tanh` que garantiza que las acciones permanezcan dentro de los límites definidos por el espacio de acción. Este esquema permite un balance controlado entre exploración y explotación, regulado por un coeficiente de entropía adaptativo.

- **Crítico doble.** Como es habitual en SAC, se emplean dos redes Q independientes para estimar el valor esperado de una acción a_t en un estado s_t. Ambas comparten la misma arquitectura (CNN + MLP), diferenciándose únicamente en sus pesos. Cada crítico recibe como entrada la observación y la acción propuesta por el actor, y devuelve una estimación escalar del retorno esperado. Este diseño, basado en el uso del mínimo entre ambas salidas, reduce el sesgo positivo en la estimación del valor y mejora la estabilidad del entrenamiento.

Exploración de variantes arquitecturales. Durante el desarrollo se implementaron y evaluaron otras arquitecturas, aunque no se emplearon en el experimento final debido a su mayor complejidad o rendimiento inferior durante las primeras fases de entrenamiento:

- **CNN preentrenadas.** Se adaptaron arquitecturas de visión profunda como `MobileNet` y `EfficientNet`, entrenadas en el conjunto ImageNet, reemplazando su primera capa convolucional para admitir múltiples canales de entrada. Si bien estas variantes son capaces de extraer características visuales más ricas, se observaron dificultades para alcanzar una convergencia temprana estable, además de una mayor exigencia en términos de recursos computacionales.

- **Modelos híbridos CNN + RNN.** También se exploraron arquitecturas que combinan una CNN para la extracción de características cuadro a cuadro y una red recurrente

(`GRU` o `LSTM`) para modelar relaciones temporales entre las imágenes. Estas variantes deberían capturar mejor la evolución dinámica del entorno, pero presentaron una alta sensibilidad a los hiperparámetros y requerimientos computacionales más elevados, sin beneficios evidentes dentro del tiempo de entrenamiento considerado.

- **Actor humano.** Se implementó una clase `HumanActor`, la cual permite controlar el entorno mediante un joystick (por ejemplo, un control de Xbox). Esta modalidad permite la recolección de datos de expertos humanos, útil como fuente de muestras de alta calidad para inicializar el entrenamiento.

2.5. Función de recompensa

La función de recompensa empleada en este trabajo fue diseñada específicamente para capturar múltiples aspectos del comportamiento del vehículo durante la conducción en *Assetto Corsa*. Su objetivo es fomentar el progreso eficiente y seguro a lo largo de la pista, penalizando conductas indeseadas como el estancamiento, las colisiones o el abandono del circuito.

Esta función toma como entrada los datos de telemetría del simulador y la acción aplicada por el agente en cada instante de tiempo. Su lógica se estructura en torno a tres componentes principales: condiciones de finalización del episodio, recompensas por comportamiento deseado y penalizaciones por errores.

Condiciones de terminación. Mecanismos para finalizar un episodio de manera anticipada:

- **Secuencia de errores:** si se acumulan errores en varios pasos consecutivos (como falta de progreso, retroceso o salida de pista), el episodio se termina, siempre que se haya alcanzado un mínimo de pasos para evitar truncamientos prematuros.
- **Daño excesivo:** si el daño máximo supera un umbral definido, el episodio se interrumpe. Este umbral fue seleccionado para tolerar roces menores, pero finalizar en caso de choques frontales o impactos severos.
- **Finalización exitosa:** al completar una vuelta válida en el circuito, se termina el episodio de forma exitosa.

Recompensas. Se otorgan recompensas positivas al agente en las siguientes situaciones:

- **Progreso en la pista:** se calcula el avance neto del vehículo mediante un buffer de posiciones históricas. Esto incentiva una conducción continua.
- **Cruce de puntos de control (checkpoints):** se recompensa el paso por tramos intermedios del circuito, ayudando a estructurar la tarea en etapas alcanzables.
- **Finalización de una vuelta:** se entrega una recompensa significativa al completar el recorrido.

Penalizaciones. Las siguientes conductas son castigadas con recompensas negativas:

- **Estancamiento o retroceso:** si el vehículo no progresa o se desplaza en sentido contrario, se penaliza proporcionalmente.
- **Velocidad insuficiente:** se penaliza linealmente en función de la distancia a un umbral mínimo de velocidad deseada.
- **Salida de pista:** la penalización es proporcional al número de neumáticos fuera del trazado.
- **Daños al vehículo:** se castigan los aumentos en las métricas de daño reportadas por el simulador.
- **Colisión continua:** se penaliza la persistencia de situaciones de *bloqueo lateral*, en las cuales la dirección del volante no se corresponde con la aceleración lateral (véase el apartado *Detección de colisiones*).

Detección de colisiones. Para identificar colisiones leves no detectadas directamente por los sensores de daño, se incorporó una lógica de *bloqueo lateral*. Esta consiste en detectar inconsistencias sostenidas entre la aceleración lateral del vehículo y la dirección del volante. Si esta condición persiste en una proporción significativa de los pasos recientes, y el vehículo se encuentra en movimiento, se interpreta como una colisión. Este mecanismo permite capturar situaciones donde el agente podría avanzar bordeando continuamente un muro, guiándose únicamente por la fuerza del impacto.

Normalización. Para evitar inestabilidades durante el aprendizaje, la recompensa total de cada paso es recortada al intervalo $[-1, 1]$. La única excepción es la recompensa por completar una vuelta, cuyo valor puede exceder dicho rango, al tratarse de una señal terminal positiva y poco frecuente.

Reinicialización. Al comienzo de cada episodio se restablecen los contadores de errores, los buffers internos de progreso y colisión, así como el historial de daño acumulado, garantizando una evaluación independiente de cada trayectoria.

Consideraciones sobre el diseño y *reward hacking*. El desarrollo de esta función implicó un proceso iterativo de ajuste fino, motivado por la aparición de comportamientos indeseados cuando ciertas señales de recompensa o penalización se sobredimensionaban. Algunos ejemplos observados incluyen:

- Penalizaciones excesivas por colisión provocaban que el agente optara por permanecer inmóvil para evitar la penalización.
- Penalizar fuertemente la baja velocidad generaba estrategias en las que el agente aceleraba al máximo sin preocuparse por mantener el control.
- Recompensas fijas por progreso neto incentivaban avanzar lentamente para maximizar la cantidad de pasos con recompensa.
- Recompensas altas por checkpoints aislados daban lugar a políticas que los alcanzaban intencionalmente y luego terminaban el episodio.
- En ausencia de penalización por colisión continua o daño, el agente aprendía a desplazarse impactando y rebotando en los muros.

Estos fenómenos ejemplifican lo que en la literatura se conoce como *reward hacking* o *specification gaming*: situaciones en las que el agente explota debilidades de la función de recompensa para maximizar su valor de forma no deseada. El diseño planteado busca no solo guiar el comportamiento del agente, sino también evitar trayectorias que resulten exitosas según la métrica, pero que sean ineficaces o artificiales en la práctica, tal como se explica con mayor detalle en Krakovna et al. [28].

3. Resultados

Los hiperparámetros utilizados en estos entrenamientos se detallan en el Anexo B.

Los episodios de prueba se ejecutan a razón de uno cada diez episodios de entrenamiento. Por ejemplo, 500 episodios de entrenamiento se corresponden con 50 episodios de prueba.

3.1. Resultados base

Se entrenó un agente en el circuito *Silverstone 1967*, una pista amplia y sencilla, sin bifurcaciones ni curvas cerradas pronunciadas. Este entorno fue escogido por su bajo nivel de complejidad visual y estructural, lo que permite aislar el efecto del aprendizaje bajo condiciones controladas (véase Figura 3.1).

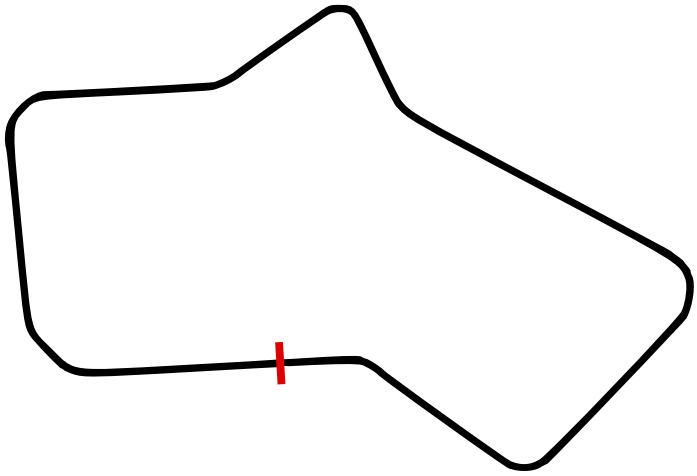


Figura 3.1: Vista cenital del circuito *Silverstone 1967*. La línea roja indica la meta. La pista se recorre en sentido horario.

El entrenamiento se inicia desde dos posiciones aleatorias dentro del circuito, seleccionadas de manera estocástica al comienzo de cada episodio. La probabilidad de inicio en cada punto es proporcional al desempeño promedio alcanzado al partir desde él, de modo que el agente dedica más tiempo de entrenamiento a las secciones que le resultan más difíciles. Este procedimiento busca favorecer la generalización del comportamiento aprendido y evitar el sobreajuste a una única zona de la pista.

Para analizar el proceso de aprendizaje, la Figura 3.2 presenta los resultados obtenidos durante los episodios de prueba, medidos de forma periódica a lo largo del entrenamiento. Se incluyen tres métricas principales: porcentaje de progreso alcanzado en la pista, longitud del episodio y recompensa acumulada por episodio.

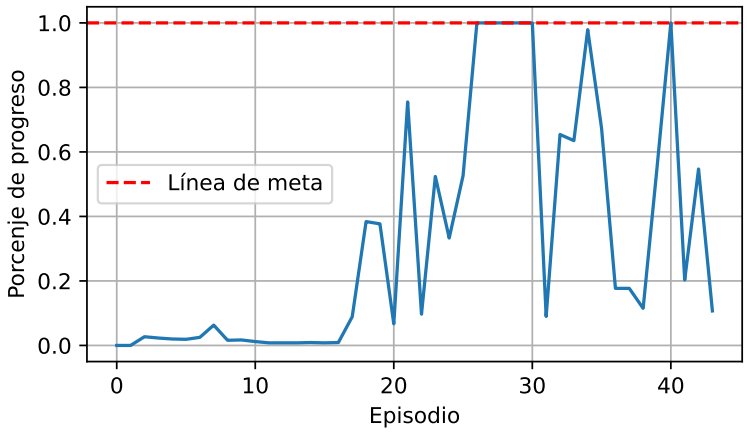
Tras completar la tarea, se continuó el entrenamiento del agente, lo cual condujo a un deterioro progresivo de su desempeño. Este fenómeno puede observarse en la Figura 3.3, que muestra la evolución de las pérdidas del actor y del crítico, así como las recompensas obtenidas en episodios de entrenamiento. Durante el proceso de aprendizaje efectivo, las pérdidas muestran estabilidad y tendencia a la convergencia. Sin embargo, después de un tiempo, el agente entra en una fase de divergencia caracterizada por un incremento sostenido de la pérdida del actor, acompañado por una disminución en la recompensa obtenida y la incapacidad de volver a completar la pista.

Cabe destacar una discrepancia entre la cantidad de episodios generados por los *RolloutWorkers* y aquellos efectivamente utilizados por el *Trainer*. En este experimento, cada trabajador generó más de 400 episodios, mientras que el entrenador procesó únicamente cerca de 230. Esta diferencia se debe al desfase inherente entre la recolección de muestras y la capacidad de entrenamiento del sistema, esperable en arquitecturas asincrónicas. Esta discrepancia explica por qué la Figura 3.3 presenta un número menor de episodios en comparación con la Figura 3.2.

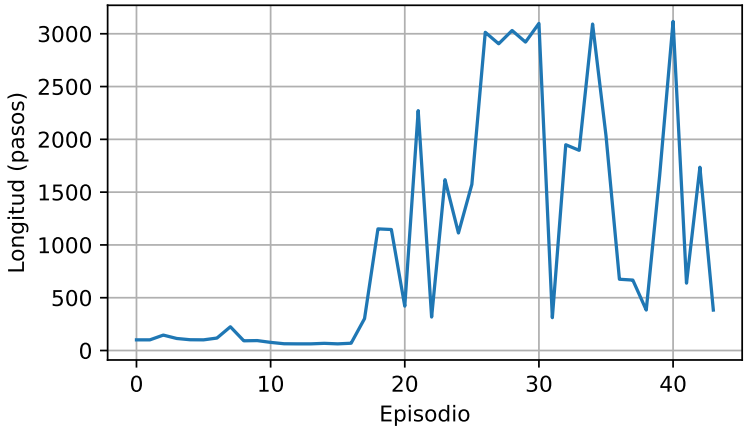
3.2. Asistencia con aprendizaje supervisado

Con el objetivo de evaluar el impacto del aprendizaje supervisado inicial sobre el rendimiento temprano del agente, se diseñó un experimento controlado en la pista *Highlands*. En esta prueba, se compararon dos configuraciones: una en la que uno de los dos trabajadores utiliza un módulo de actuación humana (*human actor*) durante el entrenamiento, y otra puramente basada en aprendizaje por refuerzo, donde ambos trabajadores exploran de forma autónoma desde el inicio. A excepción de esta diferencia, todos los parámetros del entorno, arquitectura y algoritmo se mantuvieron idénticos a los descritos en el experimento anterior.

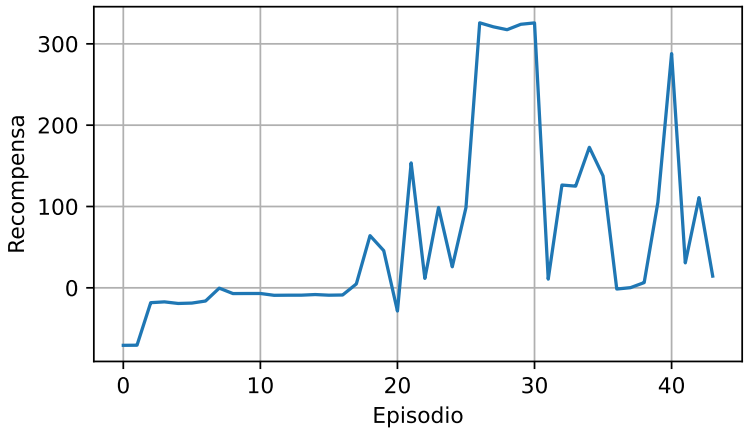
En esta pista se identifican dos secciones particularmente desafiantes al iniciar desde el *hotlap start* (véase Figura 3.4) :



(a) Progreso alcanzado por episodio.

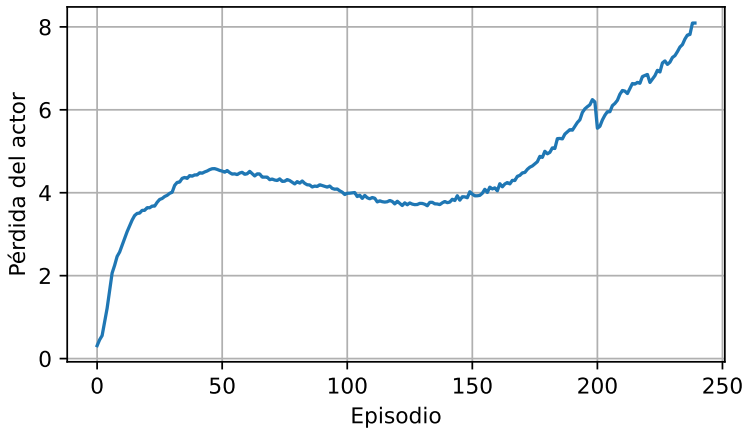


(b) Duración de episodios de prueba.

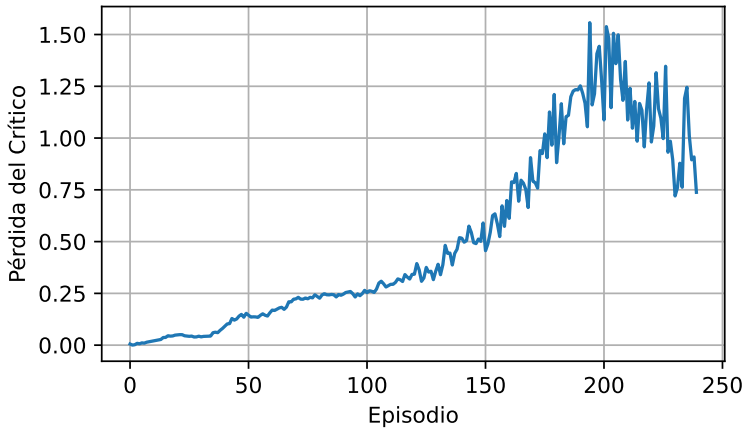


(c) Recompensa acumulada en episodios de prueba.

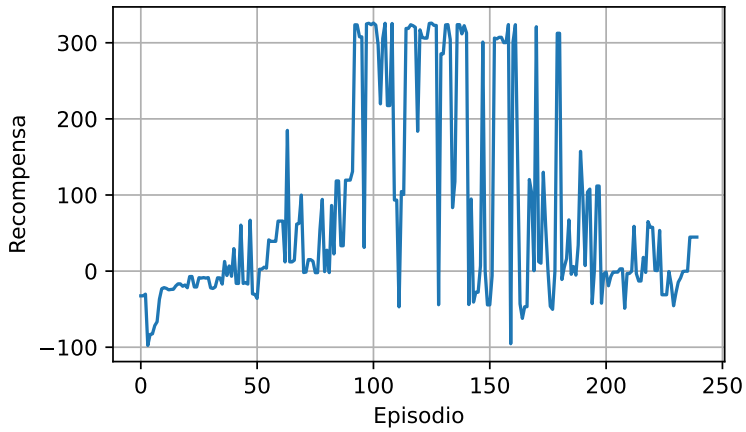
Figura 3.2: Desempeño del agente durante los episodios de prueba. Se observa que el agente logra completar satisfactoriamente la pista en el episodio 26, equivalente a aproximadamente dos horas de entrenamiento.



(a) Pérdida del actor.



(b) Pérdida del crítico.

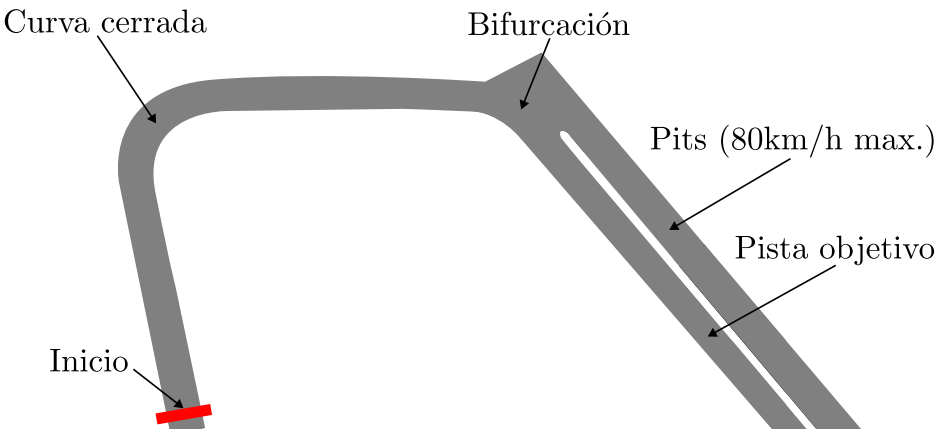


(c) Recompensas por episodio de entrenamiento (solo episodios efectivamente utilizados por el *Trainer*).

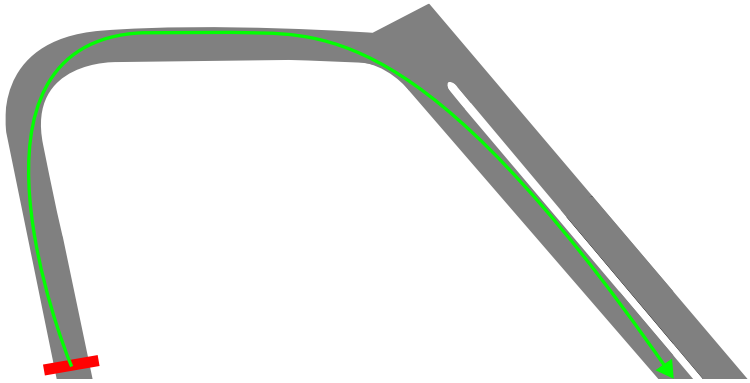
Figura 3.3: Indicadores del entrenamiento del agente. Se observa una fase de convergencia inicial, seguida de una degradación progresiva del comportamiento tras completar la tarea.

1. Una curva cerrada de menos de 90º, precedida por una recta prolongada, que exige una reducción anticipada de velocidad y un giro controlado para evitar salirse del trazado.

2. Una curva más amplia con bifurcación, en la cual una barrera central separa el camino principal de la entrada a los *pits*. El trayecto más fácil —seguir recto— conduce a los pits, una zona con límite de velocidad reducido, mientras que la salida hacia la derecha permite mantener una velocidad mayor, siendo esta última la ruta óptima.



(a) Secciones clave analizadas en los experimentos.

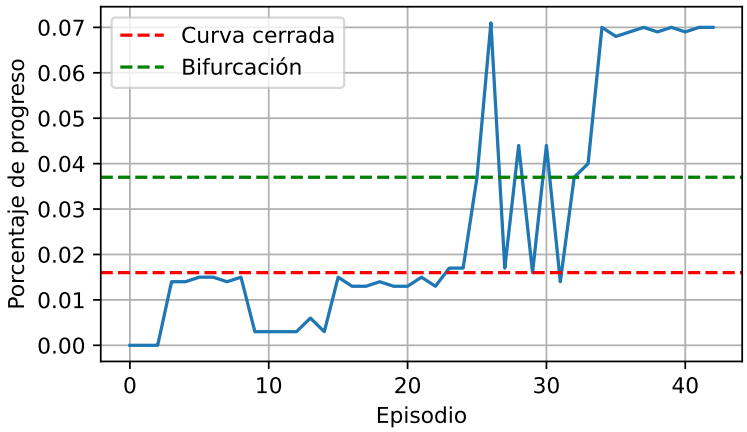


(b) Recorrido ideal para superar la curva cerrada y la bifurcación. La línea muestra el trayecto más eficiente, evitando la zona de los *pits*.

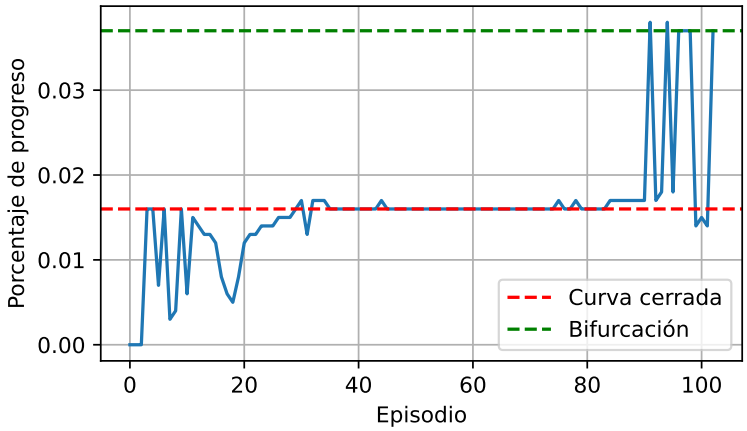
Figura 3.4: Vista cenital de la pista *Highlands*, utilizada para evaluar el impacto del aprendizaje supervisado.

Los resultados de la Figura 3.5 muestran que el agente asistido por el módulo humano logra resolver ambas curvas aproximadamente hacia el episodio 30, equivalente a cerca de una hora de entrenamiento desde cero. En contraste, el agente entrenado exclusivamente mediante

aprendizaje por refuerzo necesita alrededor de 90 episodios (cuatro horas de entrenamiento) para aprender a superar la curva cerrada, sin lograr resolver la segunda curva.



(a) Con asistencia supervisada (1 trabajador con *human actor*).



(b) Sin asistencia supervisada (ambos trabajadores en RL).

Figura 3.5: Porcentaje de pista completado por episodio de prueba en la pista *Highlands*, comparando el aprendizaje reforzado asistido con supervisión inicial frente al aprendizaje reforzado puro.

3.3. Análisis de Resultados

En el primer experimento se observó un comportamiento caracterizado por una fase inicial de mejora sostenida del desempeño, seguida de una degradación progresiva una vez alcanzado cierto nivel de rendimiento. En lugar de estabilizarse, como sería esperable en un escenario de convergencia, el agente comenzó a ejecutar trayectorias erráticas, explorando regiones del

entorno que previamente había aprendido a transitar correctamente. Este cambio de comportamiento estuvo acompañado por un incremento en la pérdida del actor y una disminución notable en la recompensa por episodio, hasta niveles cercanos a los del inicio del entrenamiento.

Se sugiere que este fenómeno podría estar relacionado con la interacción entre la presión exploratoria inducida por el término de entropía en SAC y la sensibilidad del algoritmo a sus hiperparámetros. En particular, la persistencia de una alta entropía objetivo, junto con tasas de aprendizaje y parámetros de actualización de las redes objetivo no optimizados, podría conducir a que el actor se desplace hacia regiones del espacio de acciones con estimaciones de valor poco precisas, afectando negativamente el rendimiento global [3, 29]. Asimismo, es posible que la naturaleza finita del *replay buffer* contribuya a la pérdida de experiencias clave, favoreciendo el sobreentrenamiento en muestras de menor calidad.

En cuanto al experimento realizado en la pista *Highlands*, los resultados confirman que la asistencia inicial mediante aprendizaje supervisado acelera de forma significativa la adquisición de comportamientos deseados. El agente que recibió demostraciones humanas logró superar las dos secciones críticas de la pista en un tiempo sustancialmente menor que aquel entrenado únicamente con aprendizaje por refuerzo. Además, la guía inicial introdujo patrones de comportamiento que, en ausencia de supervisión, podrían no emerger debido a que la política tiende a estabilizarse en soluciones subóptimas pero seguras, especialmente cuando los parámetros de entropía o exploración no fomentan una búsqueda suficientemente amplia del espacio de acciones.

4. Conclusiones

Este documento presenta el desarrollo de un entorno de experimentación que integra el simulador *Assetto Corsa* [2] con la arquitectura distribuida de entrenamiento *TMRL* [4], permitiendo la interacción en tiempo real y la recolección eficiente de experiencias para el entrenamiento de agentes de conducción autónoma. La relevancia principal radica en el uso de un simulador de alta fidelidad física y visual como banco de pruebas reproducible, capaz de adaptarse a diferentes configuraciones de observación, funciones de recompensa y estrategias de exploración, lo que facilita la validación y comparación de enfoques en conducción autónoma basada en visión.

Los experimentos realizados muestran que el uso de secuencias temporales de imágenes, complementadas con variables auxiliares de telemetría, dota al agente de un contexto dinámico suficiente para la toma de decisiones en escenarios de conducción. Asimismo, la incorporación de asistencia inicial mediante aprendizaje supervisado demostró ser una estrategia eficaz para acelerar la adquisición de comportamientos deseados, reduciendo de forma significativa el tiempo necesario para superar tramos de alta complejidad.

También se identificaron limitaciones inherentes al proceso de entrenamiento, tales como la degradación del rendimiento tras alcanzar el objetivo y la sensibilidad del algoritmo a sus hiperparámetros. Estos resultados resaltan la importancia de un diseño cuidadoso de la función de recompensa y de la configuración del proceso de aprendizaje, con el fin de evitar comportamientos no deseados y mantener la estabilidad de la política aprendida a lo largo del tiempo.

En conjunto, este trabajo constituye un aporte tanto metodológico como experimental para la investigación en conducción autónoma basada en visión, evidenciando que es posible obtener resultados satisfactorios en entornos de alta complejidad visual y dinámica realista, y dejando establecida una base sólida para futuras investigaciones en el área.

4.1. Trabajo futuro.

Los resultados obtenidos evidencian que, si bien el agente es capaz de aprender la tarea en las condiciones planteadas, la solución alcanzada no se mantiene estable a lo largo del entrena-

miento. Una línea de investigación inmediata corresponde a la optimización de hiperparámetros críticos, tales como el factor de descuento γ, las tasas de aprendizaje del actor y del crítico, el coeficiente de entropía α y la constante de actualización de las redes objetivo. Un ajuste sistemático de estos parámetros podría mitigar la divergencia observada tras la convergencia inicial y favorecer un desempeño más consistente.

Otra dirección relevante consiste en explorar arquitecturas más expresivas para el manejo de la información temporal. El uso de redes recurrentes (RNN, LSTM o GRU), arquitecturas basadas en Transformers [30], redes convolucionales tridimensionales [31] (3D-CNN) o el empleo de CNN preentrenadas más potentes permitiría modelar de manera más rica las dependencias entre secuencias de imágenes y extraer características visuales de mayor calidad, superando las limitaciones de la aproximación utilizada en este trabajo.

En escenarios que contemplen mayor complejidad visual, como la presencia de otros vehículos o condiciones variables de entorno, resulta prometedor incorporar técnicas avanzadas de visión por computador para el preprocesamiento de imágenes, por ejemplo, segmentación por instancias o detección de objetos en tiempo real (e.g., YOLO [32]). Esto facilitaría el aislamiento de elementos relevantes del entorno y podría mejorar la capacidad del agente para tomar decisiones seguras.

Finalmente, la integración de algoritmos recientes orientados a la eficiencia en el uso de muestras, como Randomized Ensembled Double Q-learning [33] (REDQ) o Data-Regularized Q [34] (DROQ), representa una oportunidad para acelerar el entrenamiento y alcanzar un mejor desempeño con menor cantidad de datos, lo que incrementaría la escalabilidad de este tipo de aproximaciones en entornos de mayor complejidad.

Referencias

[1] Y. Bouteiller, "rtgym: Real-time reinforcement learning environments for Gymnasium," [Online]. Available: https://github.com/yannbouteiller/rtgym, 2025, accessed: 3 May 2025.

[2] Kunos Simulazioni, "Assetto corsa," [Online]. Available: https://assettocorsa.gg/assetto-corsa/, accessed: 30 Aug. 2025.

[3] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018.

[4] E. Geze and Y. Bouteiller, "TMRL: TrackMania reinforcement learning," [Online]. Available: https://github.com/trackmania-rl/tmrl, 2025, accessed: 15 May 2025.

[5] SAE International, "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," SAE International, SAE Standard J3016_202104, 2021.

[6] K. D. Kusano, J. M. Scanlon, Y. H. Chen, T. L. McMurry, R. Chen, T. Gode, and T. Victor, "Comparison of Waymo rider-only crash data to human benchmarks at 7.1 million miles," *Traffic Injury Prevention*, vol. 25, no. sup1, pp. S66–S77, 2024.

[7] D. Pomerleau, "ALVINN: An autonomous land vehicle in a neural network," in *Proc. Neural Information Processing Systems (NeurIPS)*, D. S. Touretzky, Ed. Morgan Kaufmann, Dec. 1989, pp. 305–313.

[8] J. Zhao, W. Zhao, B. Deng, Z. Wang, F. Zhang, W. Zheng, W. Cao, J. Nan, Y. Lian, and A. F. Burke, "Autonomous driving system: A comprehensive survey," *Expert Systems with Applications*, vol. 242, p. 122836, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417423033389

[9] B. Duan, "Sensor and sensor fusion technology in autonomous vehicles," *Applied and Computational Engineering*, vol. 52, pp. 132–137, Mar. 2024.

[10] Z. Wang, J. Ma, and E. Lai, "A survey of scenario generation for automated vehicle testing and validation," *Future Internet*, vol. 16, no. 12, p. 480, Dec. 2024.

[11] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," *SAE International Journal of Transportation Safety*, vol. 4, no. 1, pp. 15–24, 2016.

[12] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proc. 1st Annual Conf. Robot Learning*, ser. Proceedings of Machine Learning Research, S. Levine, V. Vanhoucke, and K. Goldberg, Eds., vol. 78. PMLR, Nov. 2017, pp. 1–16. [Online]. Available: https://proceedings.mlr.press/v78/dosovitskiy17a.html

[13] B. Wymann, C. Espié, C. Guionneau, C. Dimitrakakis, R. Coulom, and A. Sumner, "TORCS, the open racing car simulator," [En línea]. Disponible en: https://sourceforge.net/projects/torcs/, 2000, consulta: 8 agosto 2025.

[14] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "AirSim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, ser. Springer Proceedings in Advanced Robotics, M. Hutter and R. Siegwart, Eds. Springer, Cham, 2018, vol. 5, pp. 621–635.

[15] Z. Zhu and H. Zhao, "A survey of deep RL and IL for autonomous driving policy learning," *IEEE Trans. Intell. Transport. Syst.*, vol. 23, no. 9, pp. 14 043–14 065, Sep. 2022. [Online]. Available: https://doi.org/10.1109/TITS.2021.3134702

[16] M. van Otterlo and M. Wiering, *Reinforcement Learning and Markov Decision Processes*. Berlin, Heidelberg: Springer, 2012, pp. 3–42. [Online]. Available: https://doi.org/10.1007/978-3-642-27645-3_1

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[18] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017.

[19] Z. Lai and T. Braunl, "End-to-end learning with memory models for complex autonomous driving tasks in indoor environments," *Journal of Intelligent Robotic Systems*, vol. 107, Mar. 2023.

[20] B. Peng, Q. Sun, S. Li, D. Kum, Y. Yin, J. Wei, and T. Gu, "End-to-end autonomous driving through dueling double deep Q-network," *Automotive Innovation*, vol. 4, Jun. 2021.

[21] H. Lu, "Integrating imitation learning with human driving data into reinforcement learning to improve training efficiency for autonomous driving," 2021. [Online]. Available: https://arxiv.org/abs/2111.11673

[22] J. Carreira and A. Zisserman, "Quo vadis, action recognition? A new model and the kinetics dataset," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 4724–4733.

[23] A. Dosovitskiy, P. Fischer, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 2758–2766, 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:12552176

[24] Y. Wang and J. E. Laird, "The importance of action history in decision making and reinforcement learning," *Ann Arbor*, vol. 1001, pp. 48 109–2121, 2007.

[25] B. Lee, V. Saj, M. Benedict, and D. Kalathil, "A deep reinforcement learning control strategy for vision-based ship landing of vertical flight aircraft," in *AIAA AVIATION 2021 FORUM*, 2021. [Online]. Available: https://par.nsf.gov/biblio/10318624

[26] Y. Bouteiller, "vgamepad: Virtual gamepad python package," [Online]. Available: https://pypi.org/project/vgamepad/, accessed: 30 Aug. 2025.

[27] M. Schoentgen, "mss: Python cross-platform screenshot module," [Online]. Available: https://pypi.org/project/mss/, accessed: 30 Aug. 2025.

[28] V. Krakovna, J. Uesato, V. Mikulik, M. Rahtz, T. Everitt, R. Kumar, Z. Kenton, J. Leike, and S. Legg, "Specification gaming: The flip side of AI ingenuity," *DeepMind Blog*, vol. 3, 2020.

[29] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, Jul. 2018, pp. 1587–1596. [Online]. Available: https://proceedings.mlr.press/v80/fujimoto18a.html

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[31] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 1, pp. 221–231, 2013.

[32] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788.

[33] X. Chen, C. Wang, Z. Zhou, and K. Ross, "Randomized ensembled double Q-learning: Learning fast without a model," in *Proc. 9th Int. Conf. Learning Representations (ICLR)*, May 2021. [Online]. Available: https://arxiv.org/abs/2101.05982

[34] T. Hiraoka, T. Imagawa, T. Hashimoto, T. Onishi, and Y. Tsuruoka, "Dropout Q-functions for doubly efficient reinforcement learning," *CoRR*, vol. abs/2110-02034, 2021. [Online]. Available: https://arxiv.org/abs/2110.02034

[35] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goul ao, A. Kallinteris, M. Krimmel, A. KG *et al.*, "Gymnasium: A standard interface for reinforcement learning environments," *arXiv preprint arXiv:2407.17032*, 2024.

[36] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. 35th Int. Conf. Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, Jul. 2018, pp. 1861–1870. [Online]. Available: https://proceedings.mlr.press/v80/haarnoja18b.html

[37] J. Achiam and OpenAI, "Spinning up in deep reinforcement learning," [Online]. Available: https://spinningup.openai.com/, 2018, accessed: 13 Sep. 2025.

[38] Ubisoft Nadeo, "Trackmania," [Online]. Available: https://www.trackmania.com/, accessed: 30 Aug. 2025.

Anexos

A. Descripción de Algoritmos y Herramientas utilizadas

A.1. Assetto Corsa

Assetto Corsa [2] es el simulador de conducción elegido para el entrenamiento del agente. Se trata de un videojuego de carreras lanzado en 2014 que se caracteriza por su estabilidad, física realista y fidelidad gráfica. El juego ofrece una amplia variedad de vehículos y circuitos, así como opciones avanzadas de personalización, que incluyen, por ejemplo, la condición de la pista, la presión de neumáticos, la suspensión, entre otros parámetros relevantes para la dinámica vehicular. Además, proporciona acceso a datos de telemetría en tiempo real provenientes de múltiples sensores del vehículo.

Una de las razones fundamentales para su elección es la disponibilidad de soporte para el desarrollo de aplicaciones externas mediante Python, lo cual facilita tanto la obtención de datos del entorno como la manipulación del mismo, incluyendo, por ejemplo, el reinicio automático de episodios durante el entrenamiento. El título ha alcanzado una madurez técnica que lo hace confiable para aplicaciones experimentales, al no presentar fallas críticas ni inestabilidad significativa.

El espacio de acción definido es continuo, en concordancia con la naturaleza física de los controles de un automóvil. La discretización de dichos controles implicaría una dinámica poco realista, semejante a un controlador por modulación por ancho de pulsos (PWM), lo cual no es deseable en este contexto. Se adoptó un espacio bidimensional: el primer componente regula la aceleración y el frenado, mientras que el segundo controla el ángulo de dirección del volante. Formalmente, el espacio de acciones se define como:

$$\mathbf{a}_t = \begin{bmatrix} a_t^{\text{long}} \\ a_t^{\text{lat}} \end{bmatrix} \quad \text{con} \quad a_t^{\text{long}} \in [-1, 1], \quad a_t^{\text{lat}} \in [-1, 1], \tag{A.1}$$

donde a_t^{long} representa el control longitudinal (positivo para acelerar, negativo para frenar) y a_t^{lat} corresponde al control lateral (ángulo de giro normalizado).

Las observaciones están compuestas por imágenes capturadas desde una perspectiva frontal en primera persona. Esta configuración maximiza la relevancia de la información visual, eliminando elementos constantes como partes visibles del vehículo que no aportan valor para la toma de decisiones. Además, se han desactivado todos los medidores gráficos disponibles en pantalla para evitar información redundante.

Complementariamente, se incluyen variables de telemetría que representan información cinemática del vehículo: velocidad, cambio y revoluciones por minuto (RPM). Aunque estos valores podrían inferirse indirectamente a partir de las imágenes mediante una red convolucional, su incorporación explícita permite reducir la complejidad del aprendizaje y aumentar la estabilidad del entrenamiento. En este trabajo, dichos datos se obtienen directamente desde la telemetría interna del simulador. La alternativa de extraer esta información visualmente a través de técnicas de reconocimiento óptico de caracteres (OCR) se descarta por motivos prácticos, y se considera una línea de trabajo futura.

Las observaciones en el instante t se representan como:

$$\mathbf{o}_t = \begin{bmatrix} v_t \\ g_t \\ \mathrm{rpm}_t \\ I_{t-K_I+1:t} \\ \mathbf{a}_{t-K_A:t-1} \end{bmatrix}, \tag{A.2}$$

donde v_t es la velocidad, g_t el cambio, rpm_t las revoluciones por minuto, $I_{t-K_I+1:t}$ es la secuencia de K_I imágenes recientes y $\mathbf{a}_{t-K_A:t-1}$ el historial de K_A acciones inmediatamente anteriores.

A.2. Real-Time Gym

El uso de entornos en tiempo real no diseñados originalmente para aprendizaje por refuerzo plantea desafíos significativos. Se requiere una sincronización precisa entre las decisiones del agente y el entorno físico o simulado. Este problema es común en aplicaciones como la robótica y los videojuegos, donde la ejecución del entorno debe respetar una cadencia temporal fija.

Real-Time Gym (*rtgym*) [1] se desarrolló con el fin de abordar esta problemática. Se trata de un marco construido sobre Gymnasium [35], que permite implementar entornos con restricciones temporales de forma transparente para el usuario. Su principal objetivo consiste en sincronizar la interacción entre el agente y el entorno mediante un *elastic clock* que regula el envío de acciones y la obtención de observaciones dentro de una frecuencia deseada.

rtgym proporciona una interfaz unificada que el usuario debe extender mediante la clase abstracta `RealTimeGymInterface`. Esta interfaz exige la implementación de los siguientes métodos fundamentales:

- `get_observation_space()`
- `get_action_space()`
- `get_default_action()`
- `reset()`
- `send_control()`
- `get_obs_rew_terminated_info()`

Además, pueden redefinirse opcionalmente los métodos `wait()` y `render()` para adaptar el comportamiento del entorno a necesidades particulares.

Una vez definida la interfaz, el entorno puede instanciarse mediante un diccionario de configuración predefinido, cuyos parámetros pueden modificarse según los requerimientos de la aplicación. Dicho entorno resulta compatible con la mayoría de los algoritmos de aprendizaje por refuerzo implementados sobre Gymnasium.

La Figura A.1 ilustra el comportamiento temporal de *rtgym*, en el cual los pasos de tiempo se ajustan elásticamente a una duración nominal. Si esta duración se ve excedida, ya sea por una inferencia prolongada o por demoras en la captura de observaciones, se produce un *timeout* y el siguiente paso comienza desde el instante actual.

Respecto al rendimiento, la precisión temporal alcanzable con *rtgym* depende del sistema operativo subyacente. En Windows, la granularidad del sistema limita los pasos de tiempo a

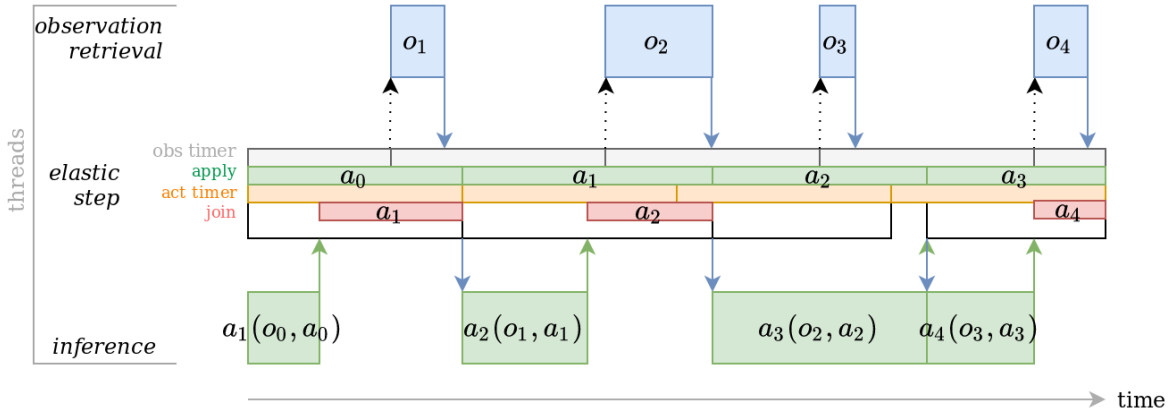


Figura A.1: Funcionamiento del entorno de `rtgym`. Figura adaptada de Yann Bouteiller [1], modificada bajo licencia MIT.

frecuencias del orden de 50 Hz, mientras que en Linux se pueden alcanzar los 500 Hz.

A.3. Soft Actor-Critic (SAC)

Soft Actor-Critic (SAC) [36] es el algoritmo de aprendizaje por refuerzo profundo seleccionado para el entrenamiento del agente. Esta elección se fundamenta en su idoneidad para entornos con espacios de acción continuos, su eficiencia en el uso de muestras y su estabilidad durante el entrenamiento. SAC pertenece a la familia de algoritmos *actor-critic* y se clasifica como un método *off-policy*, lo que permite reutilizar experiencias pasadas mediante el empleo de una memoria de repetición (*replay buffer*).

El enfoque *actor-critic* combina dos componentes fundamentales: una política o *actor*, encargada de generar acciones a partir de observaciones, y dos funciones de valor o *críticos*, que estiman el retorno esperado de una acción en un determinado estado. La incorporación de dos críticos independientes permite mitigar el sesgo en la estimación de valores al utilizar el mínimo de ambas salidas. Durante el entrenamiento, el actor es optimizado indirectamente mediante las señales proporcionadas por los críticos, mientras que en la fase de inferencia solo se requiere el actor para la generación de acciones en tiempo real.

En este trabajo se utilizó la implementación de SAC provista por la biblioteca `SpinUp` [37], la cual fue adaptada para permitir cómputo en precisión mixta y compatibilidad con la arquitectura distribuida de TMRL. Estas modificaciones fueron necesarias para integrar el algoritmo

en el flujo de entrenamiento en tiempo real sobre el simulador *Assetto Corsa*.

Finalmente, la naturaleza *off-policy* de SAC habilita el desacoplamiento entre la recolección de experiencias y la optimización del modelo. Esta propiedad hace posible el entrenamiento distribuido con múltiples trabajadores en paralelo, incrementando la eficiencia sin comprometer la coherencia del aprendizaje, lo que resulta particularmente ventajoso en entornos de simulación no diseñados originalmente para aprendizaje por refuerzo.

A.4. TMRL

Como base para la implementación de la arquitectura de entrenamiento, se utilizó el framework TMRL (TrackMania Reinforcement Learning) [4], una biblioteca en Python diseñada para entrenar agentes de aprendizaje por refuerzo profundo en entornos con restricciones de tiempo real, tales como robots o simuladores de conducción. TMRL cuenta con una arquitectura distribuida cliente-servidor que permite desacoplar la recolección de datos (realizada por los *rollout workers*) del proceso de entrenamiento (efectuado en el *trainer*), facilitando la ejecución eficiente del aprendizaje incluso en entornos donde la velocidad de simulación no puede acelerarse arbitrariamente.

Este framework fue originalmente desarrollado para entrenar agentes autónomos en el videojuego TrackMania 2020 [38], utilizando entradas visuales en tiempo real (capturas de pantalla) y actuando mediante emulación de un control análogo. Para ello, proporciona entornos compatibles con Gymnasium (basados en *rtgym*) y ofrece implementaciones listas para usar de algoritmos como Soft Actor-Critic [3] (SAC) y REDQ-SAC [33].

En el presente trabajo, se reutilizó la estructura de TMRL adaptándola para funcionar sobre el simulador Assetto Corsa, lo cual implicó la modificación de los módulos de captura de observaciones y emisión de acciones, así como el diseño de un entorno personalizado que respetara las mismas limitaciones de tiempo real impuestas por *rtgym*. Se conservaron los componentes de comunicación entre procesos, el servidor central de políticas y el sistema de entrenamiento asincrónico, lo que permitió conservar la eficiencia del ciclo de interacción sin comprometer la modularidad del sistema.

B. Configuración de entrenamiento

Con el fin de centralizar el control de los experimentos y facilitar su ajuste, todos los parámetros relevantes fueron definidos en un único archivo de configuración en formato `JSON`. Este archivo permite modificar desde un solo punto los valores empleados en el entrenamiento, asegurando consistencia entre ejecuciones y simplificando la replicación de resultados. A continuación, se presentan las configuraciones utilizadas en los experimentos, organizadas por componente del sistema.

Tabla B.1: Parámetros del entorno y visualización

Parámetro	**Valor**
Duración del paso de simulación	0.05 s
Resolución de las imágenes	128×128 píxeles
Imágenes en escala de grises	No
Historial de imágenes por observación	4
Historial de acciones por observación	4
Velocidad máxima permitida	120 km/h
Inicio en distintos puntos de la pista	Sí
Reinicio con retardo	1.5 s
Longitud máxima del episodio	10000 pasos

Tabla B.2: Parámetros de la función de recompensa

Parámetro	**Valor**
Recompensa por checkpoint	1.0
Recompensa por progreso relativo	0.05
Recompensa por completar vuelta	100.0
Penalización por no avanzar	-0.5
Penalización por baja velocidad	-0.2
Penalización por retroceder	-0.5
Penalización por salirse de pista (por rueda)	-0.1
Penalización por daño al vehículo	-1.0
Penalización por colisión continua	-0.1
Umbral velocidad mínima	40 km/h
Umbral checkpoint (porcentaje de la pista)	1 %
Daño máximo permitido	25*
Errores máximos permitidos	40 pasos
Pasos para olvidar errores	20 pasos
Pasos mínimos antes de fallo	60 pasos

* Unidades internas del simulador. Equivale una colisión con bajo ángulo de incidencia.

Tabla B.3: Parámetros arquitectónicos del modelo

Componente	**Valor**
Activación	ReLU
Capas convolucionales	4
Uso de pesos preentrenados	No
Capas ocultas del MLP	$(256, 256)$
Historial de acciones considerado	4

Tabla B.4: Hiperparámetros del algoritmo de entrenamiento

Parámetro	**Valor**
Algoritmo utilizado	Soft Actor-Critic (SAC)
Precisión mixta activada	Sí
Aprendizaje de coef. de entropía	Sí
Coeficiente inicial de entropía α	0.01
Entropía objetivo	-1
Tasa de descuento γ	0.995
Parámetro `POLYAK`	0.995
Optimizador actor / crítico	Adam
Tasas de aprendizaje actor / crítico / entropía	10^{-5} / 5×10^{-5} / 3×10^{-5}
Tamaño de batch	256
Tamaño del buffer de memoria	$500,000$ transiciones
Pasos antes de entrenar	1000
Frecuencia de actualización del modelo	cada 200 pasos
Episodios de prueba	cada 10 pasos

C. Repositorio de código

El código fuente desarrollado para este trabajo se encuentra disponible en el siguiente repositorio:

`https://github.com/SamuelOrtizV/Assetto-Corsa-RL`

Todo el código se distribuye bajo licencia MIT, y puede ser utilizado con fines académicos y de investigación.