Universidad Zaragoza
1542

Trabajo Fin de Grado

Entrenamiento y Validación de un Agente de Conducción Autónoma en un Entorno Simulado

Training and Validation of an Autonomous Driving Agent in a Simulated Environment

Autor

Juan Valle Morenilla

Directores

Eduardo Montijano Muñoz
Rubén Martínez Cantín

Escuela de Ingeniería y Arquitectura

2025

Resumen

El objetivo de este Trabajo Final de Grado (TFG) es conseguir generar un modelo para un vehículo autónomo que sea capaz de completar un circuito de la manera más correcta y eficaz posible. Esto se ha logrado utilizando algoritmos de Aprendizaje por Refuerzo (RL) que provee la librería Stable Baselines3 para Python así como Gymnasium para crear un entorno válido a utilizar por el algoritmo de RL y otras que aportan utilidad para cálculos como pueden ser Numpy u OpenCv.

Así se ha abordado el problema mediante la elaboración de diversos entrenamientos modificando la función de recompensa del algoritmo de RL y probando los resultados obtenidos contrastándolos con las métricas que nos aporta Stable Baselines3 interpretables mediante TensorBoard.

Cabe destacar que toda la plataforma desarrollada en el proyecto ha sido implementada íntegramente por el autor, tomando como base referencias y fragmentos de código disponibles públicamente en distintas fuentes online, los cuales han sido adaptados y extendidos para cumplir con los objetivos específicos del trabajo.

Abstract

The objective of this Final Degree Project (TFG) is to generate a model for an autonomous vehicle that is capable of completing a circuit as accurately and efficiently as possible. This was achieved using Reinforcement Learning (RL) algorithms provided by the Stable Baselines3 library for Python, as well as Gymnasium to create a valid environment for the RL algorithm, and other libraries that provide useful computational tools such as Numpy or OpenCv.

The problem was addressed by creating various training sessions, modifying the reward function of the RL algorithm, and testing the results against the metrics provided by Stable Baselines3, which can be interpreted using TensorBoard.

It should be noted that the entire platform developed for this project was implemented entirely by the author, based on references and code fragments publicly available from various online sources, which have been adapted and extended to meet the specific objectives of the project.

Agradecimientos

Quisiera expresar mi más sincero agradecimiento a mi familia, por su apoyo incondicional durante todo este proceso. Su confianza, paciencia y ánimo constante han sido fundamentales para que pudiera llevar a cabo este trabajo.

También quiero agradecer a mis profesores, y en especial a mis tutores de TFG, por su dedicación, orientación y por compartir sus conocimientos a lo largo de mi formación. Sus guías han sido clave para enfocar correctamente este proyecto y superar los distintos retos que han surgido durante su desarrollo.

A todos ellos, gracias por acompañarme en este camino.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Introducción general

En los últimos años, la conducción autónoma ha dejado de ser un concepto de ciencia ficción para convertirse en una realidad tecnológica en rápido desarrollo. Grandes empresas del sector del automóvil y la tecnología, como **Tesla**, **Waymo** o **NVIDIA**, han invertido enormes recursos en sistemas capaces de permitir que un vehículo se desplace sin intervención humana. Las noticias sobre coches que circulan solos por las calles de ciudades como **San Francisco**, **Phoenix** o **Roma** han captado la atención del público y alimentado un debate global sobre el futuro del transporte.

Un ejemplo reciente y especialmente ilustrativo es el caso de **Tesla**, que ha comenzado a probar su sistema de conducción autónoma en entornos reales altamente desafiantes, como el centro histórico de **Roma**. Según un reportaje publicado en junio de 2025 (El Economista, 2025), el vehículo fue capaz de circular sin intervención humana por calles estrechas, con tráfico irregular, motocicletas impredecibles y peatones cruzando sin previo aviso. Este tipo de escenarios urbanos, caóticos y sin reglas estrictamente definidas, representa uno de los mayores desafíos para cualquier sistema de conducción autónoma. Que este sistema sea capaz de navegar en tales condiciones refuerza la idea de que estamos cada vez más cerca de una autonomía total en la conducción, pero también pone de relieve la necesidad de modelos de comportamiento extremadamente robustos y adaptativos.

Pese a estos avances, la conducción autónoma sigue siendo uno de los mayores retos tecnológicos de nuestro tiempo. Lograr que un vehículo sea capaz de moverse de forma segura y eficiente en entornos complejos requiere que "vea" su entorno, entienda lo que sucede, tome decisiones rápidas y correctas, y actúe con precisión. Para ello, es necesario combinar áreas como la inteligencia artificial, la visión por computador, la robótica, y la simulación.

Actualmente, uno de los enfoques más prometedores para entrenar vehículos autónomos es el **Aprendizaje por Refuerzo** (Reinforcement Learning, RL)[1]. Este método permite que un sistema aprenda mediante la experiencia: igual que un ser humano o un animal, el agente prueba distintas acciones, observa sus consecuencias, y ajusta su comportamiento en función de recompensas o penalizaciones que recibe.

En este contexto, existen plataformas comerciales que permiten experimentar con RL aplicado a la conducción autónoma. Una de las más conocidas es **AWS DeepRacer**[2], una iniciativa de **Amazon** que proporciona un entorno virtual y físico para entrenar coches autónomos en la nube. Aunque ofrece una introducción accesible a estos conceptos, presenta limitaciones como el uso exclusivo de servicios en la nube, el alto coste económico y la baja capacidad de personalización.

Frente a esto, surge la oportunidad de construir una alternativa completamente local y abierta, que permita no sólo replicar, sino también mejorar algunos aspectos de estas plataformas comerciales. Aquí es donde se enmarca el presente **Trabajo de Fin de Grado** (TFG), que tiene como objetivo desarrollar desde cero un sistema de conducción autónoma simulado basado en aprendizaje por refuerzo, utilizando únicamente herramientas de código abierto y ejecutándose de forma local.

En este proyecto, un coche virtual aprende a conducir por sí mismo dentro de un entorno simulado. A través de una cámara virtual, el vehículo percibe el entorno como si estuviera en el mundo real, y debe decidir cómo actuar en cada momento sin recibir instrucciones directas. El modelo se entrena repitiendo acciones, observando el resultado, y ajustando su comportamiento en función de una función de recompensa definida.

Más allá de construir un sistema funcional, este trabajo busca entender en profundidad cómo funcionan estos sistemas, enfrentarse a sus principales dificultades y sentar las bases para futuras extensiones, como el uso de sensores más complejos o la transferencia a vehículos reales.

1.2. Motivación y contexto

La conducción autónoma representa uno de los desafíos más interesantes y completos que pueden abordarse desde la inteligencia artificial y la robótica. Su resolución exige una comprensión profunda de algoritmos, estructuras de software, modelado físico y estrategias de aprendizaje, lo que convierte este tipo de proyecto en una excelente oportunidad para consolidar conocimientos y desarrollar nuevas habilidades. Un ejemplo de un sistema real de conducción autónoma puede verse en la **Figura 1.1**.

Figura 1.1: La imagen muestra un ejemplo de vehículo real equipado con sensores para navegación autónoma, como cámaras, LiDAR y radares. Este tipo de hardware permite al coche percibir su entorno en tiempo real.

Además, el Aprendizaje por Refuerzo (Reinforcement Learning) se ha posicionado como una de las técnicas más prometedoras para abordar tareas donde la solución óptima no se puede programar explícitamente, sino que debe ser descubierta mediante la interacción del agente con su entorno. La idea de que un sistema pueda aprender de su experiencia, mejorar con el tiempo y adaptarse a nuevas situaciones resulta muy interesante desde el punto de vista científico y ofrece un enorme potencial de aplicación en robótica, videojuegos, automatización industrial y más.

Uno de los referentes más conocidos en este campo es la plataforma AWS DeepRacer, que ofrece un entorno virtual para entrenar coches autónomos mediante RL. Sin embargo, tras explorar su funcionamiento, se identificaron una serie de limitaciones importantes: dependencia obligatoria de servicios en la nube, escasa capacidad de personalización, uso de código cerrado y un coste económico considerable para el entrenamiento y despliegue de modelos. Estas barreras hacen que el acceso a la experimentación avanzada quede restringido a usuarios con recursos económicos o institucionales elevados.

Surge así la motivación por diseñar y construir una alternativa completamente local, accesible y abierta. El objetivo es replicar (y en algunos aspectos superar) el enfoque de plataformas comerciales, pero utilizando exclusivamente herramientas de código abierto como `ROS`, `Gazebo`, `Gymnasium` y `Stable Baselines3`. Esto no solo elimina costes, sino que abre la puerta a una personalización total del entorno, la arquitectura del agente y la función de recompensa, lo cual resulta esencial en un proyecto académico orientado al aprendizaje y la investigación.

Desde una perspectiva formativa, el proyecto también tiene una fuerte carga motivacional: implica aprender a configurar entornos simulados complejos, modelar vehículos, integrar sensores virtuales, aplicar algoritmos de RL y analizar los resultados para mejorar el sistema iterativamente. La implementación completa del sistema desde cero obliga a enfrentarse con numerosos retos técnicos, lo que convierte el proyecto en una experiencia intensa pero enriquecedora de aprendizaje autónomo y resolución de problemas reales.

Finalmente, el enfoque adoptado también responde a una visión a largo plazo: crear una base sólida sobre la cual se puedan desarrollar futuros trabajos, ya sea extendiendo la plataforma para trabajar con vehículos reales, explorando otros algoritmos de aprendizaje o introduciendo nuevos sensores y escenarios. Este TFG pretende así ser más que un trabajo puntual: una contribución inicial a una línea de trabajo abierta, accesible y con amplio margen de desarrollo.

1.3. Objetivos y alcance

Objetivos

El principal objetivo de este proyecto es desarrollar un modelo de aprendizaje por refuerzo capaz de controlar un vehículo simulado de forma autónoma, aprendiendo a recorrer un circuito con un trazado determinado. Para ello, se incluye también como parte fundamental del trabajo la construcción del entorno simulado, utilizando herramientas ampliamente adoptadas en el desarrollo de sistemas robóticos, como ROS y Gazebo. Más allá de lograr buenos resultados en ese entorno específico, se busca que el modelo adquiera cierta capacidad de generalización, permitiendo su transferencia a circuitos similares con un rendimiento comparable sin necesidad de reentrenamiento desde cero. Además, el proyecto tiene como objetivo complementar la formación técnica mediante el análisis y aplicación de técnicas avanzadas de aprendizaje por refuerzo, prestando especial atención a la experimentación con distintas configuraciones de entornos, funciones de recompensa y algoritmos de entrenamiento.

Alcance

El alcance del trabajo se ha delimitado a la implementación completa de una plataforma de conducción autónoma simulada, utilizando exclusivamente recursos locales y herramientas de código abierto. Esto incluye:

- La configuración del entorno de simulación con Gazebo.
- El desarrollo de un modelo de vehículo personalizado y sus sensores.
- La integración de este entorno con un wrapper compatible con Gymnasium.
- El diseño de una función de recompensa adecuada para guiar el comportamiento del agente.
- El entrenamiento del modelo mediante Aprendizaje por Refuerzo y su posterior evaluación en distintas condiciones.

El sistema ha sido desarrollado íntegramente desde cero, utilizando únicamente librerías de propósito general. Aunque la implementación física en un vehículo real queda

fuera del alcance de este TFG, se plantea como una posible línea futura. Este trabajo se enmarca en un proyecto más ambicioso financiado por la **Cátedra Mobility City**, en colaboración con la **Universidad de Zaragoza**, orientado a fomentar la innovación en el ámbito de la movilidad inteligente. La **Figura 1.2** muestra un esquema general del sistema desarrollado, con sus principales componentes.

Figura 1.2: Este esquema ilustra los componentes principales del sistema desarrollado, incluyendo el entorno de simulación (Gazebo), el entorno (Gymnasium) y el agente de RL (Stable Baselines3).

1.4. Planificación temporal y estructura del documento

Cronograma

A lo largo del desarrollo del proyecto, se ha seguido una planificación estructurada por fases, que ha permitido abordar de forma progresiva y ordenada cada una de las etapas necesarias para alcanzar los objetivos. La **Figura 1.3** resume las fases principales del proyecto y el periodo estimado de trabajo para cada una de ellas, desde la fase inicial de análisis hasta la evaluación de resultados y redacción final.

	oct 24	nov 25	dic 24	ene 25	feb 25	mar 25	abr 25	ma 25	jun 25
Análisis de AWS DeepRacer y definición del entorno local									
Estudio inicial sobre RL y conducción autónoma									
Configuración de Gazebo y ROS									
Desarrollo del entorno Gym personalizado									
Implementación de la función de recompensa									
Entrenamiento del modelo y ajustes de parámetros									
Evaluación de resultados y redacción del documento									

Figura 1.3: Las fases se organizan en función del tiempo utilizado para cada etapa, facilitando el seguimiento y control del progreso.

Estructura del documento

La memoria se estructura en cuatro capítulos principales. En el **Capítulo 2**, se describe de forma técnica el problema a resolver, detallando el comportamiento esperado del sistema y las restricciones que condicionan la solución. El **Capítulo 3** presenta el marco teórico necesario para comprender el desarrollo del proyecto, incluyendo los fundamentos del Aprendizaje por Refuerzo, la simulación en robótica y las tecnologías utilizadas, como Gazebo, ROS y el algoritmo PPO. En el **Capítulo 4**, se expone la arquitectura del sistema desarrollado, abarcando desde la creación del entorno simulado hasta la integración de los distintos módulos, incluyendo la definición de la función de recompensa y la lógica de entrenamiento. Finalmente, el **Capítulo 5** recoge los resultados obtenidos tras el proceso de entrenamiento, analizando el rendimiento del modelo en distintos escenarios de prueba y comentando posibles mejoras o extensiones futuras.

Capítulo 2

Planteamiento del problema y fundamentos teóricos

2.1. Descripción del problema

El presente proyecto se enmarca en el desarrollo de un sistema de conducción autónoma en un entorno simulado, donde un vehículo debe recorrer un circuito cerrado de forma eficiente y segura, sin intervención humana directa. El sistema se basa exclusivamente en la percepción visual obtenida a través de una cámara frontal montada sobre el vehículo, lo que implica que toda la toma de decisiones debe derivarse de la información contenida en las imágenes.

El escenario de trabajo consiste en una pista o circuito previamente definido, un vehículo con capacidad de movimiento en un plano bidimensional y un entorno simulado que permite controlar y monitorizar todas las interacciones. El vehículo opera en un bucle cerrado de control, en el que recibe observaciones del entorno (en forma de imágenes), procesa esa información y determina la acción más adecuada (como acelerar, frenar o girar) y la ejecuta en el entorno. Este ciclo (representado en la **Figura 2.1**) se repite de forma continua durante el entrenamiento y la evaluación del sistema.

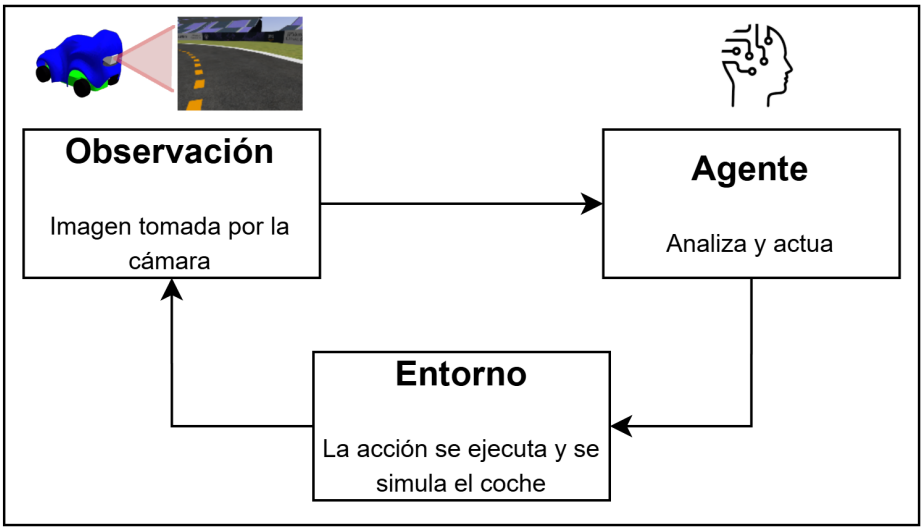


Figura 2.1: Este diagrama muestra el ciclo cerrado de control del vehículo autónomo. El agente recibe una imagen del entorno, procesa esta información para decidir una acción y la ejecuta, lo que genera una nueva observación que reinicia el ciclo.

Desde una perspectiva de inteligencia artificial, este problema se aborda mediante técnicas de Aprendizaje por Refuerzo. En este paradigma, un agente aprende a tomar decisiones óptimas a partir de la interacción repetida con un entorno, guiado por una señal de recompensa que evalúa la calidad de sus acciones. En lugar de proporcionarle instrucciones explícitas, se deja que el agente explore, falle, y gradualmente aprenda a comportarse de forma deseable a partir de la retroalimentación que recibe.

En este caso, el agente no tiene acceso a mapas, coordenadas ni información estructurada sobre el circuito. Su única fuente de información es la imagen capturada por la cámara, lo que convierte el problema en un desafío de percepción y control basado en visión. A partir de esta entrada visual, el agente debe inferir en tiempo real cómo moverse por el circuito sin salirse del trazado, evitando colisiones y manteniendo un avance constante.

Un aspecto clave del problema es la necesidad de generalización. Aunque el entrenamiento se realiza en un circuito concreto, el objetivo no es que el vehículo memorice una secuencia fija de acciones para ese trazado, sino que desarrolle un comportamiento robusto que pueda transferirse a circuitos similares no vistos. Esto implica que el agente debe aprender patrones generales de navegación basados en la percepción visual, en lugar de estrategias específicas dependientes del entorno de entrenamiento.

La complejidad del problema también radica en su naturaleza continua y en tiempo real: las acciones del agente deben calcularse rápidamente, y las decisiones deben adaptarse dinámicamente a la escena percibida en cada instante. Además, el diseño de la función de recompensa, que guía el aprendizaje del agente, debe ser lo suficientemente informativa

como para promover un comportamiento eficiente, pero también general para evitar el sobreajuste a situaciones específicas.

2.2. Aprendizaje por Refuerzo (Reinforcement Learning)

El Aprendizaje por Refuerzo [1] es una rama del aprendizaje automático en la que un agente aprende a interactuar con un entorno mediante la experimentación y la retroalimentación en forma de recompensas. A diferencia del aprendizaje supervisado, en el RL no se proporcionan pares de entrada y salida deseada, sino que el agente debe descubrir qué acciones maximizan su recompensa acumulada a lo largo del tiempo. La **Figura 2.2** muestra de forma esquemática el funcionamiento básico de la recompensa, donde se premia hacer las cosas bien y se castiga hacerlas mal, en este caso meter o no la pelota.

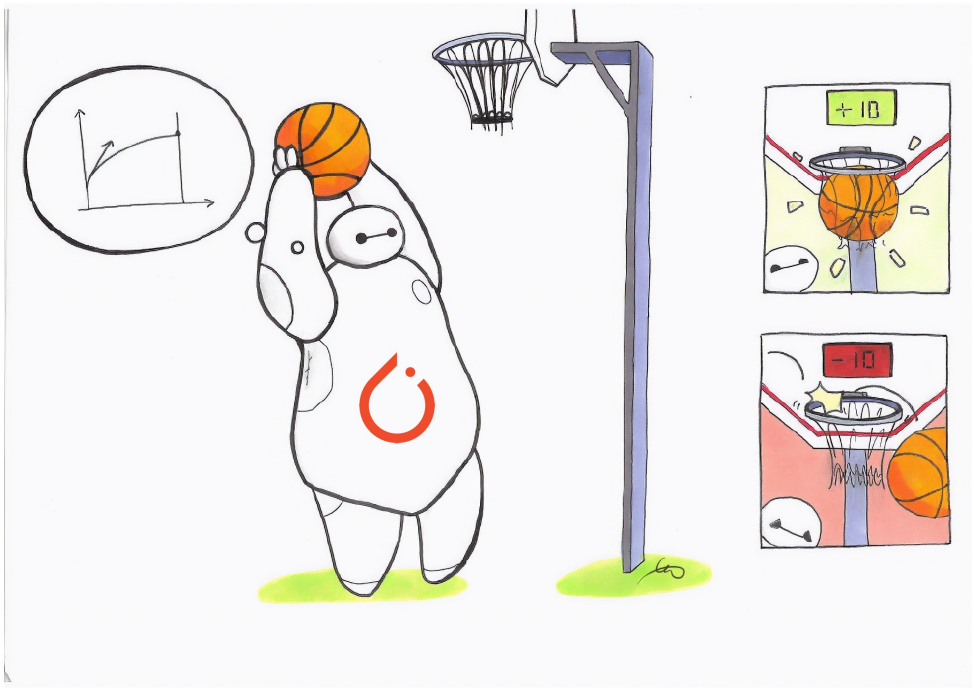


Figura 2.2: Ejemplo visual sencillo que muestra la función de recompensa en RL: se premia al agente por lograr un objetivo (meter la pelota: +10) y se penaliza por fallar (no meter la pelota: -10), fomentando el aprendizaje mediante ensayo y error.

Un entorno de RL se formaliza como un **Proceso de Decisión de Markov** (MDP)[1], definido por el cuarteto $\langle S, A, P, R \rangle$, donde:

- S: conjunto de estados posibles del entorno.

- A: conjunto de acciones que el agente puede ejecutar.
- P: función de transición de estados, que define la probabilidad de pasar de un estado a otro tras ejecutar una acción.

$$P(s_{t+1} \mid a_t, s_t)$$

- R: función de recompensa que asigna un valor (numérico) a cada transición estado-acción-estado.

$$R(s_t, a_t, s_{t+1})$$

La **Figura 2.3** representa gráficamente un modelo típico de Proceso de Decisión de Markov (MDP), donde se visualizan los elementos clave del entorno y la interacción del agente con el mismo.

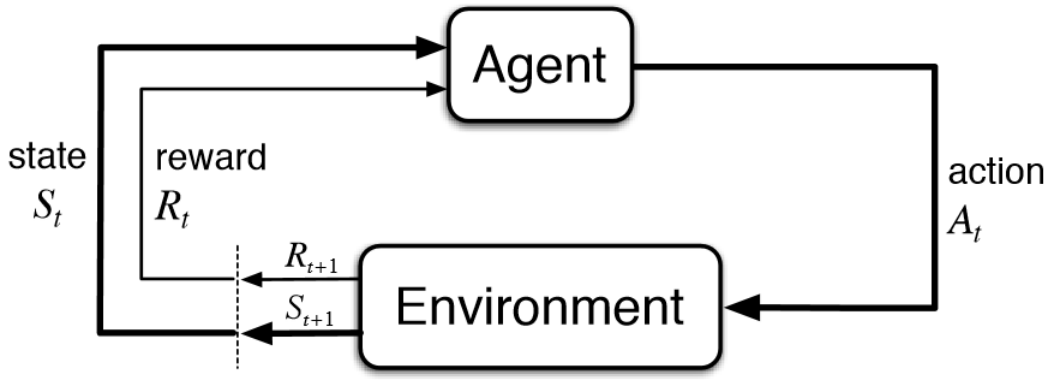


Figura 2.3: Diagrama del Proceso de Decisión de Markov, que ilustra la interacción entre estados, acciones y recompensas.

El objetivo del agente es aprender una política $\pi(a|s)$ que maximice la recompensa acumulada a largo plazo, formalizada como

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, s_{t+1})\right],$$

donde $\gamma \in [0, 1]$ es el factor de descuento que pondera la importancia de las recompensas futuras.

Este paradigma ha demostrado ser especialmente eficaz en tareas como juegos, control robótico, navegación autónoma o estrategias de trading.

Para implementar y optimizar el comportamiento del agente en un entorno de Aprendizaje por Refuerzo, se emplean ciertos componentes fundamentales que definen su estrategia de actuación y su capacidad de evaluación. Entre estos, se encuentran la **política**,

que guía la toma de decisiones del agente, y la **función de valor**, que permite estimar la calidad de los estados según la recompensa esperada. Ambos elementos son especialmente relevantes en enfoques basados en el método *actor-crítico*. Una vez procesada la entrada por las capas convolucionales y completamente conectadas de la red neuronal, el modelo genera dos salidas principales: la política y el valor del estado. A continuación, se detalla el propósito y funcionamiento de cada una:

- **Política**: La política, denotada como $\pi(s)$, es una función que toma como entrada un estado del entorno y devuelve una distribución de probabilidad sobre las posibles acciones. En este caso, dado que el espacio de acción es continuo (compuesto por un ángulo y una velocidad), la política no devuelve directamente una acción, sino los parámetros de una distribución gaussiana multivariante: las medias y desviaciones estándar correspondientes a cada dimensión de la acción. Durante la ejecución, se muestrea una acción de esa distribución. Este componente corresponde al *actor* dentro del paradigma *actor-crítico*, y su objetivo es aprender a seleccionar acciones que maximicen la recompensa esperada.
- **Valor**: El valor del estado, representado como $V(s)$, es una función que estima el valor esperado de recompensas futuras que puede obtenerse a partir de un estado, siguiendo la política actual. Este valor es un único número escalar y corresponde al *crítico* en el enfoque *actor-crítico*. Su función principal dentro de PPO es servir como una referencia para calcular la ventaja (*advantage function*), la cual indica cuánto mejor o peor fue una acción respecto a lo que se esperaba en ese estado. Es decir, permite predecir las recompensas futuras sin tener que simular todos los estados futuros, utilizando $V(s_t)$ y la recompensa r_{t+1}.

2.2.1. Proximal Policy Optimization (PPO)

PPO es un algoritmo de aprendizaje por refuerzo [3] que se basa en la optimización directa de políticas, y es ampliamente utilizado por su estabilidad y eficiencia en entornos continuos. Se basa en la idea de optimizar una política estocástica sin que los cambios entre actualizaciones consecutivas sean demasiado grandes. En este caso, la red neuronal recibe como input la observación del entorno, que suele ser una imagen o un conjunto de imágenes (por ejemplo, frames de un juego o cámaras en un entorno de conducción autónoma).

En la **Figura 2.4** se ilustra el esquema general del algoritmo PPO dentro del marco Actor-Critic. En este enfoque, el agente está compuesto por dos componentes principales: el *actor*, que representa la política π y se encarga de seleccionar acciones dadas las observaciones del entorno, y el *critic*, que estima el valor del estado (o la ventaja) y proporciona una señal que guía la mejora de la política. Ambos módulos interactúan con el entorno: el actor genera acciones que afectan al entorno, y el crítico evalúa las consecuencias de dichas acciones para actualizar la política de forma más estable.

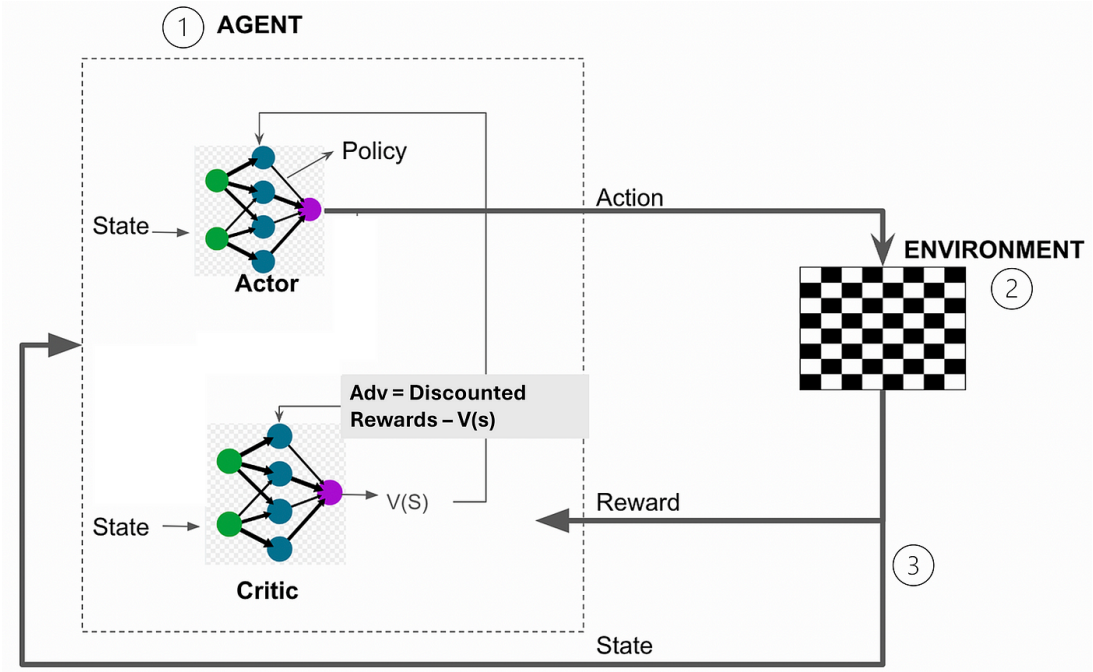


Figura 2.4: Esquema del algoritmo Proximal Policy Optimization (PPO) dentro del marco Actor-Critic. El actor genera acciones a partir de observaciones del entorno, mientras que el crítico evalúa el valor de los estados para guiar la actualización estable de la política.

Una de las claves del funcionamiento de PPO es el uso de funciones de valor que permiten mejorar la política de forma más eficiente. En particular, el crítico estima la función de valor $V^{\pi}(s)$, que representa la recompensa esperada futura al estar en el estado s y seguir la política actual π. También puede estimarse la función de acción-valor $Q^{\pi}(s, a)$, que refleja la recompensa esperada al tomar la acción a en el estado s, seguida de la política π. A partir de estas, se define la función ventaja $A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s)$, que indica si una acción es mejor o peor que la media de la política.

Para estimar esta ventaja de manera eficiente, PPO emplea *Generalized Advantage Estimation* (GAE) [4], que permite un balance controlado entre sesgo y varianza. El cálculo se realiza como una suma ponderada de las diferencias temporales.

Una vez estimadas las ventajas, PPO actualiza la política utilizando un objetivo que restringe cuánto puede cambiar esta entre iteraciones. La idea es evitar actualizaciones demasiado agresivas que puedan degradar el rendimiento. Para ello, se introduce un ratio de probabilidad entre la política nueva y la anterior:

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

Una forma teóricamente más adecuada de limitar el cambio entre políticas es penalizar

explícitamente la divergencia KL[1] entre la política nueva y la antigua. Esta penalización puede expresarse como

$$L^{KL}(\theta) = \mathbb{E}_t \left[r_t(\theta)\hat{A}_t - \beta \, \mathrm{KL}\left[\pi_{\theta_{\text{old}}}(\cdot \mid s_t) \,\|\, \pi_\theta(\cdot \mid s_t)\right] \right],$$

donde β controla el peso de la penalización. Sin embargo, en la práctica, el algoritmo PPO suele utilizar una alternativa más simple pero efectiva: la técnica de *clipping*, que evita que el ratio $r_t(\theta)$ se aleje demasiado de 1. Esta aproximación funciona como una forma práctica de limitar el cambio de política sin necesidad de calcular explícitamente la divergencia KL, y se ha observado que produce mejores resultados empíricos en muchos casos [3]. El objetivo de *clipping* se define como

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

Ambas formulaciones, con penalización KL o con *clipping*, buscan el mismo fin: evitar que la nueva política se desvíe excesivamente de la anterior. De hecho, el *clipping* puede interpretarse como una aproximación práctica de una región de confianza sobre la política, inspirada en el concepto de *trust region* de métodos como TRPO (Trust Region Policy Optimization). PPO implementa este principio de forma eficiente, evitando la complejidad computacional de TRPO, pero conservando gran parte de sus beneficios en cuanto a estabilidad y convergencia.

2.2.2. Redes neuronales y redes convolucionales

Las **redes neuronales artificiales** (ANN, por sus siglas en inglés) son modelos computacionales inspirados en el funcionamiento del cerebro humano. Están compuestas por un conjunto de nodos llamados *neuronas*, organizadas en capas, que procesan información mediante operaciones matemáticas. Cada neurona recibe entradas, las pondera a través de pesos, aplica una función de activación no lineal, y produce una salida que puede alimentar a otras neuronas. Esta arquitectura permite a las redes aprender representaciones complejas de los datos a partir de ejemplos, lo que las hace especialmente útiles en tareas como clasificación, regresión, reconocimiento de patrones y control.

Una red neuronal típica se organiza en tres tipos de capas:

- **Capa de entrada**: recibe los datos iniciales (por ejemplo, píxeles de una imagen).
- **Capas ocultas**: procesan la información internamente mediante transformaciones no lineales.

[1] La divergencia de Kullback-Leibler (KL) mide la diferencia entre dos distribuciones de probabilidad. En este contexto, cuantifica cuánto ha cambiado la política nueva respecto a la anterior. Una divergencia KL pequeña indica que ambas políticas son similares.

- **Capa de salida**: genera el resultado final (por ejemplo, una acción, una etiqueta o una predicción).

El aprendizaje de la red consiste en ajustar los pesos internos mediante un proceso llamado *retropropagación* (backpropagation), que minimiza una función de pérdida en función del error cometido.

Redes neuronales convolucionales (CNN)

Las **redes neuronales convolucionales** (CNN, por sus siglas en inglés) son una clase especializada de redes neuronales diseñadas específicamente para procesar datos con una estructura de tipo espacial, como las imágenes. A diferencia de las redes neuronales totalmente conectadas, las CNN aprovechan la estructura local de los datos mediante la operación de *convolución*, lo que les permite extraer automáticamente características relevantes como bordes, formas o texturas.

En la **Figura 2.5** se puede observar la arquitectura típica de una red CNN en la que una imagen de entrada pasa por una serie de capas convolucionales, de activación y de agrupamiento, lo que permite extraer representaciones jerárquicas de características. Estas se procesan posteriormente en capas totalmente conectadas para realizar la tarea final (por ejemplo, clasificación o estimación de valores).

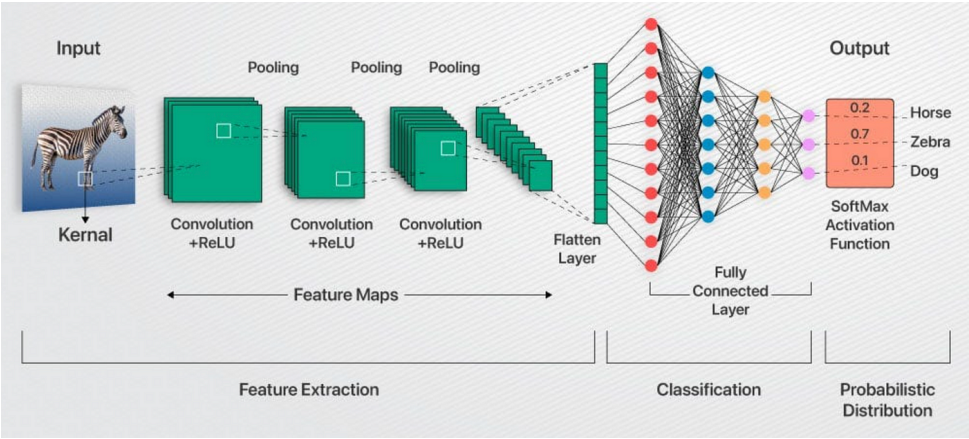


Figura 2.5: Arquitectura típica de una red neuronal convolucional (CNN).

Una CNN está compuesta típicamente por los siguientes bloques:

- **Capas convolucionales**: aplican filtros (también llamados kernels) que recorren la imagen detectando patrones locales. Cada filtro genera un *mapa de activación* que representa la presencia de una característica específica.

- **Capas de activación**: aplican funciones no lineales como ReLU (Rectified Linear Unit) para introducir no linealidad en el modelo.
- **Capas de agrupamiento** (*pooling*): reducen la dimensionalidad de los mapas de activación, manteniendo las características más importantes y reduciendo el costo computacional.
- **Capas completamente conectadas** (*fully connected*): combinan la información extraída en las capas anteriores para producir una salida final, como una clasificación o una acción.

El principal beneficio de las CNN frente a las redes densas tradicionales es su capacidad para captar relaciones espaciales jerárquicas en los datos visuales, con una eficiencia computacional significativamente mayor. Gracias a su arquitectura compartida, las CNN requieren menos parámetros, son menos propensas al sobreajuste y funcionan mejor en tareas de percepción visual, como reconocimiento de objetos, detección de obstáculos o conducción autónoma basada en imágenes.

En resumen, las redes convolucionales constituyen una herramienta fundamental para abordar problemas donde la entrada del sistema es una imagen o secuencia visual. Su capacidad para aprender representaciones espaciales complejas las convierte en una elección natural en el campo del Aprendizaje por Refuerzo basado en visión, donde el agente debe interpretar visualmente su entorno para tomar decisiones en tiempo real.

2.3. Tecnologías y herramientas utilizadas

En este proyecto se ha hecho uso de diversas tecnologías y herramientas que permiten el desarrollo, entrenamiento y evaluación de un agente de Aprendizaje por Refuerzo en un entorno de simulación de robótica móvil. Estas tecnologías abarcan desde simuladores físicos hasta librerías especializadas en aprendizaje automático y visualización de métricas.

El objetivo de esta sección es describir cada una de estas herramientas y explicar cómo se integran entre sí para formar un sistema funcional. Esta combinación de herramientas permite crear una infraestructura modular y reproducible, adecuada para investigar soluciones de conducción autónoma mediante técnicas modernas de RL.

2.3.1. Simulación y robótica móvil

La conducción autónoma es una aplicación compleja de la robótica móvil, ya que requiere percibir el entorno, tomar decisiones en tiempo real y ejecutar acciones precisas. Para probar algoritmos sin depender de hardware físico, se recurre a entornos de simulación realistas.

ROS (Robot Operating System)

Robot Operating System (ROS) [5] es un conjunto de herramientas, bibliotecas y convenciones diseñadas para simplificar el desarrollo de software para sistemas robóticos. Aunque no es un sistema operativo en el sentido tradicional, ROS proporciona servicios similares, como abstracción de hardware, control de dispositivos, paso de mensajes entre procesos y gestión de paquetes.

ROS se basa en una arquitectura distribuida en nodos, donde cada nodo realiza una función específica (por ejemplo, adquisición de sensores, control de movimiento o planificación). Estos nodos se comunican entre sí mediante tópicos, servicios y acciones, lo que facilita la creación de sistemas modulares, escalables y reutilizables. ROS es ampliamente utilizado tanto en la academia como en la industria por su flexibilidad, compatibilidad con numerosos lenguajes de programación (como Python y C++) y su gran ecosistema de paquetes comunitarios.

Gazebo

Gazebo [6] es un simulador 3D de código abierto que permite crear entornos virtuales realistas para pruebas de algoritmos de robótica sin necesidad de hardware físico. Proporciona una simulación precisa de dinámicas físicas (colisiones, fricción, gravedad), sensores (cámaras, LIDAR, IMU) y actuadores, lo que lo convierte en una herramienta poderosa para el desarrollo y validación de sistemas autónomos.

El simulador permite definir modelos complejos de robots y mundos mediante archivos de configuración y scripts, y ofrece interfaces de integración con otros marcos de software, como ROS. Gracias a su motor físico, Gazebo permite ejecutar simulaciones en tiempo real o aceleradas, proporcionando una plataforma segura, repetible y escalable para experimentación en robótica móvil, manipulación y vehículos autónomos.

2.3.2. Aprendizaje Automático

El desarrollo del sistema de Aprendizaje por Refuerzo en este proyecto se ha apoyado en un conjunto de librerías especializadas que permiten tanto la definición de entornos de simulación como la implementación, entrenamiento y análisis de algoritmos de RL.

Stable Baselines3 y PyTorch

Stable Baselines3 [7] es una implementación de algoritmos de RL basada en PyTorch [8], que ofrece modelos robustos y reproducibles. Se eligió por su integración con Gy-

mnasium, su rendimiento en GPU y por incluir gran variedad de algoritmos, entre los que destaca PPO, y arquitecturas de redes neuronales, como CNN, gestion de rollouts[2] y protocolos de evaluación.

Gymnasium y entornos personalizados

`Gymnasium` es una biblioteca ampliamente utilizada en el campo del Aprendizaje por Refuerzo [9], que proporciona una interfaz estandarizada para la creación, uso y evaluación de entornos. Esta interfaz se basa en tres funciones principales:

- `reset()`: reinicia el entorno y devuelve el estado inicial del agente.
- `step(action)`: ejecuta una acción en el entorno y devuelve una tupla con el nuevo estado, la recompensa obtenida, una señal booleana de finalización por una acción del robot, como salir de la pista o terminar una vuelta (`done`), y otra de truncamiento para acciones externas al robot, como la expiración de un timer (`truncated`), así como información adicional en un diccionario opcional (`info`).
- `render()`: permite visualizar el entorno, ya sea en modo gráfico o como texto, dependiendo de la implementación.

Para este trabajo se ha desarrollado un entorno personalizado compatible con la API de `Gymnasium`, que actúa como puente entre el simulador y el agente de aprendizaje por refuerzo. El entorno implementa todas las funciones necesarias para integrarse con Stable Baselines3 y muchas otra librerías de RL, facilitando la interacción entre el agente y el entorno simulado.

Gracias a esta integración, es posible entrenar agentes de RL utilizando bibliotecas como `Stable Baselines3`, garantizando compatibilidad, modularidad y reproducibilidad en los experimentos.

TensorBoard

`TensorBoard` es una herramienta de visualización desarrollada por Google como parte del ecosistema de `TensorFlow`, que permite inspeccionar en tiempo real el comportamiento de modelos de aprendizaje automático durante el entrenamiento. Entre sus funcionalidades más destacadas se encuentra la representación gráfica de métricas como la función de pérdida, las recompensas acumuladas, la evolución de los parámetros del modelo, así como histogramas, distribuciones y comparativas entre sesiones de entrenamiento.

[2]Un *rollout* es una secuencia de pasos (estado, acción, recompensa) generada al interactuar con el entorno usando la política actual del agente, y que se utiliza para actualizar dicha política.

Capítulo 3

Arquitectura y diseño del sistema

El sistema desarrollado para este proyecto se compone de varios módulos que interactúan de forma coordinada para permitir el entrenamiento y evaluación de un agente de control autónomo. La arquitectura se ha diseñado siguiendo un enfoque modular, permitiendo flexibilidad, reutilización de componentes y facilidad en la depuración y análisis.

3.1. Flujo de datos entre ROS, Gazebo y el agente

El sistema de conducción autónoma propuesto integra diversos componentes que se comunican entre sí de manera sincronizada para permitir el entrenamiento y evaluación del agente inteligente. El flujo de datos sigue una arquitectura cíclica y coordinada, donde cada módulo cumple una función específica dentro del lazo de control. A continuación, se detalla el flujo completo de información:

1. **Simulación en Gazebo:** El simulador Gazebo representa el entorno físico virtual en el que opera el vehículo autónomo. Este entorno incluye el modelo dinámico del coche, las características del terreno (carretera, bordes, curvas) y la cámara montada en el vehículo. En cada ciclo de simulación, Gazebo genera una imagen desde la cámara frontal del coche, que simula una cámara RGB. Esta imagen representa la percepción visual del entorno en tiempo real.
2. **Transmisión mediante ROS:** El sistema operativo robótico (ROS) actúa como middleware y se encarga de la comunicación entre Gazebo y los componentes de alto nivel. ROS publica las imágenes de la cámara como mensajes en un tópico específico.
3. **Preprocesamiento en el entorno Gym:** El entorno de entrenamiento, implementado mediante la interfaz Gym, actúa como un contenedor que abstrae las interacciones entre el agente y el entorno simulado. Este entorno recibe las imágenes desde ROS y las presenta al agente como una observación.

4. **Inferencia del agente:** El agente, basado en Aprendizaje por Refuerzo Profundo (PPO), procesa la observación y determina la mejor acción posible en función de su política actual. Esta acción consiste en comandos de velocidad y ángulo de giro. La acción se codifica como un mensaje ROS y se publica en el tópico correspondiente, para ser interpretada por el simulador.

5. **Aplicación de la acción en Gazebo:** ROS reenvía el mensaje de acción al simulador Gazebo, donde se aplica directamente al modelo del vehículo. Como resultado, el coche actualiza su posición, orientación y estado dinámico dentro del entorno simulado. Esta interacción permite cerrar el bucle de control en tiempo casi real, simulando la ejecución física de la acción en el mundo real.

6. **Cálculo de recompensa y retroalimentación:** El entorno Gym, en conjunto con los datos ofrecidos por ROS, calcula una función de recompensa basada en la nueva observación del estado del vehículo. Esta recompensa refleja qué tan buena fue la acción ejecutada según criterios definidos previamente. Luego, esta señal de recompensa se utiliza para actualizar la política del agente y se inicia un nuevo ciclo de interacción.

Este flujo de datos en bucle permite que el agente aprenda de la experiencia, ajustando su comportamiento con cada iteración. La arquitectura modular basada en ROS facilita además la integración de nuevos sensores o el intercambio de componentes (como cambiar el simulador o modificar la red neuronal) sin necesidad de rediseñar el sistema completo.

La **Figura 3.1** muestra la arquitectura general del sistema desarrollado para entrenar el agente de Aprendizaje por Refuerzo en el entorno simulado. En ella se visualizan los principales componentes: el simulador Gazebo, el middleware ROS que gestiona la comunicación entre nodos, el entorno Gym personalizado y el agente PPO implementado con Stable Baselines3. Esta estructura modular permite una integración fluida entre simulación, percepción, entrenamiento y evaluación del modelo.

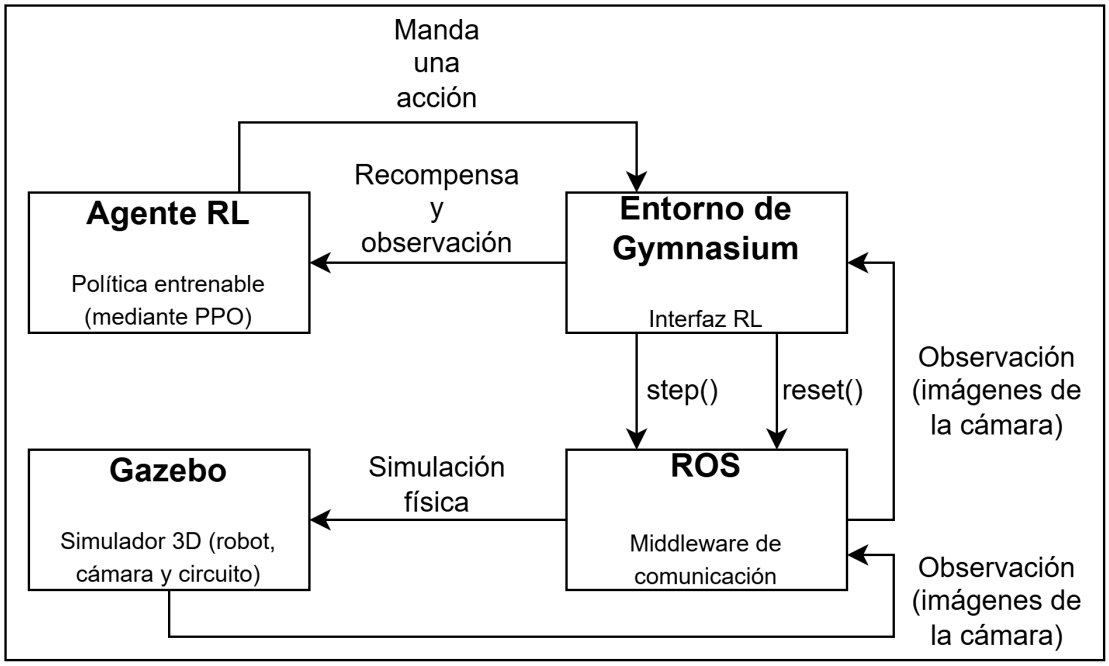


Figura 3.1: Diagrama que muestra la interacción modular entre el simulador Gazebo, el middleware ROS, el entorno Gym personalizado y el agente PPO. Este flujo cíclico permite el entrenamiento y evaluación del agente de control autónomo basado en imágenes captadas en tiempo real.

3.2. Componentes principales del sistema

A grandes rasgos, el sistema está compuesto por tres bloques principales:

1. **Interfaz de comunicación**: proporcionada por ROS, permite el intercambio de información entre los diferentes módulos, como la percepción del entorno y las órdenes de control del agente.
2. **Simulación del entorno**: gestionada mediante Gazebo, representa el mundo virtual, incluyendo el circuito, el vehículo y su sensor de cámara.
3. **Agente de Aprendizaje por Refuerzo**: implementado con Stable Baselines3 y Gymnasium, este módulo procesa la información visual, decide las acciones a ejecutar y ajusta su política mediante entrenamiento continuo.

3.2.1. Interfaz de comunicación mediante ROS

En este proyecto, ROS cumple una función central como middleware que facilita la comunicación entre los diferentes módulos del sistema, en particular entre el entorno de simulación en Gazebo y el agente de Aprendizaje por Refuerzo implementado con Stable Baselines3. ROS permite estructurar el sistema de forma modular, donde cada componente se comunica mediante la publicación y suscripción a *tópicos* y el uso de *servicios*, lo que mejora la escalabilidad y mantenibilidad del proyecto.

El nodo principal desarrollado para este sistema es `DeepRacerEnv`, que actúa como la interfaz entre el entorno de entrenamiento y el simulador Gazebo. La comunicación con Gazebo se realiza mediante la publicación de comandos de movimiento sobre el tópico `/vesc/low_level/ackermann_cmd_mux/output`, y la suscripción a los tópicos `/gazebo/model_states` y `/camera/zed/rgb/image_rect_color` para obtener información sobre la posición del vehículo y su percepción visual, respectivamente. Además, se utilizan los servicios `/gazebo/pause_physics`, `/gazebo/unpause_physics` y `/gazebo/set_model_state` para controlar la simulación y reiniciar el estado del entorno al inicio de cada episodio.

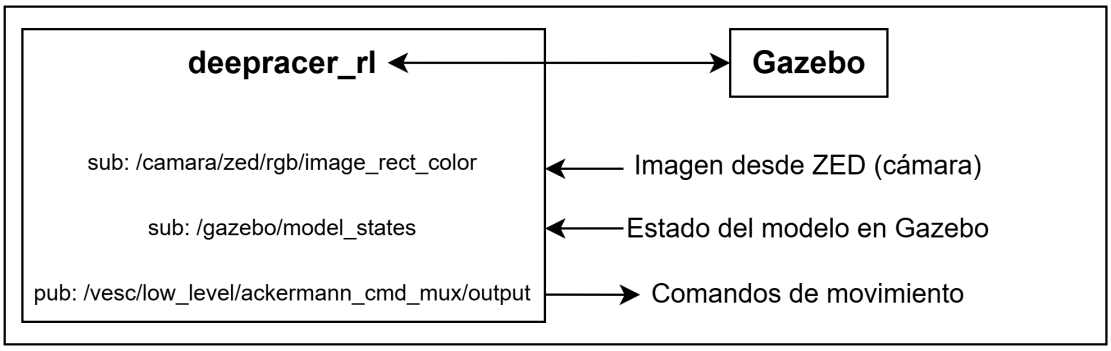


Figura 3.2: Diagrama que muestra la interacción modular entre el nodo principal de ejecución del entorno y sus suscriptores y publicadores.

En la **Figura 3.2** se representa la arquitectura del nodo principal `DeepRacerEnv`, evidenciando cómo se organiza la comunicación mediante ROS. Los componentes que conforman el nodo incluyen:

- **Imágenes de la cámara**: Uno de los componentes se encarga de suscribirse al tópico que publica las imágenes RGB captadas por la cámara virtual montada en el frontal del vehículo simulado. Estas imágenes son capturadas en tiempo real desde el entorno de Gazebo y enviadas al agente, que las utiliza como observaciones del entorno para tomar decisiones. Antes de ser procesadas por la red neuronal, las imágenes pueden ser redimensionadas o normalizadas para adecuarse al formato de entrada esperado por el modelo.

- **Comandos de control (velocidad y dirección)**: Otro elemento actúa como interfaz de salida, publicando en un tópico específico los comandos de acción generados por el agente. Estos comandos están compuestos por dos valores continuos: la velocidad lineal y el ángulo de dirección del vehículo. ROS se encarga de redirigir estos comandos al controlador del vehículo en Gazebo, cerrando así el bucle de control entre percepción y acción.
- **Posición y orientación del vehículo (evaluación, depuración y cálculo de la recompensa)**: Un componente adicional se suscribe a la información de estado del vehículo (por ejemplo, mensajes del tipo `Odometry`), que proporciona datos sobre la posición, orientación y velocidad actuales del agente en el entorno. Esta información no se utiliza como entrada del modelo, pero sí se registra con fines de evaluación, análisis del comportamiento y depuración. A partir de estos datos se pueden calcular métricas como distancia recorrida, número de colisiones, o desviación respecto al centro del carril.

Gracias a esta arquitectura basada en ROS, el sistema puede ejecutarse de forma sincronizada y controlada. Además, se facilita la integración de componentes adicionales como sensores virtuales, sistemas de monitorización o visualización en tiempo real. Esta capacidad de orquestar la interacción entre simulador y agente hace de ROS una herramienta clave para el desarrollo de soluciones avanzadas de conducción autónoma en entornos simulados.

3.2.2. Simulación del entorno con Gazebo

En este proyecto, Gazebo se ha utilizado para recrear un entorno de pruebas controlado, que incluye un circuito cerrado con geometría definida (curvas, rectas, márgenes) y un vehículo simulado equipado con una cámara frontal. Esta simulación permite realizar pruebas seguras y reproducibles, lo que resulta ideal para el desarrollo y evaluación de sistemas de conducción autónoma. Además, tanto el vehículo como los circuitos han sido diseñados de forma que permiten introducir cambios fácilmente (nuevos sensores, diferentes trazados, modificaciones físicas del entorno) sin necesidad de rehacer el sistema desde cero.

1. Modelo del vehículo

El modelo del vehículo en Gazebo ha sido diseñado con un equilibrio entre simplicidad computacional y realismo funcional. Si bien se omiten ciertos elementos físicos complejos, se preservan las dinámicas esenciales necesarias para simular la conducción, como la dirección, el movimiento y la percepción visual desde una cámara frontal. Esta simplificación permite entrenar de forma eficiente sin comprometer el aprendizaje de comportamientos relevantes. La **Figura 3.3** ilustra una comparación visual entre el modelo virtual y un coche real de referencia.

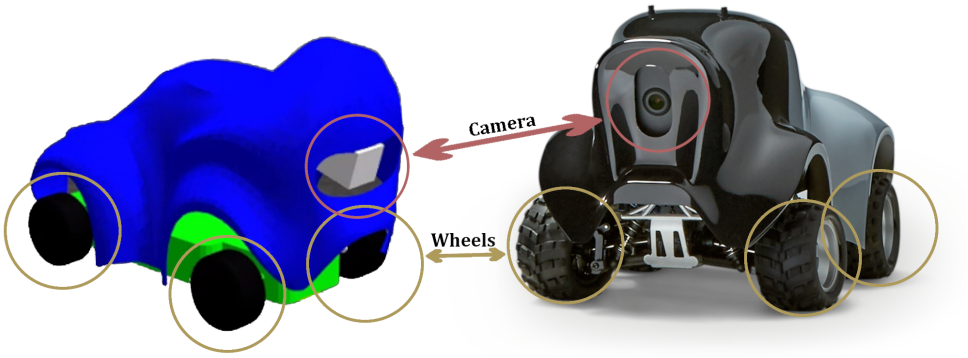


Figura 3.3: Comparación entre el modelo virtual y el modelo real de vehículo.

2. Geometría del circuito

Gazebo permite diseñar circuitos virtuales con gran precisión geométrica, lo que es ideal para pruebas sistemáticas y controladas. Las pistas simuladas presentan superficies limpias y delimitaciones claras, lo que facilita la evaluación del desempeño del agente sin interferencias externas. En la **Figura 3.4** se compara visualmente una pista simulada en Gazebo con una pista real.

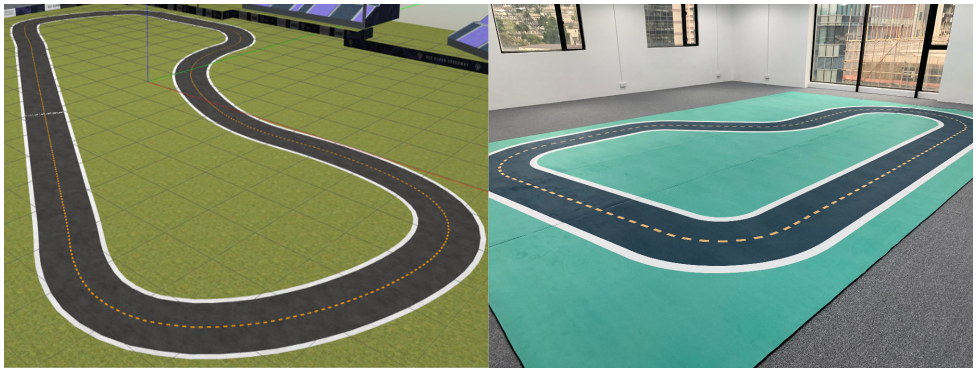


Figura 3.4: Comparación entre el modelo virtual y el modelo real de la pista.

La **Figura 3.5** muestra una vista general del entorno construido en Gazebo, incluyendo el circuito y las marcas viales, que permiten simular de forma eficiente escenarios de navegación autónoma.

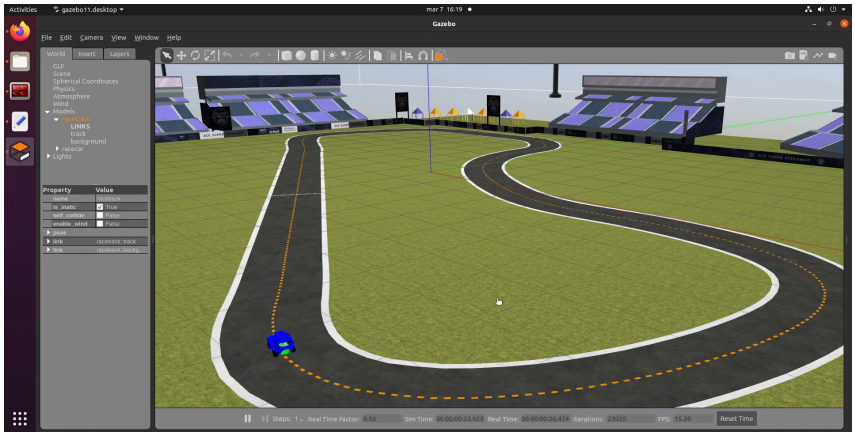


Figura 3.5: Visualización del simulador Gazebo con pista y vehículo cargados.

3.2.3. Agente RL con Stable Baselines3

El agente de control desarrollado en este proyecto está construido sobre el paradigma de Aprendizaje por Refuerzo, utilizando la librería `Stable Baselines3` como base para la implementación de algoritmos. En particular, se emplea el algoritmo **Proximal Policy Optimization**, ampliamente utilizado por su equilibrio entre rendimiento y estabilidad durante el entrenamiento.

El entrenamiento del agente se lleva a cabo dentro de un entorno personalizado conforme a la interfaz `Gymnasium`. Este entorno gestiona los datos referidos la conducción de un vehículo autónomo en un circuito virtual, proporcionando observaciones visuales en forma de imágenes RGB y recibiendo acciones continuas compuestas por ángulo de dirección y velocidad.

Arquitectura de la red neuronal

La política del agente —es decir, el modelo encargado de tomar decisiones en cada instante— está parametrizada mediante una red neuronal convolucional (CNN), diseñada específicamente para procesar entradas visuales. Esta red actúa como el núcleo del agente PPO, generando tanto la política (actor, $\pi(s)$) como el valor del estado (crítico, $V(s)$), y permitiendo al sistema aprender directamente a partir de imágenes sin requerir características manuales.

El núcleo del agente de Aprendizaje por Refuerzo está constituido por una red neuronal profunda que permite procesar observaciones visuales en bruto (imágenes RGB) y transformarlas en decisiones de control. Esta red actúa como función de aproximación tanto para la política (actor) como para el valor del estado (crítico), siguiendo el paradigma *actor-crítico* característico del algoritmo PPO.

Dado que la entrada al sistema son imágenes, se emplea una arquitectura convolucional especializada en extraer patrones espaciales relevantes del entorno, como bordes de pista, curvas o cambios en la dirección. Estas características visuales se convierten progresivamente en representaciones abstractas que permiten al agente inferir acciones óptimas.

La **Figura 3.6** muestra un esquema detallado de la arquitectura convolucional empleada como política en el algoritmo PPO. Esta red está diseñada para procesar imágenes RGB capturadas del entorno y extraer características espaciales mediante capas convolucionales, que actúan como detectores de patrones visuales relevantes para la toma de decisiones.

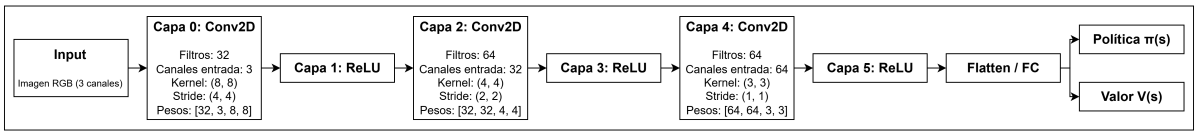


Figura 3.6: Esquema de la arquitectura convolucional utilizada como política en PPO. La red procesa imágenes RGB capturadas del entorno simulado mediante varias capas convolucionales que extraen características espaciales jerárquicas. La salida incluye tanto la política (actor) para seleccionar acciones, como el valor del estado (crítico) para evaluar las decisiones, siguiendo el paradigma actor-crítico.

Codificador visual

La política está parametrizada por una red neuronal convolucional (CNN) que recibe como *input* la observación del entorno. En nuestro caso, esta observación es una imagen con 3 canales (RGB) y tamaño específico según la tarea.

Concretamente, la red utilizada tiene las siguientes capas convolucionales:

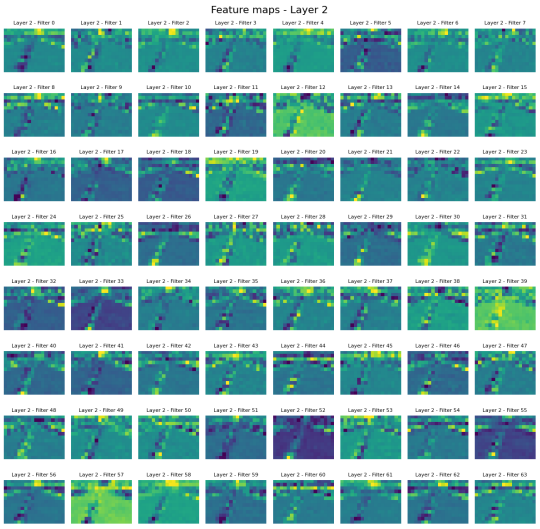
- **Capa 0 (Conv2D)**: Esta primera capa utiliza 32 filtros con un tamaño de kernel de 8×8 y un stride (paso) de 4×4. El tensor[1] de entrada tiene forma `[batch_size, 3, H, W]`, donde 3 corresponde a las tres canales de color (RGB). El tamaño relativamente grande del kernel[2] permite capturar patrones espaciales amplios en la imagen, mientras que el stride[3] de 4 reduce significativamente la resolución espacial, acelerando el procesamiento y disminuyendo la dimensionalidad para las siguientes capas.

[1]Un *tensor* es una estructura de datos multidimensional que generaliza vectores y matrices, utilizada para representar datos como imágenes en redes neuronales.

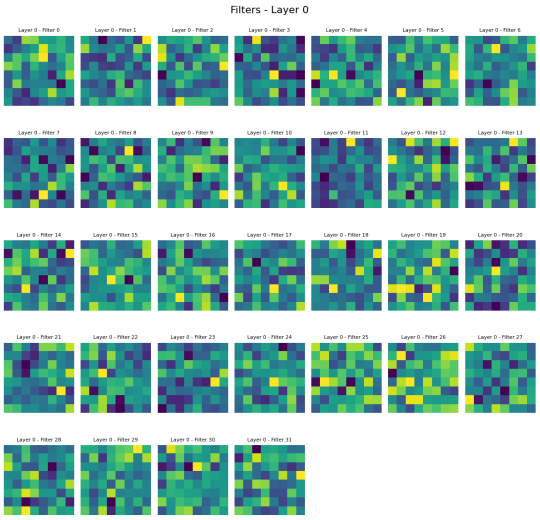
[2]El *kernel* o filtro es una pequeña matriz que se desliza sobre la imagen para extraer características locales.

[3]El *stride* es el tamaño del paso con que el kernel se mueve sobre la imagen, afectando la resolución espacial de la salida.

En la **Figura 3.7a** se muestran algunos ejemplos de filtros aprendidos por esta capa, mientras que la **Figura 3.7b** presenta los mapas de activación (feature maps) resultantes al aplicar dichos filtros a una imagen de entrada.



(a) Ejemplos de filtros (kernels) aplicados en la Capa 0.

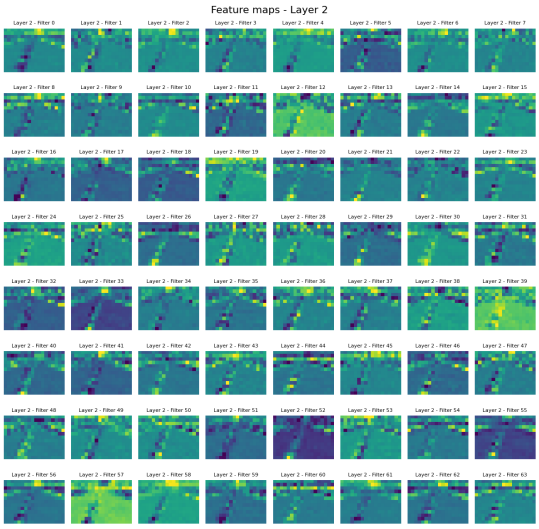


(b) Feature maps resultantes tras aplicar la Capa 0.

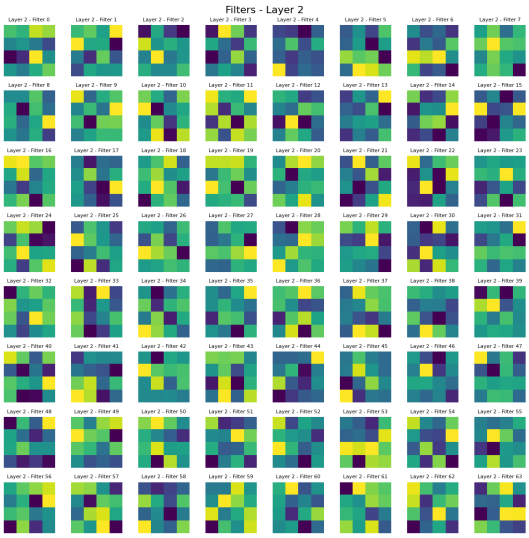
Figura 3.7: En la Capa 0 de la red convolucional, los filtros (kernels) aprendidos están diseñados para captar patrones visuales amplios, como bordes horizontales y verticales, así como texturas globales presentes en la imagen de entrada. Al aplicar estos filtros, se generan mapas de activación que resaltan las regiones donde se detectan patrones relevantes, como líneas y contornos, proporcionando una primera representación estructurada de la información visual.

- **Capa 2 (Conv2D)**: Aquí se aplican 64 filtros con un kernel de tamaño 4×4 y un stride de 2×2. Al aumentar el número de filtros, la red puede aprender una mayor variedad de características más complejas. El tamaño más pequeño del kernel en comparación con la primera capa permite enfocarse en detalles más finos, y el stride de 2 continúa reduciendo la resolución espacial pero de forma más moderada, manteniendo suficiente información para un análisis detallado.

  Los filtros aprendidos en esta segunda capa se muestran en la **Figura 3.8a**, y los correspondientes mapas de activación pueden observarse en la **Figura 3.8b**.



(a) Ejemplos de filtros (kernels) aplicados en la Capa 2.

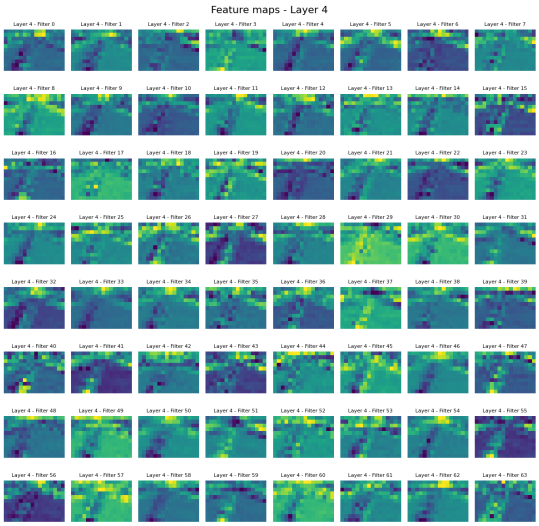


(b) Feature maps resultantes tras aplicar la Capa 2.

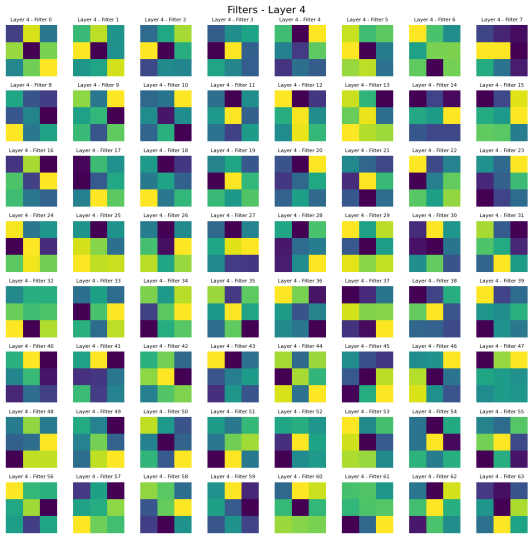
Figura 3.8: En la Capa 2, los filtros aprendidos presentan una mayor profundidad y un tamaño de kernel reducido, lo que les permite capturar detalles visuales más específicos, como esquinas, intersecciones o variaciones locales de textura. Al aplicar estos filtros, los mapas de activación resultantes muestran respuestas más refinadas, capaces de detectar estructuras complejas mediante la combinación de patrones previamente identificados en capas anteriores.

- **Capa 4 (Conv2D)**: Esta capa utiliza 64 filtros con kernels de 3×3 y stride de 1×1. Este es el tamaño de kernel más común en arquitecturas convolucionales debido a su capacidad para extraer patrones locales con gran precisión. Al mantener el stride en 1, esta capa no reduce la resolución espacial, sino que se centra en refinar las características extraídas, permitiendo a la red aprender combinaciones más complejas de las activaciones previas.

  La **Figura 3.9a** muestra algunos de los filtros empleados en esta capa, mientras que en la **Figura 3.9b** se pueden observar los correspondientes mapas de activación.



(a) Ejemplos de filtros (kernels) aplicados en la Capa 4.



(b) Feature maps resultantes tras aplicar la Capa 4.

Figura 3.9: En la Capa 4, se aplican filtros con un kernel pequeño (3×3) y un stride de 1, lo que permite refinar las activaciones mediante combinaciones de características más abstractas. Los mapas de activación generados en esta etapa reflejan la consolidación de patrones de alto nivel extraídos de las capas anteriores, preparando así una representación visual más rica y estructurada para la toma de decisiones por parte del agente.

Estas tres capas convolucionales actúan de forma secuencial, disminuyendo progresivamente la resolución espacial de la imagen mientras incrementan la profundidad del tensor de características, lo que permite a la red capturar y representar información visual desde patrones generales hasta detalles específicos.

3.2.4. Integración y diseño del entorno Gymnasium

Para posibilitar la integración del agente de Aprendizaje por Refuerzo con el simulador Gazebo, se ha desarrollado una clase personalizada en Python denominada `DeepRacerEnv`, que extiende la interfaz estándar de `gym.Env`, compatible con la librería Gymnasium. Esta clase encapsula toda la lógica necesaria para la interacción entre ROS, Gazebo y el agente de entrenamiento, gestionando el ciclo completo de ejecución del entorno: `reset()`, `step()` y, en caso necesario, `render()`.

Aunque el entorno `DeepRacerEnv` se ha diseñado siguiendo la estructura de un Proceso de Decisión de Markov (MDP), en la práctica se trata de un **Proceso de Decisión de Markov Parcialmente Observable** (POMDP), ya que el agente no tiene acceso completo al estado del entorno. En su lugar, recibe una observación parcial, representada por una imagen RGB de la cámara frontal, que no contiene toda la información relevante

del estado subyacente (por ejemplo, la posición y velocidad del vehículo no son observables directamente). Las acciones corresponden a comandos de control continuo: dirección de las ruedas y velocidad del vehículo.

La transición entre estados se produce a través de la dinámica simulada del entorno (definida por Gazebo y el modelo físico del vehículo), y cada acción ejecutada genera una recompensa escalar, diseñada para fomentar un comportamiento de conducción eficiente y seguro. Esta formulación permite aplicar algoritmos de Aprendizaje por Refuerzo que buscan aprender una política que maximice la recompensa acumulada, aunque el agente debe inferir el estado real a partir de observaciones parciales, lo cual incumple la propiedad de Markovianidad[4].

El entorno define de forma clara los espacios de observación y acción requeridos por la API de Gymnasium. El espacio de observación está constituido por imágenes RGB de tamaño $120 \times 160 \times 3$, captadas por la cámara frontal del vehículo simulado. El espacio de acción se define como un espacio continuo bidimensional, representado mediante `gym.spaces.Box`, donde el primer valor corresponde al ángulo de dirección del volante (rango $[-1, 1]$) y el segundo a la velocidad lineal del vehículo (rango $[0, 5]$).

Durante la inicialización, el entorno establece los mecanismos de comunicación con el simulador a través de ROS, mediante la suscripción y publicación en distintos tópicos, así como el uso de servicios. Se suscribe a los tópicos `/camera/zed/rgb/image_rect_color` para obtener la imagen de la cámara delantera y `/gazebo/model_states` para conocer la posición y orientación del vehículo. Por otra parte, publica comandos de control en el tópico `/vesc/low_level/ackermann_cmd_mux/output`, empleando mensajes del tipo `AckermannDriveStamped`. Además, se utilizan servicios ROS para pausar y reanudar la simulación, lo que permite mantener un control preciso sobre el entorno físico durante el entrenamiento. Esta información se considera únicamente para el cálculo de la reward y de las situaciones en las que el robot comete errores durante el entrenamiento.

El vehículo simulado implementa un modelo de dirección Ackermann[10], utilizado en la mayoría de automóviles reales. Este modelo asegura que las ruedas delanteras puedan girar en ángulos diferentes pero coordinados durante una curva, permitiendo que todas las ruedas describan trayectorias circulares con un centro de giro común. Esto minimiza el deslizamiento lateral de los neumáticos, lo cual es esencial para una conducción realista y estable.

En la **figura 3.10** se ilustra el principio del modelo de dirección Ackermann, donde se observa que, al tomar una curva, las ruedas delanteras giran con ángulos distintos para que todas las ruedas del vehículo sigan trayectorias circulares con un centro común de rotación, las ruedas traseras no giran. Este diseño geométrico permite que las ruedas traseras y delanteras no deslicen lateralmente, lo cual es crucial para mantener la estabilidad del

[4]La propiedad de Markovianidad implica que el próximo estado del sistema depende únicamente del estado actual y la acción tomada. En un POMDP, el agente no observa directamente el estado completo, por lo que debe trabajar con observaciones parciales que no garantizan dicha propiedad.

vehículo y replicar con fidelidad las condiciones de conducción reales en el simulador.

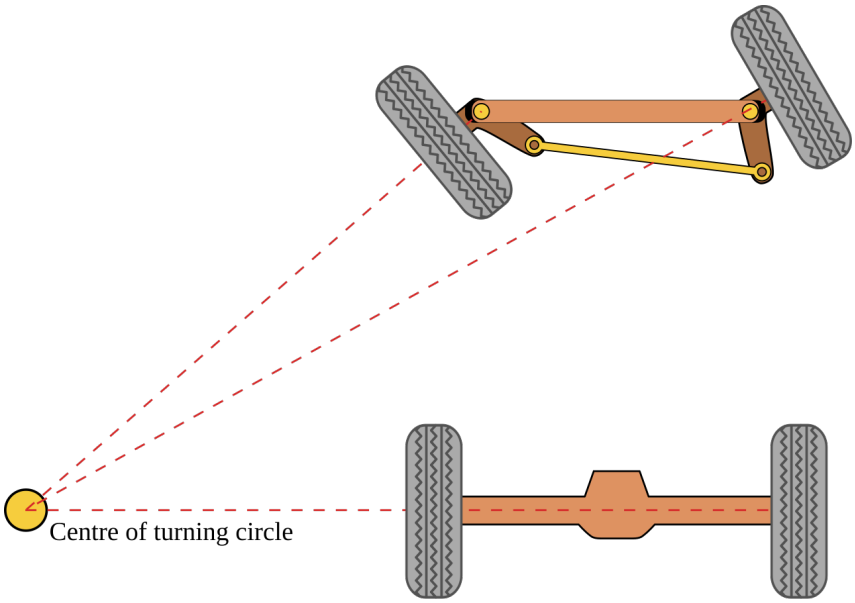


Figura 3.10: Visualización del modelo de conducción Ackermann. En este modelo, las ruedas delanteras giran en diferentes ángulos, de forma que todas las ruedas siguen trayectorias con un centro de curvatura común, minimizando el deslizamiento lateral.

Para controlar vehículos con geometría Ackermann en ROS, se emplean mensajes del tipo `ackermann_msgs/AckermannDriveStamped`, definidos por el paquete `ackermann_msgs`. Estos mensajes encapsulan la información de control en una estructura compuesta por una cabecera temporal (`std_msgs/Header`) y una instancia de `AckermannDrive`, que incluye los siguientes campos:

- `steering_angle`: ángulo de giro de las ruedas delanteras (en radianes).
- `steering_angle_velocity`: velocidad a la que cambia el ángulo de dirección.
- `speed`: velocidad lineal del vehículo.
- `acceleration`: aceleración lineal deseada.
- `jerk`: tasa de cambio de la aceleración.

En el contexto de este proyecto, se utilizan únicamente los campos `steering_angle` y `speed` para definir la acción del agente en cada paso del entorno. El resto de parámetros se dejan por defecto, ya que no son estrictamente necesarios para un control básico durante el entrenamiento.

La **Figura 3.11** muestra una representación esquemática del ciclo de un MDP aplicado al entorno `DeepRacerEnv`. En este esquema se ilustran las interacciones entre el agente y el entorno, donde en cada paso el agente recibe una observación del estado actual, selecciona una acción basada en su política, y a continuación el entorno devuelve una nueva observación junto con una recompensa que guía el aprendizaje. Este ciclo iterativo refleja la dinámica fundamental del Aprendizaje por Refuerzo implementado en el simulador Gazebo, que permite al agente optimizar su comportamiento de conducción autónoma mediante la maximización de la recompensa acumulada.

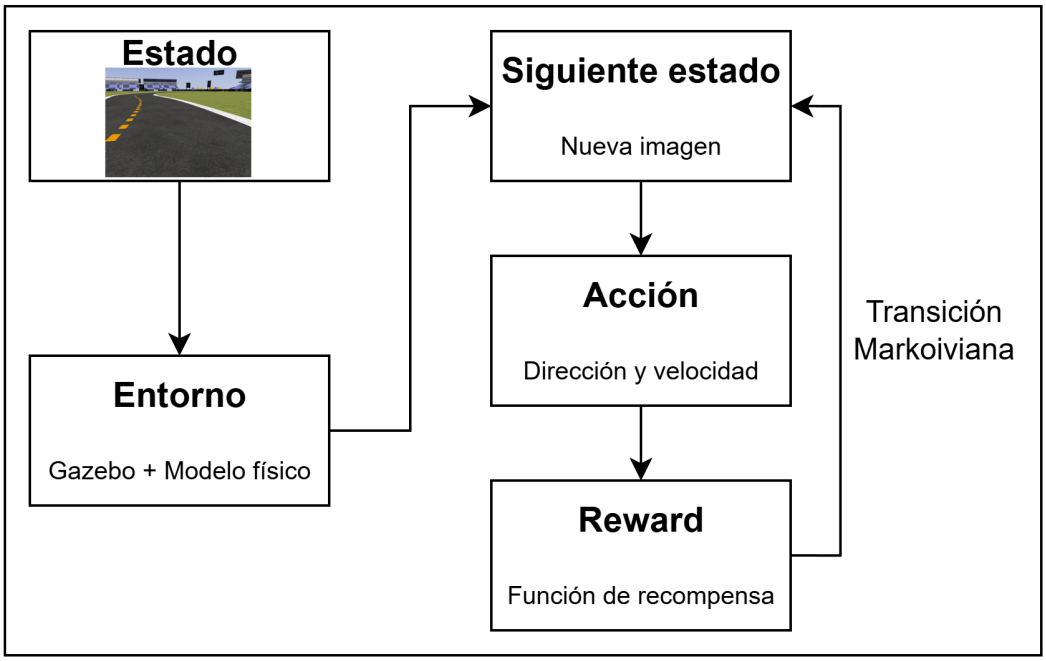


Figura 3.11: Representación del ciclo de Proceso de Decisión de Markov (MDP) aplicado al entorno `DeepRacerEnv`. El esquema ilustra la interacción secuencial entre el agente y el entorno: el agente observa el estado actual (imagen de cámara), selecciona una acción (control de dirección y velocidad), y recibe una nueva observación junto con una recompensa.

Cada episodio de entrenamiento comienza con una llamada al método `reset()`, que restablece la posición y orientación del vehículo a una configuración inicial predeterminada, reinicia las variables internas del entorno y vacía la memoria de trayectorias. Por su parte, el método `step(action)` aplica una acción generada por el agente al entorno, publicando los comandos necesarios mediante un mensaje `AckermannDriveStamped` y esperando brevemente (`time.sleep(0.025)`) para garantizar que la acción haya tenido efecto antes de evaluar el nuevo estado. Esta pausa, ajustada empíricamente, mejora la estabilidad del entrenamiento al sincronizar mejor la simulación física con el ciclo de aprendizaje.

Una vez aplicada la acción, el entorno devuelve una nueva observación (imagen RGB),

una recompensa escalar, indicadores de finalización del episodio (`done` y `truncated`),[5] y un diccionario opcional con información adicional. La recompensa se calcula a partir de una función compuesta que combina distintos factores, como la distancia al centro del carril, la proximidad a los waypoints definidos y la orientación del vehículo respecto al recorrido. Estos criterios se ponderan adecuadamente para fomentar un comportamiento eficiente y seguro.

Durante el entrenamiento, el entorno almacena las trayectorias recorridas por el vehículo en archivos CSV, lo que permite un análisis posterior detallado del comportamiento del agente. Para el cálculo de la recompensa se requiere encontrar el waypoint más cercano al vehículo, en un principio esta búsqueda se realizaba mediante una búsqueda secuencial (fuerza bruta), lo cual resultaba ineficiente al manejar grandes cantidades de puntos. Para resolver esta limitación, se implementó una estructura de datos `KDTree`[11], que permite realizar búsquedas vecinas de forma mucho más rápida, sin sacrificar precisión. Dicha mejora temporal se puede observar en la **Tabla 3.1**.

Método de búsqueda	**Tiempo promedio (s)**	**Mejora**
Fuerza bruta	0.0030	-
`KDTree`	0.0003	10x

Tabla 3.1: Comparación de tiempos promedio entre búsqueda por fuerza bruta y `KDTree`.

Gracias a esta arquitectura modular, el entorno `DeepRacerEnv` permite una integración fluida con algoritmos de Aprendizaje por Refuerzo basados en imágenes y acciones continuas. La combinación de Gymnasium, ROS y Gazebo ofrece un marco robusto y realista para el entrenamiento de agentes inteligentes, con control preciso sobre cada aspecto de la simulación. La implementación del modelo de dirección Ackermann, el uso de observaciones visuales en tiempo real y la incorporación de técnicas eficientes como `KDTree` consolidan este entorno como una plataforma potente y extensible para el desarrollo de sistemas de conducción autónoma.

3.3. Diseño de la función de recompensa

Una parte crucial del entrenamiento por Aprendizaje por Refuerzo es el diseño de la función de recompensa. Esta función define lo que el agente debe considerar como comportamiento deseado, y guía el aprendizaje hacia una política eficiente. En este proyecto, se ha diseñado una función de recompensa personalizada, adaptada al entorno simulado y al comportamiento esperado del vehículo autónomo.

[5]El indicador `done` señala que el episodio ha finalizado por alcanzar una condición terminal definida por la tarea, como salirse del circuito, ir dirección contraria o estar demasiado tiempo quieto. En cambio, `truncated` indica una finalización no relacionada con el comportamiento del agente, como alcanzar un límite máximo de pasos o una interrupción externa.

El objetivo principal de esta función es fomentar que el coche:

- Mantenga una orientación coherente con el trazado del circuito.
- Permanezca dentro de los límites de la pista, lo más centrado posible.
- Avance progresivamente hacia los waypoints que conforman el recorrido.
- Evite comportamientos indeseados como ir marcha atrás, salirse del circuito o quedarse parado.

La **Figura 3.12** ilustra de forma esquemática los factores que influyen en la función de recompensa. Se muestran las variables espaciales relevantes (distancia al centro, orientación, y avance) y así esta visualización sirve como mapa conceptual para entender cómo se integran distintas fuentes de información durante el entrenamiento.

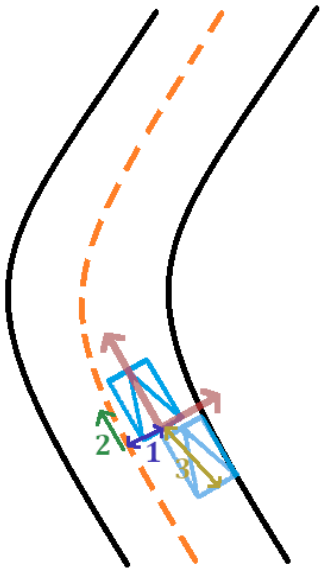


Figura 3.12: Representación gráfica de la función de recompensa. Se visualizan los elementos clave que influyen en el cálculo de la recompensa: la distancia lateral al centro de la pista, la orientación del vehículo respecto al trazado y la progresión hacia los waypoints.

La función de recompensa implementada combina varias métricas parciales ponderadas, que se detallan a continuación:

1. **Distancia al centro de la pista:** Se penaliza el alejamiento del vehículo respecto al centro de la pista. Si se supera un umbral máximo, también se considera que el vehículo ha salido del circuito y se termina el episodio con penalización.

2. **Orientación del vehículo:** Se calcula el coseno del ángulo entre el vector de orientación del coche y el vector que une el waypoint actual con el siguiente. Si este valor es bajo (inferior a 0.3), significa que el coche circula en dirección contraria al trazado, por lo que se penaliza con una recompensa negativa y se termina el episodio.

3. **Movimiento efectivo:** Se calcula la distancia recorrida en cada paso en comparación con un valor máximo estimado. Este valor representa si el vehículo avanza de forma constante y se usa como incentivo para evitar que se quede parado.

4. **Progreso por waypoints:** Se mantiene un contador de waypoints superados. Si completa todos los waypoints del circuito, recibe una bonificación adicional como recompensa por haber completado el circuito.

5. **Penalización por comportamiento incorrecto:** Si la velocidad del vehículo es negativa (marcha atrás), sale de la pista o permanece estancado durante demasiados pasos consecutivos, se penaliza con una recompensa negativa y se termina el episodio.

La recompensa total se calcula como una combinación ponderada de las tres componentes principales

$$\text{total\_reward} = r_{\text{centro}} \cdot w_{\text{center}} + r_{\text{orientación}} \cdot w_{\text{orient}} + r_{\text{avance}} \cdot w_{\text{avance}}.$$

Donde:

- **Recompensa por centrado en el carril** (r_{centro}):

$$r_{\text{centro}} = 1 - \frac{d}{d_{\text{máx}}} \tag{3.1}$$

  donde d es la distancia lateral entre el vehículo y el waypoint más cercano (centro del carril), y $d_{\max}$ es la distancia máxima permitida (mitad del grosor de la pista). *(Ver Figura 3.13a)*

- **Recompensa por orientación** ($r_{\text{orientación}}$):

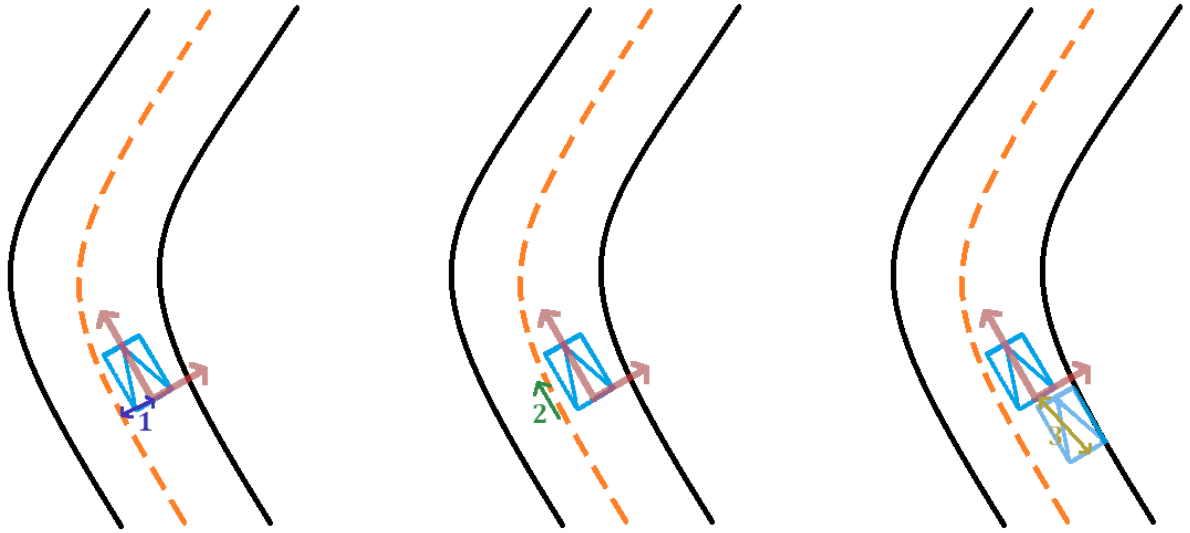
$$r_{\text{orientación}} = \cos(\theta) \tag{3.2}$$

  donde θ es el ángulo entre la dirección del vehículo (según su orientación actual) y el vector tangente al trazado en el waypoint actual. *(Ver Figura 3.13b)*

- **Recompensa por avance** (r_{avance}):

$$r_{\text{avance}} = \frac{||\vec{p}_t - \vec{p}_{t-1}||}{\delta_{\max}} \tag{3.3}$$

  donde $\vec{p}_t$ y $\vec{p}_{t-1}$ son las posiciones actuales y anteriores del vehículo respectivamente, y $\delta_{\max}$ es el desplazamiento máximo esperado por paso. *(Ver Figura 3.13c)*



(a) Recompensa por centrado.

(b) Recompensa por orientación.

(c) Recompensa por avance.

Figura 3.13: Visualización de las tres componentes principales que conforman la función de recompensa del entorno `DeepRacerEnv`: (a) la distancia lateral al centro del carril, que penaliza salidas de la pista; (b) el alineamiento de la orientación del vehículo con el eje de la pista, fomentando trayectorias suaves; y (c) el avance hacia adelante, que recompensa el progreso continuo del agente. Estas métricas se combinan de forma ponderada para guiar el aprendizaje del agente hacia un comportamiento de conducción óptimo.

Los pesos utilizados son $w_{\text{center}} = 1$, $w_{\text{orient}} = 1$ y $w_{\text{avance}} = 2$, con el objetivo de priorizar el progreso del vehículo a lo largo del trazado sobre las demás métricas.

Dado que cada una de las recompensas parciales está normalizada en el rango $[0, 1]$, y considerando los pesos aplicados, la recompensa total está acotada en el intervalo

$$\text{total\_reward} \in [-2.5,\ 1 \cdot 1 + 1 \cdot 1 + 1 \cdot 2] = [-2.5, 4].$$

Este límite superior permite mantener la función de recompensa dentro de una escala numérica controlada y coherente durante el entrenamiento del agente. El límite inferior es la reward negativa que se da durante el entrenamiento por hacer algo indebido.

Esta función ha demostrado ser efectiva para guiar al agente hacia una conducción estable, eficiente y generalizable dentro del entorno simulado. En la **Figura 3.14** se observa la trayectoria descrita por el vehículo tras completar el entrenamiento con la función de recompensa compuesta. La trayectoria resulta continua, centrada en la pista y sin oscilaciones bruscas, lo que evidencia que el agente ha aprendido una política de navegación robusta y alineada con los objetivos deseados.

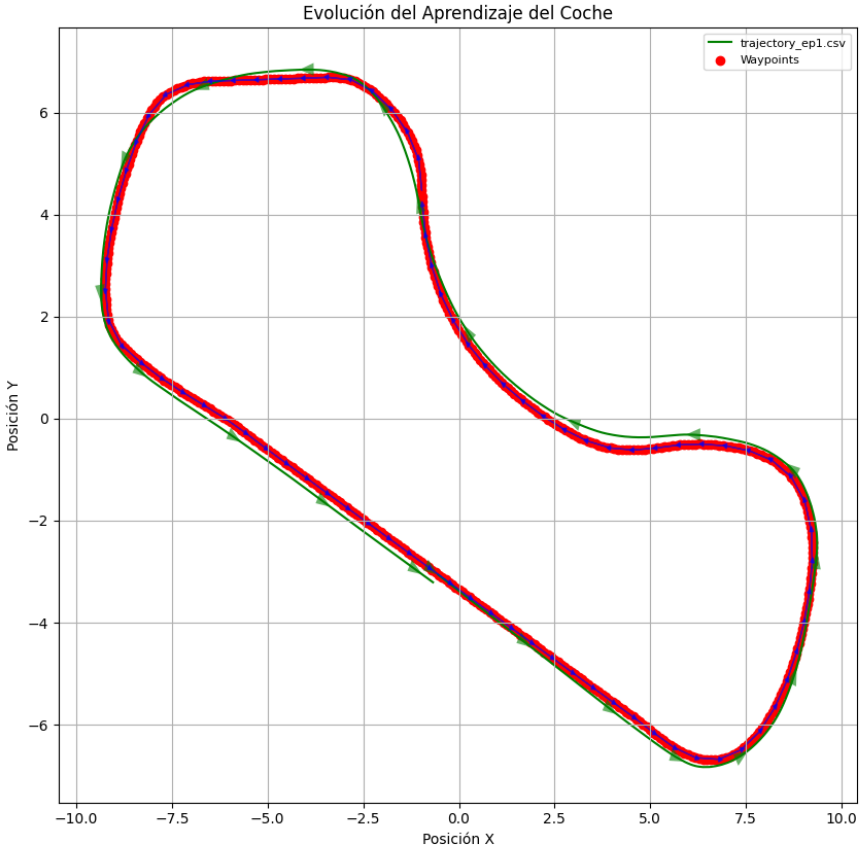
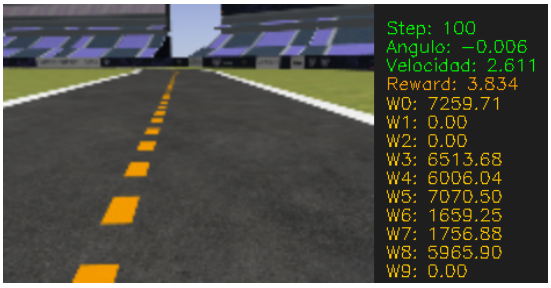


Figura 3.14: Trayectoria recorrida por el vehículo tras el entrenamiento con la función de recompensa compuesta. Se observa un desplazamiento continuo y estable, centrado en la pista y sin oscilaciones bruscas, lo que indica que el agente ha aprendido una política eficaz de conducción autónoma, alineada con los objetivos de centrado, orientación y avance definidos en la función de recompensa.

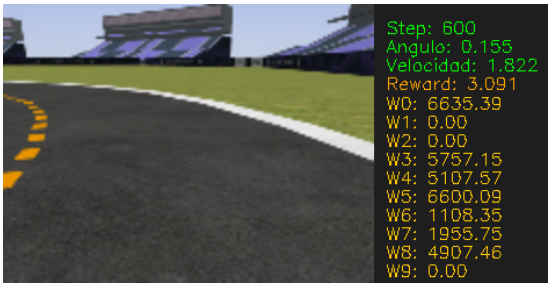
Con el fin de comprender mejor cómo el agente percibe el entorno y cómo reacciona ante distintas situaciones, se desarrolló una herramienta que permite capturar imágenes del entorno desde la perspectiva del vehículo, junto con la información procesada por el sistema en cada instante. Esto facilita un análisis cualitativo del comportamiento del agente, ya que permite visualizar tanto la observación como las acciones tomadas y la recompensa recibida.

En la **Figura 3.15** se muestran diversas capturas representativas obtenidas durante un episodio de evaluación del modelo final entrenado. Cada imagen incluye la vista frontal del entorno percibido por el agente, junto con los valores de acción generados (ángulo de dirección y velocidad) y la recompensa correspondiente en ese instante (en un color ana-

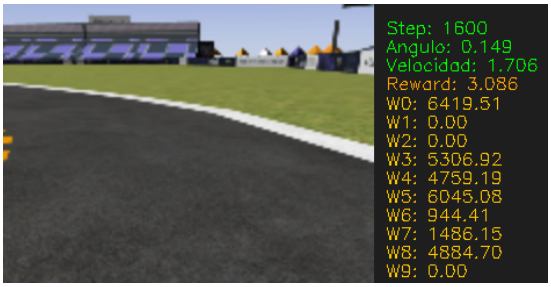
ranjado). Esta visualización permite analizar el comportamiento del sistema en situaciones reales y verificar que la función de recompensa responde adecuadamente. Al tratarse de un modelo ya entrenado, también se visualizan los pesos activos de la red neuronal en cada paso, lo que ofrece una visión más profunda del proceso de toma de decisiones del agente.



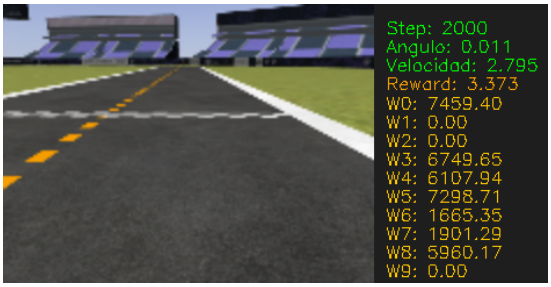
(a) Paso 100.



(b) Paso 600.



(c) Paso 1600.



(d) Paso 2000.

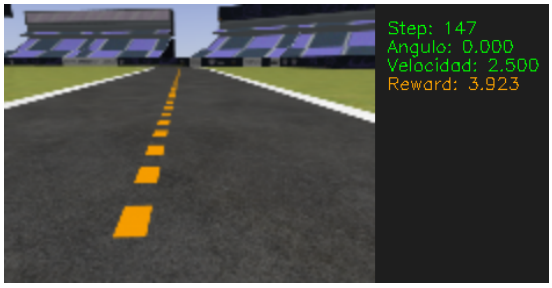
Figura 3.15: Diagnóstico visual del comportamiento del agente durante distintos pasos del episodio de evaluación. Cada subfigura muestra la imagen observada por el vehículo, junto con los valores de acción (ángulo de dirección y velocidad) y la recompensa obtenida en ese instante. Estos ejemplos ilustran cómo la función de recompensa responde ante diferentes configuraciones espaciales, permitiendo validar que el agente toma decisiones coherentes con la política aprendida y que la recompensa refuerza adecuadamente comportamientos deseados como mantener el carril, avanzar y orientarse correctamente. También se pueden observar los pesos específicos utilizados en los cálculos de la red neuronal para cada imagen.

Además, se desarrolló un script que permite controlar el vehículo de forma manual dentro del entorno simulado, con el objetivo de probar y depurar la función de recompensa de manera más precisa. Este script permite registrar, paso a paso, las observaciones, recompensas y acciones generadas al mover el agente de forma controlada, sin necesidad de un modelo entrenado.

Esta funcionalidad resultó especialmente útil durante el diseño y ajuste de la función de recompensa, ya que permitió verificar cómo respondía ante diferentes situaciones: desviaciones laterales, cambios en la orientación, proximidad a los bordes o inactividad del vehículo. El análisis detallado de estas pruebas permitió identificar errores o inconsisten-

cias en la formulación de la recompensa, facilitando su corrección antes del entrenamiento final del agente.

En la **Figura 3.16** se muestra una secuencia de situaciones concretas registradas con este método, donde se evalúa la respuesta de la función de recompensa ante distintas configuraciones espaciales del agente.

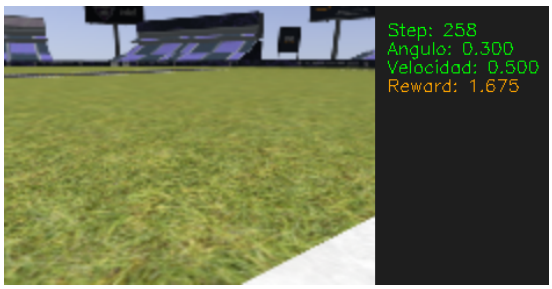


(a) En el centro y bien orientado.

(b) En el centro y mal orientado.

(c) En el centro y peor orientado.

(d) Lejos del centro y mal orientado.



(e) Más alejado del centro y mal orientado.

(f) Fuera de pista y peor orientado.

(g) Lejos del centro y bien orientado.

Figura 3.16: Evaluación cualitativa de la función de recompensa en diferentes configuraciones espaciales del agente. Se observa cómo la recompensa disminuye progresivamente a medida que el agente se desvía del centro de la pista o se orienta incorrectamente. En particular: (a) recompensa alta por estar centrado y bien orientado, además de llevar una velocidad elevada lo que implica mayor avance; (b–c) penalización por orientación deficiente, a pesar de estar centrado; (d–f) castigo severo por alejamiento del centro y mala orientación; (g) compensación parcial debido a buena orientación, aunque con posición alejada. Esta secuencia permite comprobar que la función de recompensa combina de forma efectiva la posición y orientación del agente para proporcionar señales coherentes durante el aprendizaje.

Se aprecia cómo la recompensa disminuye progresivamente a medida que el agente se desvía del centro de la pista o se orienta incorrectamente. En particular:

(a) Recompensa alta por estar centrado, bien orientado y avanzando bastante entre paso y paso.

(b–c) Penalización por orientación deficiente, a pesar de estar centrado.

(d–f) Castigo severo por alejamiento del centro y mala orientación, llegando a penalizaciones negativas (límite inferior de $total\_reward$).

(g) Compensación parcial gracias a una buena orientación, aunque el agente esté lejos del centro.

Esta secuencia permite comprobar que la función de recompensa combina de forma efectiva la posición y orientación del agente para proporcionar señales coherentes durante el aprendizaje.

Capítulo 4

Experimentación y resultados

4.1. Metodología experimental

Para evaluar la eficacia del sistema desarrollado, se diseñaron y llevaron a cabo una serie de experimentos controlados que permitieron analizar el rendimiento del agente en diferentes condiciones. Esta sección introduce la metodología general, mientras que las siguientes secciones describen de forma individual cada conjunto de experimentos realizados.

4.1.1. Circuitos empleados

Para el desarrollo del proyecto se han utilizado circuitos extraídos de un repositorio público en GitHub, el cual proporciona varios trazados listos para ser utilizados en entornos de simulación con Gazebo y ROS.

Estos circuitos fueron integrados en el entorno de simulación local mediante la importación directa de sus archivos de mundo (`.world`) y modelos (`.sdf`/`.urdf`/`.dae`). Su uso permitió ahorrar tiempo de diseño, manteniendo un entorno consistente para evaluar el desempeño del agente.

Los circuitos empleados son los siguientes y se ilustran en la **Figura 4.1**, donde se muestra una vista general de cada uno en el entorno de simulación Gazebo. La diversidad en su diseño permitió validar la capacidad de generalización del agente a distintos contextos geométricos y de navegación.

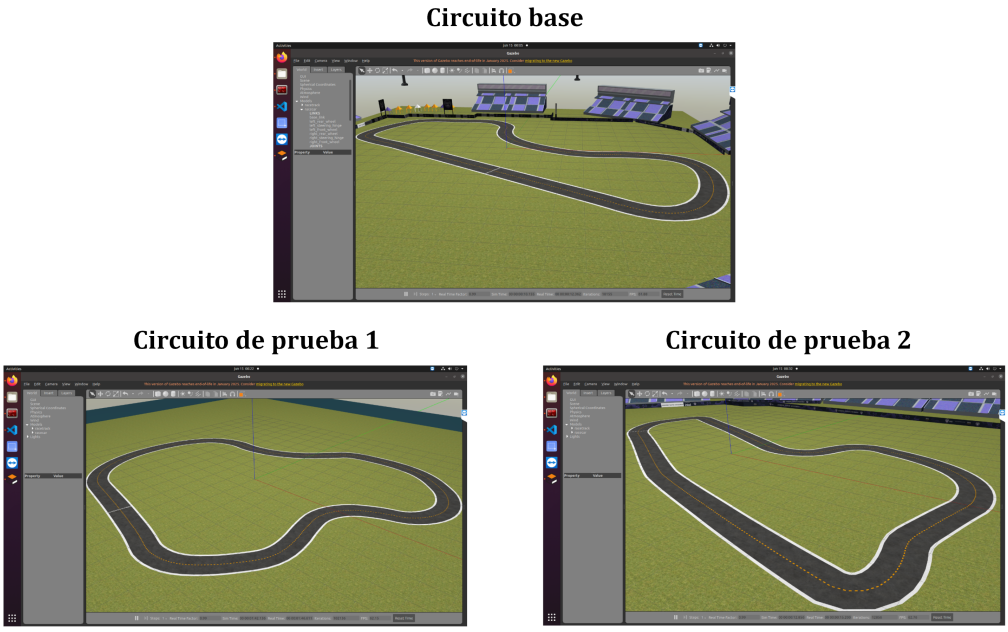


Figura 4.1: Los mapas mostrados en la Figura corresponden a los entornos virtuales empleados en las simulaciones con Gazebo.

4.1.2. Configuración del agente e hiperparámetros

El modelo ha sido entrenado mediante el algoritmo PPO (Proximal Policy Optimization), implementado con Stable Baselines3. Las observaciones del entorno consisten en imágenes, lo que motivó el uso de una política basada en convoluciones (`CnnPolicy`). Los hiperparámetros utilizados son:

- **learning_rate**: `1e-4` – tasa de aprendizaje fija para actualizar la red neuronal.
- **gamma**: `0.995` – factor de descuento que da más peso a recompensas futuras.
- **gae_lambda**: `0.92` – parámetro de suavizado para el cálculo de la ventaja generalizada (GAE).
- **n_steps**: `2048` – número de pasos a recolectar por actualización del modelo.
- **batch_size**: `128` – tamaño de los mini-lotes usados durante la optimización.
- **clip_range**: `0.2` – rango de recorte usado para estabilizar las actualizaciones de la política.
- **ent_coef**: `0.01` – coeficiente de la pérdida por entropía, que favorece la exploración.
- **vf_coef**: `0.5` – coeficiente de la pérdida del valor estimado.

- **device**: `"cuda"` – entrenamiento acelerado por GPU utilizando PyTorch.

Estos parámetros han sido elegidos tras una fase inicial de experimentación y ajuste manual, priorizando la estabilidad del aprendizaje, la eficiencia del entrenamiento y la capacidad de generalización del modelo. Aunque no se ha realizado ajuste automático de hiperparámetros (mediante optimización bayesiana o grid search), los resultados obtenidos han mostrado un rendimiento satisfactorio en los escenarios planteados. Cabe destacar que los experimentos a discutir a continuación, todos ellos asociados a la función de recompensa anteriormente descrita, se han realizado utilizando estos hiperparámetros y en el denominado *circuito base*, lo que permitió un entorno controlado para analizar el impacto individual de cada componente de la reward en el comportamiento del agente.

Además de estas métricas, cabe destacar el uso de TensorBoard para comprobar que el entrenamiento del modelo era correcto, permitiendo una visualización clara y detallada de múltiples indicadores durante el proceso de aprendizaje. Entre las métricas más relevantes que se pueden monitorear se incluyen la recompensa media por episodio y la entropía de la política. Estas gráficas facilitan la detección de comportamientos anómalos, como colapsos en la política o sobreajuste, y permiten validar que el modelo converge de manera estable hacia una solución óptima.

En la **Figura 4.2** se muestran diversas métricas extraídas con TensorBoard durante el entrenamiento del agente. Estas gráficas permiten observar la evolución de la pérdida total, la calidad de la estimación del valor (value_loss), la estabilidad de las actualizaciones (approx_kl, clip_fraction), así como la reducción progresiva de la entropía de la política, indicativa de una menor exploración.

Cabe señalar que, si bien estas métricas se han tenido en cuenta para verificar la estabilidad y eficacia del entrenamiento, no se ha profundizado en su análisis ni en el comportamiento interno del algoritmo PPO. El foco principal de este trabajo ha sido la integración del agente con el entorno simulado y la validación de su rendimiento, más que una exploración detallada de los mecanismos internos del algoritmo de optimización.

Figura 4.2: Las gráficas corresponden a métricas obtenidas durante el entrenamiento del agente, donde se puede observar la evolución de la pérdida total (total loss), la calidad de la estimación del valor (value loss), la estabilidad de las actualizaciones del modelo (aproximación de KL y fracción de clip), la entropía de la política que refleja el grado de exploración, y la varianza explicada (explained variance) que indica la capacidad del crítico para modelar correctamente el entorno.

En la **Figura 4.3** se muestra un ejemplo de las gráficas generadas por TensorBoard, a partir de los datos registrados automáticamente por la librería Stable Baselines3. Estas gráficas resultan fundamentales para evaluar el rendimiento del agente y ajustar los hiperparámetros del modelo de forma informada.

Figura 4.3: Visualización de métricas clave durante el entrenamiento del agente mediante TensorBoard, incluyendo la recompensa media por episodio en evaluación, así como su duración.

Como se puede observar en esta gráfica, conforme avanzan los episodios las evaluaciones obtienen mejores resultados, estos se pueden observar tanto en la duración de los episodios como en la recompensa media de estos. Además se puede llegar a la conclusión de que un episodio largo es sinónimo de una reward igualmente alta.

4.1.3. Métricas de evaluación

Para valorar objetivamente el rendimiento del agente se han utilizado las siguientes métricas cualitativas y cuantitativas:

- **Velocidad media de avance:** Se observa si el agente es capaz de conducir a una velocidad adecuada, evitando comportamientos ineficientes como detenerse sin motivo o avanzar de forma extremadamente lenta. Aunque la velocidad no se impone como una restricción, se espera una conducción fluida y constante.
- **Recompensa acumulada por episodio:** Se analiza la recompensa total obtenida en cada episodio. Una tendencia creciente durante el entrenamiento indica que el agente está aprendiendo una política más eficaz. Esta métrica también se utiliza para comparar distintas versiones del modelo.
- **Duración de la trayectoria:** Se mide cuánto tiempo (en pasos de simulación) es capaz de mantenerse el agente conduciendo de forma válida antes de salirse o cometer un error crítico. Cuanto mayor es la duración, mejor se considera el desempeño del agente.

- **Distribución de acciones y recompensa por paso:** Se han registrado las acciones ejecutadas por el agente (como velocidad lineal y angular) en cada paso de simulación, junto con la recompensa obtenida. El análisis de estos datos permite identificar patrones de comportamiento y evaluar la estabilidad y coherencia de la política aprendida.
- **Trayectoria recorrida sobre el circuito:** Se ha visualizado el recorrido real efectuado por el agente sobre el trazado, lo que permite comparar su comportamiento frente a la trayectoria ideal. Este tipo de representación es especialmente útil para detectar desviaciones sistemáticas, oscilaciones o errores de guiado.

Estas métricas combinan evaluación automática y análisis cualitativo, permitiendo tener una visión completa del rendimiento del agente en tareas de conducción simulada. La **Figura 4.4** muestra la evolución de las acciones ejecutadas por el agente (velocidad lineal y angulación de las ruedas) y la recompensa obtenida paso a paso durante un episodio de evaluación. Este gráfico permite observar la coherencia del comportamiento aprendido.

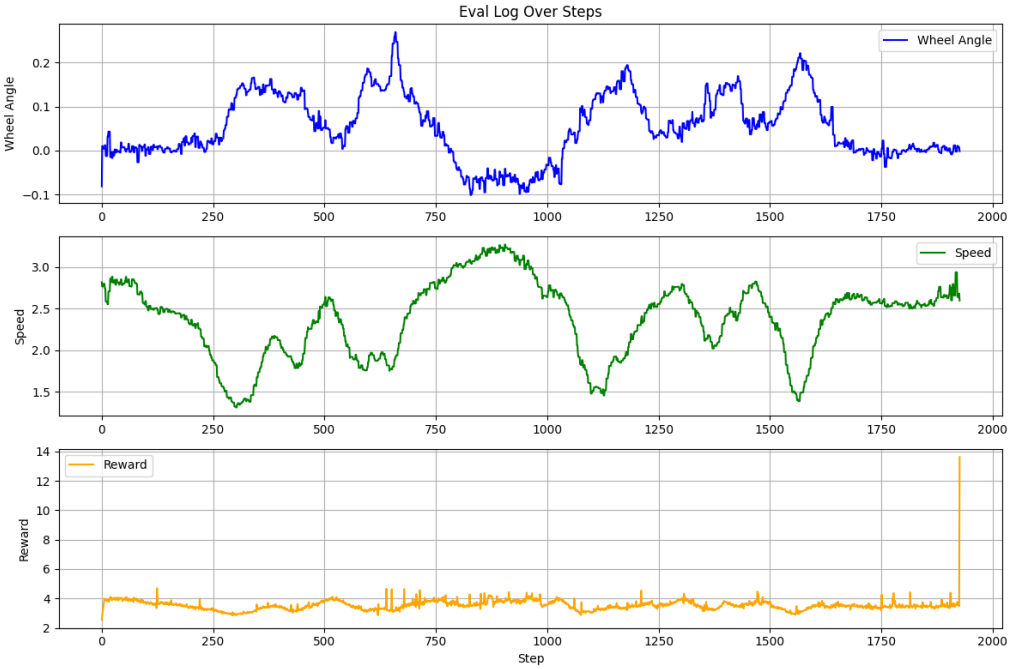


Figura 4.4: La Figura ilustra cómo el agente adapta progresivamente sus acciones, modificando la velocidad y el ángulo de las ruedas, mientras maximiza la recompensa recibida. Esta visualización se ha extraído de una prueba de ejecución de un modelo con la reward comentada. Se puede apreciar que el ángulo de las ruedas (gráfico superior) presenta variaciones coherentes con los ajustes necesarios en la trayectoria, la velocidad (gráfico medio) oscila en función del entorno y la política del agente, y la recompensa (gráfico inferior) refleja picos en los momentos de desempeño más eficiente, el pico final es una recompensa extra por haber completado la vuelta a la pista exitosamente. Esto indica una política de control que busca estabilidad y maximización de la recompensa acumulada.

Por otra parte, la **Figura 4.5** representa la trayectoria real seguida por el agente durante un episodio en el entorno simulado. Esta visualización permite detectar de forma clara si el agente sigue el centro del carril, si corrige adecuadamente en curvas o si presenta oscilaciones. Una trayectoria eficiente y estable se aproxima a una línea fluida y centrada. En este caso se puede observar claramente una evolución en los intentos que ha ido haciendo el agente, desde trayectorias muy curvadas hasta una muy correcta y fluida que completa el recorrido.

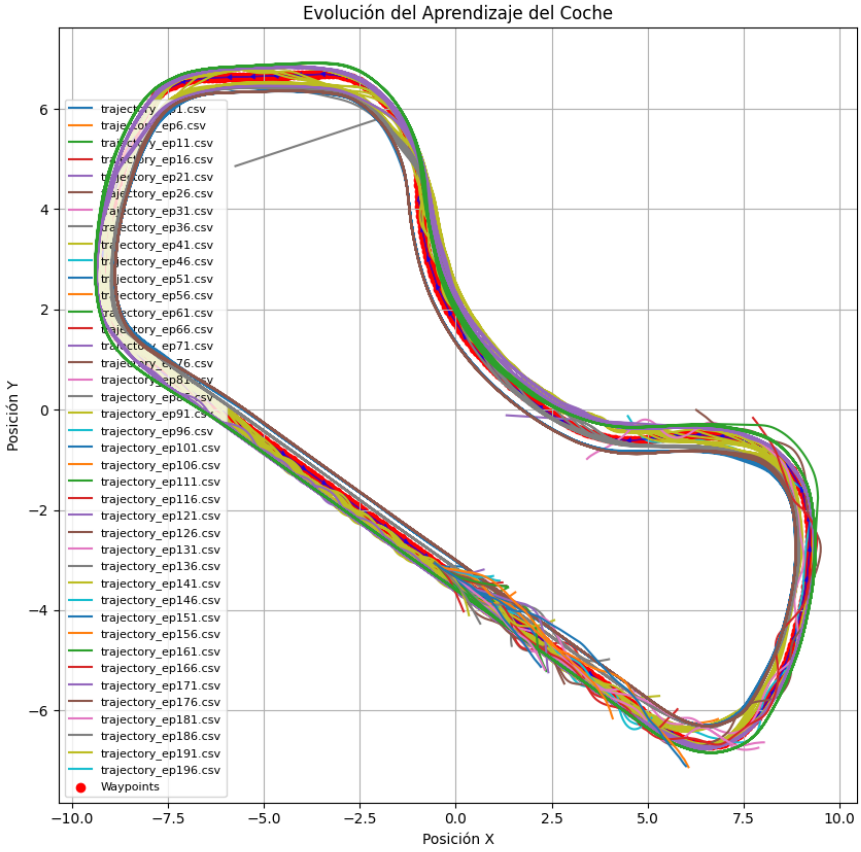


Figura 4.5: La Figura muestra claramente la mejora progresiva en la trayectoria del agente, evidenciando su capacidad para aprender a seguir el centro del carril y realizar correcciones suaves en las curvas, lo que se traduce en un comportamiento de conducción cada vez más estable y eficiente.

4.2. Conjunto de experimentos

Como ya se ha comentado, se ha logrado programar una función de recompensa que proporciona resultados satisfactorios, permitiendo que el agente aprenda a conducir de forma estable y eficiente en los circuitos simulados. Sin embargo, antes de alcanzar esta configuración óptima, se realizaron otros experimentos que no lograron resultados exitosos o concluyentes.

Estos intentos iniciales, aunque no alcanzaron el desempeño esperado, resultaron fundamentales para comprender mejor el comportamiento del agente y para guiar el diseño

final de la función de recompensa y la configuración del entrenamiento. Por esta razón, también es importante destacar estos experimentos fallidos o con resultados subóptimos, ya que aportan valiosas lecciones sobre la sensibilidad del aprendizaje a distintos parámetros y estructuras de recompensa.

En esta sección se describen tanto los experimentos previos que no lograron buenos resultados, como otros experimentos de interés realizados detallando las causas principales y los aprendizajes derivados.

4.2.1. Estudio de ablación de la función de recompensa

Con el fin de validar el correcto funcionamiento de la función de recompensa propuesta y entender en profundidad las razones detrás de su efectividad, se ha realizado un estudio de ablación. Este tipo de análisis permite aislar el impacto de cada uno de los componentes que conforman la recompensa total, evaluando su contribución específica al comportamiento del agente durante el entrenamiento.

La motivación principal de este estudio no es únicamente comparar configuraciones, sino evidenciar por qué la combinación propuesta logra inducir un comportamiento de conducción activo, estable y eficiente. Para ello, se han eliminado selectivamente diferentes términos de la función de recompensa, manteniendo constantes el resto de variables del entorno, y se ha observado el efecto de cada configuración sobre diversas métricas de rendimiento.

Se consideraron tres componentes clave:

- r_{centro}: recompensa basada en la distancia al centro de la pista.
- $r_{\text{orientación}}$: recompensa basada en la orientación del vehículo respecto al eje del trazado.
- r_{avance}: recompensa basada en el avance del agente entre pasos.

Cada configuración fue entrenada desde cero durante 50000 timesteps, además de tener todos los pesos de sus recompensas aisladas igualados a 1, y evaluada en el mismo circuito, bajo condiciones idénticas y probadas durante 5 ejecuciones distintas cada una de ellas. Las métricas utilizadas para la evaluación fueron:

- **Distancia media al centro de la pista (m)**: cuanto menor, mejor centrado está el vehículo (la anchura de la pista sobre la que se han hecho las pruebas es de 0.991 m, es decir, la distancia máxima al centro es la mitad, 0.4955).
- **Velocidad media (m/s)**: refleja la fluidez del movimiento (siendo la velocidad máxima 5 m/s).

- **Tiempo de ejecución (s)**: representa la eficiencia general y el tiempo durante el que se ejecuta el modelo, este puede terminar al completar una vuelta o al activar alguno de los flags de terminación (como puede ser quedarse demasiado tiempo quieto o con una velociadad muy reducida).
- **Porcentaje de vuelta completada (%)**: mide la capacidad del agente para terminar el circuito sin errores críticos.

Componentes activas	**Distancia media al centro (m)**	**Velocidad media (m/s)**	**Tiempo de ejecución (s)**	**Vuelta completada (%)**
Solo r_{centro}	0.122 ± 0.003	0.063 ± 0.000	468.8 ± 8.125	$16.49\,\% \pm 0.001$
Solo $r_{\text{orientación}}$	0.016 ± 0.000	0.000 ± 0.000	14.184 ± 2.622	$0.00\,\% \pm 0.000$
Solo r_{avance}	0.238 ± 0.002	2.184 ± 0.003	105.118 ± 0.855	$100.00\,\% \pm 0.000$
$r_{\text{centro}} + r_{\text{orientación}}$	0.016 ± 0.000	0.000 ± 0.000	13.56 ± 0.346	$0.00\,\% \pm 0.000$
$r_{\text{centro}} + r_{\text{avance}}$	0.158 ± 0.002	0.312 ± 0.001	744.536 ± 2.647	$100.00\,\% \pm 0.000$
$r_{\text{orientación}} + r_{\text{avance}}$	0.378 ± 0.002	0.519 ± 0.002	470.826 ± 9.698	$100.00\,\% \pm 0.000$
Todas las componentes	0.132 ± 0.003	2.371 ± 0.016	103.23 ± 2.113	$100.00\,\% \pm 0.000$

Tabla 4.1: Resultados cuantitativos del estudio de ablación de la función de recompensa

A partir de los resultados recogidos en la **Tabla 4.1**, se pueden extraer varias conclusiones significativas sobre el papel que juega cada componente de la función de recompensa en el aprendizaje del agente:

- Los modelos entrenados únicamente con r_{centro} o $r_{\text{orientación}}$ no fueron capaces de completar ninguna vuelta (16.49 % y 0 %, respectivamente). En el caso de r_{centro}, el agente logra mantener una distancia media aceptable al centro (0.122 m), pero con una velocidad media muy baja (0.063 m/s). Para $r_{\text{orientación}}$, la distancia media es notablemente mejor (0.016 m), pero la velocidad es nula, indicando que el agente permanece inmóvil. Esto sugiere que, por sí solos, estos componentes no inducen desplazamiento real.
- El término r_{avance} demuestra ser el principal motor del comportamiento activo. El modelo que lo emplea de forma aislada completa el 100 % del circuito, con una alta velocidad media (2.184 m/s), a pesar de tener una distancia al centro relativamente alta (0.238 m). Esto evidencia que, sin restricciones adicionales, el agente puede priorizar el avance sin preocuparse por la estabilidad o el centrado.

- La combinación $r_{\text{centro}} + r_{\text{orientación}}$ obtiene una distancia al centro baja (0.015 m, una distancia despreciable dentro del margen que se considera estar centrado en la pista), pero la velocidad es nula y el tiempo de ejecución es 13.56 s, lo que indica que el agente simplemente no se mueve. Este comportamiento trivial maximiza la recompensa sin desplazarse, lo que pone de relieve la necesidad de un término explícito que promueva el movimiento. El episodio de prueba del modelo termina al no haber avanzado ningún waypoint en ese tiempo.

- Las combinaciones que incluyen r_{avance} junto con otras componentes logran completar el 100 % de las vueltas. Sin embargo, presentan diferentes compromisos en términos de velocidad y centrado. Por ejemplo, la combinación $r_{\text{centro}} + r_{\text{avance}}$ reduce la distancia al centro a 0.158 m, aunque con una velocidad media baja (0.312 m/s) y el mayor tiempo de ejecución (744.536 s), lo que sugiere un comportamiento prudente. En cambio, $r_{\text{orientación}} + r_{\text{avance}}$ muestra un desempeño con una mayor velocidad (0.519 m/s) y tiempo moderado (470.826 s), a costa de una mayor desviación del centro (0.378 m).

- El modelo con todas las componentes activas obtiene resultados globalmente superiores: completa el circuito en el menor tiempo (103.23 s), con la mayor velocidad media (2.371 m/s) que es prácticamente la mitad de la velocidad que puede alcanzar el robot y una distancia al centro razonablemente baja (0.132 m). Esto sugiere que la combinación balanceada de incentivos permite un comportamiento eficiente y controlado.

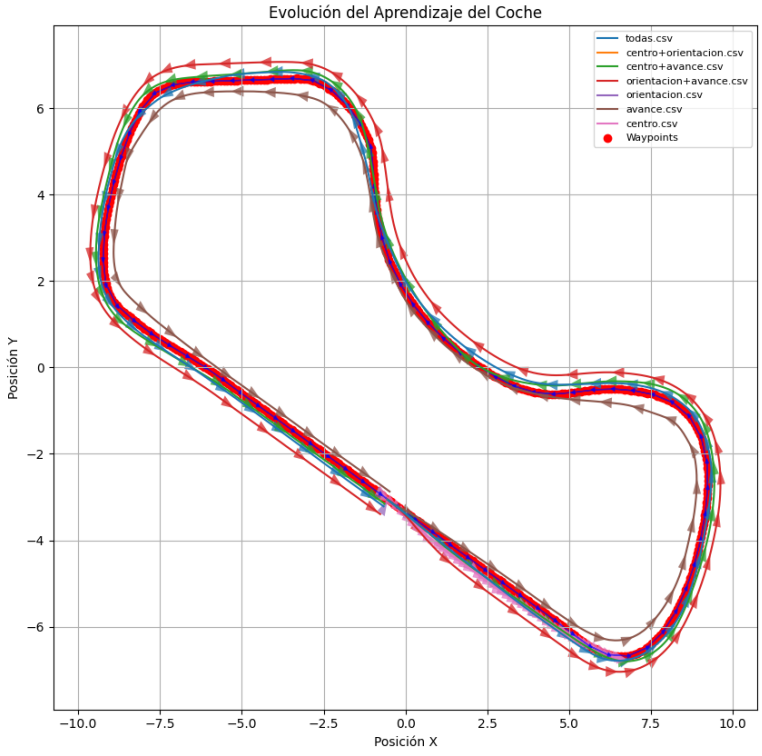
El estudio de ablación ha demostrado que la función de recompensa diseñada cumple eficazmente su propósito: guiar al agente hacia un comportamiento de conducción autónoma que sea tanto activo como estable y preciso. La evaluación sistemática de cada componente revela que el término de **avance** es indispensable para inducir movimiento y completar el circuito, actuando como motor principal del aprendizaje. Sin embargo, su uso aislado tiende a generar trayectorias rápidas pero menos centradas ya que esto no es una necesidad en la recompensa.

Los términos de **centrado** y **orientación**, por otro lado, aunque no suficientes por sí solos para inducir desplazamiento, aportan restricciones esenciales que estabilizan y afinan el comportamiento del agente. Cuando se combinan con el término de avance, permiten equilibrar la recompensa entre velocidad, trayectoria y precisión, lo que se refleja en métricas notablemente mejores.

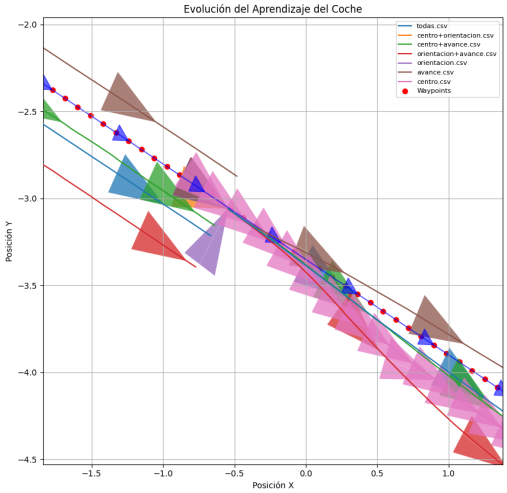
La configuración completa, que incluye los tres componentes, es la única que logra un desempeño alto y equilibrado en todas las métricas evaluadas. Esta evidencia empírica valida que la función de recompensa no solo está correctamente formulada, sino que su funcionamiento se fundamenta en la sinergia entre incentivos complementarios. En conjunto, este estudio confirma que un diseño de recompensa bien balanceado no solo promueve la acción, sino que también asegura una conducción eficaz, segura y generalizable.

Para complementar el análisis cuantitativo presentado en la **Tabla 4.1**, en la **Figura 4.6** se muestran las trayectorias representativas obtenidas durante las pruebas de cada configuración del estudio de ablación. Estas trayectorias incluyen la vuelta completa, la salida del circuito y tres curvas características, permitiendo observar de forma visual y cualitativa cómo influyen los distintos componentes de la función de recompensa en el comportamiento del agente.

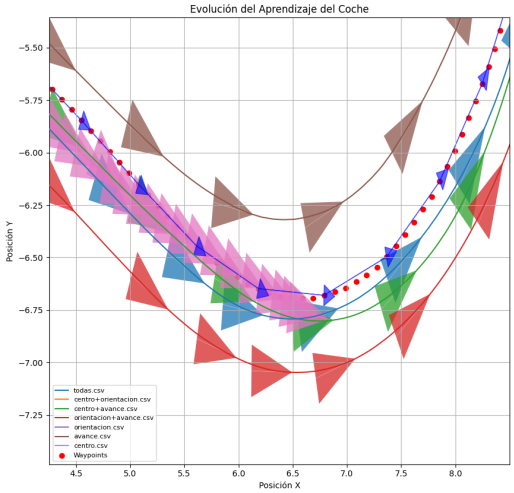
La comparación gráfica facilita la comprensión del impacto que tiene cada término sobre la capacidad del agente para seguir el trazado de la pista, su estabilidad lateral y su fluidez de conducción. De esta manera, se puede corroborar visualmente la interpretación de las métricas, evidenciando las diferencias en la precisión y el estilo de conducción entre los modelos entrenados con distintos componentes activos.



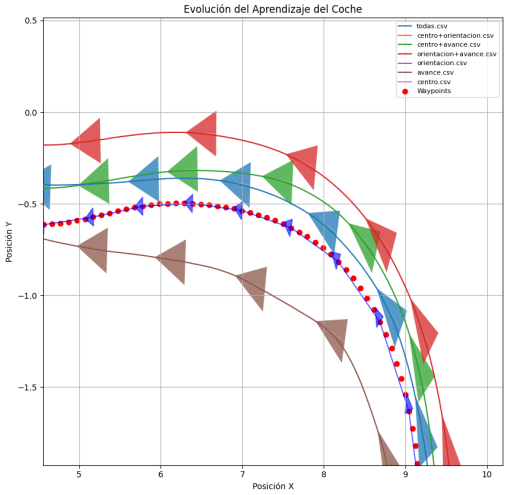
(a) Trayectoria completa del circuito.



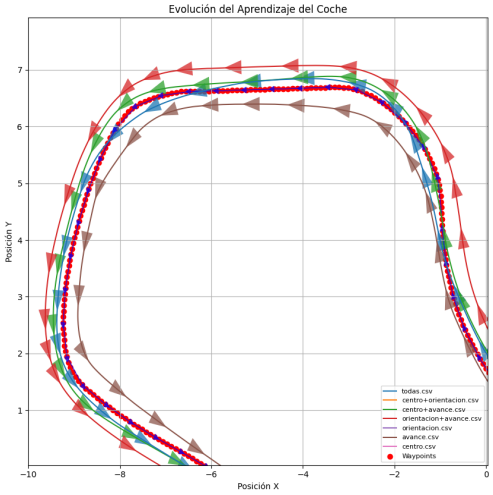
(b) Salida del circuito aumentada.



(c) Primera curva aumentada.



(d) Segunda curva aumentada.



(e) Tercera curva aumentada.

Figura 4.6: Trayectorias representativas del agente bajo diferentes configuraciones de la función de recompensa. La figura muestra: (a) la vuelta completa, (b) la salida del circuito y (c)-(e) tres curvas seleccionadas del trazado.

Como se refleja en las trayectorias mostradas en la **Figura 4.6a**, se confirma lo observado en las métricas anteriores: el modelo entrenado solo con r_{centro} abandona la prueba prematuramente, mostrando un avance muy limitado; el modelo con únicamente $r_{\text{orientación}}$ permanece prácticamente inmóvil (triángulo direccional visible en dirección incorrecta en

la **Figura 4.6b**), sin desplazarse a lo largo del circuito; la combinación de ambas sin el término de avance no logra movimiento alguno y termina rápidamente (mismo caso que el anterior); mientras que las configuraciones que incluyen r_{avance} permiten completar la vuelta completa con diferentes estilos de conducción, desde un comportamiento más conservador y lento cuando se combina con r_{centro}, hasta trayectorias más rápidas pero menos centradas con $r_{\text{orientación}}$. Finalmente, la configuración con todas las recompensas activas consigue un equilibrio óptimo, reflejado en una conducción eficiente, rápida y razonablemente centrada, corroborando la importancia de la combinación sinérgica de los tres términos en la función de recompensa.

Este comportamiento también se aprecia con mayor detalle en las figuras ampliadas: en la **Figura 4.6c**, correspondiente a la primera curva del circuito, se evidencia cómo los modelos que incluyen r_{avance} son capaces de tomar la curva con distintas trayectorias, algunas más amplias pero continuas; en la **Figura 4.6d** se aprecia la capacidad de anticipación del modelo con todas las recompensas activas, que toma una línea más fluida y centrada en comparación con los demás; y en la **Figura 4.6e**, correspondiente a la última curva antes de cerrar la vuelta, se observa nuevamente cómo dicho modelo logra mantener una trayectoria estable y balanceada, completando el recorrido con éxito. Estas observaciones refuerzan el valor del diseño cuidadoso de la función de recompensa para lograr un comportamiento robusto y eficaz en tareas de conducción autónoma.

4.2.2. Capacidad de generalización

Para que un agente de conducción autónoma sea realmente útil, no basta con que funcione adecuadamente en un único entorno de entrenamiento. Es crucial que sea capaz de generalizar su comportamiento a situaciones nuevas o no vistas durante el aprendizaje, adaptándose a variaciones del entorno sin requerir una reentrenamiento completo.

Con este objetivo, se entrenó al agente exclusivamente en un **circuito base**, y posteriormente se evaluó su rendimiento en dos **circuitos distintos**, con geometrías diferentes pero comparables en términos de escala y dificultad. Para completar el análisis, también se probó al agente en los mismos tres circuitos (el de entrenamiento y los dos nuevos) pero **en sentido contrario**, es decir, recorriéndolos en dirección opuesta a la usada durante el entrenamiento.

Estas pruebas permiten evaluar no solo la capacidad del agente para enfrentarse a nuevos escenarios, sino también su habilidad para adaptarse a inversiones topológicas del entorno, que implican variaciones en las curvas, puntos de decisión y distribución de waypoints.

Las métricas empleadas para la evaluación fueron las mismas utilizadas en el estudio de ablación: distancia media al centro de la pista, velocidad media, tiempo de ejecución y porcentaje de vuelta completada.

Otros circuitos

Para analizar la capacidad del agente de transferir su comportamiento a nuevos trazados, se realizaron pruebas en varios circuitos diferentes a aquel utilizado durante el entrenamiento. Estos circuitos presentaban variaciones en la geometría.

Los resultados mostraron que el agente era capaz de avanzar en todos los circuitos, evitando quedarse bloqueado, lo que demuestra una cierta capacidad de locomoción genérica. Estas evidencias apuntan a que el modelo tiene cierta capacidad de generalización estructural (mantiene su política de avance y evita comportamientos triviales como detenerse).

Como se muestra en la **Figura 4.7**, el agente es capaz de generar trayectorias exitosas en diferentes circuitos de prueba, lo que evidencia su capacidad para adaptarse a variaciones en la geometría de los trazados.

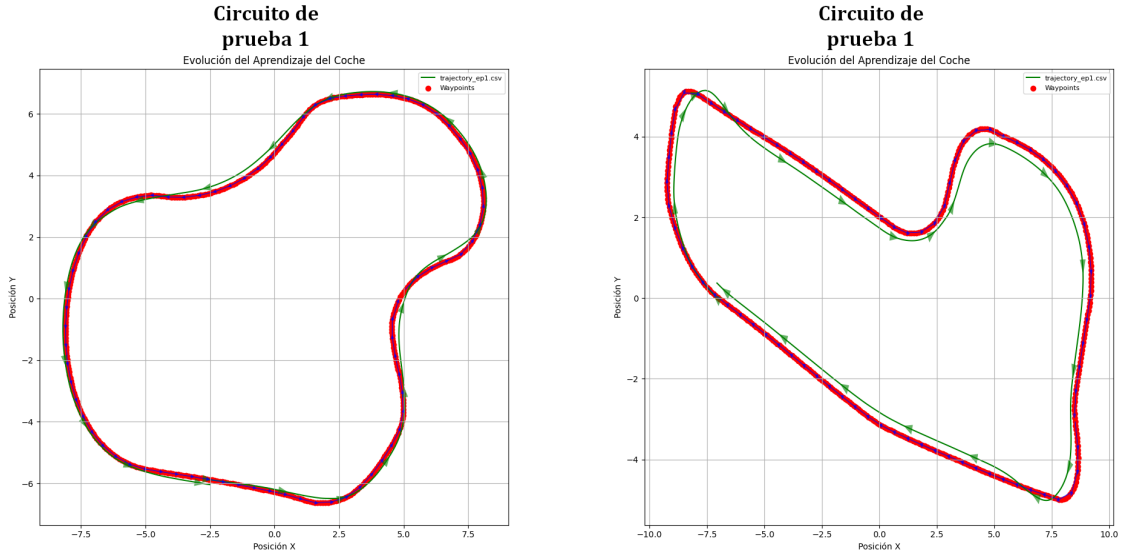


Figura 4.7: Las trayectorias ilustradas en la Figura demuestran que el agente puede generalizar su política aprendida para mantener un comportamiento de conducción coherente y efectivo en circuitos no vistos durante el entrenamiento, adaptándose a diferentes configuraciones geométricas.

Los resultados cuantitativos de estas pruebas se recogen en la **Tabla 4.2**, donde puede observarse que el agente mantiene una velocidad estable y completa la vuelta con éxito en ambos trazados, aunque presenta diferencias en el centrado, lo que refleja la variabilidad geométrica entre circuitos. Se realizaron cinco ejecuciones independientes en cada circuito para calcular las medias y desviaciones estándar correspondientes.

Circuito	Distancia media al centro (m)	Velocidad media (m/s)	Tiempo de ejecución (s)	Vuelta completada (%)
Circuito de prueba 1	0.115 ± 0.002	2.324 ± 0.015	98.39 ± 2.587	$100.00\,\% \pm 0.000$
Circuito de prueba 2	0.239 ± 0.003	2.216 ± 0.012	106.03 ± 2.104	$100.00\,\% \pm 0.000$

Tabla 4.2: Resultados cuantitativos en otros circuitos con el modelo entrenado (todas las componentes activas)

Desde el punto de vista cuantitativo, los resultados obtenidos reflejan un comportamiento coherente con el objetivo de navegación eficiente:

- En el primer circuito, el agente logró completar una vuelta en 98.39 segundos, con una velocidad media de 2.324 m/s y una distancia media al centro de tan solo 0.115 m, lo que indica una conducción rápida y centrada, incluso en un entorno no visto durante el entrenamiento.
- En el segundo circuito, con curvas más cerradas, se observa una ligera disminución del rendimiento: la velocidad media descendió a 2.216 m/s, la distancia media al centro aumentó a 0.239 m, y el tiempo de vuelta se elevó a 106.03 segundos. Este comportamiento sugiere que el agente tiene más dificultades para mantener una trayectoria óptima en geometrías más exigentes, aunque sigue siendo capaz de completar el recorrido pese a tener alguna trayectoria un poco peor de lo esperado.

En conjunto, estos resultados refuerzan la hipótesis de que el agente ha aprendido una política de navegación basada en patrones generales de la pista, y no simplemente una secuencia de acciones memorística. Esta capacidad de generalización es un indicador positivo de la robustez del modelo aprendido.

Sentido contrario

Una buena prueba para comprobar la robustez del modelo consiste en invertir el sentido de la marcha en el mismo circuito en el que fue entrenado. Al hacer esto, los patrones espaciales y visuales que el agente encuentra son distintos, y se requiere una capacidad de adaptación a una distribución de observaciones diferente.

En la **Figura 4.8** se muestran las trayectorias generadas por el agente al recorrer todos los circuitos en sentido contrario. A pesar del cambio en la dirección, se observa que el comportamiento sigue siendo coherente y la trayectoria se mantiene estable y centrada, lo cual indica que el modelo ha aprendido una política que no depende exclusivamente

de secuencias visuales o patrones en una dirección fija, sino que posee cierto grado de generalización espacial.

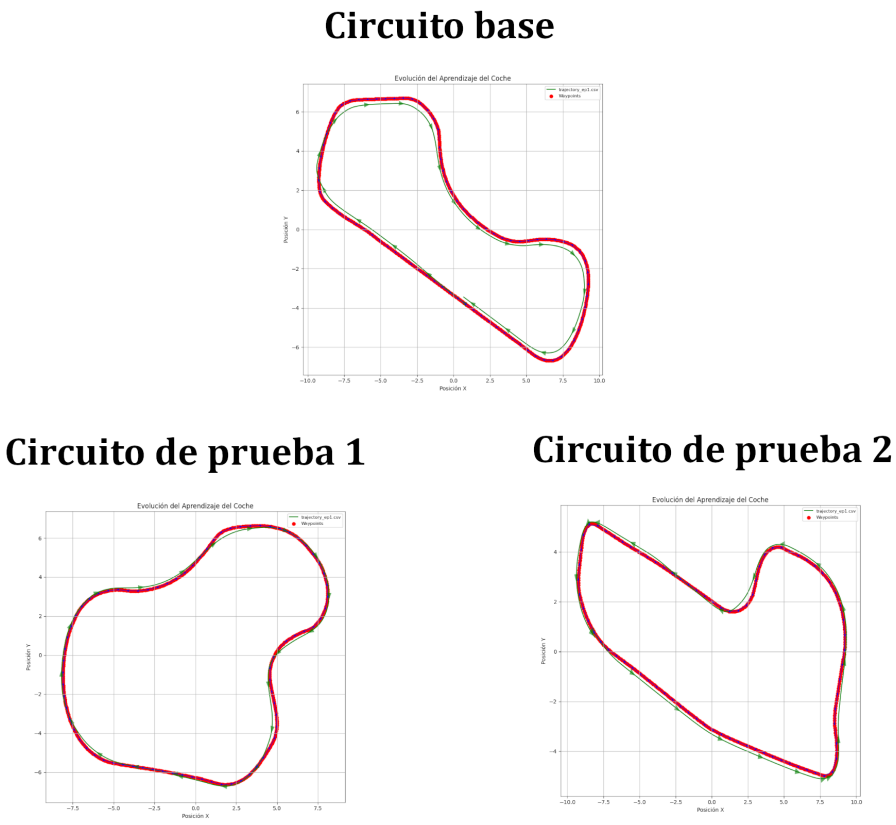


Figura 4.8: Las trayectorias en sentido contrario ilustran la capacidad del agente para mantener un comportamiento consistente y adaptativo, demostrando que su política no está restringida a una única dirección de recorrido y presenta generalización frente a cambios en la distribución visual y espacial del entorno.

Los resultados cuantitativos se presentan en la **Tabla 4.3**. Todas las vueltas fueron completadas satisfactoriamente en los tres circuitos, lo que confirma que el agente mantiene un desempeño robusto bajo inversión de dirección. Las métricas de velocidad media y tiempo de ejecución son comparables a las obtenidas en el sentido original, aunque se observan ligeras variaciones en la distancia media al centro. Al igual que en los demás experimentos, se llevaron a cabo cinco ejecuciones para obtener los datos.

Estas diferencias pueden atribuirse a la asimetría geométrica de los trazados o a la distribución distinta de curvas y obstáculos visuales en el nuevo sentido. Aun así, el agente es capaz de adaptarse sin necesidad de ser reentrenado, lo cual constituye una evidencia clara de su capacidad de generalización y resiliencia en escenarios no vistos durante el entrenamiento.

Circuito	**Distancia media al centro (m)**	**Velocidad media (m/s)**	**Tiempo de ejecución (s)**	**Vuelta completada (%)**
Circuito base	0.241 ± 0.003	2.529 ± 0.014	92.48 ± 1.755	$100.00\,\% \pm 0.000$
Circuito de prueba 1	0.107 ± 0.002	2.595 ± 0.017	88.27 ± 1.632	$100.00\,\% \pm 0.000$
Circuito de prueba 2	0.150 ± 0.002	2.181 ± 0.013	110.17 ± 2.749	$100.00\,\% \pm 0.000$

Tabla 4.3: Resultados cuantitativos en otros circuitos con el modelo entrenado (todas las componentes activas)

Los resultados obtenidos al invertir el sentido de la marcha evidencian que el agente ha desarrollado una política de navegación robusta y generalizable. La capacidad de completar exitosamente todas las vueltas en dirección contraria, sin necesidad de reentrenamiento y manteniendo un rendimiento comparable en términos de velocidad y estabilidad, demuestra que el modelo no se limita a memorizar trayectorias específicas, sino que ha aprendido una representación más abstracta del entorno. Esto refuerza la validez del enfoque propuesto y su aplicabilidad en escenarios no vistos o con ligeras modificaciones respecto al entorno de entrenamiento.

Capítulo 5

Conclusiones y líneas futuras

Este proyecto ha consistido en el desarrollo de un agente autónomo de conducción basado en Aprendizaje por Refuerzo, entrenado en un entorno simulado con ROS y Gazebo, utilizando imágenes como entrada principal de la red neuronal. La arquitectura implementada permite cerrar el ciclo entre simulación, percepción y acción, mediante un flujo de datos en tiempo real que integra ROS, Gazebo, Gym y Stable Baselines3.

Entre los logros más destacados se encuentra la correcta integración del entorno de simulación con el agente de control basado en PPO, así como el diseño de una función de recompensa eficaz que guía el aprendizaje hacia una conducción centrada, fluida y eficiente. El estudio de ablación realizado demuestra empíricamente la importancia de cada componente de esta recompensa y respalda las decisiones de diseño adoptadas. A nivel técnico, se ha conseguido que el agente aprenda comportamientos razonables de conducción con entrada visual, lo cual supone un reto importante por la complejidad inherente al espacio de observaciones y acciones.

No obstante, el proyecto también ha presentado dificultades importantes. El ajuste de hiperparámetros fue especialmente sensible, y la falta de herramientas para evaluar automáticamente la calidad del comportamiento del agente exigió una supervisión manual intensiva. Asimismo, el entorno de simulación no siempre ofreció estabilidad total, lo cual requirió solucionar diversos problemas de compatibilidad y sincronización.

Además de los resultados cuantitativos obtenidos durante la fase de evaluación —como las recompensas medias, la velocidad media alcanzada o el número de episodios completados con éxito—, el desarrollo del proyecto ha generado una serie de resultados cualitativos que también son relevantes y reflejan el valor del trabajo realizado. Se ha conseguido una integración funcional entre distintas tecnologías de código abierto (`Gazebo`, `ROS`, `Gymnasium`, `Stable Baselines3`) dentro de un flujo de entrenamiento completamente local. Este logro implica resolver incompatibilidades entre librerías, gestionar la sincronización entre simulador y agente, y asegurar una comunicación eficiente entre los distintos

componentes del sistema. A diferencia de plataformas comerciales como AWS DeepRacer, que dependen de la nube, esta solución es completamente local, lo que permite una total autonomía del usuario, evita costes asociados y ofrece un control completo sobre el entorno y el código.

Otro resultado destacable ha sido la construcción de un entorno de simulación personalizable y modular. Tanto el vehículo como los circuitos fueron diseñados para permitir modificaciones sencillas, como la incorporación de nuevos sensores, la introducción de diferentes trazados o la alteración de las condiciones físicas del entorno. Esto facilita la experimentación futura sin necesidad de rehacer el sistema desde cero.

A nivel de comportamiento, se ha observado que el agente entrenado no solo aprende a seguir el trazado de un circuito específico, sino que tiende a desarrollar un estilo de conducción fluido y adaptable. En pruebas cualitativas realizadas en circuitos no vistos o invertidos, el vehículo mostró una capacidad razonable para mantener la trayectoria y reaccionar ante curvas o bifurcaciones de forma coherente, aunque con un rendimiento algo inferior al del circuito original. Este comportamiento sugiere que el modelo no memorizó trayectorias concretas, sino que captó ciertos patrones generales de navegación, lo cual indica una incipiente capacidad de generalización.

Cabe destacar también el valor formativo del proceso. El desarrollo completo del sistema desde cero, ejecutado exclusivamente en local, ha requerido adquirir y aplicar conocimientos avanzados sobre robótica, simulación, inteligencia artificial y arquitectura de software. Este aprendizaje transversal y profundo se traduce no solo en un sistema funcional, sino en una comprensión sólida que permitirá escalar o adaptar el proyecto con mayor eficacia en el futuro.

Pese a los avances logrados, el sistema desarrollado presenta algunas limitaciones conocidas que conviene señalar. La política aprendida por el agente, al haber sido entrenada en un entorno relativamente controlado y específico, podría no generalizar adecuadamente a zonas no vistas si no se introducen técnicas adicionales como el *domain randomization*[1]. Además, los entrenamientos se realizaron siempre desde el mismo punto de partida, lo cual puede limitar la diversidad de experiencias que el agente acumula durante el aprendizaje y reducir su robustez ante situaciones inesperadas o variantes iniciales.

En cuanto a la valoración de los objetivos alcanzados, los principales propósitos definidos al inicio del proyecto se han cumplido satisfactoriamente. Se ha desarrollado una plataforma de conducción autónoma completamente local, empleando exclusivamente herramientas de código abierto y prescindiendo de servicios en la nube, lo que ha aportado flexibilidad, reducción de costes y control total sobre el entorno de experimentación. Además, se ha entrenado con éxito un agente de conducción autónoma mediante Aprendizaje por Refuerzo, utilizando como entrada imágenes captadas por una cámara frontal. El

[1]Técnica utilizada en Aprendizaje por Refuerzo que consiste en variar aleatoriamente aspectos del entorno simulado (como texturas, iluminación o dinámicas) durante el entrenamiento, para mejorar la capacidad de generalización del agente a entornos reales o desconocidos.

agente ha demostrado un comportamiento coherente, estable y generalizable en entornos simulados, siendo capaz de mantenerse centrado en la pista, avanzar de forma fluida y tomar curvas sin salirse, todo ello basándose únicamente en la percepción visual.

Asimismo, se ha verificado la capacidad de generalización del modelo al probarlo en circuitos no vistos durante el entrenamiento, observando que mantiene un rendimiento aceptable sin necesidad de reentrenamiento. También se ha realizado un estudio de ablación de la función de recompensa, lo que ha permitido entender mejor los factores que guían el aprendizaje del agente. Algunos objetivos secundarios, como la comparación con otros algoritmos de Aprendizaje por Refuerzo o el uso de múltiples modalidades sensoriales, no se han abordado debido a limitaciones de tiempo y recursos. No obstante, se identifican como posibles líneas de desarrollo futuro que podrían enriquecer el trabajo realizado.

Entre las posibles mejoras y líneas futuras se encuentra la transferencia del modelo a un robot real, lo que implicaría aplicar técnicas de sim-to-real para reducir la brecha entre la simulación y el entorno físico, como el ajuste en el mundo real. También sería interesante explorar algoritmos alternativos al PPO, como SAC (Soft Actor-Critic) o TD3, que podrían ofrecer mejoras en determinadas condiciones. Otro aspecto a considerar es la ampliación del entorno simulado, incorporando escenarios más complejos y realistas que incluyan diferentes variaciones de iluminación o desniveles en el terreno, con el fin de evaluar la robustez del modelo. Finalmente, entrenar al agente en múltiples mapas con geometrías distintas desde el inicio podría fomentar una mayor capacidad de generalización y transferencia.

En conjunto, el proyecto sienta una base sólida para el desarrollo de sistemas de conducción autónoma basados en visión y Aprendizaje por Refuerzo, y deja abiertas numerosas vías para su mejora, ampliación y aplicación en escenarios más exigentes y realistas.

Apéndice A

Anexos

A.1. Especificaciones del entorno de simulación

A continuación se describen las características técnicas del entorno utilizado para el desarrollo y entrenamiento del agente. Se detallan tanto los aspectos de software como de hardware que han condicionado el rendimiento y la compatibilidad del sistema implementado.

- **Sistema operativo:** Ubuntu 20.04.6 LTS
- **Simulador:** Gazebo 11.15.1 + ROS Noetic
- **Framework de entrenamiento:** Stable Baselines3 (versión 2.2.1) + Gymnasium (versión 0.29.1)
- **Backend de red neuronal:** PyTorch (versión 2.4.1)
- **Hardware:** NVIDIA TITAN Xp, 64 GB RAM, Intel i7-6700

A.2. Composición del entorno y ejecución

En esta sección se describe la forma en la que se estructura y lanza el entorno de simulación. Se indican los comandos principales de ejecución y los nodos que intervienen en el proceso de entrenamiento y prueba del agente.

- **Comandos de ejecución principal:**

```
roslaunch deepracer_simulation train.launch

roslaunch deepracer_simulation try.launch
```

- **Componentes involucrados:**
  - Nodo de control del agente
  - Nodo de captura de observaciones desde cámara
  - Nodo de servo (control de movimiento)
  - Nodo de publicación de odometría
  - Nodo de publicación de estados articulares
  - Carga y publicación del modelo del robot (URDF)

A.3. Arquitectura de la red neuronal

Se muestra aquí la estructura de la red neuronal utilizada para aprender la política de control del agente. La arquitectura está compuesta por capas convolucionales para la extracción de características visuales y capas densas que generan las acciones continuas de salida.

- **Entrada:** imágenes RGB de $120 \times 160 \times 3$
- **Capas convolucionales:**
  - Conv2D(32 filtros, 8x8, stride 4) + ReLU
  - Conv2D(64 filtros, 4x4, stride 2) + ReLU
  - Conv2D(64 filtros, 3x3, stride 1) + ReLU
- **Capas totalmente conectadas:**
  - Dense(512) + ReLU
  - Output: acciones continuas (velocidad, giro)

A.4. Infraestructura de referencia: AWS DeepRacer

Con el fin de contextualizar la arquitectura propuesta en este trabajo, a continuación se presenta un resumen de la infraestructura utilizada por AWS DeepRacer, una de las

plataformas más conocidas para el entrenamiento de agentes de conducción autónoma mediante Aprendizaje por Refuerzo.

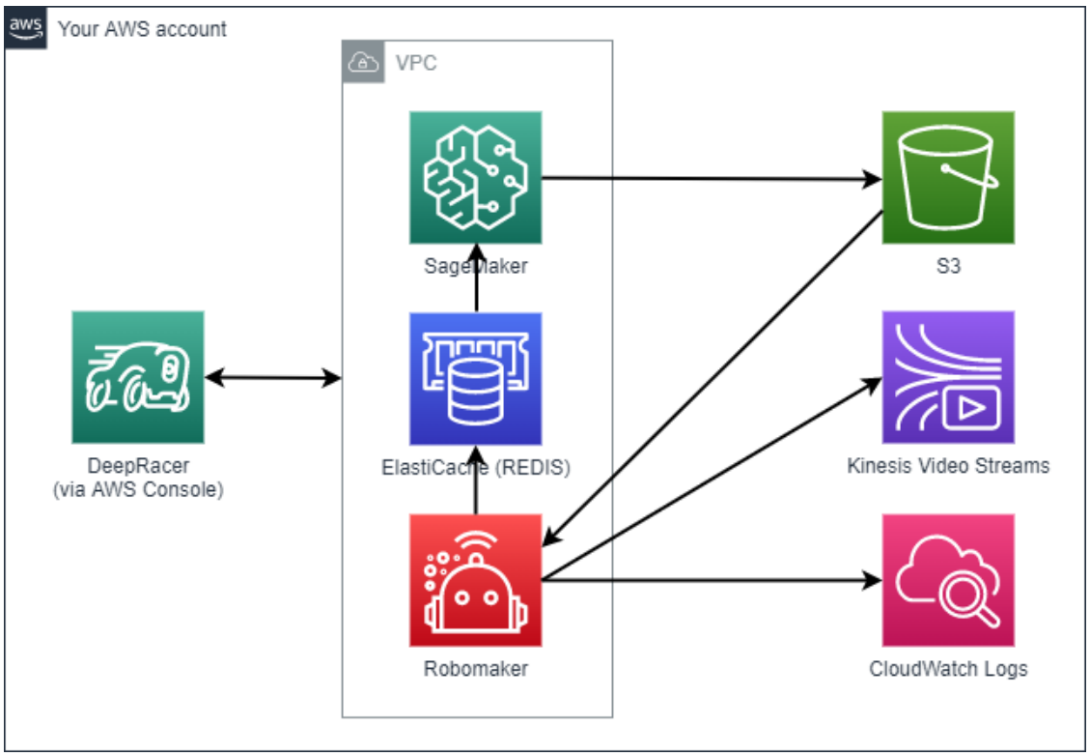


Figura A.1: Figura explicativa de la infraestructura de DeepRacer de AWS.

Este sistema se basa en una infraestructura en la nube donde el entrenamiento se realiza de forma remota mediante servicios gestionados como Amazon SageMaker, AWS RoboMaker y Amazon S3, como se puede ver en la **Figura A.1**. El entorno de simulación y la función de recompensa se definen en la consola web, y los modelos entrenados se descargan posteriormente al vehículo físico para pruebas reales.

En contraste, el sistema desarrollado en este trabajo se ejecuta completamente en local, sin depender de servicios externos, lo que permite mayor control sobre los recursos, personalización del entorno y reducción de costes. Esta independencia también plantea desafíos adicionales, como la necesidad de resolver manualmente problemas de integración y configuración entre componentes, pero proporciona una plataforma más flexible y replicable en otros contextos.

Bibliografía

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. MIT Press, 2018. [Online]. Available: https://www.andrew.cmu.edu/course/10-703/textbook/BartoSutton.pdf

[2] Amazon Web Services, "Aws deepracer developer guide," https://docs.aws.amazon.com/deepracer/latest/developerguide/deepracer-rl.html, 2025, technical documentation for DeepRacer environment, reward function, and simulation.

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[4] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2016.

[5] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[6] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2004, pp. 2149–2154.

[7] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, N. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," https://github.com/DLR-RM/stable-baselines3, 2021, accessed: 2025-06-14.

[8] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[9] F. Foundation, "Gymnasium documentation," https://gymnasium.farama.org, 2023, accessed: 2025-06-14.

[10] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. MIT Press, 2011.

[11] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, I. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, and C. R. Harris, "Scipy 1.0: Fundamental algorithms for scientific computing in python," *Nature Methods*, vol. 17, pp. 261–272, 2020, kDTree module available at https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.KDTree.html. [Online]. Available: https://www.nature.com/articles/s41592-019-0686-2