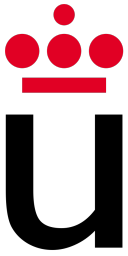


Universidad
Rey Juan Carlos

ESCUELA DE INGENIERÍA DE FUENLABRADA

GRADO EN INGENIERÍA DE ROBÓTICA SOFTWARE

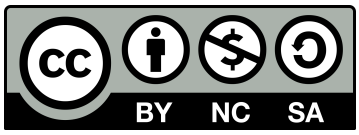
TRABAJO FIN DE GRADO

APARCAMIENTO DE UN COCHE AUTÓNOMO MEDIANTE APRENDIZAJE POR REFUERZO

Realizado por
David Duro Aragonés

Dirigido por
José María Cañas Plaza
David Gualda Gómez

Curso académico 2024/2025



Agradecimientos

Este trabajo no habría sido posible sin el apoyo y acompañamiento de todas aquellas personas que han estado presentes durante esta etapa formativa y, especialmente, durante la realización de este proyecto.

Quiero expresar mi más sincero agradecimiento a mis padres, por su apoyo incondicional y por ser mi pilar fundamental en todo momento. A mis amigos, compañeros y las nuevas amistades forjadas durante la carrera, por hacer este camino más llevadero y enriquecedor.

A mis tutores y profesores, en especial a aquellos que han sabido guiarme y motivarme cuando más lo necesitaba. Su orientación y consejos han sido de un valor incalculable para el desarrollo de este trabajo.

Un agradecimiento muy especial a mi pareja, no solo por su apoyo emocional en los momentos más complejos, sino también por su ayuda como pre-redactora y crítica constructiva durante todo el proceso.

A todos, mi más profundo reconocimiento por contribuir, cada uno desde su lugar, a que este logro académico sea hoy una realidad.

Resumen

En este Trabajo Fin de Grado (TFG) se aborda la tarea de realizar el aparcamiento autónomo de un agente a partir de algoritmos de aprendizaje por refuerzo, dentro del entorno de desarrollo SMARTS, que ha sido seleccionado por su especialización en algoritmos de aprendizaje para la conducción autónoma.

El trabajo incluye un importante componente de desarrollo en esta plataforma, ya que la adaptación del simulador requiere realizar modificaciones sustanciales en sus parámetros para adaptarlo a los requisitos de estacionamiento en entornos controlados.

Los resultados relativos a distintas propuestas de aparcamiento autónomo (1-D y diversas variaciones en 2-D) demuestran tasas de acierto superiores al 80 % en la mayoría de los casos, siendo del 100 % en el escenario más simple, aportando conclusiones de gran valor sobre el uso de técnicas de aprendizaje por refuerzo orientadas al aparcamiento autónomo, así como proveer de soluciones prácticas para la adaptación de simuladores a problemas específicos de la robótica móvil.

Palabras clave: aparcamiento autónomo, aprendizaje por refuerzo, conducción autónoma, simulador, entornos controlados, robótica móvil.

Abstract

This Final Degree Project addresses the task of performing the autonomous parking of an agent based on reinforcement learning algorithms, within the SMARTS development environment, which has been selected for its specialization in learning algorithms for autonomous driving.

The work includes an important development component on this platform, as the adaptation of the simulator requires substantial modifications to its parameters to adapt it to the requirements of parking in controlled environments.

The results for different autonomous parking proposals (1-D and several 2-D variations) demonstrate success rates of over 80 % in most cases, with 100 % in the simplest scenario, providing valuable insights into the use of reinforcement learning techniques for autonomous parking, as well as providing practical solutions for adapting simulators to specific mobile robotics problems.

Keywords: autonomous parking, reinforcement learning, autonomous driving, simulator, controlled environments, mobile robotics.

Acrónimos

TFG - Trabajo de Fin de Grado
LiDAR - Light Detection and Ranging
RLiDAR - Rotating LiDAR
ROS - Robot Operating System
CPU - Central Processing Unit
GPU - Graphics Processing Unit
SLAM - Simultaneous Localization and Mapping
SAE - Sociedad de Ingenieros Automotrices
MDP - Markov Decision Process
SUMO - Simulation of Urban MObility
DDPG - Deep Deterministic Policy Gradient
SAC - Soft Actor-Critic
DBSCAN - Density-Based Spatial Clustering of Applications with Noise
DQN - Deep Q-Network
DDQN - Double Deep Q-Network
DL - Deep Learning
RL - Reinforcement Learning
DRL - Deep Reinforcement Learning
PID - Proportional-Integral-Derivative
ReLU - Rectified Linear Unit

Índice general

Índice de figuras

Índice de tablas

Índice de extractos de código

Índice de ecuaciones

1. Introducción

1.1. Robótica

La robótica es una disciplina tecnológica que integra principios de ingeniería mecánica, eléctrica, informática y ciencias cognitivas para el diseño, construcción y programación de sistemas robóticos. Estos sistemas, conocidos como robots, son capaces de interactuar con su entorno físico, ejecutando tareas mediante percepción, procesamiento y acción.

El desarrollo histórico de la robótica ha experimentado una notable evolución, desde sus orígenes en dispositivos mecánicos simples hasta complejos sistemas autónomos. Esta evolución ha estado marcada por hitos tecnológicos significativos, como la creación de los primeros brazos robóticos industriales en la década de 1960 o el desarrollo de vehículos autónomos y robots colaborativos en el siglo XXI.

Uno de los campos más consolidados de la robótica es la robótica industrial, centrándose en la automatización de procesos de fabricación. Los robots industriales han transformado sectores como la automoción, la electrónica o la metalurgia, permitiendo una producción más rápida, precisa y segura. Estos robots suelen operar en entornos controlados, y están diseñados para tareas específicas como soldadura, ensamblaje o manipulación de materiales. Su fiabilidad y eficiencia los han convertido en piezas clave en las cadenas de producción modernas.

En contraste, la robótica de servicios se orienta a la asistencia directa a las personas o a tareas en entornos dinámicos. Aquí se incluyen desde robots de limpieza o reparto hasta asistentes personales o robots quirúrgicos. A diferencia de los robots industriales, los robots de servicios deben enfrentarse a escenarios cambiantes, interactuar con humanos y, a menudo, tomar decisiones en tiempo real. Esta área está en rápido crecimiento gracias a los avances en visión e inteligencia artificial y sistemas de navegación autónoma.

A lo largo del tiempo, el concepto de robot ha sido influenciado tanto por avances científicos como por la cultura popular. Desde los autómatas diseñados por inventores como Leonardo da Vinci hasta las representaciones futuristas de robots en la literatura de ciencia ficción, la robótica ha capturado la imaginación

colectiva y ha orientado los esfuerzos tecnológicos hacia la creación de máquinas cada vez más inteligentes y autónomas.

Actualmente, la robótica desempeña un papel clave en sectores como la medicina, la agricultura, la logística o la exploración. Gracias a la integración de sensores avanzados y técnicas de inteligencia artificial y aprendizaje automático, los robots modernos pueden tomar decisiones en tiempo real, adaptarse a entornos dinámicos y colaborar de manera segura con los seres humanos. Este avance ha generado un cambio de paradigma: los robots ya no son solo herramientas programadas, sino agentes capaces de aprender y evolucionar con la experiencia.

1.1.1. Componentes fundamentales

Un sistema robótico se estructura en tres subsistemas principales que interactúan de manera sinérgica:

- **Sensores:** Constituyen el conjunto de dispositivos de percepción que permiten al robot captar información sobre su entorno interno y externo. Algunos de los más usados son; cámaras RGB, RGB-D o térmicas; sistemas LiDAR, encoders rotacionales y lineales y unidades de medición inercial (IMUs).

Figura 1.1: Representacion sensorial. [1]

- **Actuadores:** Comprenden los mecanismos que permiten al robot interactuar físicamente con su entorno. Uno de los tipos más empleados son los actuadores eléctricos, especialmente los motores eléctricos, debido a su precisión, capacidad de control y facilidad de integración con sistemas electrónicos.

También se emplean actuadores que funcionan mediante fluidos o gases comprimidos, como los hidráulicos y neumáticos, que se utilizan en entornos industriales.

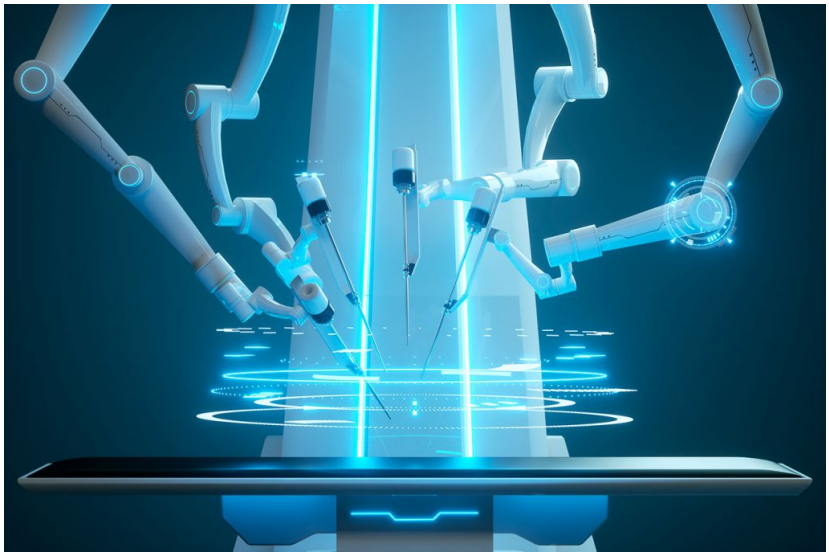


Figura 1.2: Representación de mecanismos de actuación. [2]

- **Procesadores:** Representa un conjunto de componentes electrónicos que reciben datos de los sensores, los procesan y envían las órdenes correspondientes a los actuadores. Básicamente, es lo que permite al robot tomar decisiones y actuar en consecuencia.

  Estos dispositivos pueden ir desde simples circuitos integrados hasta sistemas avanzados basados en microcontroladores, FPGAs o procesadores más potentes como CPUs y GPUs. Las CPUs gestionan tareas de control, lógica y planificación, mientras que las GPUs resultan especialmente útiles para procesar grandes volúmenes de datos en paralelo, como ocurre en visión por computador o redes neuronales.

Figura 1.3: Representación de control. [3]

Aunque los componentes físicos como sensores, actuadores y procesadores son fundamentales para el funcionamiento de un robot, representan solo una parte del sistema. El verdadero valor y complejidad de la robótica reside en el software que gestiona, interpreta y coordina toda esa información. La programación, los algoritmos de control, los procesos de depuración y análisis de datos son cruciales para lograr un comportamiento eficiente y adaptable.

1.1.2. Software robótico

El desarrollo de sistemas robóticos complejos requiere de herramientas software especializadas que permitan integrar los diferentes componentes del sistema. Los middlewares robóticos han surgido como soluciones fundamentales para este propósito, actuando como capas intermedias entre el hardware y las aplicaciones de alto nivel. Estos sistemas son clave para la comunicación entre sensores, actuadores y algoritmos de control, simplificando significativamente el proceso de desarrollo.

Entre las plataformas más utilizadas destaca ROS, para la investigación y desarrollo en robótica. Su arquitectura distribuida y modular permite conectar componentes mediante mecanismos de publicación-suscripción, facilitando la creación de sistemas complejos. Junto a los middlewares, los entornos de simulación como Gazebo, Webots o el propio SMARTS utilizado en este trabajo, permiten probar y validar algoritmos en entornos virtuales antes de su implementación en sistemas físicos.

Estas herramientas son particularmente valiosas para interactuar con robots, ya que proporcionan interfaces estandarizadas para diferentes tipos de sensores y

actuadores. La combinación de middlewares robustos con simuladores avanzados ha acelerado significativamente el desarrollo de aplicaciones robóticas, permitiendo a los investigadores centrarse en los aspectos algorítmicos mientras se abstraen de los detalles de bajo nivel.

1.1.3. Robótica móvil

La robótica móvil abarca el diseño de sistemas robóticos autónomos que pueden moverse por diversos entornos. A diferencia de los robots estacionarios tradicionales, como los brazos robóticos en las líneas de producción, estos sistemas incluyen mecanismos de movimiento autónomo que les permiten funcionar en espacios no estructurados y dinámicos.

La clasificación fundamental de estos robots se establece según su medio de locomoción, dando lugar a tres categorías principales: los robots terrestres, diseñados para moverse sobre superficies sólidas; los robots aéreos, capaces de navegar en el espacio tridimensional; y los robots acuáticos, especializados en operar en medios líquidos con diferentes grados de profundidad. Cada tipología presenta características mecánicas y sensoriales específicas adaptadas a las particularidades de su entorno operativo.

El software para robótica móvil constituye el núcleo de estos sistemas. En su funcionamiento se encuentran tres capacidades principales: la navegación global, que planifica rutas óptimas considerando el mapa completo del entorno; la navegación local, que ejecuta ajustes en tiempo real; y la autolocalización, que permite al robot conocer su posición en todo momento.

Frameworks como Nav2 [12] han estandarizado la implementación de estas capacidades, proporcionando una arquitectura robusta para sistemas de navegación autónoma. Estos sistemas integran algoritmos de mapeado con LiDAR mediante técnicas SLAM, que construyen representaciones tridimensionales del entorno mientras simultáneamente determinan la posición del robot.

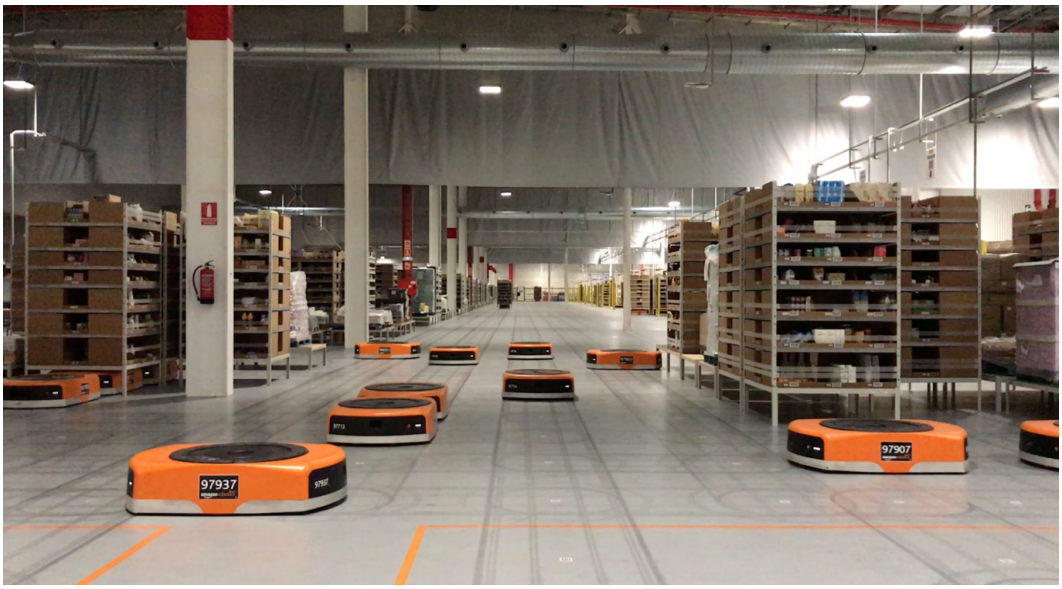


Figura 1.4: Ejemplo de almacén con vehículos móviles autónomos. [4]

La imagen muestra un ejemplo real de robótica móvil aplicada a la logística moderna, similar a los sistemas utilizados por empresas como Amazon, GEODIS y Ocado. Los robots navegan de forma autónoma por las instalaciones, optimizando rutas y reduciendo tiempos de operación, lo que permite a estas empresas aumentar significativamente su productividad.

1.1.4. Robótica de servicios

1.2. Conducción autónoma

La conducción autónoma emerge como uno de los campos más innovadores dentro de la robótica de servicios, a modo de respuesta a los desafíos de movilidad y seguridad vial, planteando un paradigma donde los vehículos son capaces de interpretar su entorno y tomar decisiones sin intervención humana directa. Este concepto es generado mediante la convergencia de múltiples tecnologías, desde sistemas avanzados de percepción hasta complejos algoritmos de control.

Figura 1.5: Coche autónomo. [5]

La evolución de los sistemas de conducción autónoma ha transitado desde las primeras ayudas a la conducción, como el control de crucero adaptativo, hasta soluciones integrales que gestionan completamente la navegación. Este progreso tecnológico ha incorporado funcionalidades cada vez más avanzadas: asistencia para adelantamientos, mantenimiento activo de carril, frenado automático de emergencia, detección de ángulos muertos, sistemas de autoaparcamiento y, más recientemente, capacidades de conducción autónoma en autopistas. Cada una de estas innovaciones ha ido ampliando gradualmente los escenarios donde la automatización puede asistir o reemplazar al conductor.

Figura 1.6: Los 5 niveles de conducción autónoma SAE. [6]

El potencial de esta tecnología puede servir para convertir la conducción, una actividad cotidiana cargada de subjetividad y riesgo, en un proceso sistemático y optimizado. Al eliminar los factores humanos que contribuyen a la siniestralidad vial (la fatiga, las distracciones o los errores de juicio) los sistemas autónomos establecen nuevos estándares de seguridad y eficiencia. Esta transición progresiva desde la asistencia a la conducción hasta la autonomía plena representa no solo un hito tecnológico, sino un cambio en nuestra comprensión de la movilidad, marcando el camino hacia un futuro donde el transporte sea intrínsecamente más seguro, accesible y sostenible.

Actualidad

La conducción autónoma ha experimentado un desarrollo acelerado gracias al impulso de compañías como Waymo, Cruise y Tesla, cada una con enfoques distintos en su camino hacia la autonomía total. Estas iniciativas comparten su origen en competencias pioneras como el *DARPA Grand Challenge* (2004) y *Urban Challenge* (2007), que demostraron la viabilidad técnica y fomentaron la investigación en este campo.

Tecnologías clave

El desarrollo de sistemas de conducción autónoma se basa en un conjunto de tecnologías fundamentales que trabajan de forma coordinada. En el núcleo del sistema se encuentran arquitecturas software como Autoware y Apollo, que proporcionan frameworks para integrar los diferentes componentes del vehículo

autónomo.

La percepción del entorno se logra mediante fusión sensorial, que combina datos de múltiples fuentes: LiDARs para modelado 3D, cámaras estereoscópicas para reconocimiento de objetos, y radares para medición de velocidades. La comunidad científica utiliza datasets de referencia como KITTI y nuScenes para entrenar y validar estos algoritmos de percepción, permitiendo comparaciones objetivas entre diferentes aproximaciones.

Entornos de desarrollo

Simuladores como CARLA y SMARTS se han convertido en herramientas esenciales, permitiendo probar los sistemas en escenarios virtuales antes de su implementación real. Estos entornos reproducen fielmente las condiciones del mundo real, tráfico denso y comportamientos impredecibles de peatones y otros vehículos.

1.2.1. Aparcamiento autónomo

El aparcamiento autónomo es uno de los problemas más interesantes de la conducción autónoma, ya que implica múltiples requisitos. Para ejecutar una maniobra de aparcamiento eficaz, un sistema debe ser capaz de reconocer y medir el espacio de aparcamiento disponible, el tamaño de la plaza de aparcamiento y los obstáculos estáticos y dinámicos que haya en sus proximidades. Una vez que el sistema encuentra y selecciona una plaza de aparcamiento, debe calcular una trayectoria factible para maximizar la eficiencia de sus maniobras.

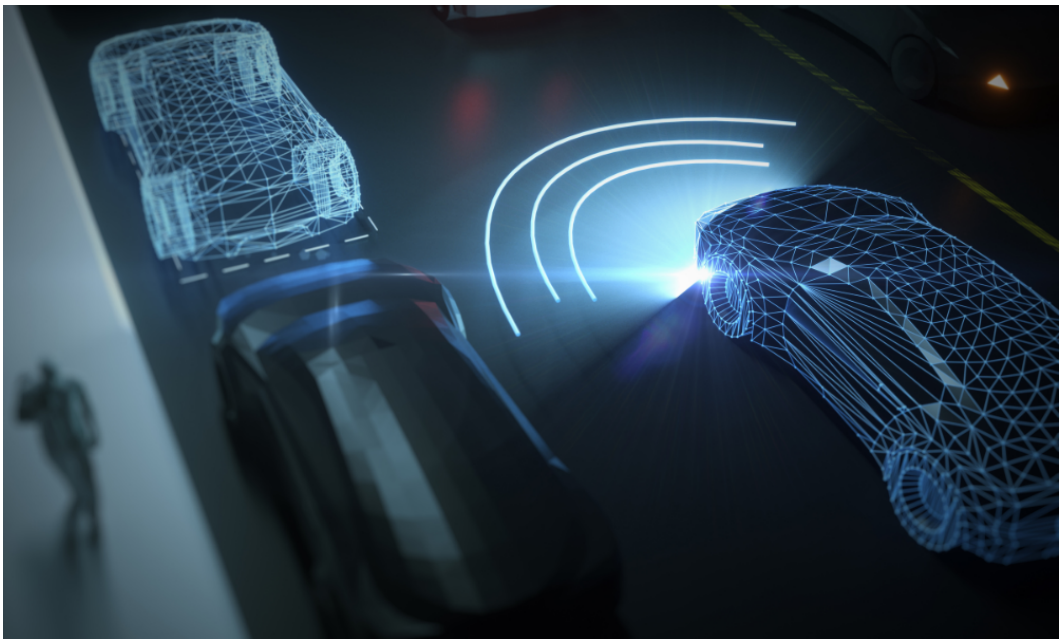


Figura 1.7: Reconocimiento autónomo de hueco. [7]

Para realizar el procedimiento, los actuadores se controlan junto con la velocidad y los ángulos de giro de forma simultánea. Además, algunos de los sistemas más sofisticados disponen también de corrección en tiempo real, con la que el vehículo puede realinearse si un obstáculo dinámico aparece a la vista de forma inesperada o la situación se vuelve demasiado complicada para que el vehículo circule por sí solo.

Los sistemas de aparcamiento autónomo actuales tienden a basarse en soluciones híbridas, adquiriendo la geometría del espacio por medios geométricos. La fusión sensorial (cámaras, ultrasonidos, LiDAR) crea una imagen precisa del espacio circundante a pesar de la disminución de la visibilidad.

El desarrollo de este tipo de sistemas permite que los coches puedan aparcar o incluso maniobrar en lugares extremadamente estrechos de forma totalmente autónoma. Estos inventos transforman la experiencia del usuario e incluso la construcción de plazas de aparcamiento.

1.2.2. Impacto en seguridad vial y eficiencia

Aunque no existen estudios estadísticos oficiales que cuantifiquen con precisión el porcentaje de accidentes durante el estacionamiento, los talleres de reparación y las aseguradoras reportan frecuentes daños por roces y golpes ocurridos durante estas operaciones, especialmente en espacios reducidos.

La principal ventaja de los sistemas automatizados radica en su capacidad para realizar maniobras de estacionamiento con precisión milimétrica. A diferencia de los conductores humanos, que suelen dejar márgenes de seguridad excesivos por inseguridad o falta de percepción espacial, estos sistemas aprovechan al máximo el espacio disponible, permitiendo un diseño más eficiente de plazas de parking sin necesidad de ampliar su superficie.

En cuanto a eficiencia, el proceso de estacionamiento se optimiza eliminando movimientos innecesarios. Mientras un conductor puede requerir múltiples intentos para posicionar el vehículo correctamente (especialmente en situaciones de estrés o falta de visibilidad) un sistema automatizado calcula y ejecuta la trayectoria óptima.

Al prescindir de factores humanos como la fatiga o la presión, estos sistemas garantizan un rendimiento constante y repetible en cualquier condición, asegurando que cada operación de estacionamiento se realice con la misma precisión, independientemente de las circunstancias externas.

El verdadero valor del aparcamiento autónomo no reside en la prevención de accidentes, sino en su capacidad para automatizar una de las tareas más repetitivas y tediosas de la conducción cotidiana. Esta automatización representa un avance en la experiencia del conductor, transformando una operación mecánica en un proceso transparente.

1.3. Aprendizaje por refuerzo en robótica

El aprendizaje por refuerzo (RL) es una técnica de aprendizaje automático (ML) que entrena al software para que tome decisiones y logre los mejores resultados. Imita el proceso de aprendizaje por ensayo y error que los humanos utilizan para lograr sus objetivos. Las acciones de software que trabajan para alcanzar su objetivo se refuerzan, mientras que las que se apartan del objetivo se ignoran[1].

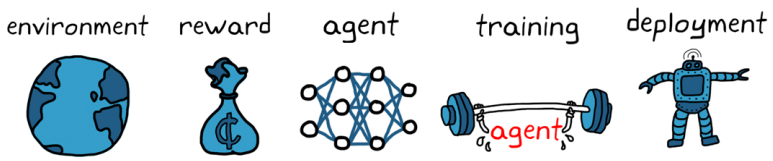


Figura 1.8: Pasos RL. [8]

[1] https://aws.amazon.com/what-is/reinforcement-learning/

El aprendizaje por refuerzo representa un enfoque innovador para resolver problemas complejos en robótica, aunque aún se encuentra en fase de experimentación y desarrollo. Esta técnica de aprendizaje automático permite a los sistemas robóticos adquirir habilidades mediante la interacción continua con su entorno, a través de un proceso de ensayo y error guiado por recompensas. Como demuestra este TFM sobre la conducción autónoma, el RL está abriendo nuevas posibilidades en aplicaciones donde los enfoques tradicionales de control encuentran limitaciones.

1.3.1. Q-Learning

El Q-Learning [13] es un algoritmo fundamental de aprendizaje por refuerzo que se basa en el concepto de función de valor acción-estado (Q). Este método aprende una política óptima estimando iterativamente los valores Q, que representan la recompensa acumulada esperada al tomar una acción específica en un estado determinado y seguir luego la política óptima. La base teórica del Q-Learning se basa en dos fundamentos: el proceso de decisión de Markov (MDP) y la ecuación de Bellman. Los MDPs proporcionan un marco matemático donde la transición entre estados cumple con la propiedad markoviana, es decir, el próximo estado y la recompensa dependen únicamente del estado actual y la acción tomada, independientemente de los estados pasados.

$$Q(s,a) = Q(s,a) + \alpha \left[r + \gamma \, maxQ(s',a') - Q(s,a) \right]$$

Ecuación 1.1: Ecuación de Bellman.

- $Q(s,a)$: valor actual de tomar acción a en estado s.
- α: tasa de aprendizaje.
- r: recompensa obtenida al tomar la acción a.
- γ: factor de descuento (cuánto valoramos las recompensas futuras).
- s': nuevo estado después de la acción.
- $maxQ(s',a')$: valor estimado de la mejor acción posible en el nuevo estado s'.

Esta ecuación implementa una actualización temporal diferencial, combinando la recompensa observada con las estimaciones futuras para mejorar progresivamente la política. En robótica, el Q-Learning ha demostrado ser

particularmente efectivo para problemas con espacios de estado discretos y accionables limitados.

1.3.2. DQN y DDQN

Los algoritmos Deep Q-Network (DQN) [14] representan una evolución fundamental del Q-Learning clásico al incorporar redes neuronales profundas para aproximar la función de valor Q en espacios de estado continuos y de alta dimensionalidad. La arquitectura típica de DQN consiste en una red neuronal con dos capas ocultas completamente conectadas. Esta estructura permite manejar entradas complejas transformándolas en representaciones abstractas para la estimación final de los valores Q. Sin embargo, DQN presenta limitaciones como la sobreestimación de valores Q, lo que puede llevar a políticas subóptimas y problemas de convergencia.

Double DQN (DDQN) surge como mejora sustancial al problema de sobreestimación, desacoplando la selección de acciones de su evaluación mediante dos redes separadas: una red principal (current network) que selecciona las acciones, y una red objetivo (target network) que las evalúa. Esta división reduce las sobreestimaciones, llevando a políticas más estables y precisas.

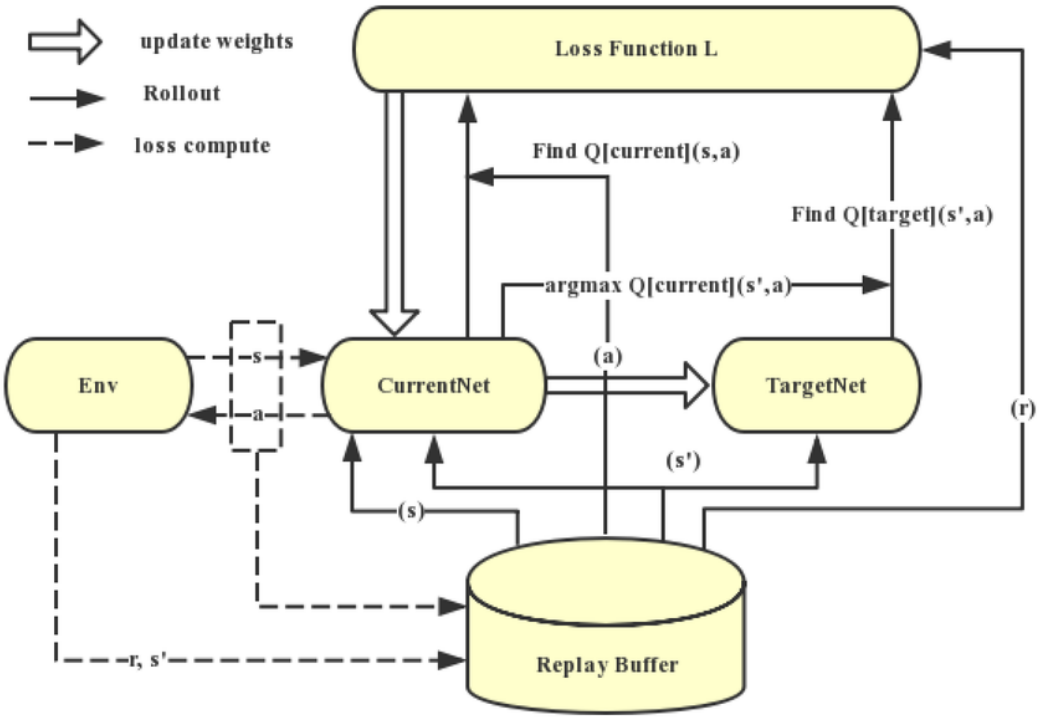


Figura 1.9: Esquema de funcionamiento DDQN. [9]

En robótica, esta arquitectura ha demostrado especial eficacia para tareas complejas como navegación en entornos no estructurados o manipulación precisa, donde la estabilidad del aprendizaje resulta crítica. La combinación de DDQN con técnicas como *experience replay* y actualizaciones periódicas de la red objetivo ha establecido este enfoque como referencia en aplicaciones de RL profundo para control robótico.

1.3.3. Comparativa entre algoritmos RL

Características	**Q-Learning**	**DQN**	**DDQN**
Espacio de estados	Finito y discreto	Muy grande	Muy grande
Uso de redes neuronales	No	Sí	Sí
Función Q almacenada como	Tabla	Red neuronal	Red neuronal
Problema de sobreestimación	Sí	Sí	Corregido
Estabilidad del aprendizaje	Alta	Media	Alta
Memoria de experiencias	No	Sí	Sí
Entornos grandes	No	Parcialmente	Mejorado

Tabla 1.1: Comparación entre Q-Learning, DQN y DDQN.

1.3.4. Comparativa con métodos tradicionales

El aprendizaje por refuerzo representa un cambio fundamental en el paradigma de control robótico, ofreciendo un cambio significativo con los enfoques tradicionales. Mientras que la robótica clásica se fundamenta en la descomposición analítica de tareas y el diseño manual de controladores para cada subsistema, los métodos modernos de aprendizaje automático buscan capturar el comportamiento del sistema en su totalidad.

Los sistemas robóticos convencionales operan mediante una jerarquía de percepción, planificación y ejecución. Esta arquitectura, aunque bien comprendida teóricamente, enfrenta limitaciones cuando se trata de sistemas complejos con interacciones no lineales o entornos parcialmente observables. El aprendizaje por refuerzo, en cambio, permite comportamientos complejos a través de la interacción directa con el entorno, sin requerir un modelado explícito de cada componente.

La adaptabilidad constituye otra diferencia crucial. Los controladores clásicos requieren recalibración manual cuando cambian las condiciones operativas,

mientras que los sistemas basados en aprendizaje pueden adaptarse automáticamente dentro de los límites de su espacio de entrenamiento. Esta capacidad es particularmente valiosa en aplicaciones como la interacción humano-robot, donde la variabilidad es infinita.

Sin embargo, el aprendizaje por refuerzo no debe verse como un reemplazo absoluto de los métodos tradicionales, sino como un complemento. Las técnicas clásicas siguen siendo insuperables en aplicaciones que demandan garantías formales de seguridad o estabilidad. El auténtico poder surge al integrar ambos métodos: utilizando el aprendizaje para manejar la complejidad de alto nivel mientras se mantienen controladores tradicionales para funciones críticas de bajo nivel.

2. Objetivos y metodología

El proyecto se centra en el desarrollo y validación de un sistema de aparcamiento autónomo mediante técnicas de aprendizaje por refuerzo, utilizando un entorno de simulación como plataforma de prueba.

2.1. Objetivos

Este trabajo se estructura progresivamente, abordando problemas de complejidad creciente, permitiendo validar cada componente antes de enfrentar desafíos más avanzados.

2.1.1. Configuración del simulador

El primer objetivo consiste en preparar y adaptar el entorno de simulación para cumplir con los requisitos específicos del proyecto. Esto incluye la configuración de sensores, la creación de escenarios personalizados y la implementación de interfaces para el control de los agentes autónomos.

2.1.2. Parking 1-D

Se aborda inicialmente el problema unidimensional como caso de estudio simplificado, permitiendo verificar la efectividad del aprendizaje por refuerzo en un contexto controlado. Esta fase sirve como prueba de concepto y permite ajustar los parámetros fundamentales del sistema antes de enfrentar configuraciones más complejas.

2.1.3. Parking 2-D

El trabajo se extiende al caso bidimensional, primero con la configuración holonómica que ofrece mayor libertad de movimiento y posteriormente con la cinemática Ackermann más restrictiva. Este enfoque progresivo permite comprender cómo las diferentes capacidades de maniobra afectan a las estrategias

de estacionamiento autónomo, desde la solución relativamente sencilla holonómica hasta el desafío más complejo que representa el modelo Ackermann con sus limitaciones de giro y movimiento.

2.2. Metodología

2.2.1. Avance de trabajo

El desarrollo del trabajo se organizó mediante reuniones semanales siguiendo la metodología SCRUM, lo que permitió adaptar continuamente los objetivos a los avances y desafíos encontrados. Estas sesiones regulares sirvieron para evaluar progresos y mantener un trabajo constante.

2.2.2. Control de versiones

El proyecto se estructuró en dos repositorios GitHub independientes pero complementarios. El primero alberga una versión personalizada del simulador SMARTS, modificada específicamente para los requerimientos de esta investigación, donde se encuentra todo el código desarrollado.

Paralelamente, se mantuvo un segundo repositorio dedicado a la documentación técnica, implementado como blog mediante GitHub Pages. Este espacio se actualizó semanalmente.

2.2.3. Comunicación y seguimiento

La coordinación del trabajo se apoyó en la plataforma Slack para la comunicación asíncrona y la resolución rápida de dudas.

2.2.4. Plan de trabajo SCRUM

Configuración y adaptación del simulador

Esta primera fase se centró en la configuración y personalización del simulador SMARTS 4, un proceso que requirió adentrarse profundamente en su arquitectura interna. Durante esta etapa, se identificaron y resolvieron diversos problemas que afectaban al comportamiento del vehículo autónomo, lo que nos llevó a desarrollar

una versión personalizada del simulador con modificaciones específicas para nuestro caso de uso. Para comprender y adaptar el simulador, fue necesario estudiar en profundidad su integración con SUMO, utilizado para la simulación del tráfico, así como su interfaz con Gymnasium, que nos permite implementar algoritmos de aprendizaje por refuerzo.

Parking 1-D

El desarrollo del sistema de aparcamiento unidimensional se basó en un enfoque básico de aprendizaje por refuerzo, implementando el algoritmo Q-learning. Esta elección se justificó por la relativa simplicidad del problema 1D, donde el espacio de estados discretizado permitía obtener buenos resultados.

Parking 2-D

Para abordar el problema de aparcamiento bidimensional y el objetivo final de este trabajo, se implementó una arquitectura de red neuronal profunda utilizando PyTorch como framework principal.

2.2.5. Cronograma de trabajo

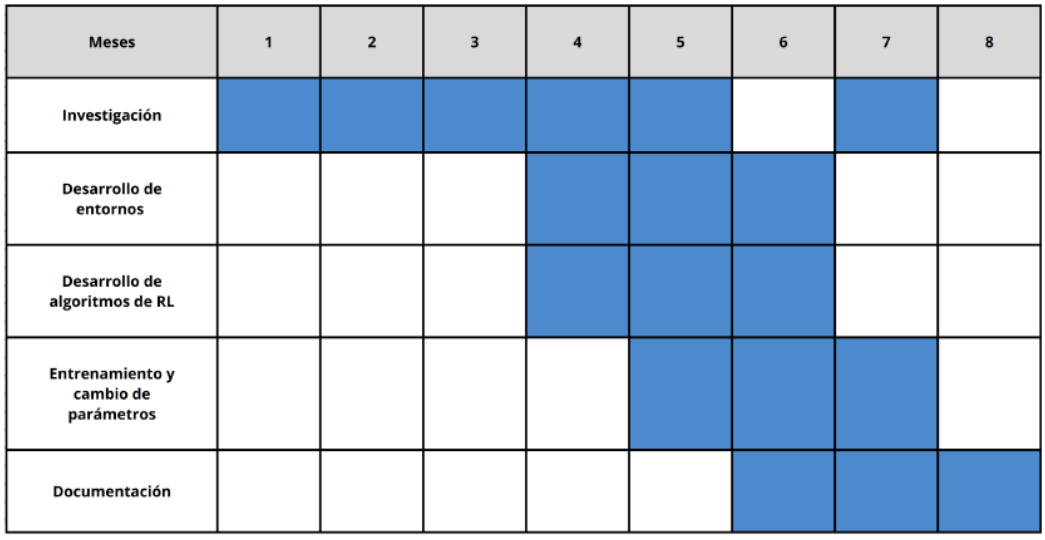


Figura 2.1: Diagrama de Gantt.

En la figura 2.1 se muestra un cronograma con las tareas realizadas a lo largo de la duración del trabajo.

3. Herramientas utilizadas

Este capítulo describe las principales herramientas empleadas en el desarrollo de este TFG. El análisis abarca desde los entornos de desarrollo hasta las bibliotecas especializadas que han permitido implementar algoritmos propuestos.

3.1. Simulador SMARTS

El simulador SMARTS (Scalable Multi-Agent Reinforcement Learning Training System) [15] representa una plataforma especializada para el desarrollo de sistemas de conducción autónoma mediante aprendizaje por refuerzo. Su arquitectura está diseñada específicamente para abordar los desafíos que presentan los entornos de tráfico.

La principal fortaleza de SMARTS reside en su capacidad para generar escenarios de tráfico altamente configurables, permitiendo desde simulaciones de cruces simples hasta intersecciones complejas con múltiples agentes interactuando simultáneamente. Esta flexibilidad se complementa con un sistema de sensores virtuales que emulan dispositivos reales como LiDARs y sistemas de posicionamiento, proporcionando a los agentes de aprendizaje por refuerzo las mismas limitaciones y desafíos que enfrentarían en implementaciones físicas.

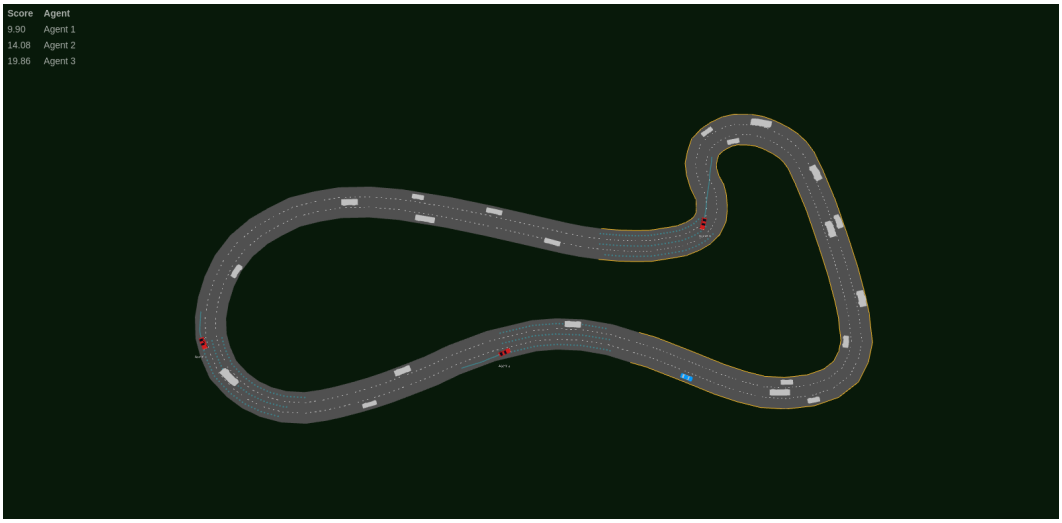


Figura 3.1: Escenario de ejemplo en SMARTS.

3.1.1. Arquitectura de SMARTS

Ruta	Descripción
`/examples/*`	Ejecutables.
`/scenarios/sumo/*`	Escenarios en formato SUMO.
`/smarts/core/lidar.py`	Módulo que gestiona el LiDAR.
`/smarts/core/controllers/*`	Controladores de velocidad y comportamiento de bajo nivel.
`/smarts/core/chassis.py`	Chasis y su configuración física.
`/smarts/core/utils/*`	Herramientas de depuración, visualización y gestión (control de episodios).
`/smarts/env/gymnasium/wrappers/*`	Gestionan recompensas, observaciones y estados.
`/smarts/env/utils/*`	Funciones de depuración y filtrado, como rangos de valores de las acciones.
`/smarts/sstudio/*`	Integración de SUMO y la generación/gestión de tráfico.

Tabla 3.1: Estructura de archivos en SMARTS.

3.1.2. Características y ventajas frente a CARLA

Al comparar SMARTS con CARLA [16], otro simulador ampliamente utilizado en investigación de conducción autónoma, emergen diferencias fundamentales. Mientras CARLA se centra en proporcionar un entorno visualmente realista con gráficos de alta fidelidad, SMARTS prioriza la escalabilidad y el control preciso sobre el comportamiento de los agentes, características esenciales para la investigación en aprendizaje por refuerzo.

La integración nativa de SMARTS con SUMO [17] le permite generar patrones de tráfico realistas, algo particularmente valioso para simular escenarios complejos. Esta combinación usa la precisión en la simulación de tráfico de SUMO y la capacidad de SMARTS para modelar agentes inteligentes, ofreciendo un entorno muy interesante para el desarrollo de algoritmos de conducción autónoma que deben operar en entornos de tráfico.

Donde CARLA sobresale en realismo visual, SMARTS ofrece ventajas en abstracción computacional y eficiencia. La capacidad de SMARTS para operar en modo *"headless"* (sin renderizado gráfico) permite ejecutar simulaciones, acelerando significativamente el ciclo de desarrollo. Esta característica lo hace particularmente adecuado para la investigación.

Aspecto	**SMARTS**	**CARLA**
Gráficos	Muy bajos	Muy altos
Realismo visual	Bajo	Muy alto
Simulación de tráfico	Alto (via SUMO)	Medio
Peso computacional	Muy bajo	Alto
Simplicidad de uso	Medio	Medio
Facilidad de modificación	Alta	Media
Facilidad de integración con otros sistemas	Alta	Media
Adecuado para aprendizaje por refuerzo	Muy adecuado	Adecuado
Soporte para ejecución sin interfaz gráfica	Sí	Limitado

Tabla 3.2: Comparación entre SMARTS y CARLA en aspectos clave.

Cabe destacar que SMARTS, aunque eficiente para simulación de tráfico, presenta limitaciones en la integración de nuevos sensores y en el soporte nativo para sistemas LIDAR. En concreto, carece de funcionalidades básicas como la generación automática de coordenadas relativas, lo que requirió transformar y procesar adecuadamente los datos de percepción.

3.1.3. SUMO

La herramienta SUMO [17] ha desempeñado un papel fundamental en nuestro trabajo como plataforma para el diseño y generación de escenarios de tráfico. Su integración por defecto con SMARTS nos permitió crear entornos de simulación donde probar nuestros algoritmos de aparcamiento autónomo. El profundo estudio de SUMO fue necesario para comprender su modelo de simulación y cómo interactúa con los agentes controlados por aprendizaje por refuerzo.

El uso principal de SUMO en nuestro proyecto se centró en la creación de mapas y configuraciones de carreteras para escenarios de estacionamiento. A través de su formato de archivos *.net.xml*, pudimos diseñar las dimensiones de carreteras y espacios de aparcamiento. Un aspecto clave fue aprender a configurar

adecuadamente los parámetros de tráfico para simular vehículos estáticos que funcionaran como obstáculos, lo que requirió un análisis detallado de los tipos de vehículos y sus comportamientos en SUMO.

El estudio de SUMO nos permitió, además de crear los escenarios necesarios para nuestro proyecto (4.8), localizar y solucionar tanto problemas que afectaban el comportamiento de nuestro agente autónomo como problemas en la generación de escenarios.

3.1.4. Biblioteca Gymnasium para RL

SMARTS implementa nativamente la interfaz de Gymnasium para el aprendizaje por refuerzo, lo que nos permitió integrar directamente nuestros algoritmos con el simulador. Esta compatibilidad estándar resultó esencial para establecer una comunicación consistente entre el entorno de simulación y nuestros modelos.

Gymnasium, sucesor del popular OpenAI Gym [18], es una biblioteca de referencia para el desarrollo y comparación de algoritmos de aprendizaje por refuerzo. Surgió como evolución del proyecto original de OpenAI para mantener y mejorar el ecosistema de entornos estandarizados. Su principal función es proporcionar una interfaz común para interactuar con diversos entornos de simulación, definiendo una relación clara para las observaciones, acciones y mecanismos de recompensa.

Al analizar el funcionamiento interno de la interfaz Gymnasium en SMARTS, pudimos verificar cómo se generaban y formateaban las observaciones del LiDAR, la odometría del vehículo y otros estados relevantes del entorno. Esta comprensión profunda nos permitió adaptar nuestros algoritmos para procesar eficientemente estas observaciones sin necesidad de modificar la interfaz básica proporcionada por el simulador.

```
env = gym.make(
        "smarts.env:hiway-v1",
        scenarios=scenarios,
        agent_interfaces={AGENT_ID: agent_interface},
        headless=headless,
    )
env = SingleAgent(env)

```

```
for episode in episodes(n=num_episodes):
  agent = KeepLaneAgent()
  observation, _ = env.reset()
  episode.record_scenario(env.unwrapped.scenario_log)

  terminated = False
  while not terminated:
      action = agent.act(observation)
      observation, reward, terminated, truncated, info = env.step(action)
      episode.record_step(observation, reward, terminated, truncated,
    info)

env.close()
```

Extracto de código 3.1: Ejemplo de SMARTS usando gymnasium.

3.2. Pytorch

La elección de PyTorch [19] como entorno para implementar nuestros modelos de aprendizaje profundo se fundamenta en su diseño centrado en la flexibilidad y el control sobre la arquitectura de las redes neuronales. A diferencia de otros entornos más restrictivos, PyTorch ofrece una interfaz que permite moldear cada aspecto del proceso de aprendizaje.

La principal ventaja de PyTorch frente a alternativas como TensorFlow reside en su ejecución *eager*, que evalúa las operaciones inmediatamente en lugar de construir un grafo estático de computación. Este enfoque resulta particularmente valioso durante la fase de investigación y desarrollo, cuando se requiere experimentar rápidamente con diferentes arquitecturas.

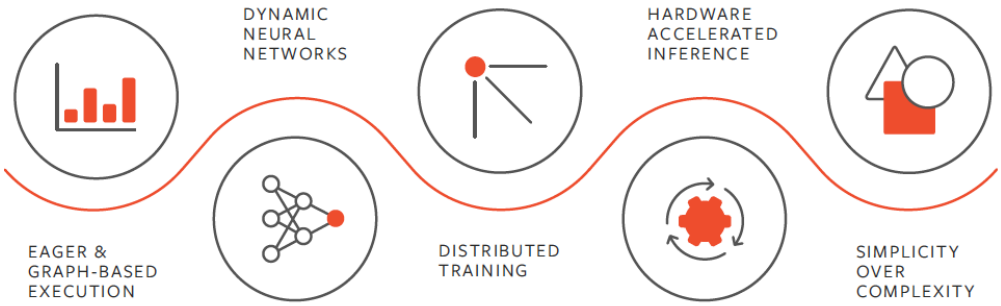


Figura 3.2: Características de Pytorch. [10]

Su elección se debió principalmente a su integración natural con Python, la abundante documentación disponible y la flexibilidad que ofrece para modificar la arquitectura de las redes neuronales. Durante el desarrollo, pudimos experimentar fácilmente con diferentes configuraciones de capas, funciones de activación y parámetros, lo que resultó clave para ajustar el rendimiento de nuestro agente de aprendizaje por refuerzo. La sencillez para intercambiar componentes permitió optimizar el modelo final sin necesidad de complejas reestructuraciones del código.

4. Modificaciones a SMARTS

SMARTS, al ser un simulador de código abierto, nos ha permitido realizar adaptaciones específicas para cumplir con los requisitos de nuestro sistema de estacionamiento autónomo. Estas modificaciones han sido esenciales para superar las limitaciones encontradas en la versión estándar. Los cambios implementados no solo han mejorado la precisión de la simulación, sino que también han facilitado la fase de entrenamiento de nuestros agentes al proporcionar un entorno más realista y consistente. Algunas de estas adaptaciones provienen del hecho de que SMARTS se ha utilizado típicamente para navegación de robots basada en posición y con escenarios globales (rotondas e intersecciones) con muchos otros vehículos. Ha sido necesario adaptarlo para centrarnos en escenarios locales, con una única carretera y navegación basada fundamentalmente en el sensor LiDAR.

4.1. Corrección de errores

Durante el desarrollo surgieron distintos problemas relacionados con el simulador que requirieron ajustes o modificaciones completas. Esta sección detalla las principales correcciones implementadas y su impacto en el sistema.

4.1.1. Medidas del sensor LiDAR

El sensor LiDAR emite pulsos láser para medir distancias a los obstáculos circundantes. SMARTS devuelve estas mediciones como coordenadas absolutas (x,y,z) de los puntos de impacto, formando una nube de puntos tridimensional. Esta representación permite al sistema analizar la disposición espacial del entorno del vehículo.

Durante las pruebas iniciales del entorno, se detectaron anomalías en el comportamiento del sensor LiDAR. Con la siguiente representación de la nube de puntos detectados (distintos de NaN o Inf):

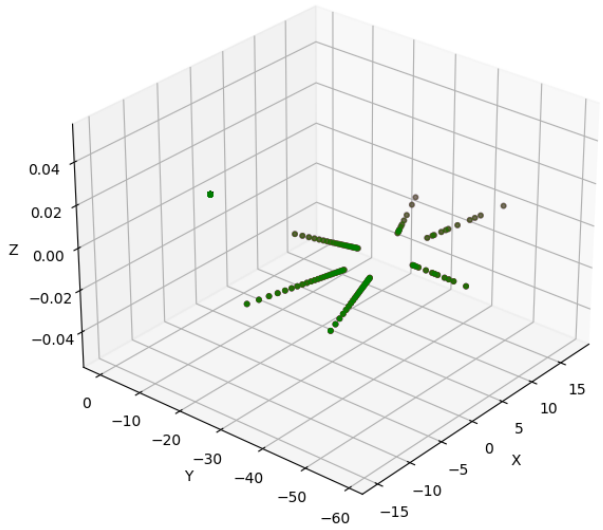


Figura 4.1: Nube de puntos por defecto.

Aunque a primera vista parecía estar funcionando correctamente, tras un análisis más detallado se comprobó que los rayos del sensor estaban detectando únicamente el suelo. Esto se debía a la configuración por defecto, en la cual el eje y del haz se extendía verticalmente, provocando una orientación demasiado inclinada hacia abajo.

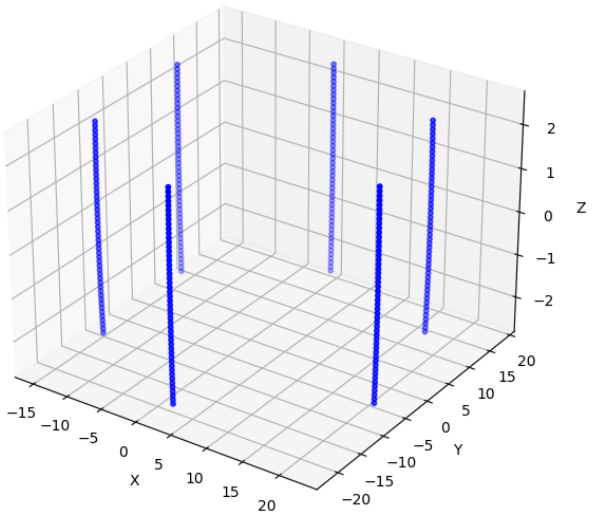


Figura 4.2: Vectores del LiDAR por defecto.

Para investigar más a fondo, se inspeccionaron todas las librerías relacionadas con el funcionamiento del sensor. Fue entonces cuando se identificó el archivo responsable de su configuración base. Dentro de este archivo, se encontró la clase *SensorParams*, que permitía modificar algunos parámetros.

Se realizó la siguiente modificación para reajustar el ángulo y reducir el campo de visión vertical, limitando así el contacto constante con el suelo:

```
BasicLidar = SensorParams(
    start_angle=0,
    end_angle=2 * np.pi,
    laser_angles=np.linspace(-np.radians(0.01), np.radians(0.01), 1),
    angle_resolution=0.02094,
    max_distance=20,
    noise_mu=0,
    noise_sigma=0.078,
)
```

Extracto de código 4.1: Parámetros para LiDAR funcional.

Nuevos vectores tras la modificación:

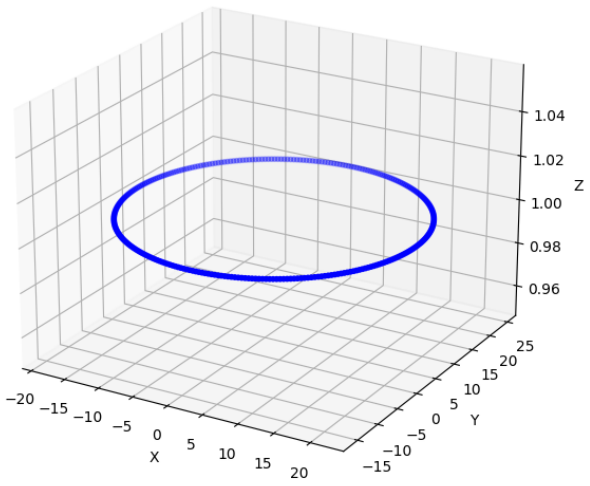


Figura 4.3: Vectores del LiDAR arreglados.

Tras varias pruebas se detectó un problema en el comportamiento del agente en 1-D, nunca parecía converger en una solución tras haber llegado a la posición

destino. Se analizaron los datos y se detectaron anomalías en las medidas del LiDAR cuando el agente se encontraba estático. Estas anomalías, curiosamente, coincidían con su posición actual.

SMARTS genera un conjunto de rayos desde un origen definido para simular el sensor LiDAR. Por eficiencia, estos rayos no se recalculan completamente en cada iteración, sino que se trasladan en función del movimiento del vehículo. Sin embargo, se detectó que, si el origen no varía entre dos pasos de simulación consecutivos, los rayos detectan el propio origen como un obstáculo, provocando lecturas incorrectas y comportamientos erráticos por parte del agente.

Este comportamiento se traduce en situaciones en las que el vehículo detecta un obstáculo inexistente justo en su centro, lo que afecta directamente al cálculo de recompensas, penalizaciones y a la toma de decisiones.

Para evitar que el sensor "se vea a sí mismo", se introdujo una pequeña variación aleatoria en la posición del origen en cada paso de simulación. Esta perturbación es mínima, por lo que no afecta a la precisión del sensor, pero es suficiente para evitar que se produzca una falsa detección.

El código utilizado es el siguiente:

```
1 def add_origin_perturbation(origin, magnitude=0.001):
2     """Add noise to origin"""
3     perturbation = np.random.uniform(-magnitude, magnitude, size=2)
4     return origin + np.array([perturbation[0], perturbation[1], 0])
```

Extracto de código 4.2: Ruido intencionado al origen del LiDAR.

4.1.2. Colisiones

Otro de los problemas detectados durante la personalización del entorno fue el mal funcionamiento del sistema de colisiones. Aunque SMARTS incorpora detección de colisiones de forma nativa y se registraban impactos entre el vehículo y otros obstáculos dinámicos como vehículos, el Lidar (ya arreglado) no era capaz de detectarlos.

Para solucionarlo, se implementó un sistema en PyBullet que utilizaba los modelos físicos de SMARTS para comprobar las detecciones del Lidar. Esta solución híbrida nos permitió validar de forma fiable las colisiones gráficamente.

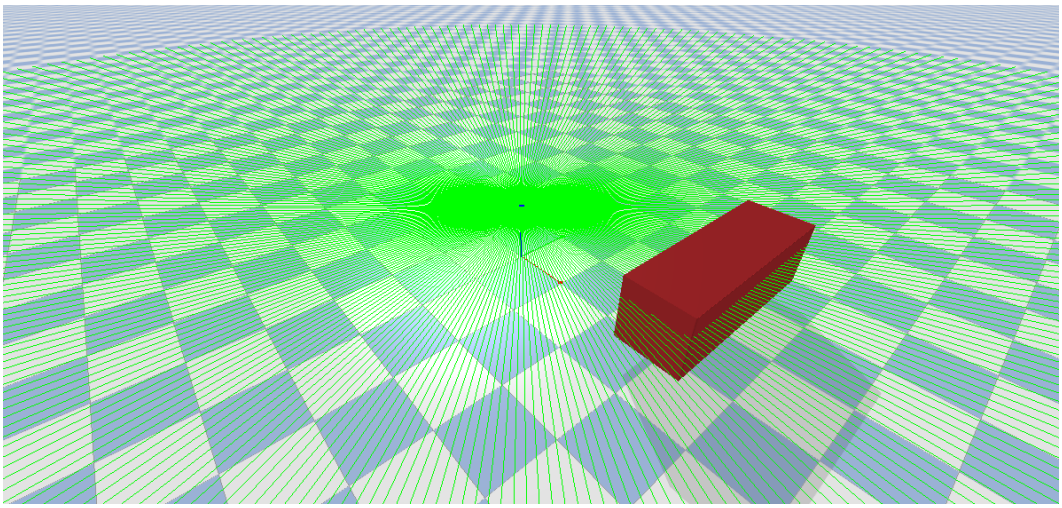


Figura 4.4: BoxChassis sin físicas.

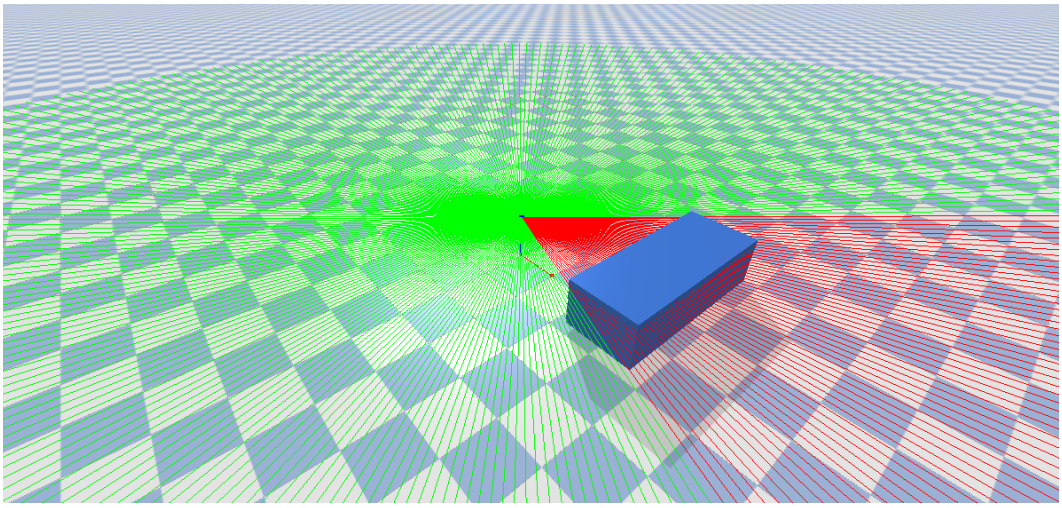


Figura 4.5: AckermannChassis con físicas.

Para todos los obstáculos dinámicos (vehículos), SMARTS usa obligatoriamente el *BoxChassis*, ya que se genera simplemente con un cubo de x dimensiones.

El sistema de físicas de SMARTS se basa en PyBullet, y tras un análisis exhaustivo del código se descubrió que el problema no estaba en el modelo del vehículo ni en el entorno, sino en la configuración del grupo de colisiones del objeto físico del coche.

Se le asignaba al vehículo un grupo y máscara de colisiones con valor 0x0, lo cual equivale a decir que no pertenece a ningún grupo de colisión. Como resultado, los vehículos no podían ser registrados como obstáculos desde Pybullet. Por simplificación, SMARTS no calcula las colisiones mediante Pybullet, sino que se

ayuda de la posición de cada obstáculo y su perímetro. Por ese motivo, el sistema detectaba colisiones físicas, pero los rayos del Lidar no.

Tras modificar el grupo de colisiones, el sistema empezó a funcionar correctamente.

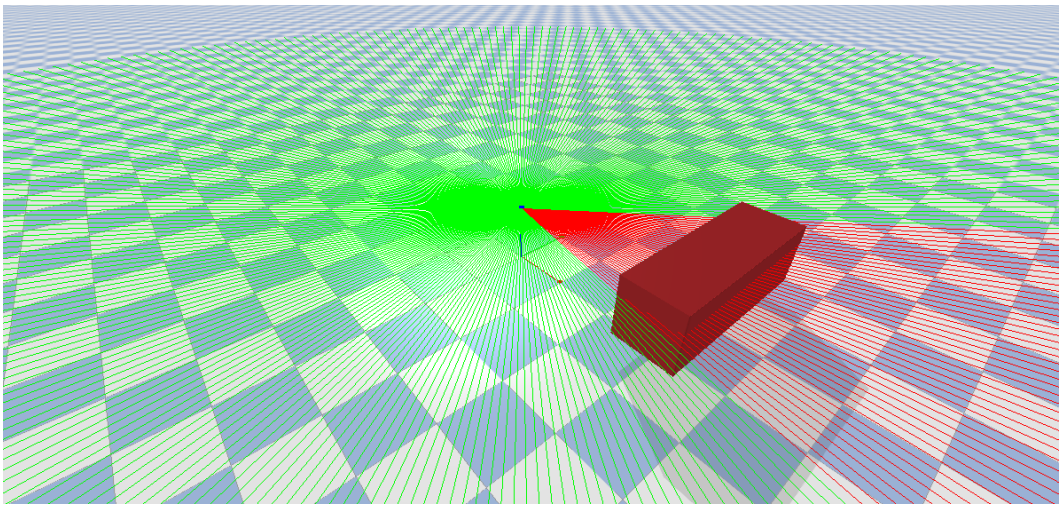


Figura 4.6: BoxChassis con físicas.

Este cambio no solo resolvió el problema en nuestro entorno, sino que se identificó como un bug general en la implementación por defecto de SMARTS. Por ello, la solución fue enviada al repositorio oficial mediante un pull request, y posteriormente fue aceptada e integrada por los desarrolladores del simulador.

commit oficial

4.1.3. Velocidades negativas

El soporte para velocidades negativas en SMARTS, esencial para maniobras de aparcamiento realistas como la marcha atrás, no fue una simple opción de configuración, sino un desafío. La arquitectura original de SMARTS, optimizada para flujos de tráfico convencionales, asumía que los vehículos solo se moverían hacia adelante. Para que esto se cumpliese, se usaban múltiples capas de control, eliminando cualquier valor negativo de velocidad antes de que afectara a la simulación.

El problema se agudizaba al comparar los dos modelos de vehículo disponibles. Mientras que el *BoxChassis*, al operar con aceleraciones/velocidades directas, permitía cierta flexibilidad para movimientos inversos, el *AckermannChassis*, diseñado para emular la física realista de un coche convencional,

bloqueaba cualquier intento de retroceso. Su sistema de control, basado en *throttle*, *brake* y *steering*, incorporaba validaciones que truncaban las velocidades negativas, interpretándolas como errores. Esto limitaba drásticamente las capacidades del vehículo, imposibilitando maniobras básicas como ajustar la posición con marcha atrás durante el estacionamiento.

La solución requirió una intervención en tres niveles críticos del simulador. Primero, en los controladores, donde se redefinieron los rangos válidos para el throttle, permitiendo valores que pudieran generar movimiento inverso. Segundo, en el cálculo de la física del chasis, donde se adaptaron las ecuaciones para interpretar correctamente las velocidades negativas, asegurando que la inercia y el frenado se comportaran como en el mundo real. Por último, en la conversión de acciones, donde se eliminaron filtros que descartaban cualquier indicación de retroceso.

Estos cambios no estuvieron exentos de complicaciones. SMARTS, al no estar diseñado para este uso, presentaba inconsistencias al calcular velocidades negativas; el último filtro se aseguraba de que la velocidad devuelta siempre fuese positiva. Una vez resuelto este problema, ya teníamos la nueva funcionalidad para conducir en sentido contrario.

4.2. Diseño de escenarios

4.2.1. Entorno dinámico

El simulador SUMO está diseñado para replicar situaciones de tráfico realista, donde los vehículos interactúan entre sí siguiendo reglas de conducción dinámica. En este contexto, los coches generados circulan a velocidades normales, manteniendo distancias de seguridad y ajustando su comportamiento en función del flujo de tráfico. Esta capacidad lo convierte en una herramienta ideal para simular escenarios en vías con circulación activa, donde el vehículo autónomo debe percibir y reaccionar ante el movimiento de otros agentes.

Sin embargo, SUMO no está pensado para manejar obstáculos completamente estáticos de forma convencional. Su motor de planificación de rutas asume que todos los vehículos son entidades dinámicas, lo que significa que si se intenta definir un coche como estático (velocidad cero), el sistema interpreta que está bloqueando la vía y genera errores en la generación de trayectorias para otros vehículos. Esto obliga a buscar soluciones alternativas para modelar situaciones

donde se requieren obstáculos inmóviles, como vehículos estacionados que delimitan los huecos de aparcamiento.

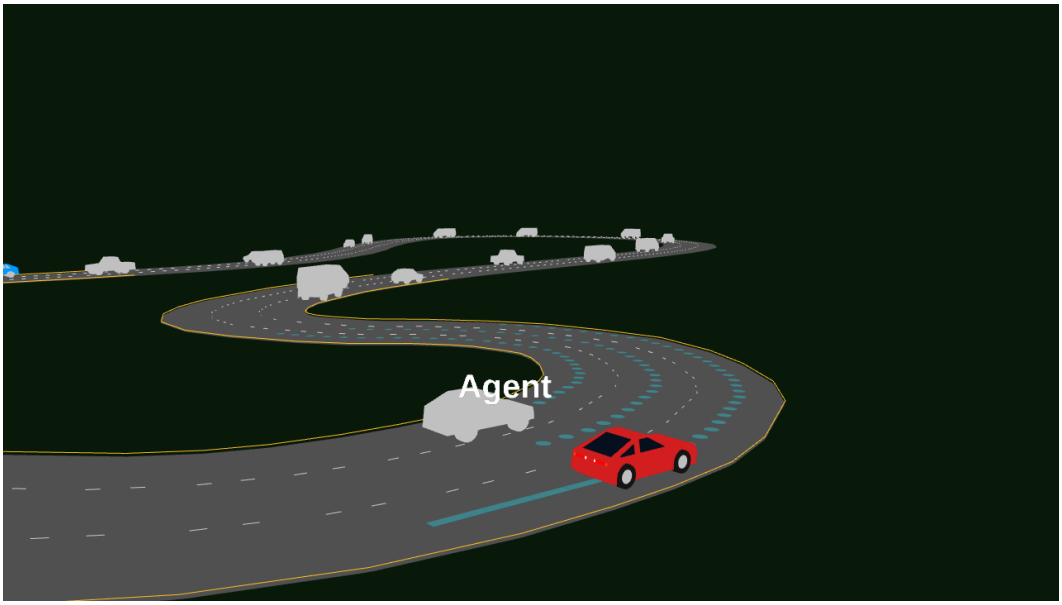


Figura 4.7: Escenario dinámico por defecto de SMARTS.

4.2.2. Entorno estático

Para simular obstáculos estáticos en SUMO sin provocar inestabilidades en el simulador, se implementó una solución basada en parámetros de tráfico modificados. En lugar de definir vehículos con velocidad cero (lo que causaría fallos en el cálculo de rutas), se establece una velocidad máxima extremadamente reducida en el archivo de configuración XML (por ejemplo, 0.01 mm/s). Este valor es lo suficientemente bajo como para que, en la práctica, los vehículos apenas se muevan durante la simulación, dando la apariencia de estar estacionados.

De este modo, aunque técnicamente todos los agentes siguen siendo dinámicos, el sistema logra emular un entorno estático donde los coches "estacionados" permanecen en sus posiciones iniciales, creando huecos de aparcamiento consistentes. Esta aproximación garantiza que el simulador no falle mientras se mantiene la utilidad del escenario para el entrenamiento del algoritmo de aprendizaje por refuerzo.

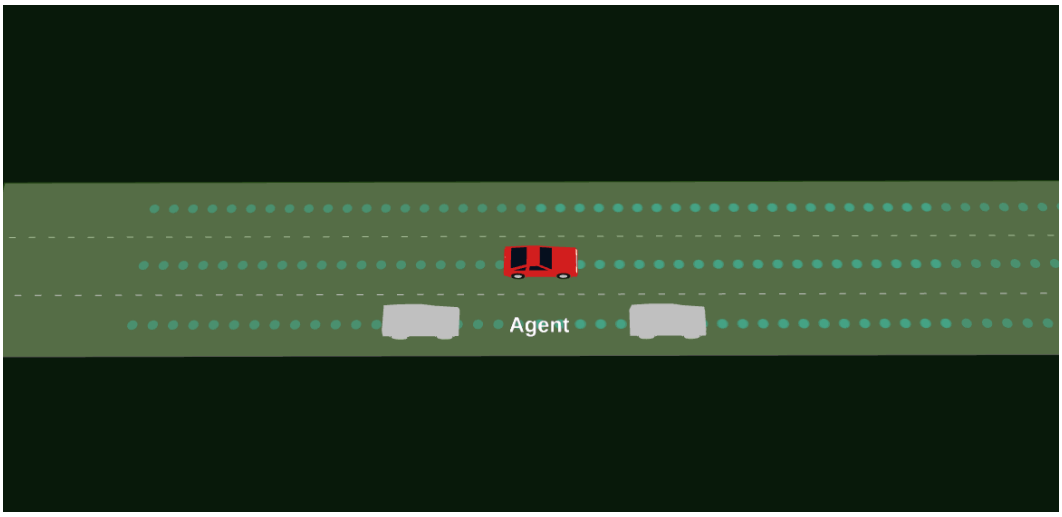


Figura 4.8: Escenario estático personalizado.

5. RL para autoparking

Este capítulo presenta el enfoque de aprendizaje por refuerzo aplicado al problema de estacionamiento autónomo, donde se combinan técnicas de RL y DRL con distintos escenarios.

5.1. Fundamentos y preprocesamiento

El aprendizaje por refuerzo aplicado al aparcamiento autónomo requiere transformar las observaciones del entorno en un formato adecuado para el entrenamiento del agente. Las lecturas del LiDAR se convierten a coordenadas relativas al vehículo, normalizando los datos para garantizar estabilidad en el aprendizaje.

La inicialización aleatoria del vehículo en distintas posiciones durante el entrenamiento asegura que el agente no memorice trayectorias fijas, sino que aprenda políticas generalizables.

En el caso del estacionamiento en 2D, se implementa un detector de huecos que identifica espacios válidos entre obstáculos. Este preprocesamiento es esencial para reducir la complejidad del espacio de estados y acelerar la convergencia del algoritmo, adaptando las observaciones brutas del simulador a una representación que el agente pueda aprender eficientemente.

5.1.1. Tratamiento de datos del LiDAR

El procesamiento de las nubes de puntos para el sistema de aparcamiento autónomo parte de una premisa fundamental: el vehículo debe entender su entorno no en términos absolutos, sino en relación con su propia posición y orientación. Esto no es una mera conveniencia técnica, sino una necesidad, donde el agente toma decisiones basadas en percepciones locales, que son las únicas disponibles en los sensores de los vehículos reales.

La transformación de LiDAR-SMARTS a coordenadas relativas del coche no es trivial. Cuando el vehículo captura datos del entorno mediante el LiDAR, la nube de puntos inicial está referenciada al sistema global del simulador. Sin embargo,

para que el modelo de aprendizaje por refuerzo pueda generalizar, es crucial que perciba los obstáculos y los posibles huecos de aparcamiento en función de su propia perspectiva. Aquí es donde entra en juego la matriz de transformación homogénea, que no solo traslada los puntos, sino que también los rota según la orientación actual del vehículo.

Pero hay un detalle crítico: antes de que el coche comience a maniobrar, debe guardar la orientación original de la carretera. ¿Por qué? Porque el espacio de aparcamiento no existe en el vacío; está alineado con una vía, y esa dirección inicial define qué es ”paralelo” o ”perpendicular”. Si ignoramos este paso, el vehículo podría malinterpretar la geometría del entorno al rotar.

5.1.2. Inicialización aleatoria del vehículo

Como SMARTS no permite generación aleatoria de escenarios estáticos (una limitación particularmente relevante para nuestro caso de aparcamiento), se implementó la clase *Desalignment* para introducir variabilidad en las posiciones iniciales de entrenamiento. Esta solución proporciona el elemento de aleatoriedad necesario para robustecer el aprendizaje y evitar el sobreajuste.

La lógica consiste en desplazamientos controlados del vehículo en dos dimensiones:

- **Desplazamiento longitudinal:** Aplica offsets aleatorios en el eje principal de la vía, seleccionados de un conjunto discreto de valores que cubren diferentes grados de desalineación inicial.
- **Rotación angular:** Opcionalmente introduce pequeñas variaciones en la orientación inicial (solo para parking 2-D).

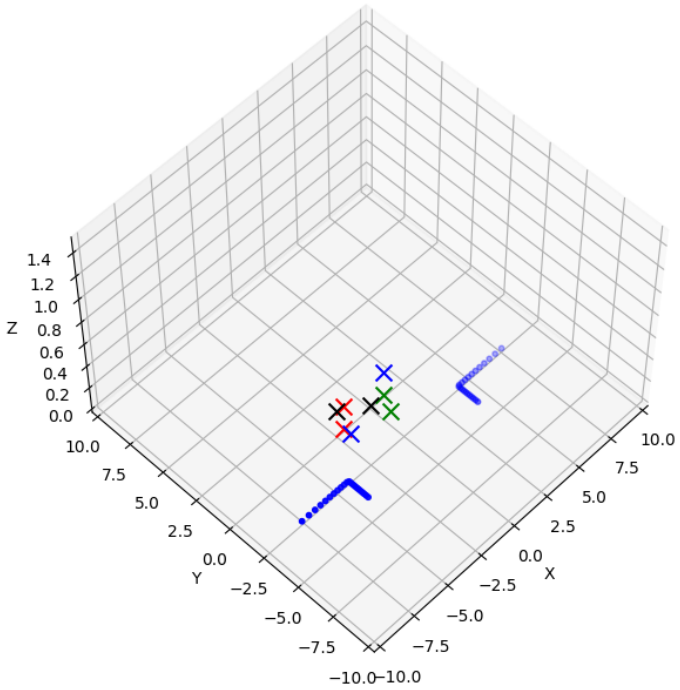


Figura 5.1: Ejemplo de posiciones aleatorias iniciales.

Durante el entrenamiento 1-D, se mantiene fija la orientación del vehículo (rotación desactivada), ya que este modo solo requiere ajustes de posición longitudinal. Para el parking 2-D, se activan ambas componentes de aleatorización (posición y orientación), simulando condiciones realistas donde el vehículo puede aproximarse al espacio de aparcamiento desde diferentes ángulos.

La transición a la posición inicial aleatoria se realiza mediante un controlador que acelera/desacelera hasta alcanzar la posición objetivo y frena cuando se aproxima al destino. Este proceso ocurre antes de comenzar el entrenamiento, asegurando que:

- No afecta a las métricas de aprendizaje (recompensas/estados).
- Mantiene constante el número total de episodios.

La implementación muestra cómo superar estas limitaciones del simulador mediante soluciones técnicas, proporcionando la variabilidad necesaria para un entrenamiento efectivo sin comprometer la integridad del proceso de aprendizaje.

5.1.3. Detección de huecos

El sistema desarrollado analiza la nube de puntos preprocesada para identificar espacios entre dos vehículos estacionados. Utilizando el algoritmo DBSCAN [20], agrupa los puntos en *clusters* que representan cada obstáculo, requiriendo que cada grupo cumpla con una densidad mínima de puntos y mantenga una separación adecuada entre sí. Estos parámetros aseguran que solo se consideren objetos bien definidos y suficientemente separados.

Para localizar el espacio disponible, el algoritmo busca el par de puntos más cercano entre los dos *clusters* obtenidos. Estos puntos representan las esquinas enfrentadas de los vehículos que delimitan el posible hueco de aparcamiento. El sistema calcula el punto medio entre estas esquinas y aplica un ajuste lateral que tiene en cuenta la geometría del propio vehículo.

Este ajuste posiciona el punto objetivo ligeramente desplazado del centro geométrico, garantizando que cuando el vehículo alcance esta posición, quede correctamente centrado en el espacio disponible. El resultado es una coordenada precisa que guía la maniobra de estacionamiento.

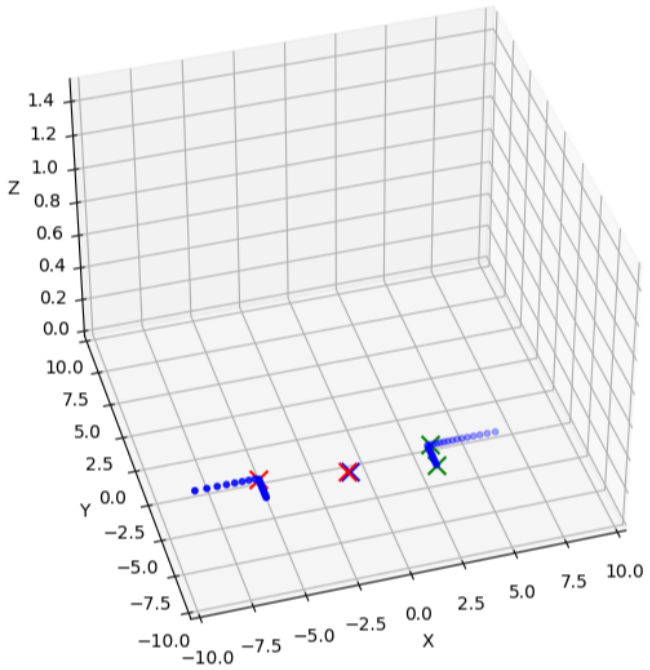


Figura 5.2: Nube de puntos con delimitadores del hueco.

5.1.4. Actualización dinámica del objetivo

El sistema implementa un mecanismo de referencia relativa para mantener un seguimiento preciso del objetivo durante toda la maniobra. La estrategia se basa en un principio fundamental: en lugar de recalcular constantemente la posición del hueco, el sistema establece una referencia espacial relativa en el momento de la detección inicial.

Cuando se identifica por primera vez un espacio apto para aparcar, el sistema almacena su posición relativa al vehículo en ese instante exacto. Esta referencia inicial se convierte en el punto de anclaje para todos los cálculos posteriores. A medida que el vehículo se mueve, el sistema actualiza dinámicamente la posición del objetivo.

Este enfoque ofrece varias ventajas clave:

- Elimina la necesidad de redetectar el hueco en cada frame.
- Reduce el error acumulativo que surgiría de cálculos independientes.
- Mantiene coherencia espacial durante toda la maniobra.

La transformación se realiza mediante operaciones vectoriales que consideran tanto la traslación como la rotación del vehículo, garantizando que el objetivo se mantenga estable mientras el vehículo ajusta su posición. Se actualiza en cada lectura sensorial aplicando la matriz de rotación para mantener la alineación correcta con el espacio de aparcamiento.

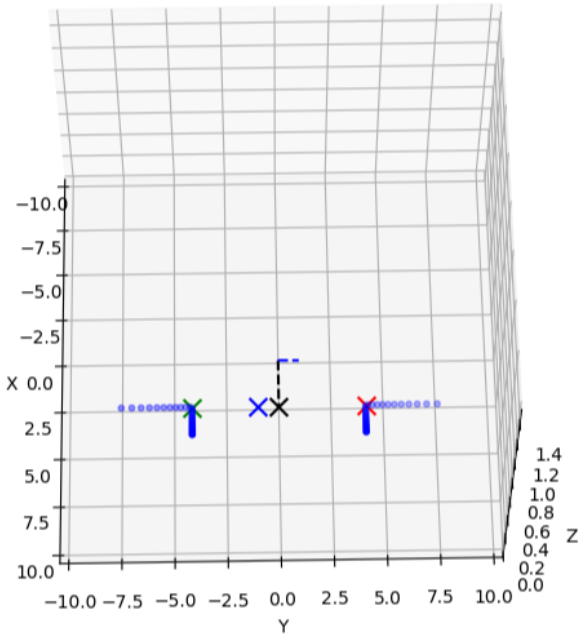


Figura 5.3: Nube de puntos con vectores de objetivo.

Explicación de la figura 5.3:

- **Puntos:**
  - **Verde y rojo:** Esquinas de los vehículos.
  - **Negro:** Punto medio (ejemplo sin añadir profundidad).
  - **Azul:** Posición actual del vehículo.
- **Vectores:** Todos en coordenadas relativas (desde (0,0)).
  - **Negro:** Vector desde la posición inicial hasta el objetivo. Este es el cálculo inicial.
  - **Azul:** Vector que muestra posición del objetivo respecto a la posición actual.

5.2. Parking 1-D

El estacionamiento autónomo en una dimensión sirvió como caso de estudio inicial para validar los fundamentos del sistema antes de abordar problemas más

complejos. El escenario consistía en posicionar un vehículo en el punto medio entre dos obstáculos fijos, simulando una plaza de parking entre otros coches.

El enfoque unidimensional eliminó variables como la orientación o el movimiento lateral, concentrándose exclusivamente en la coordinación entre aceleración, frenado y percepción de distancias.

Este ejercicio demostró la viabilidad del aprendizaje por refuerzo para problemas de posicionamiento preciso, sentando las bases técnicas que luego se extenderían al caso bidimensional. La simplicidad controlada del escenario 1-D facilitó la depuración temprana de componentes críticos como la función de recompensa y la discretización de estados, esenciales para el rendimiento final del sistema. Aunque muchos de ellos serían modificados y ampliados más adelante.

5.2.1. Algoritmo

El sistema implementa Q-Learning clásico, donde el agente aprende mediante una tabla que asocia estados discretos a valores de acción. La política se optimiza iterativamente a través de la interacción con el entorno, siguiendo la ecuación de Bellman.

En código:

```
# Hiperparametros clave
alpha = 0.2  # Tasa de aprendizaje
gamma = 0.9  # Factor de descuento
epsilon = 0.99  # Probabilidad inicial de exploracion
min_epsilon = 0.0  # Limite inferior para exploracion
decay_rate = 0.99  # Decaimiento de epsilon por episodio

# Ecuacion de actualizacion Q-Learning
q_table[state][action] += alpha * (reward + gamma * max(q_table[
    next_state]) - q_table[state][action])
```

Extracto de código 5.1: Hiperparámetros y función de actualización 1D.

El espacio de estados se discretiza en bins de 0.25 metros para la posición y 0.1 m/s para la velocidad, mapeando las observaciones continuas del simulador a un conjunto finito de casos. Las acciones disponibles son tres valores discretos de

aceleración (acelerar en sentido negativo, mantener velocidad, o acelerar en positivo).

La exploración inicial se guía por una política ϵ-greedy, donde el agente alterna aleatoriamente entre:

- Explotar el conocimiento actual (seleccionar la acción con mayor valor Q).
- Explorar nuevas opciones (acciones aleatorias).

```python
def choose_action(self, state):
        """Selecciona una accion basada en la politica epsilon-greedy."""
        if np.random.rand() < self.epsilon:
            # Explorar: Elegir una accion aleatoria
            return np.random.choice(self.actions)

        # Explotar: Elegir la mejor accion conocida
        if state not in self.q_table:
            # Inicializar valores Q para acciones en el estado si no existen
            self.q_table[state] = {action: 0.0 for action in self.actions}

        # Devolver la accion con el valor Q mas alto en este estado
        return max(self.q_table[state], key=self.q_table[state].get)
```

Extracto de código 5.2: Función exploración/explotación 1D.

La probabilidad de exploración (ϵ-greedy) decae exponencialmente con cada episodio para priorizar progresivamente el conocimiento aprendido.

$$\epsilon = max(\epsilon_{\text{min}},\ \epsilon \cdot \text{decay\_rate})$$

Ecuación 5.1: Decaimiento exponencial de ϵ.

Esta implementación minimalista demostró ser suficiente para resolver el problema 1D, donde la tabla Q converge a una política óptima en aproximadamente 250 episodios de entrenamiento. El éxito del enfoque radica en la adecuada discretización del espacio y el diseño de recompensas, no en la complejidad algorítmica.

5.2.2. Recompensas

El sistema de recompensas implementado sigue los principios fundamentales del Q-learning, donde el agente aprende a través de retroalimentación positiva y negativa basada en sus acciones. En este escenario unidimensional de aparcamiento, la función de recompensa se centra en dos aspectos clave: el correcto posicionamiento respecto al espacio de aparcamiento y la seguridad durante la maniobra.

La recompensa principal se calcula analizando la simetría del entorno mediante lecturas LiDAR en ángulos específicos. El sistema compara las distancias a obstáculos delanteros y traseros del vehículo, generando una recompensa inversamente proporcional a la diferencia entre estas mediciones.

- **Recompensas:**

  Estándar: Inversamente proporcional a la diferencia de distancias de los obstáculos delanteros/traseros, incentivando el movimiento hacia la posición central.

  Estacionamiento perfecto: Se otorga cuando el vehículo está completamente centrado en el espacio de aparcamiento y completamente detenido.

- **Penalizaciones:**

  Riesgo de colisión: Se aplica cuando las lecturas del LiDAR muestran una proximidad excesiva a los obstáculos.

  Velocidad inadecuada: Se activa cuando el vehículo supera cierta velocidad, considerada peligrosa para una maniobra de estacionamiento.

- **Casos especiales:**

  Lecturas inválidas del LiDAR: Cuando los sensores no detectan obstáculos (distancias infinitas), el sistema no otorga recompensa.

El sistema incluye un umbral mínimo para la diferencia de distancias, evitando divisiones por cero y estabilizando el aprendizaje.

Este esquema proporciona al agente una señal de aprendizaje clara y gradual, donde las mayores recompensas corresponden a los estados más deseables (centrado y detenido), mientras que las penalizaciones desalientan comportamientos ineficientes.

5.2.3. Estados y observaciones

El sistema utiliza como entrada las observaciones estándar proporcionadas por SMARTS[1]. Estas observaciones brutas se procesan para crear una representación del estado adecuada para el aprendizaje por refuerzo.

El estado del agente se compone de dos características principales derivadas de las observaciones ya procesadas, como se explicó anteriormente. Por un lado, se analiza la diferencia entre las distancias a obstáculos delanteros y traseros, obtenidas mediante las lecturas LiDAR en las direcciones frontal (90°) y posterior (270°). Esta diferencia con signo se discretiza para crear una representación más manejable del entorno, proporcionando información crucial sobre la posición relativa del vehículo respecto al espacio de aparcamiento, indicando si necesita avanzar o retroceder para centrarse.

La segunda componente del estado es la velocidad del vehículo, que también se somete a un proceso de discretización. Este tratamiento de la velocidad permite al agente distinguir claramente diferentes situaciones sin verse afectado por pequeñas variaciones irrelevantes.

La diferencia entre las distancias constituye una característica particularmente importante, ya que proporciona el grado de alineación con el aparcamiento. Esta medida de asimetría, combinada con la información de velocidad, forma un espacio de estados bidimensional que resulta suficiente para el problema 1D mientras mantiene la complejidad manejable. El proceso de discretización aplicado a ambas variables ayuda a generalizar el aprendizaje y reduce la sensibilidad a pequeñas variaciones en las observaciones.

5.3. Parking 2-D Holonómico

El sistema de estacionamiento autónomo implementado controla un vehículo con capacidades de movimiento similares a un robot diferencial: puede desplazarse hacia adelante/atrás y girar sobre su propio eje, pero no tiene movimiento lateral (restricciones del simulador). Esta característica simplifica las maniobras en comparación con un automóvil convencional, pero mantiene restricciones realistas en cuanto a radios de giro y cinemática.

[1] https://smarts.readthedocs.io/en/latest/sim/obs_action_reward.html

5.3.1. Arquitectura

Inicialmente, se exploraron diferentes enfoques de aprendizaje por refuerzo. Primero se implementó Q-learning tradicional, pero rápidamente se encontraron limitaciones debido al gran tamaño del espacio de estados, lo que hacía inviable su convergencia. Posteriormente, se probó con Deep Q-Network (DQN), pero este presentaba problemas de sobreestimación de los valores Q. Finalmente, se logró estabilizar el aprendizaje mediante Double DQN.

5.3.2. Implementación

El sistema implementado utiliza un enfoque de aprendizaje por refuerzo profundo. A continuación, se detallan los componentes clave:

Arquitectura de la red neuronal.

Se optó por una arquitectura de red neuronal sencilla pero efectiva, tras experimentar con diferentes configuraciones. Inicialmente, probamos con una sola capa oculta, pero resultó insuficiente. Finalmente, adoptamos el enfoque clásico de dos capas ocultas, que ofrece el equilibrio perfecto entre capacidad de aprendizaje y simplicidad para nuestro caso de uso.

```
class DQN(nn.Module):
    def __init__(self, state_size, action_size):
        super(DQN, self).__init__()
        self.fc1 = nn.Linear(state_size, 128)
        self.fc2 = nn.Linear(128, 64)
        self.fc3 = nn.Linear(64, action_size)

        self.layer_norm1 = nn.LayerNorm(128)
        self.layer_norm2 = nn.LayerNorm(64)

    def forward(self, x):
        x = torch.relu(self.layer_norm1(self.fc1(x)))
        x = torch.relu(self.layer_norm2(self.fc2(x)))
        return self.fc3(x)
```

Extracto de código 5.3: Red neuronal con dos capas ocultas.

Esta estructura estándar nos permite:

- Aprender representaciones útiles del estado sin caer en sobreingeniería.
- Mantener tiempos de entrenamiento razonables.
- Evitar la complejidad innecesaria de arquitecturas más profundas.
- Lograr un buen rendimiento sin requerir ajustes excesivos.

Política de acción *Softmax*

La selección de acciones no sigue el enfoque ϵ-greedy tradicional, sino que utiliza una política *softmax* con temperatura. Esta estrategia asigna probabilidades a cada acción según sus valores Q estimados, donde τ (temperatura) controla el nivel de exploración. Cuando τ es alto, todas las acciones tienden a tener probabilidades similares, favoreciendo la exploración. A medida que el agente aprende, τ puede disminuir para favorecer acciones con mayores valores Q.

```
def act(self, state):
        if np.random.rand() < self.epsilon:
            return random.choice(self.actions)

        state_tensor = torch.FloatTensor(state).unsqueeze(0).to(self.
    device)
        with torch.no_grad():
            q_values = self.model(state_tensor)

        try:
            tau = 0.5
            probs = torch.nn.functional.softmax(q_values / tau, dim=1)  #
     Q - probs
            action_index = torch.multinomial(probs, num_samples=1).item()
            return self.actions[action_index]
        except IndexError:
            return random.choice(self.actions)  # error - random
```

Extracto de código 5.4: Política Softmax.

Double DQN

El algoritmo Double DQN aborda el problema de sobreestimación de valores Q que afecta al DQN estándar. Utiliza dos redes neuronales separadas: una red principal para seleccionar acciones y una red objetivo para evaluarlas. Esta separación evita el sesgo positivo que surge cuando la misma red estima tanto los

valores como las políticas. La red objetivo se actualiza periódicamente mediante "*soft updates*", donde sus parámetros se mezclan gradualmente con los de la red principal, lo que proporciona mayor estabilidad durante el entrenamiento.

Replay Buffer

El sistema implementa un buffer de experiencias con capacidad para x transiciones. Este componente almacena tuplas (estado, acción, recompensa, siguiente estado, flag de terminación) que se generan durante la interacción con el entorno. Durante el entrenamiento se muestrean lotes aleatorios de este buffer, rompiendo así las correlaciones temporales entre experiencias consecutivas y permitiendo el aprendizaje fuera de política. El muestreo aleatorio ayuda a que el modelo generalice mejor y evite sobreajustarse a secuencias específicas.

```
class ReplayBuffer:
    def __init__(self, capacity):
        self.buffer = deque(maxlen=capacity)

    def push(self, state, action, reward, next_state, done):
        self.buffer.append((state, action, reward, next_state, done))

    def sample(self, batch_size):
        return random.sample(self.buffer, batch_size)

    def __len__(self):
        return len(self.buffer)
```

Extracto de código 5.5: Replay Buffer.

Espacio de estados y acciones

El espacio de estados incluye información discretizada sobre:

- Distancia al objetivo.
- Error de orientación.
- Velocidad.
- Distancia de obstáculo más cercano.

Las acciones disponibles son discretas y representan combinaciones de movimiento lineal y angular:

- Movimiento hacia atrás.

- Giro a la izquierda.
- Mantener posición (acción neutra).
- Giro a la derecha.
- Movimiento hacia adelante.

Este diseño permite maniobras complejas mediante secuencias de acciones simples, aprovechando la capacidad del vehículo para girar sobre su propio eje.

Mecanismo de recompensas

El sistema de recompensas utiliza un enfoque híbrido y discreto que combina múltiples componentes para guiar eficazmente al vehículo hacia la posición de estacionamiento. Se ha optado por un diseño con transiciones y umbrales definidos, con el que se han superado los mínimos locales.

Componentes clave del sistema de recompensas:

- **Recompensa por distancia:**
  - Utiliza una función de recompensa sigmoide inversa para la distancia euclídea al objetivo.
  - Penalización máxima cuando el vehículo se aleja demasiado o cruza ciertos límites virtuales (pared/bordillo).
- **Recompensa por orientación:**
  - Solo se activa cuando el vehículo está cerca del objetivo. Dependiendo del error de orientación puede ser una penalización o recompensa.
  - Penalización máxima cuando la desorientación supera 75º grados. Se utiliza como medida de seguridad para que el vehiculo no se coloque perpendicular a la carretera.
- **Control de velocidad:**
  - Penaliza las velocidades excesivas.
  - Recompensa las velocidades bajas cerca del objetivo.
- **Bonificación por estacionamiento perfecto:**
  - Gran recompensa proporcional al tiempo restante cuando se alcanza la posición exacta. Una vez obtenida se termina el episodio.
- **Detección de colisiones:**

- Penalización máxima por acercarse demasiado a cualquier obstáculo.

5.4. Parking 2-D Ackermann

La transición del modelo holonómico al sistema Ackermann requirió modificaciones en el control del vehículo y en la representación de las acciones. A diferencia del modelo anterior, donde el vehículo podía girar sobre su eje, el sistema Ackermann introduce restricciones cinemáticas realistas que afectan directamente a la estrategia de control.

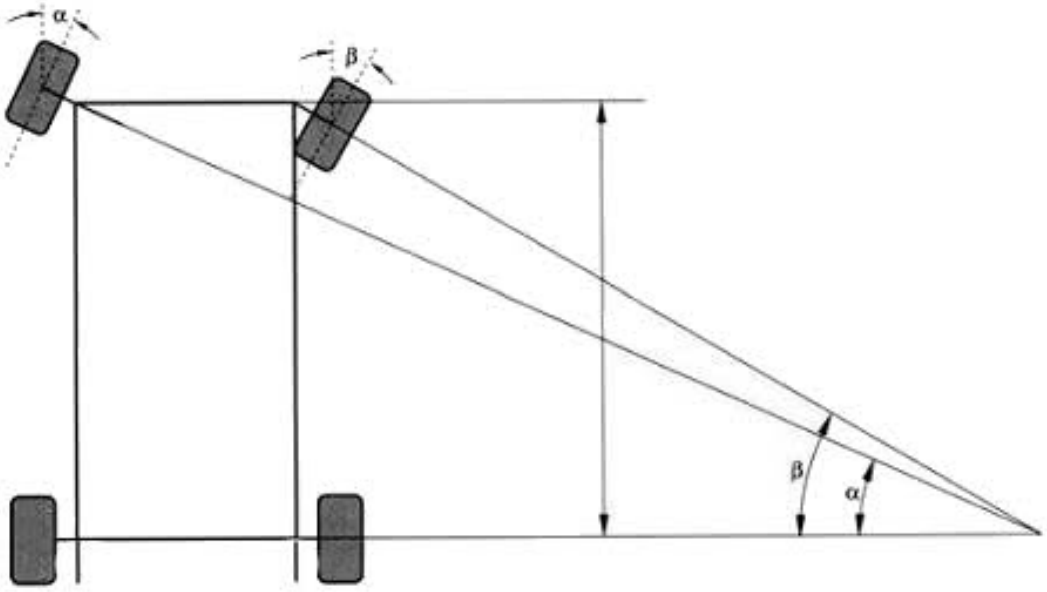


Figura 5.4: Técnica de dirección Ackermann. [11]

El principal cambio fue la redefinición del espacio de acciones. Donde antes teníamos movimientos discretos combinando traslación y rotación, ahora trabajamos con un sistema de control continuo que comprende tres componentes fundamentales: *throttle* (aceleración), *brake* (frenado) y *steering* (ángulo de dirección). Esta modificación obligó a rediseñar completamente la capa de interacción con el simulador.

5.4.1. Arquitectura

Para mantener la coherencia con el enfoque anterior y aprovechar el conocimiento adquirido, conservamos la estructura básica de Double DQN

(DDQN) que había demostrado buenos resultados en la versión holonómica. La red neuronal mantiene su arquitectura de dos capas ocultas, pero ahora recibe como entrada un estado ampliado que incluye información adicional sobre la dinámica del vehículo.

5.4.2. Implementación

Aunque la arquitectura base mantiene los componentes principales del sistema holonómico, la transición al modelo Ackermann introdujo algunos cambios clave. Estos ajustes fueron necesarios para manejar las restricciones cinemáticas propias de un vehículo con dirección Ackermann, donde el control requiere una coordinación más precisa. A continuación, nos centraremos en estas modificaciones esenciales, destacando cómo afectan al aprendizaje del agente y por qué fueron críticas para lograr un estacionamiento realista y estable.

Espacio de estados y acciones

El sistema de estados fue completamente rediseñado para el modelo Ackermann, pasando de una representación simple a un conjunto de variables discretizadas que capturan con precisión la relación entre el vehículo y su objetivo de estacionamiento. Estos son los componentes clave del estado:

- Distancia horizontal discretizada: Representa la distancia lateral entre el vehículo y el centro del espacio de estacionamiento.
- Distancia vertical discretizada: Indica la distancia longitudinal respecto al punto objetivo.
- Error de orientación: Mide la diferencia angular entre la orientación actual y la deseada.
- Velocidad discretizada.
- Distancia mínima a obstáculos.

El principal cambio fue la descomposición de la distancia absoluta en componentes horizontales y verticales independientes. Mientras que en el modelo holonómico podíamos trabajar con una representación unificada del espacio gracias a su mayor libertad de movimiento, la naturaleza restrictiva de la dinámica Ackermann nos obligó a desarrollar un modelo de estados más granular y preciso. ¿Por qué separar horizontal y vertical? En el estacionamiento real, un metro de distancia no significa lo mismo si es lateral (horizontal) o frontal (vertical).

Horizontalmente, incluso pequeñas desviaciones requieren correcciones agresivas de dirección. Verticalmente, el vehículo puede permitirse mayor margen. Esta separación permite al agente aprender estrategias distintas para cada eje de movimiento.

Por otro lado, el espacio de acciones experimentó una transformación radical al migrar al modelo Ackermann. Donde antes teníamos movimientos discretos simples, ahora implementamos un sistema multidimensional que combina tres variables críticas de control:

- Frenar.
- Giro a la izquierda.
- Marcha atrás.
- Acelerar.
- Giro a la derecha.
- Acelerar suave y giro a la derecha.
- Acelerar suave y giro a la izquierda.
- Marcha atrás suave y giro a la derecha.
- Marcha atrás suave y giro a la izquierda.

La inclusión de acciones compuestas (aceleración + dirección) fue particularmente importante para superar las nuevas limitaciones, permitiendo al vehículo ejecutar las maniobras de aproximación al estacionamiento típicas de los vehículos reales.

Mecanismo de recompensas

El diseño del sistema de recompensas para el modelo Ackermann representó uno de los desafíos más complejos del proyecto, requiriendo varios modelos hasta alcanzar un esquema efectivo que guiara adecuadamente el aprendizaje del agente.

Inicialmente, se implementaron dos enfoques contrastantes:

- Recompensa respecto al objetivo final: Un sistema premiaba la reducción progresiva de la distancia euclídea y la correcta orienatción del agente.
- Enfoque por submetas (*reward shaping*): Dividiendo la maniobra en fases discretas (alineación, aproximación, corrección angular). Este segundo

enfoque, pese a su fundamentación teórica, no logró converger en la práctica debido a la dificultad de balancear las recompensas parciales.

La solución final implementa un sistema continuo de recompensas que evita las transiciones bruscas de los condicionales discretos, permitiendo un aprendizaje más estable.

Componentes sistema de recompensas:

- **Recompensa por distancia:**
  - Basada en una función sigmoide inversa que varía suavemente con la distancia euclidiana al punto de aparcamiento.
  - Penalización máxima cuando el vehículo se aleja demasiado o cruza ciertos límites virtuales (pared/bordillo).
- **Recompensa por orientación:**
  - Comienza a ser efectiva cuando el vehículo está cerca del objetivo. Dependiendo del error de orientación puede ser una penalización o recompensa.
  - Recompensa el progreso en la corrección del ángulo. Incluye penalizaciones para evitar estancamiento en orientaciones incorrectas.
  - Penalización máxima cuando la desorientación supera 75º grados. Se utiliza como medida de seguridad para que el vehiculo no se coloque perpendicular a la carretera.
- **Incentivo direccional:**
  - Recompensa específicamente las aproximaciones marcha atrás.
  - Penaliza las aproximaciones frontales.
- **Bonus progresivo por precisión final:**
  - Un término gaussiano que premia la simultánea aproximación en posición y orientación, maximizándose cuando el vehículo alcanza el estado objetivo con precisión.
- **Detección de colisiones:**
  - Penalización máxima por acercarse demasiado a cualquier obstáculo.
- **Control de velocidad:**
  - Penaliza las velocidades excesivas.

- Recompensa progresivamente las velocidades bajas cerca del objetivo.

El componente direccional (marcha atrás vs frontal) resultó particularmente crucial para resolver los problemas de orientación observados inicialmente. Al incentivar explícitamente las aproximaciones marcha atrás, el vehículo aprendió estrategias más eficientes para alinearse correctamente con el espacio de aparcamiento.

6. Validación experimental

El proceso de validación del sistema se ha estructurado en varias fases, cada una diseñada para evaluar aspectos específicos del rendimiento y la eficacia de los algoritmos desarrollados. Durante todo el desarrollo del TFG se implementó un sistema de métricas de evaluación que incluyó un programa de pruebas interactivo para la teleoperación controlada del agente. Esta herramienta permitió analizar en detalle maniobras específicas, calibrar parámetros como la duración de episodios y recopilar datos en condiciones controladas para la depuración.

La metodología incorporó además herramientas avanzadas de visualización para representar gráficamente las nubes de puntos procesadas, para depurar el sistema de percepción y validar la correcta interpretación del entorno. Para cada modalidad de aparcamiento (1D y 2D) se implementaron protocolos de evaluación adaptados a sus particularidades. Estas capacidades de diagnóstico permitieron una optimización de todos los componentes del sistema, desde los algoritmos de procesamiento sensorial hasta las políticas de control.

6.1. Parking 1-D

6.1.1. Métricas de evaluación

La validación del sistema de aparcamiento unidimensional se desarrolló en torno a dos puntos: el análisis de la evolución de los parámetros de aprendizaje y un programa de pruebas específicamente diseñado para verificar el correcto funcionamiento con la tabla Q generada.

Antes de iniciar las sesiones de aprendizaje, se realizó un estudio del parámetro de exploración (ϵ), generando curvas teóricas que modelaban su evolución a lo largo de los episodios. Este análisis inicial permitió establecer una política de equilibrio entre exploración y explotación, ajustando la tasa de decaimiento basada en el número de episodios.

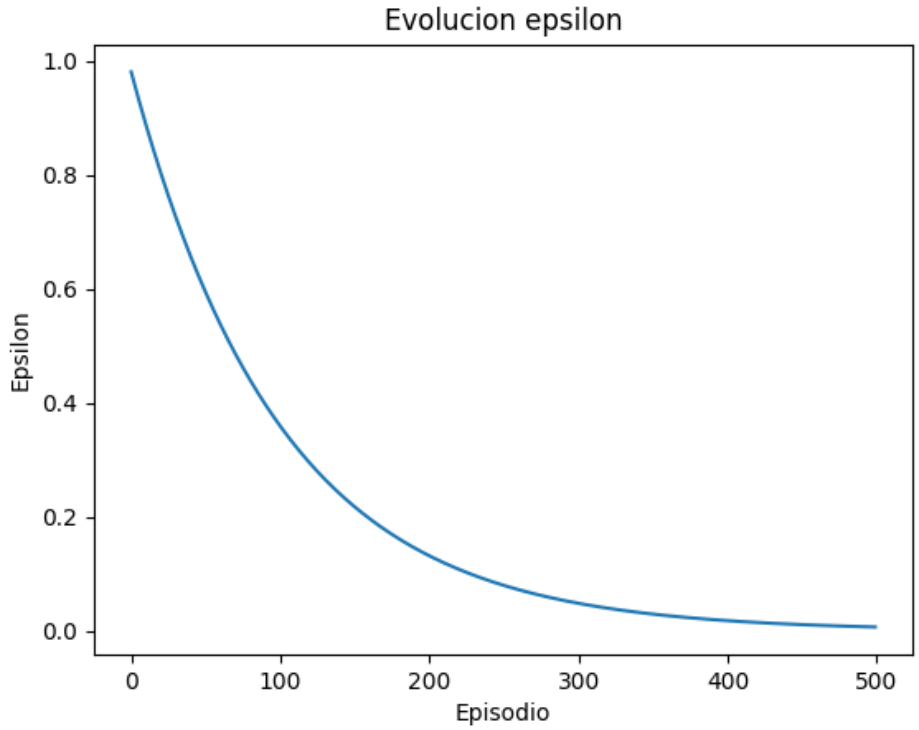


Figura 6.1: Evolución de ϵ en aparcamiento 1D.

Durante el entrenamiento, el sistema recopiló de forma sistemática un conjunto de métricas que capturaban la dinámica del proceso de aprendizaje. Se registraron medidas como el número de pasos, las recompensas obtenidas y la diferencia de distancia al objetivo. Estos datos se almacenaron en un formato estructurado que permitiría su posterior análisis.

Una vez completado el entrenamiento, el análisis de los datos brindaba información significativa sobre el comportamiento del agente. Las visualizaciones mostraron claramente cómo la política de decisión evolucionaba de manera consistente. Se observó una correlación directa entre la reducción de la distancia al objetivo y el aumento de las recompensas obtenidas, confirmando la validez del diseño del espacio de estados y la efectividad del mecanismo de recompensas implementado.

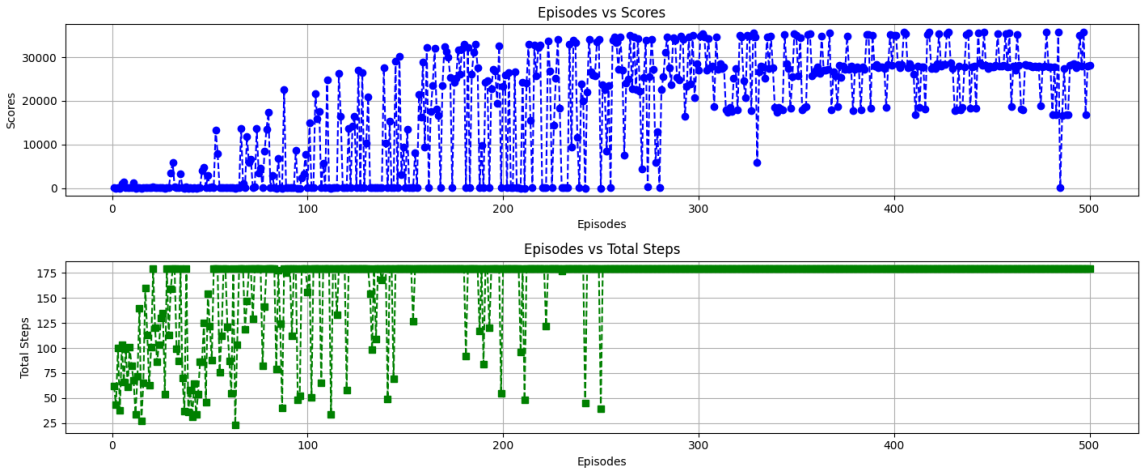


Figura 6.2: Recompensas y pasos por episodio en aparcamiento 1D.

Las gráficas muestran dos aspectos fundamentales de la evolución del aprendizaje del agente. En la gráfica de pasos por episodio, se observa claramente cómo el agente comienza utilizando menos pasos de los disponibles, pero progresivamente aprende a aprovechar el tiempo máximo asignado para cada episodio. Este comportamiento refleja el aprendizaje de estrategias más sofisticadas para evitar colisiones y realizar aproximaciones más precisas al objetivo.

La curva de recompensa por episodio complementa esta información, mostrando una mejora progresiva a medida que avanza el entrenamiento. Inicialmente, las recompensas son bajas e incluso nulas, correspondientes a la fase de exploración aleatoria. Conforme el agente acumula experiencia, se aprecia una tendencia ascendente y más estable, logrando finalmente la convergencia del aprendizaje. La correlación entre ambas gráficas demuestra cómo el agente no solo aprende a evitar penalizaciones, sino que también optimiza su política para maximizar las recompensas obtenidas.

6.1.2. Resultados

El sistema de aparcamiento 1D demostró su eficacia tras completar satisfactoriamente la fase de entrenamiento y alcanzar una convergencia estable en la política de aprendizaje. Para validar su rendimiento, se realizaron pruebas utilizando la tabla Q generada, evaluando el comportamiento del agente desde distintas posiciones iniciales. Estos tests mostraron cómo el vehículo era capaz de converger correctamente al punto de estacionamiento objetivo en todos los casos, independientemente de su posición de partida. Los resultados confirmaron que el

enfoque de Q-learning, a pesar de su relativa simplicidad, resultó completamente adecuado para resolver el problema unidimensional de aparcamiento.

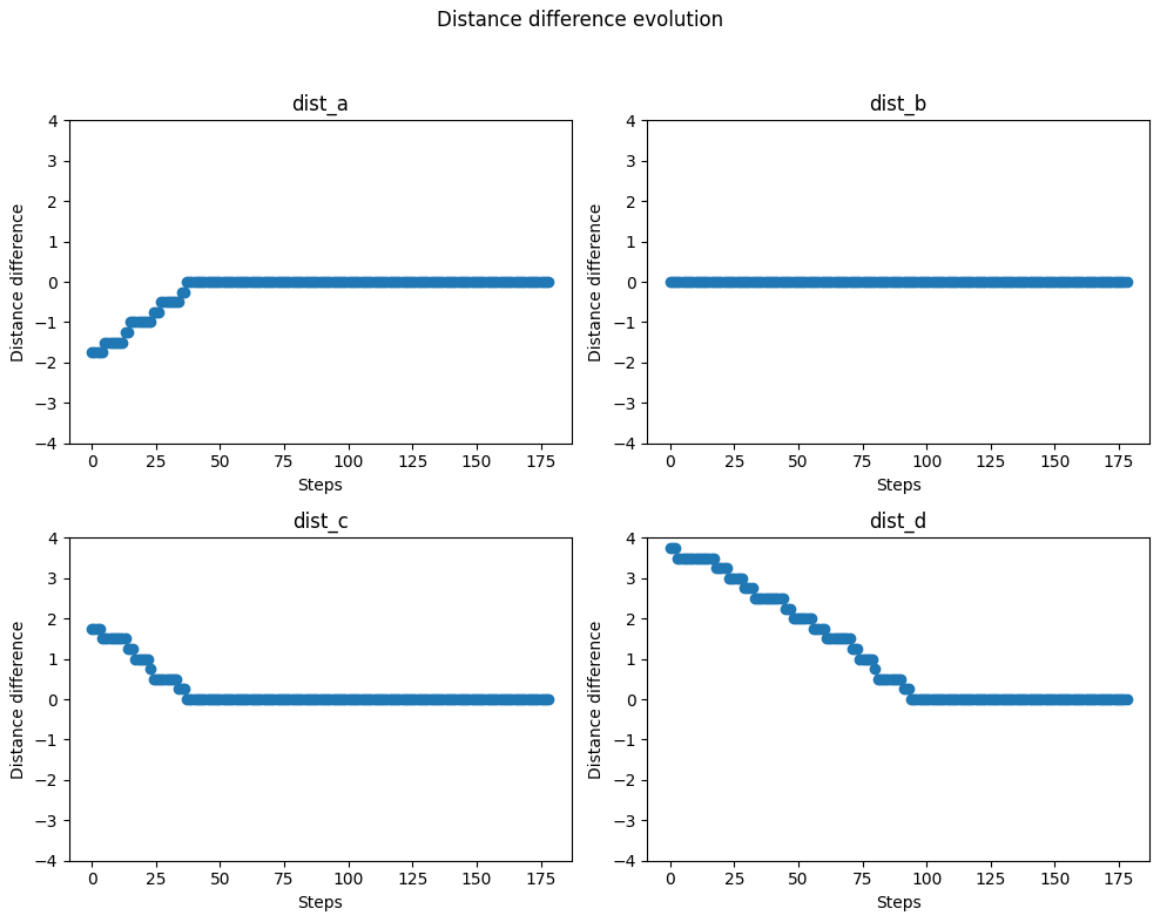


Figura 6.3: Test de eficiencia en aparcamiento 1D.

La figura 6.3 muestra la evolución del error de posicionamiento para cuatro pruebas independientes con condiciones iniciales aleatorias. En las cuatro gráficas se observa claramente cómo el error de distancia disminuye progresivamente, convergiendo finalmente a un valor nulo en todos los casos. Esta convergencia sistemática demuestra la robustez del sistema, capaz de alcanzar un posicionamiento perfecto independientemente del punto de partida inicial. La repetibilidad del éxito en las cuatro pruebas valida la consistencia del enfoque Q-learning implementado para resolver este tipo de maniobras unidimensionales.

A continuación, se muestra un vídeo que ilustra el comportamiento del agente de aparcamiento 1D, partiendo desde posiciones iniciales aleatorias. Como se explicó previamente, el sistema comienza con una fase de desalineación controlada. Una vez posicionado en la posición de partida, el agente inicia su maniobra aprendida mediante Q-learning.

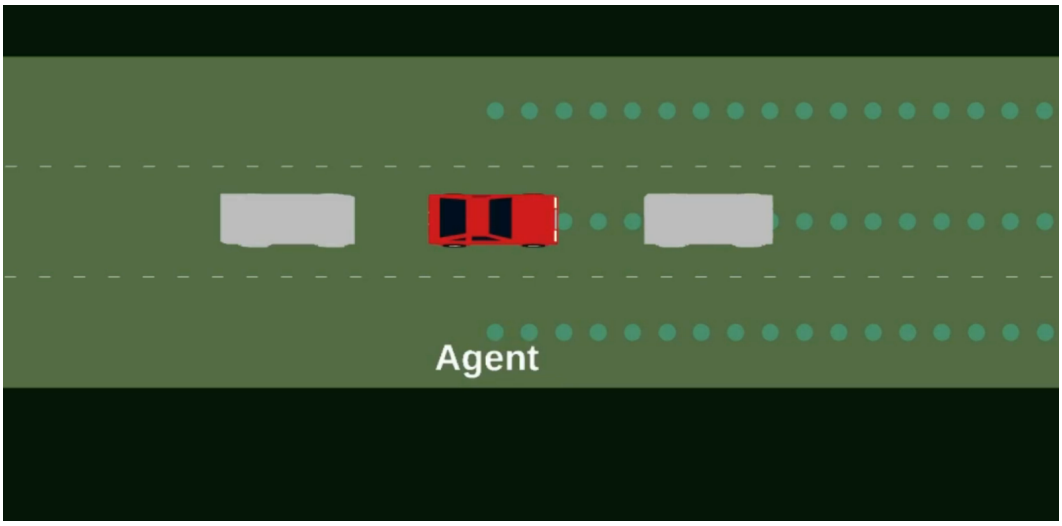


Figura 6.4: https://youtu.be/S0tpI1oVO_I

6.2. Parking 2-D Holonómico

La validación del sistema para esta aplicación requirió un enfoque más sofisticado que en el caso unidimensional, acorde con la mayor complejidad del problema. El proceso de evaluación combinó tres componentes fundamentales: análisis post-entrenamiento mediante gráficas, monitorización en tiempo real durante el aprendizaje, y un nuevo programa de evaluación y pruebas en tiempo de inferencia diseñado específicamente para este escenario. Este último incorporó un mayor grado de aleatoriedad en las condiciones iniciales, evaluando la capacidad del agente para manejar diferentes configuraciones que pusieran a prueba su adaptabilidad.

6.2.1. Métricas de evaluación

El proceso de validación del sistema holonómico incorporó una innovadora herramienta de visualización en tiempo real que permitió monitorizar de forma continua el progreso del entrenamiento. A diferencia del enfoque 1D, donde el análisis se realizaba post-entrenamiento, esta implementación mostraba dinámicamente la evolución del aprendizaje de nuestro agente mediante gráficos que se actualizaban tras cada episodio. Se visualizaba de esta forma múltiples aspectos del aprendizaje: la curva de recompensas acumuladas, la distancia al objetivo, el parámetro ϵ y las pérdidas del modelo.

Este sistema de monitorización en tiempo real resultó particularmente valioso

para identificar problemas tempranos en el entrenamiento, como la aparición de oscilaciones en la política o estancamientos en mínimos locales. Los gráficos mostraron claramente cómo el agente superaba estos desafíos a medida que el aprendizaje progresaba.

La visualización también permitió relacionar eventos con anomalías en el entrenamiento, facilitando el diagnóstico y la mejora del sistema.

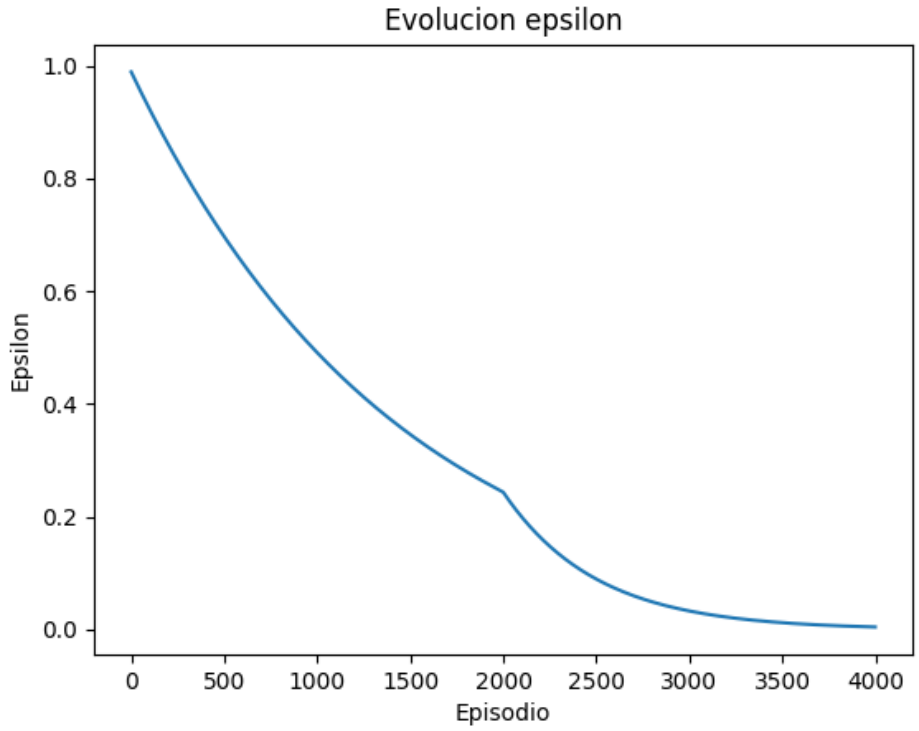


Figura 6.5: Evolución de ϵ por etapas en aparcamiento holonómico.

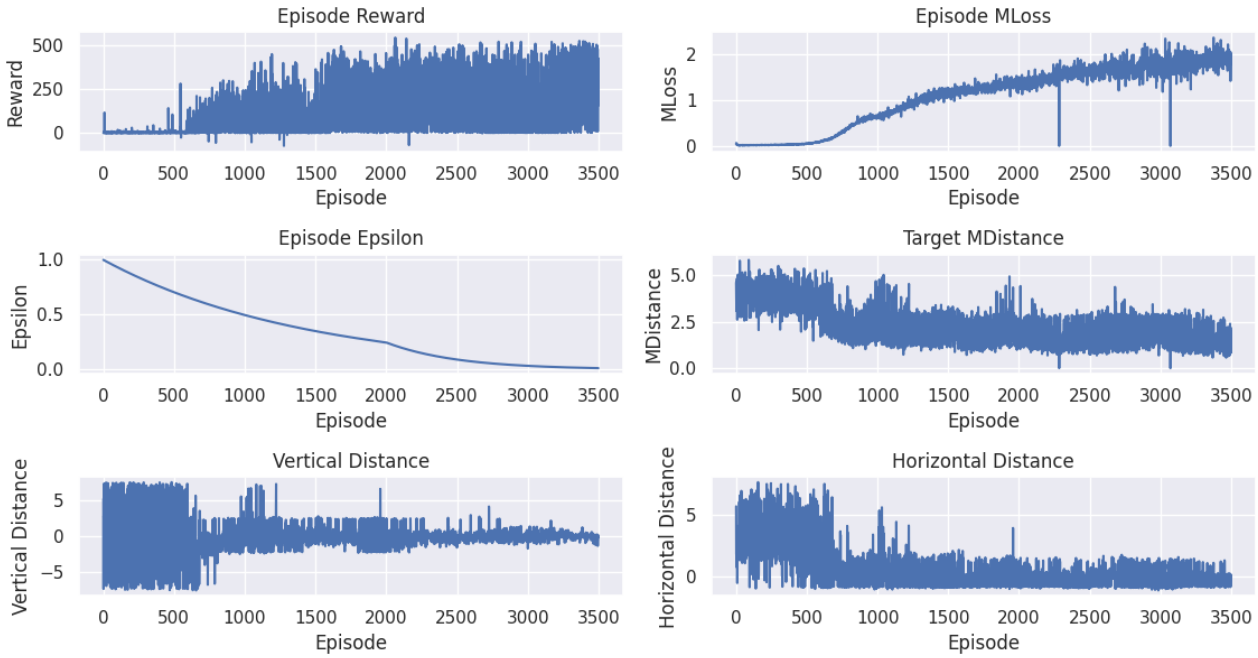


Figura 6.6: Evolución de entrenamiento en aparcamiento holonómico.

La figura 6.6 presenta un conjunto de gráficas que muestran el proceso de aprendizaje del agente. La curva de ϵ muestra el comportamiento esperado, comenzando con una alta exploración inicial que gradualmente decae para dar prioridad a la explotación de la política aprendida, manteniendo siempre un mínimo de exploración aleatoria.

La evolución de la función de pérdida revela un patrón característico del aprendizaje profundo: un incremento inicial mientras el agente explora el espacio de estados, seguido de una estabilización progresiva conforme se consolida el conocimiento adquirido. Este comportamiento confirma la efectividad del mecanismo de actualización de la red neuronal.

Las gráficas de recompensa muestran una mejora consistente a lo largo del entrenamiento. Este progreso se correlaciona directamente con la reducción observable en las tres métricas de distancia. La convergencia simultánea de estas tres medidas de distancia confirma que el agente ha aprendido a coordinar movimientos complejos, combinando desplazamientos y rotaciones para alcanzar la posición óptima. Los resultados validan especialmente la capacidad del sistema para manejar tanto el posicionamiento como la orientación final del vehículo, resolviendo así completamente el problema bidimensional de estacionamiento.

6.2.2. Resultados

Para evaluar el rendimiento real del sistema, se diseñó un programa de validación que consiste en 100 ejecuciones independientes con condiciones iniciales aleatorias. Este conjunto de pruebas contempló diversas combinaciones de posición y orientación inicial, simulando situaciones realistas de estacionamiento. Los resultados demostraron la robustez del sistema, con una tasa de éxito comprendida entre el 80 % y 90 %, donde se consideró éxito la capacidad de estacionarse completamente dentro del espacio designado con la orientación correcta.

El análisis de los casos fallidos reveló que estos se concentraban principalmente en configuraciones iniciales particularmente desafiantes, donde la posición de partida requería maniobras complejas de reorientación.

Finalmente, se incluye un vídeo demostrativo que muestra una ejecución exitosa. Nuevamente, se observará una primera fase de preparación de la posición inicial , como se explicó previamente en el capítulo 5, seguida de una fase donde ya el agente realiza las maniobras aprendidas.

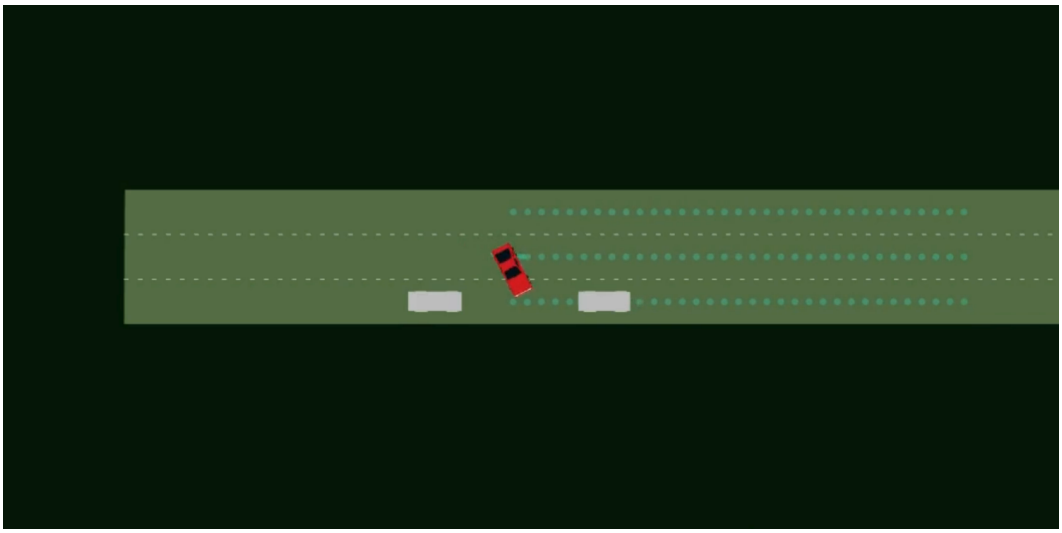


Figura 6.7: https://youtu.be/q5a7agrdaak

6.3. Parking 2-D Ackermann

El análisis del sistema de aparcamiento con dinámica Ackermann presenta desafíos debido a sus restricciones de movimiento, lo que requirió adaptar y mejorar nuestra metodología de evaluación a la vez que se mantenía gran parte de

los métodos del aparcamiento holonómico. La evaluación se centró en capturar cómo el agente aprende mediante maniobras secuenciales inteligentes.

6.3.1. Métricas de evaluación en los entrenamientos

Para comprender el comportamiento del sistema, implementamos un esquema de monitorización que amplía el utilizado en la configuración holonómica. Se incorporó un análisis detallado del sistema de recompensas mediante gráficas que revelan la relación entre los distintos estados y sus respectivas recompensas. Estas visualizaciones surgieron como herramienta clave para diagnosticar y resolver un problema fundamental durante el entrenamiento: el agente frecuentemente quedaba atrapado en mínimos locales, quedándose atrapado en maniobras repetitivas que no progresaban hacia el estacionamiento completo. La gráfica de recompensas detallada se diseñó específicamente para identificar estos estancamientos y modificar el sistema en consecuencia. Esta adaptación fue crucial para superar los problemas de rendimiento donde el vehículo mostraba comportamientos oscilatorios o se conformaba con soluciones subóptimas, particularmente en las fases críticas de reorientación donde la dinámica Ackermann impone sus mayores restricciones.

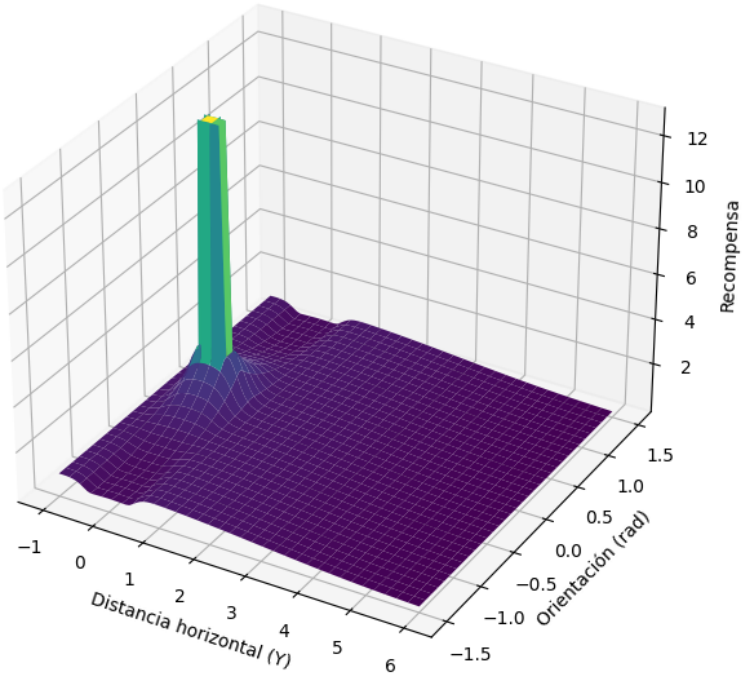


Figura 6.8: Recompensas según orientación y aproximación horizontal.

El sistema de aparcamiento con dinámica Ackermann representó el desafío más complejo de este TFG, debido a sus fuertes restricciones de movimiento. Durante el proceso de entrenamiento, se implementó una estrategia de guardado progresivo de la red neuronal, ya que se observó que el aprendizaje no seguía una mejora continua. Curiosamente, la política más efectiva se obtuvo alrededor de los 2000 episodios, donde el agente había desarrollado maniobras razonablemente eficientes. Sin embargo, pasado este punto, el entrenamiento mostró un comportamiento paradójico: aunque las métricas indicaban que el agente seguía "aprendiendo", esto se traducía en maniobras más erráticas y menos consistentes, sin llegar a alcanzar una verdadera convergencia y sin completar la maniobra de estacionamiento.

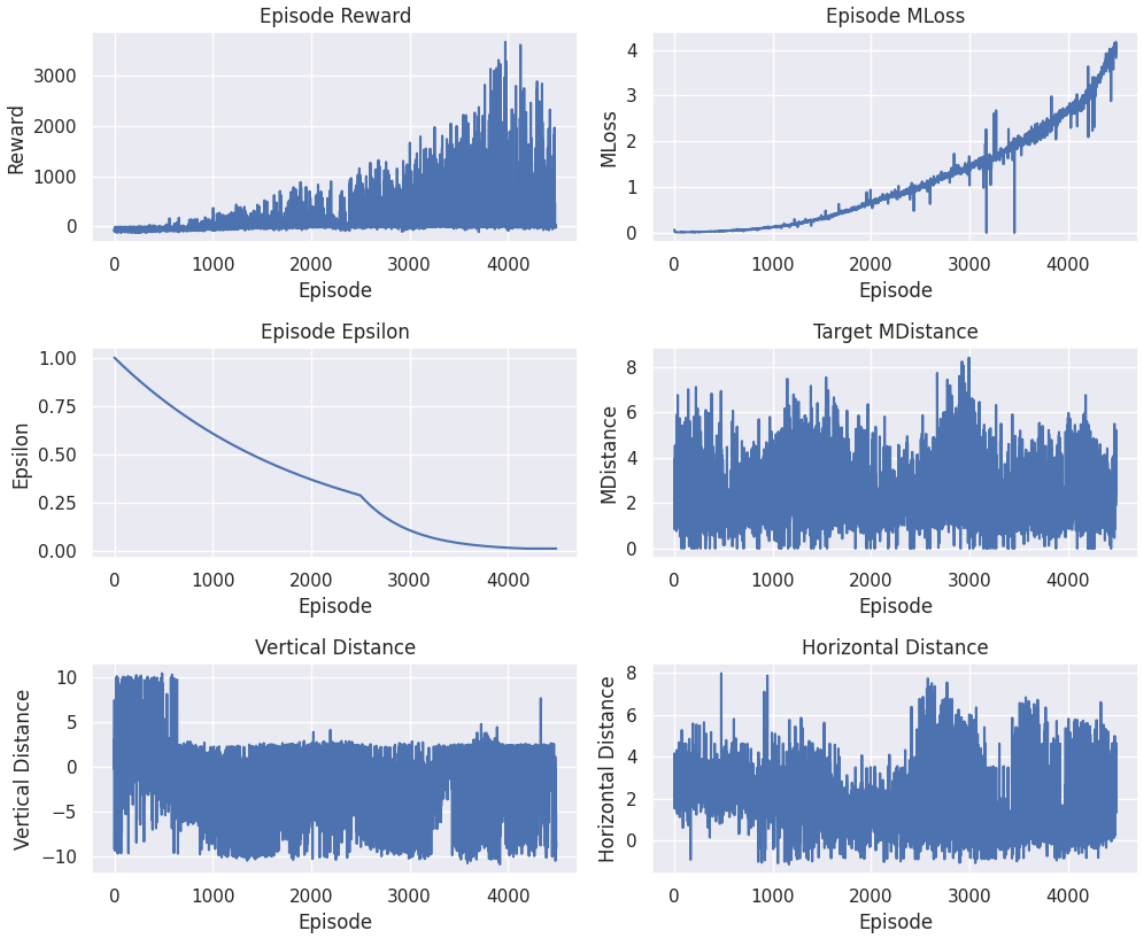


Figura 6.9: Evolución de entrenamiento en aparcamiento Ackermann.

En la figura 6.9 podemos observar cómo el agente comenzó a sobreajustar para situaciones específicas, comprometiendo su habilidad para generalizar, un desafío conocido en el aprendizaje por refuerzo profundo cuando se enfrenta a problemas particularmente complejos.

6.3.2. Resultados finales en inferencia

El sistema de evaluación para la configuración Ackermann mantuvo la misma estructura del caso holonómico, adaptándolo a esta dinámica más restrictiva. Las pruebas realizadas con 100 inicializaciones aleatorias mostraron una tasa de éxito entre el 80 % y 85 %, resultado significativo dado el mayor desafío que supone la naturaleza no holonómica del vehículo.

En las primeras etapas de desarrollo surgió un patrón problemático donde el vehículo persistía en aproximarse frontalmente al espacio de aparcamiento. Este comportamiento, capturado en un vídeo demostrativo, revelaba cómo el agente quedaba frecuentemente atrapado en configuraciones angulares incorrectas, incapaz de corregir su orientación final mediante las limitadas maniobras disponibles.

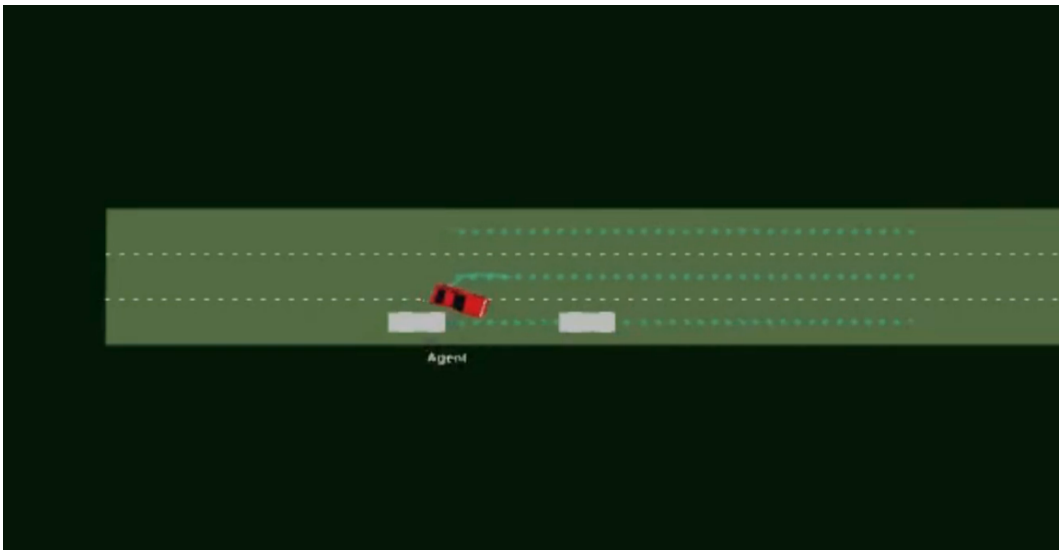


Figura 6.10: https://youtu.be/9BpQTvAPXbA

La solución se alcanzó al modificar el sistema de recompensas, incorporando incentivos específicos para aproximaciones en marcha atrás. Este cambio se hace evidente al comparar con el video posterior, donde ahora se observa una estrategia más sofisticada que combina fases de reorientación con aproximaciones controladas. El vehículo muestra capacidad para calcular secuencias eficientes de movimientos que compensan sus limitaciones dinámicas.

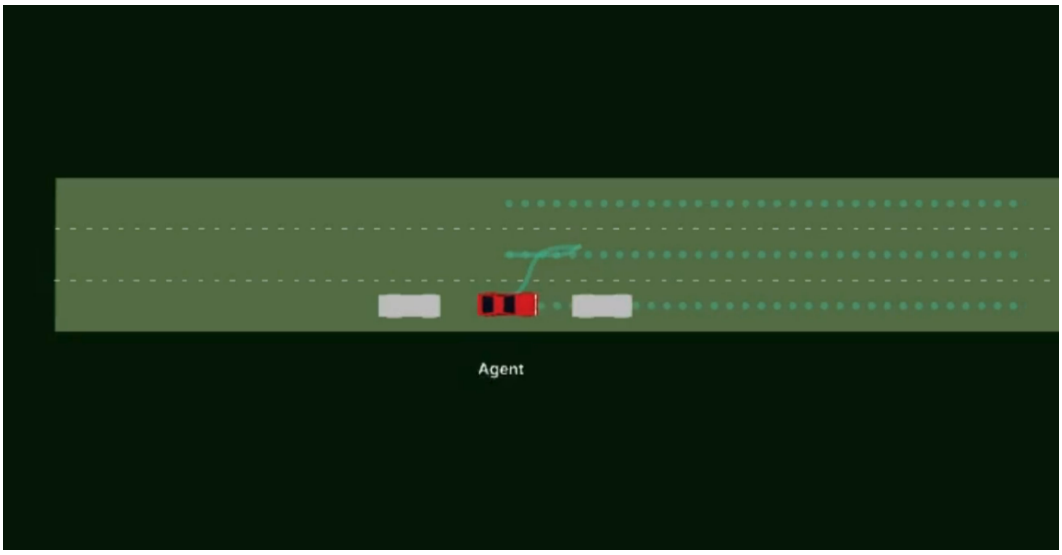


Figura 6.11: https://youtu.be/frut9MC-MMM

La evolución desde el comportamiento inicial problemático hasta la solución final queda perfectamente ilustrada en la comparativa de videos, mostrando cómo un diseño cuidadoso del esquema de recompensas puede superar limitaciones estructurales aparentemente complicadas.

7. Conclusiones

7.1. Conclusiones finales

En este TFG, se han implementado diversas técnicas de aprendizaje por refuerzo para el desarrollo de sistemas de conducción autónoma en el simulador SMARTS, abordando una primera aplicación simple de aparcamiento autónomo en 1-D, otra en 2-D holonómica y la más completa en 2-D utilizando el modelo de coche con dinámica Ackermann. Aparte de abordar con éxito los casos descritos anteriormente, se han realizado diversas modificaciones a SMARTS, corrigiendo algunos errores y diseñando escenarios personalizados.

En cuanto a resultados, para todos los casos se han obtenido métricas de evaluación satisfactorias, implementando el algoritmo de Q-Learning clásico para el caso del aparcamiento autónomo en 1-D, consiguiendo que el agente aparque correctamente el 100 % de las veces; y aplicando el aprendizaje mediante Double DQN para los casos en 2-D, que mostró una mejor adaptación que DQN, obteniendo una tasa de éxito de entre el 80 % y el 90 % para el aparcamiento holonómico y de entre el 80 % y el 85 % si se utiliza el modelo Ackermann, en conjuntos de 100 pruebas diferentes.

Estos resultados demuestran la capacidad de los modelos de aprendizaje por refuerzo en realizar tareas complejas, sin necesidad de abordarlas de manera tradicional, pero con la dificultad de encontrar el equilibrio adecuado entre todos los parámetros para su correcto funcionamiento, incluyendo tanto las observaciones sensoriales, los estados y las funciones de recompensa como hiperparámetros del entrenamiento por refuerzo.

7.2. Competencias empleadas

Este trabajo ha demandado la integración de conocimientos adquiridos a lo largo de la formación en el grado de ingeniería robótica software. El dominio avanzado de Python y sus ecosistemas científicos (PyTorch, NumPy, PyBullet) fue fundamental para implementar tanto los algoritmos de control como las herramientas de análisis y visualización.

Los conceptos de robótica móvil resultaron cruciales para comprender y modelar las diferencias entre dinámicas, mientras que los fundamentos de aprendizaje por refuerzo permitieron diseñar sistemas de recompensa efectivos y arquitecturas neuronales adecuadas. La experiencia en técnicas de depuración y optimización fue particularmente valiosa para resolver los problemas imprevistos del simulador.

7.3. Competencias adquiridas

A lo largo del desarrollo de este trabajo se consolidaron y expandieron significativamente diversas competencias. La implementación práctica del aprendizaje por refuerzo profundo permitió adquirir un conocimiento aplicado sobre el diseño de funciones de recompensa, la selección de hiperparámetros y el diagnóstico de problemas en el entrenamiento de agentes inteligentes. La necesidad de adaptar el simulador SMARTS fomentó el desarrollo de habilidades avanzadas en ingeniería de software, particularmente en la modificación de sistemas complejos existentes. El proyecto también permitió profundizar en técnicas de experimentación en robótica, desde el diseño de protocolos de validación hasta el análisis de resultados.

Las habilidades obtenidas van más allá del ámbito particular del proyecto, constituyendo un valioso bagaje para futuros desarrollos en robótica autónoma e inteligencia artificial.

7.4. Trabajos futuros

Una principal línea de mejora del trabajo desarrollado es optimizar el algoritmo actual para aumentar su robustez, centrándonos en algunas mejoras clave: refinar el sistema de recompensas e implementar un mecanismo de lanzamiento paralelo en servidores remotos que permita ejecutar múltiples instancias de entrenamiento simultáneamente.

Estas mejoras buscan resolver los principales limitantes actuales: el tiempo excesivo de entrenamiento y la sensibilidad al estancamiento en mínimos locales. Estas optimizaciones podrían elevar la tasa de éxito por encima del 95 % en todos los escenarios.

En conclusión, la principal línea futura se orienta a transformar el actual

prototipo funcional en un sistema robusto y eficiente, listo para implementación real. La combinación de recompensas inteligentes y entrenamiento paralelizado representa el camino más prometedor para superar las limitaciones actuales y alcanzar un rendimiento óptimo en todas las condiciones.

Bibliografía

[1] Freepik. Imagen de ojo robótico publicada, n.d. URL https://smart-lighting.es/wp-content/uploads/2022/03/0-freepik-ojo-robot-tech-retina.jpg.

[2] Advantecnia. Imagen del robot da vinci, n.d. URL https://advantecnia.com/wp-content/uploads/2024/02/da-vinci1-1024x682.jpg.

[3] Clarín. Robots brain, n.d. URL https://www.clarin.com/img/2019/06/19/ZSspnOjcu_1256x620__2.jpg.

[4] Blogs.es. Robots autónomos, n.d. URL https://i.blogs.es/57f295/robots_sin_carga/1366_2000.png.

[5] Applesfera. Vehículos autónomos y percepción del entorno, n.d. URL https://i.blogs.es/7982fb/zenseact-ap/650_1200.jpeg.

[6] Europeos Viajeros. Imagen sobre los niveles de automatización en coches, 2020. URL https://www.europeosviajeros.com/wp-content/uploads/2020/04/niveles-automatizacion-coches-1024x519.jpg.

[7] Erum Vial. Reconocimiento autónomo de aparcamiento, n.d. URL https://erumvial.com/wp-content/uploads/2023/03/Imagenes-blog-Honimunn-2023-03-22T175655.401.png.

[8] MathWorks. Reinforcement learning, n.d. URL https://la.mathworks.com/discovery/reinforcement-learning/_jcr_content/mainParsys3/discoverysubsection_603098216/mainParsys3/image_1570014212.adapt.full.medium.png/1701680709240.png.

[9] ResearchGate. Imagen ddqn technical principle framework, n.d. URL https://www.researchgate.net/publication/335182599/figure/fig2/AS:1143128143127382 6@1746789631448/DDQN-technical-principle-framework.tif.

[10] NVIDIA. Pytorch for deep learning, n.d. URL https://www.nvidia.com/content/dam/en-zz/Solutions/glossary/data-science/pytorch/img-1.png.

[11] Wordpress.com. Imagen de esquema de giro sobre dinámica ackermann, n.d. URL https://decarreteres.wordpress.com/wp-content/uploads/2015/12/244bbcw.jpg.

[12] Navigation 2 Team. Navigation 2 documentation, n.d. URL https://docs.nav2.org/.

[13] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8: 279–292, 1992.

[14] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[15] Ming Zhou, Jun Luo, Julian Villella, Yaodong Yang, David Rusu, Jiayu Miao, Weinan Zhang, Montgomery Alban, Iman Fadakar, and Zheng Chen. Smarts: Scalable multi-agent reinforcement learning training school for autonomous driving. *arXiv preprint arXiv:2010.09776*, 2020.

[16] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Conference on robot learning*, pages 1–16. PMLR, 2017.

[17] Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. Sumo–simulation of urban mobility: an overview. In *Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation*. ThinkMind, 2011.

[18] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[19] Nikhil Ketkar, Jojo Moolayil, Nikhil Ketkar, and Jojo Moolayil. Introduction to pytorch. *Deep learning with python: learn best practices of deep learning models with PyTorch*, pages 27–91, 2021.

[20] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. Dbscan revisited, revisited: why and how you should (still) use dbscan. *ACM Transactions on Database Systems (TODS)*, 42(3):1–21, 2017.

Anexos

A1

Blog y archivos de interés:

- https://github.com/RoboticsLabURJC/2024-tfg-david-duro

Instalación personalizada de SMARTS:

- https://github.com/dduro2020/SMARTS