Universidad Rey Juan Carlos

ESCUELA DE MÁSTERES OFICIALES

MÁSTER UNIVERSITARIO EN VISIÓN ARTIFICIAL

CURSO ACADÉMICO 2023/2024

TRABAJO FIN DE MÁSTER

CONDUCCIÓN AUTÓNOMA VISUAL BASADA EN APRENDIZAJE POR REFUERZO

Autor: Francisco C. Vázquez Donaire

Directores: José María Cañas Plaza

Roberto Calvo Palomino

Agradecimientos

A mis tutores, José María y Roberto, por su paciencia y comprensión, y por acompañarme durante todo el desarrollo del trabajo.
A mis compañeros del CAPO, por su apoyo en el día a día.
A Alejandra, por compartir conmigo sus conocimientos matemáticos que han servido para aportar mayor rigor a esta memoria.
A Blanca, por acompañarme desde que tenemos 3 años.
A mi familia y amigos, por estar siempre que los he necesitado.

Resumen

La conducción autónoma de vehículos ha ganado atención en los últimos años debido al progreso de las tecnologías de Inteligencia Artificial (IA), y también al gran interés de la sociedad por automatizar la conducción. Sin embargo, la conducción autónoma presenta un escenario complejo, poco amigable y dinámico desde el punto de vista del conductor. Se pueden utilizar diferentes técnicas para crear un comportamiento de conducción inteligente que permita controlar el vehículo de forma autónoma. En este Trabajo Fin de Máster, nos centramos en los sistemas de conducción autónoma que integran técnicas de visión por computador y Aprendizaje por Refuerzo para proporcionar las aplicaciones de seguir la línea y seguir el carril. Proponemos utilizar técnicas de visión por computador basadas en Inteligencia Artificial para el módulo de percepción, que se encarga de la detección robusta del carril de la carretera, y utilizar el Aprendizaje por Refuerzo *Q-learning* para el módulo de control. Nuestras soluciones han sido validadas experimentalmente bajo diferentes condiciones en el simulador *CARLA*, una herramienta ampliamente utilizada en la comunidad. Este trabajo contribuye al campo de la conducción autónoma al aprovechar las técnicas de visión artificial basadas en IA y de Aprendizaje por Refuerzo para mejorar las capacidades de la conducción autónoma.

Abstract

Autonomous vehicle driving has gained attention in the recent years due to the progress of artificial intelligence technologies, and also to the great interest of the society in automating driving. However, autonomous driving presents a complex, unfriendly and dynamic scenario from the driver point of view. Different techniques can be used to create a smart driving behaviour to autonomously control the vehicle. In this master's thesis, we focus on autonomous driving systems which integrate computer vision and Reinforcement Learning techniques to provide the follow-line and the follow-lane applications. We propose to use AI-based computer vision techniques for the perception module, which is in charge of the robust detection of the lane of the road, and to use the *Q-learning* Reinforcement Learning for the control module. Our solutions have been experimentally validated under different conditions in the *CARLA* simulator, a widely used tool in the community. This work contributes to the field of autonomous driving by leveraging AI-based computer vision and Reinforcement Learning techniques to improve the capabilities of self-driving. The results obtained pave the way for safer and more efficient autonomous vehicles.

Índice

Página

Índice de figuras

Índice de tablas

1. Introducción

En este TFM (TFM) se pretende conseguir que un agente de Aprendizaje por Refuerzo resuelva varias aplicaciones de conducción autónoma únicamente utilizando información visual; por este motivo, este trabajo se sitúa en la intersección de tres dominios de creciente interés: la robótica, la visión artificial y el *machine learning*.

En esta sección se presenta el tema principal del proyecto: el Aprendizaje por Refuerzo aplicado al control visual de un vehículo. También se explora la motivación detrás de la búsqueda de una solución para este problema, al mismo tiempo que se establecen los objetivos específicos a tratar.

1.1. Motivación y contexto

En los últimos años, el interés en la robótica ha experimentado un crecimiento explosivo. Gracias a avances en hardware y software, la robótica se ha convertido en una parte integral de la vida moderna, desde vehículos autónomos hasta asistentes domésticos inteligentes. Su impacto se extiende a campos como la medicina, la agricultura y la exploración espacial, y su creciente presencia promete continuar transformando la sociedad y la tecnología en el futuro cercano. Por ejemplo, en la Figura 1 se puede observar un dron que está siendo utilizado para realizar un mantenimiento predictivo de las redes eléctricas. Este tipo de trabajo era previamente realizado con un operario a bordo de un helicóptero, lo que conllevaba un mayor riesgo y un mayor gasto. Por otro lado, en la Figura 2 se puede observar otra aplicación de la robótica. En ella se utilizan robots para aparcar los coches de los clientes en un parking al aire libre.

Por su parte, la visión artificial también ha experimentado un aumento constante en su interés y relevancia en los últimos años. Este campo se ha beneficiado de avances tecnológicos en procesamiento de imágenes y cámaras más avanzadas, lo que ha permitido una amplia gama de aplicaciones en diversas industrias. Desde sistemas de reconocimiento de objetos en la industria manufacturera hasta soluciones de diagnóstico por imagen en la atención médica, la visión artificial se ha convertido en una herramienta fundamental para la automatización y mejora de procesos en una variedad de sectores. Su adopción

Figura 1: Aplicación de la robótica en el ámbito de la inspección industrial basada en un dron que realiza mantenimiento predictivo de la red eléctrica en España.

Figura 2: Aplicación de la robótica en el ámbito del automovilismo en la cual un robot es capaz de aparcar un coche de forma autónoma.

continua promete optimizar aún más la eficiencia y la precisión en numerosas aplicaciones. Uno de los ámbitos en los cuales la visión por computador está ganando relevancia es en el ámbito deportivo, siendo ésta utilizada, entre otros, para medir el recorrido total de un jugador a lo largo de un partido o calcular la probabilidad de que una cierta jugada termine en gol (Figura 3).

Figura 3: Ejemplo de una aplicación de visión artificial en el fútbol, en este caso, calculando la probabilidad de que la jugada termine en gol.

Gracias a avances en algoritmos y acceso a grandes cantidades de datos, el *machine learning* ha transformado una amplia variedad de campos. Desde la recomendación de contenido en plataformas de *streaming* hasta la detección de fraudes en transacciones financieras, el *machine learning* se ha convertido en una herramienta esencial para automatizar tareas, tomar decisiones basadas en datos y desarrollar soluciones personalizadas. Su continua evolución promete seguir impulsando la innovación en una variedad de aplicaciones en constante expansión. Una de las técnicas más importantes dentro de esta disciplina es la conocida como *Reinforcement Learning* (RL), la cual no necesita de conjuntos de datos etiquetados, sino que basa su aprendizaje en interacciones con el entorno. Por ejemplo, estas técnicas se han utilizado para lograr que un agente multiarticular aprenda a caminar en distintos escenarios (Figura 4, izquierda [1]), o para que un vehículo aprenda a aparcar (Figura 4, derecha [2]).

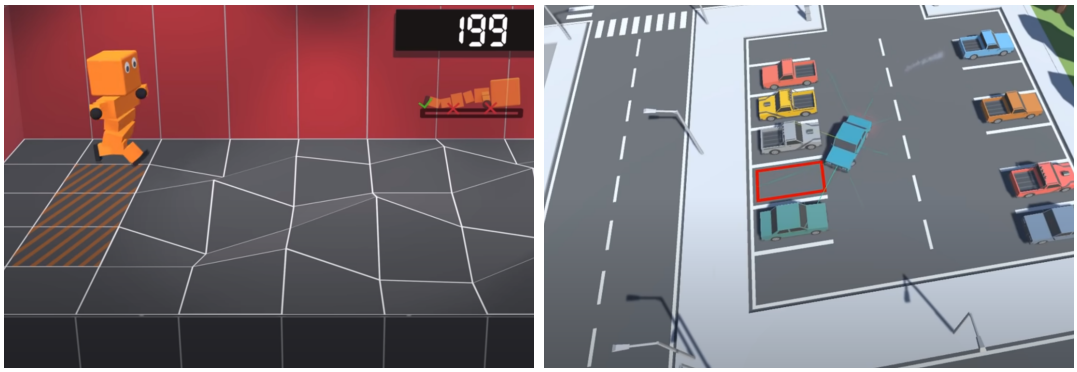

Figura 4: A la izquierda se observa un agente multiarticular aprendiendo a caminar mediante RL. A la derecha, un vehículo aprendiendo a aparcar, también con RL.

Conducción autónoma

Según las últimas publicaciones de la Administración Nacional de Seguridad del Tráfico en las Carreteras (NHTSA), el 94 % de los accidentes de coche son causados por los humanos. Como solución a esta problemática, y aprovechando los grandes avances en el conocimiento de la dinámica del coche, la visión por computador y el auge del *Deep Learning*, los Sistemas de Conducción Autónoma o ADS por sus siglas en inglés (*Autonomous Driving Systems*) han ganado protagonismo en los últimos años. Estos sistemas se crean con el propósito de reducir el número de accidentes así como las emisiones de CO2 y el estrés que conlleva la conducción por sitios muy frecuentados a diario [3].

Estos sistemas necesitan de una etapa de percepción robusta y precisa a la hora de identificar los elementos del entorno y de actuar en consecuencia. Además, el problema se complica si tenemos en cuenta que en ocasiones el entorno puede ser incierto, ya que depende de las decisiones que toman los demás usuarios de la vía. Un ejemplo de una situación de este tipo es el cruce entre dos vehículos en una carretera estrecha. Por este motivo los ADS deben ser capaces de interpretar las señales de otros conductores y de ser interpretados por los mismos [4].

En la actualidad, las empresas que lideran el desarrollo de estos ADS son en primer lugar *Tesla*, con su *AutoPilot*, *Mercedes* y *Waymo*, esta última siendo una empresa que forma

parte de Alphabet (Google). Estos sistemas se clasifican según su nivel de automatización siguiendo el estándar [5] de la siguiente manera:

- Nivel 0: Sin automatización.
  No existe ningún tipo de ayuda a la conducción.

- Nivel 1: Asistencia en la conducción.
  El vehículo cuenta con sistemas de ayuda a la conducción, como mantenimiento en el carril o controles de velocidad adaptativos.

- Nivel 2: Automatización parcial.
  El vehículo cuenta con control de movimiento longitudinal y lateral pero no tiene la capacidad de detectar objetos

- Nivel 3: Automatización condicionada.
  Aunque en este nivel el vehículo sea capaz de decidir cuándo cambiar de carril, frenar para evitar colisiones o ante un semáforo en rojo, el conductor debe estar preparado para intervenir en cualquier momento ya que el factor humano seguirá teniendo un papel importante. Actualmente, *Tesla* se encuentra en este nivel.

- Nivel 4: Automatización elevada.
  Dentro de algunos casos de uso, el vehículo tomará todas las decisiones, desde la ruta a seguir para llegar a un destino hasta las decisiones de circulación pequeñas que deben tomarse cada pocos segundos. No se necesitará ningún tipo de intervención por parte del conductor dentro de estos casos de uso. Por su parte, los *robotaxis* de *Waymo* y *General Motors* se encuentran en este nivel.

- Nivel 5: Automatización completa.
  El vehículo estará habilitado para responder a solicitudes realizadas a través de la interfaz, permitiéndole desplazarse a cualquier destino sin requerir volante, pedales ni controles convencionales. En este nivel, se prescinde de la figura del conductor humano; se entrará en el automóvil, se establecerá el destino y el vehículo comenzará a moverse automáticamente.

Por el momento, no se ha conseguido desarrollar un ADS completamente autónomo, aunque sí se han conseguido numerosos hitos como la capacidad del coche de mantenerse en

el carril, frenar si prevé que puede colisionar con otro vehículo o la capacidad de aparcar y des-aparcar por sí solo. Además, se debe destacar que este tipo de sistemas se enfrentan también a diversas incertidumbres a nivel legal y de aceptabilidad por parte de los usuarios, debiendo ser aceptadas por las respectivas agencias gubernamentales de cada país. Incluso si estas empresas han desarrollado con éxito sistemas completamente autónomos y sus coches están conduciendo de forma autónoma en escenarios reales (Waymo y Cruise ofrecen servicio de robotaxi en San Franscisco), todavía hay trabajo por hacer. Tesla está siendo investigada por la Administración Nacional de Seguridad del Tráfico en las Carreteras (NHTSA) debido a que el piloto automático de Tesla estuvo involucrado en casi 750 accidentes desde 2019, incluyendo 17 víctimas mortales[1]. Los robotaxis Waymo y Cruise han sido los principales responsables de accidentes leves y saltos de tráfico en San Francisco por un mal funcionamiento del software[2]. No obstante, el desarrollo y las soluciones de conducción autónoma de estas empresas arrojan luz sobre el futuro de la conducción autónoma.

Figura 5: Robotaxi de Waymo circulando de forma autónoma.

Uno de los principales motivos por los cuales la conducción autónoma ha ganado mayor robustez y mejores prestaciones ha sido el auge de las técnicas de tipo *Deep Learning* que

[1]https://www.caranddriver.com/news/a44185487/report-tesla-autopilot-crashes-since-2019/

[2]https://www.sfchronicle.com/bayarea/article/sffd-says-two-robotaxis-blocked-ambulance-18343808.php

se han ido introduciendo en cada uno de los distintos módulos de este ámbito; como es la *toma de decisiones* y, en especial, en la *percepción del entorno.*

En este sentido, las *Convolutional Neural Networks*, de ahora en adelante, CNN, se utilizan a la hora de detectar peatones, obstáculos, señales de tráfico, ofreciendo muy buenos resultados. Por ejemplo, en [6] mezclan técnicas de visión por computador clásicas con el uso de las CNN para detectar objetos desde un vehículo, obteniendo una tasa de detección del 86.21 % y una tasa de falsos positivos del 0.21 %.

Como se ha comentado anteriormente, también existen otro tipo de técnicas de Aprendizaje Automático que basan su aprendizaje en la interacción con el entorno. Estas técnicas, conocidas como técnicas de Aprendizaje por Refuerzo, se utilizan en multitud de problemas, desde el ámbito de los videojuegos clásicos como los de la marca *Atari*, hasta en juegos de habilidad mental como el ajedrez o el póker.

Otro aspecto reseñable en el desarrollo de los sistemas de conducción autónoma es la comparativa entre los sistemas modulares y los sistemas extremo a extremo, siendo los **modulares** sistemas en los cuales se desarrolla, para cada subtarea, un módulo específico de *software*, utilizando distintos algoritmos en cada uno de ellos. En cambio, en los sistemas **extremo a extremo**, típicamente, un modelo recibe los datos sensoriales de entrada y toma todas las decisiones de control necesarias, que se envían a los actuadores. En este proyecto se ha utilizado un enfoque extremo a extremo, en el cual se recibe una imagen como entrada, y el modelo de A elige una de entre varias acciones para comandarla a los actuadores del vehículo.

En el presente TFM, se utilizarán las CNN para la parte de la percepción del entorno del agente, y el Aprendizaje por Refuerzo para la toma de decisiones, proponiendo así la implementación de un controlador visual para un vehículo autónomo.

1.2. Objetivos

El objetivo general del presente TFM se enfoca en explorar y aplicar diversas técnicas de visión artificial y Aprendizaje por Refuerzo en el contexto de la conducción autónoma

utilizando el simulador *CARLA*. Este objetivo se divide en tres subobjetivos concretos:

1. **Realizar un estudio sobre el Estado del Arte de la robótica con visión y el Aprendizaje por Refuerzo**:
   El segundo subobjetivo de este proyecto es realizar un investigación previa sobre aquellos trabajos que utilizan el Aprendizaje por Refuerzo para abordar problemas típicos en la robótica, en concreto, en la robótica con visión. Más específicamente, se realizará un estudio sobre los algoritmos de Aprendizaje por Refuerzo que se han aplicado al problema de la conducción autónoma.

2. **Programar un prototipo de un controlador visual para la aplicación de seguir una *línea* utilizando técnicas de RL**:
   El tercer subobjetivo se centra en abordar el desafío de desarrollar un sistema de conducción autónoma capaz de seguir una línea en carretera utilizando como único sensor la cámara, técnicas clásicas de procesamiento de imagen y Aprendizaje por Refuerzo para la toma de decisiones.

3. **Programar un prototipo de un controlador visual para la aplicación de seguir un *carril* utilizando técnicas de RL**:
   El cuarto y último subobjetivo se enfoca en aplicar técnicas de *Deep Learning*, como son las CNNs, y, una vez más, algoritmos de Aprendizaje por Refuerzo (*Q-learning*) para desarrollar un controlador visual para un vehículo que debe seguir un carril de forma autónoma.

1.3. Metodología

En este trabajo se ha decidido utilizar algunos aspectos de la metodología *Scrum* (Figura 6) debido a su capacidad para adaptarse a cambios, algo esencial en la mayoría de los proyectos de investigación donde los requisitos pueden cambiar a medida que avanza el trabajo. *Scrum* es una metodología ágil de gestión de proyectos que se utiliza comúnmente en el desarrollo de software [7]. Se caracteriza por:

- Equipos interdisciplinarios: *Scrum* se basa en equipos autoorganizados y multifuncionales, compuestos por miembros con diversas habilidades y roles.
- Tres roles principales: *Scrum master*, *product owner* y el equipo de desarrollo.
- *Backlog* del Producto: Lista priorizada de todas las características, tareas y mejoras que se desean en el producto. El *product owner* es responsable de mantenerlo actualizado y priorizado.
- *Sprint*: Un período de tiempo fijo y corto (generalmente de 2 a 4 semanas) durante el cual se trabaja en un conjunto de elementos del *Backlog* del Producto.
- Incremento potencialmente entregable: Al final de cada *Sprint*, el Producto debe estar en un estado que podría entregarse al cliente, aunque no sea necesario hacerlo. Esto garantiza que haya un producto funcional en desarrollo en todo momento.
- Adaptación continua: *Scrum* promueve la mejora continua del proceso mediante la inspección y adaptación regulares. Los equipos analizan su desempeño en las retrospectivas y realizan ajustes para ser más efectivos en futuros *Sprints*.

En este caso, el equipo estaba formado por tres personas, los dos tutores y el autor del trabajo. Los tutores cumplirían el rol de *scrum master* y *product owner*, y el autor, el de equipo de desarrollo. Los *Sprints* tuvieron una duración media de alrededor de 6 semanas, en los cuales se fueron incrementando las funcionalidades implementadas progresivamente.

Por otro lado, se fijaron las reuniones de forma semanal, siendo éstas realizadas a través de la plataforma *Meet* (*Google*). Además, con el objetivo de que la comunicación entre el autor y los tutores fuera ágil, se creó un canal en *Slack* (*Google*), el cual ha sido utilizado

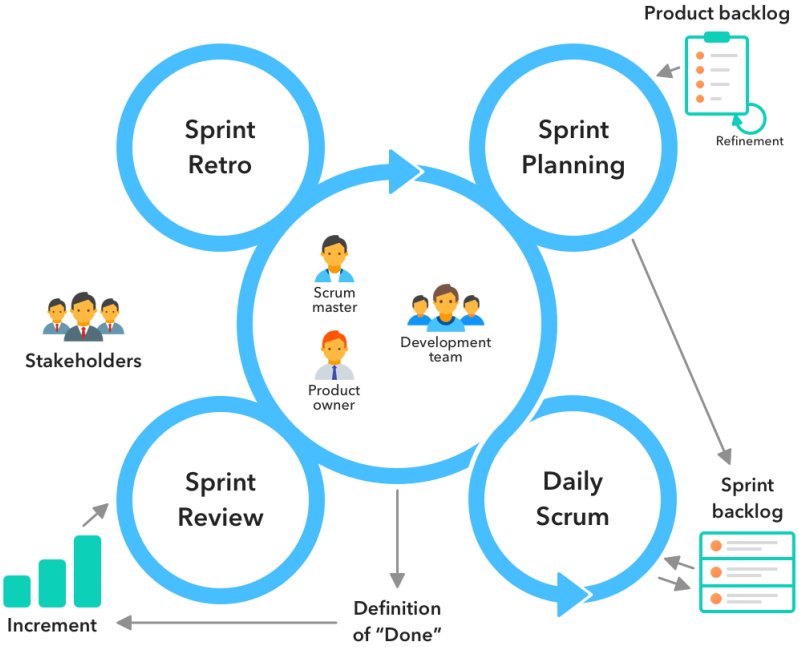

Figura 6: Resumen visual de la metodología *Scrum*.

a lo largo de todo el trabajo.

A su vez, para la realización de este trabajo se creó un *blog*[3] con *GitHub Pages*, el cual se fue actualizando semanalmente con cada uno de los avances del proyecto.

El código que ha sido necesario desarrollar para la realización de este proyecto se encuentra público en la plataforma *GitHub*[4].

[3]https://roboticslaburjc.github.io/2022-tfm-francisco-vazquez/
[4]https://github.com/RoboticsLabURJC/2022-tfm-francisco-vazquez/

2. Estado del Arte

En esta sección se va a realizar una revisión sobre los distintos algoritmos de Aprendizaje por Refuerzo y sobre cómo se pueden utilizar para resolver distintas problemáticas de conducción autónoma.

El Aprendizaje Automático se puede desglosar en varias áreas. Por un lado, existen los métodos supervisados, los cuales se caracterizan por necesitar de un conjunto de datos etiquetados, entre otros, las redes neuronales, las máquinas de vector soporte (SVM) o los árboles de decisión se utilizan en la actualidad tanto para problemas de clasificación como de regresión. En cambio, existen los llamados métodos no supervisados, los cuales no necesitan etiquetas. Dentro de estos métodos se incluye el *clustering*, que se utiliza para agrupar datos en conjuntos o grupos basados en similitudes inherentes entre las muestras. También existen los métodos basados en Aprendizaje por Refuerzo, en los cuales el agente aprende a realizar una tarea mediante interacciones con su entorno. Estos agentes no son programados de manera explícita, sino que son evaluados mediante una *función de recompensa*, que les aportará recompensas mayores si deciden tomar las acciones correctas en cada uno de sus estados.

El objetivo de este agente será el de acumular el mayor número de recompensas posibles, lo cual puede realizar *explotando* su conocimiento obtenido previamente, es decir, eligiendo la acción que le proporciona una mayor recompensa. Por otro lado, para encontrar estas acciones que maximizan la recompensa, debe tomar ciertos riesgos y elegir otras acciones que podrían repercutir en recompensas aún mayores. El agente de Aprendizaje por Refuerzo debe *explotar* lo que ya sabe para obtener recompensas, pero también debe *explorar* lo que desconoce para encontrar posibles recompensas mejores en el futuro. Se debe llegar a un compromiso entre *explotación-exploración* [4].

Dentro del Aprendizaje por Refuerzo existen varias familias de métodos. Por un lado, existen los métodos basados en valores; entre otros, *Q-learning*. Y por otro lado, los métodos basados en políticas, entre los que destacan *Proximal Policy Optimization* (PPO) o *Trust Region Policy Optimization* (TRPO).

A continuación, se explicarán los Procesos de Decisión Finitos de Markov (MDP), para posteriormente comentar los principales métodos de Aprendizaje por Refuerzo basados en valores y basados en políticas. Además, se citarán trabajos que han utilizado este tipo de técnicas para aplicarlos a la robótica.

2.1. Procesos de Decisión Finitos de Markov (MDP)

Los Procesos de Decisión de Markov consisten en un conjunto finito de estados S, un conjunto finito de acciones A, una función de transición T, y una función de recompensa R de tal manera que cuando se parte de un estado $s \in S$ y se toma una acción $a \in A$ se termina en un estado $s' \in S$ con una probabilidad $T(s, a, s') \in (0, 1)$ y se obtiene una recompensa $R(s, a)$. Estos MDP son el marco teórico que sirve para entender la mayoría de algoritmos de Aprendizaje por Refuerzo. El objetivo de estos procesos de decisión de Markov es encontrar la política óptima π^* que resulte en las máximas recompensas [8]:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left\{\sum_{k=0}^{H-1} \gamma^k r_{k+1} \mid s_0 = s\right\}$$

donde $r_k = R(s_k, a_k)$ es la recompensa en el momento k y el parámetro $\gamma \in [0, 1]$ controla qué recompensas toman más importancia, las pasadas o las futuras. Un valor bajo se enfocará en maximizar recompensas en el corto plazo.

Los MDP se pueden diferenciar según la longitud de su horizonte H. Un MDP puede tener horizonte infinito $H = \infty$ o bien horizonte finito, los cuales se denominan *episódicos* y terminan cuando se ha *sobrevivido* a un número de *pasos* o bien cuando se ha alcanzado un estado determinado.

2.2. Métodos de Aprendizaje por Refuerzo basados en valores

Uno de los algoritmos más conocidos en el ámbito del Aprendizaje por Refuerzo es el *Q-learning algorithm* [9]. Este algoritmo hace uso de la información adquirida al dar un *paso*, es decir, tomar una acción, y con esta información actualiza el *Q-value*. En *Q-learning* la estimación del valor Q se obtiene a través del siguiente proceso iterativo [10]:

$$Q_{t+1}(x, a) = (1 - \alpha)Q_t(x_t, a_t) + \alpha\left[r_t + \gamma V(y) - Q_t(x_t, a_t)\right]$$

donde

- r_t es la recompensa obtenida al pasar del estado s_t al estado s_{t+1}.
- γ, el factor de descuento, el cual actúa como un factor de 'memoria' que influye en la ponderación de las recompensas pasadas en comparación con las futuras dentro de una secuencia de acciones.
- $V(y)$ se calculará como la máxima recompensa obtenida al cambiar al nuevo estado.
- α, la tasa de aprendizaje, marca cuánto se actualiza la tabla Q en cada iteración del algoritmo. Se debe tener un compromiso entre valores muy altos y muy bajos, ya que valores cercanos a 1 tenderán a producir el efecto de 'olvidar' lo aprendido previamente, y valores cercanos a 0 no producen ningún aprendizaje.

Otros métodos que extienden al anterior han ido surgiendo en la literatura. En [11] se propone un nuevo algoritmo llamado *Double Q-learning* que obtiene un mejor rendimiento en entornos muy estocásticos, donde el *Q-learning* tiene algunas limitaciones. Estas limitaciones se deben a que en *Q-learning* se utiliza la acción que ha ofrecido mayores recompensas como aproximación a la acción que obtendrá la máxima recompensa. En este trabajo, esta aproximación se realiza utilizando un estimador doble que se aplica sobre *Q-learning* para obtener el *Double Q-learning*, que es capaz de converger a la política óptima en circunstancias donde el *Q-learning* tenía un mal rendimiento. El algoritmo fue probado en juegos de ruleta y laberintos en los cuales se obtuvieron buenos resultados experimentales de una manera más rápida.

Por otro lado, se han creado métodos que mezclan redes neuronales con este tipo de algoritmos clásicos de Aprendizaje por Refuerzo. Muchos de los grandes hitos alcanzados en el *Deep Reinforcement Learning* simplemente se han basado en escalar el trabajo previo a problemas de dimensionalidad mayor. Uno de los algoritmos de este tipo más conocidos es el *Deep Q Network* propuesto por investigadores del laboratorio *Deep Mind* en [12]. Este algoritmo es capaz de aprender políticas óptimas desde señales de entrada en muy alta dimensión. En este trabajo se comprobó que este tipo de *agentes*, recibiendo como entrada únicamente los píxeles y los resultados del juego, eran capaces de superar con creces a todos los algoritmos probados anteriormente, alcanzando el nivel de jugadores de

videojuegos *Atari* profesionales.

Además, este algoritmo ha sido mejorado recientemente en [13], combinando el *DQN* con el *Double Q-learning*. Una vez más, este algoritmo se ha testeado en los *Atari 2600 games*, superando los resultados obtenidos en el trabajo anterior [12] como se muestra en la Figura 7.

	no ops		human starts		
	DQN	DDQN	DQN	DDQN	DDQN (tuned)
Median	93%	**115**%	47%	88%	**117**%
Mean	241%	**330**%	122%	273%	**475**%

Figura 7: Resumen de rendimiento normalizado en 49 juegos durante un período de hasta 5 minutos, con un máximo de 30 acciones sin efecto al inicio de cada episodio, y durante un período de hasta 30 minutos con puntos de inicio humanos seleccionados al azar. Los resultados para DQN provienen de los estudios de Mnih y otros (sin acciones sin efecto) [12] y Nair y colaboradores (2015) (inicio humano) [14].

Uno de los retos predominantes en el campo de la robótica gira en torno a la cuestión de la navegación por entornos desconocidos. Como se evidencia en [15], el Aprendizaje por Refuerzo parece ofrecer una solución sólida y completa a este problema. En este estudio, los investigadores utilizan una imagen RGB-D capturada por una cámara *Kinect* e introducen un novedoso algoritmo llamado *Dueling Double DQN* (D3QN). Los hallazgos presentados en el artículo respaldan de manera sólida la idea de que el aprendizaje autónomo del entorno, sin supervisión externa, es altamente efectivo. El robot puede navegar autónomamente por el entorno al mismo tiempo que evita colisiones con obstáculos.

2.3. Métodos de Aprendizaje por Refuerzo basados en políticas

Los métodos basados en políticas se centran en aprender una política directamente, en lugar de estimar funciones de valor. Una política define la correspondencia entre estados y acciones, indicando la mejor acción a tomar en cada estado. Estos métodos buscan optimizar la política ajustando sus parámetros, a menudo utilizando técnicas de ascenso de gradiente.

PPO, o *Proximal Policy Optimization*, se erige como uno de los algoritmos más populares en este campo, siendo utilizado en diferentes aplicaciones prácticas en diversas áreas, incluido el control de estabilización de cuadricópteros. En un estudio realizado por Cano y colaboradores [16], se utilizó PPO para que el agente adquiriera una política de control efectiva para la estabilización precisa de un dron con estas especificaciones dentro de un entorno simulado. Los resultados demostraron una mejora sustancial en comparación con esfuerzos de investigación previos ([17]), reduciendo notablemente el tiempo de convergencia mientras se mantenía un alto rendimiento.

Otro algoritmo que utiliza el Aprendizaje por Refuerzo basado en políticas combinado con redes neuronales es DDPG (*Deep Deterministic Policy Gradient*). Si bien DDPG ha encontrado aplicaciones en diversos dominios de problemas, es destacable el trabajo realizado por los investigadores de DeepMind en [18], quienes emplearon este algoritmo para la planificación de trayectorias en robots móviles. Los investigadores identificaron debilidades en DDPG, como la baja eficiencia del entrenamiento y la lenta convergencia. Para abordar estas limitaciones, introdujeron mejoras en el algoritmo, incorporando una pequeña cantidad de conocimiento *a priori* para reducir la prueba y error. Además, adoptaron una política de exploración adaptativa basada en el algoritmo *epsilon-greedy*. En su estudio, se comparó el rendimiento de este algoritmo modificado con el de *Q-learning* y SARSA, y los resultados mostraron una planificación de rutas superior, un menor tiempo computacional y una convergencia más rápida.

2.4. Aprendizaje por Refuerzo en Conducción Autónoma

Algunas tareas en las cuales se aplican los algoritmos de Aprendizaje por Refuerzo son:

- Optimización del controlador
- Planificación de rutas
- Optimización de trayectorias
- Planificación del movimiento
- Planificación dinámica del movimiento

A continuación, se recopilan algunos ejemplos ilustrativos, sin vocación de ser exhaustivos. En [19] se aplica el algoritmo *Q-learning* para crear un controlador capaz de hacer que un robot resuelva la aplicación del sigue-línea. Concretamente, utilizan una técnica conocida como *SA-based Q-learning* para resolver el dilema de la *explotación-exploración* planteado en la Sección 2, alcanzando unos resultados que muestran que esta técnica supera al algoritmo *e-greedy* y al *P-controlled* cuando se entrena durante los suficientes episodios. Los resultados obtenidos en este trabajo se muestran en la Figura 8. Además, se puede observar el robot real en la Figura 9.

The best score out of five tries

Algorithm	Score
P controlled	516.32
ε-greedy Q-learning MISO	432.93
SA-based Q-learning MISO	433.34
SA-based Q-learning MIMO	518.95

Figura 8: Resultados experimentales en el circuito complejo [19]

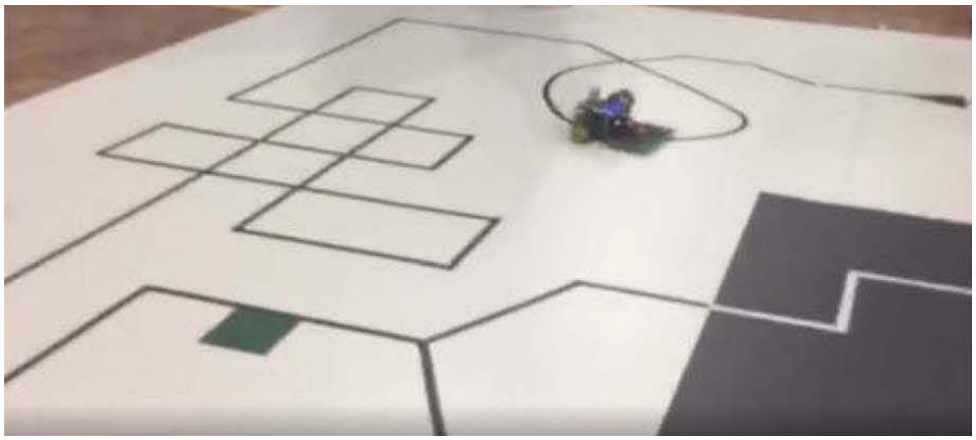

Figura 9: Prueba experimental del robot en un mapa de líneas complejo [19]

Una de las mayores preocupaciones que pueden surgir cuando se aplican algoritmos de Aprendizaje por Refuerzo al problema de la conducción autónoma se da cuando el *agente* se enfrenta a escenarios que no ha visto durante su entrenamiento. En este caso, el comportamiento del *agente* puede ser impredecible. Una solución a este problema se propone en [20], en el cual los autores proponen combinar el Aprendizaje por Refuerzo profundo con técnicas de control basadas en seguridad que demuestran tener un buen rendimiento

a la hora de evitar colisiones con vehículos en los alrededores.

Un marco de trabajo muy útil en los últimos años para resolver problemas relacionados con la conducción autónoma mediante Aprendizaje por Refuerzo es *AWS DeepRacer*, el cual tiene su propia liga competitiva en la que los desarrolladores pueden probar sus sistemas de aprendizaje automático, tanto en su dispositivo físico como en su simulador 3D. Un trabajo ilustrativo que utiliza este entorno es el realizado en [21], en el cual logran mejorar los modelos predeterminados de la plataforma al encontrar la ruta óptima evitando obstáculos, combinando algoritmos como A* y LoS con Aprendizaje por Refuerzo.

Otro trabajo que aprovecha esta plataforma es el que se realiza en [22], donde los autores aplicaron la planificación de rutas en *DeepRacer* basado en RL. Propusieron un marco de sistema novedoso llamado VNARM (*Vehicle Network Autonomous Racing Model*), que permite que varios autos *DeepRacer* cooperen entre sí para lograr un objetivo común. VNARM superó a los algoritmos de conducción autónoma de última generación en términos de tiempo de carrera y tasa de finalización. Específicamente, el rendimiento del vehículo al completar una vuelta se redujo de casi 30 segundos a menos de 9 segundos, manteniendo al mismo tiempo un alto porcentaje de finalización.

En el artículo [23], los autores argumentan que el costo de entrenar un modelo en *DeepRacer Console* es demasiado caro para principiantes, por lo que presentan un nuevo proceso de simulación en *DeepRacer* basado en la plataforma EC2 [24], que resulta más económico.

En el trabajo presentado en [25], se incorporan Redes Neuronales Recurrentes (RNN) junto con modelos de atención con el objetivo de reducir la complejidad computacional para implementar el sistema de conducción autónoma en hardware integrado. Más específicamente, los autores desarrollan un marco de trabajo de extremo a extremo que utiliza RNN en el Aprendizaje Profundo por Refuerzo para abordar escenarios de Problemas de Decisión Parcialmente Observables (POMDP). Específicamente, logran desarrollar un marco para el mantenimiento del carril que fue probado en el simulador TORCS.

2.4.1. Sistemas multi-agente

Cuando se quiere resolver un problema de conducción autónoma, se debe tener en cuenta que en el entorno real habrá otro tipo de actores como ciclistas, peatones u otros vehículos. Por este motivo, se dice que la conducción autónoma es fundamentalmente un problema multi-agente [4].

Algunas investigaciones han centrado sus esfuerzos en crear simuladores ultra-realistas en los cuales se puedan introducir multitud de actores diferentes con los que interactuar. En [26] los investigadores del *Microsoft AI + R* crearon un sistema de este tipo en el cual es posible probar los algoritmos de DAprendizaje por Refuerzo sobre varios agentes al mismo tiempo y coordinarlos. Además, en este trabajo se propone el uso de los juegos de Markov parcialmente observables (POSG) con el fin de formular problemas de conducción autónoma conectada realizando asunciones realistas. Los resultados obtenidos en este trabajo demuestran que los agentes entrenados en la plataforma desarrollada son capaces de aprender las políticas de comportamiento correctas de forma independiente a partir de los datos en alta dimensión (píxeles captados por la cámara).

Figura 10: La Figura muestra un entorno de aprendizaje multiagente heterogéneo desarrollado utilizando MACAD-Gym. En la imagen de la izquierda, se presenta una visión panorámica del escenario en su conjunto. La imagen central representa el escenario simulado en detalle, y en la imagen de la derecha se muestran vistas focalizadas de las observaciones individuales de cada agente [26].

Estos algoritmos son de especial importancia a la hora de coordinar grupos grandes de vehículos, negociar intersecciones sin señalización o realizar adelantamientos en autopistas. Además, estos sistemas pueden tener un rol muy importante a la hora de desarrollar

políticas más robustas y seguras para la conducción autónoma como se muestra en [27], en el cual se plantea que debe haber un equilibrio entre anticipar los comportamientos inesperados por parte de otros actores de la escena y no ser demasiado defensivo de tal forma que se mantenga la fluidez normal del tráfico.

3. Herramientas

En esta sección se van a describir todas las herramientas utilizadas, tanto software como hardware, que han sido necesarias para llevar a cabo este TFM. En primer lugar se describirá la infraestructura hardware utilizada para entrenar los distintos agentes. A continuación, se describe el simulador utilizado para finalmente introducir las distintas herramientas de desarrollo utilizadas.

3.1. Hardware

Una de las principales limitaciones en el campo de la robótica y la inteligencia artificial es la necesidad de contar con infraestructuras de hardware que dispongan de suficiente capacidad de cómputo para realizar entrenamientos prolongados y demandantes. Esta necesidad se vuelve especialmente relevante cuando se aborda la automatización de tareas complejas, como la conducción autónoma.

En este caso, la infraestructura disponible incorpora los componentes contenidos en la Tabla 1.

Especificaciones	Características
Procesador	AMD Ryzen 7 2700x @ 3.6GHz
Memoria RAM	16 GB
Sistema operativo	Ubuntu 22.04
Tarjeta gráfica	Gigabyte GeForce GTX 1660 SUPER

Tabla 1: Descripción del equipo utilizado

3.2. Simulador *CARLA*

La utilización de simuladores en el campo de la robótica es una práctica esencial y cada vez más importante. Estos simuladores proporcionan entornos virtuales que imitan situaciones del mundo real, permitiendo a los investigadores, ingenieros y desarrolladores probar y validar algoritmos, diseños y sistemas robóticos antes de ejecutarlos en los robots físicos.

Uno de los simuladores más populares y que ha ganado más relevancia en los últimos tiempos, sobre todo en el ámbito de la conducción autónoma, es *CARLA Simulator* [28], cuya última versión es la 0.9.14 en el momento de la escritura de esta memoria. Sin embargo, por motivos de coordinación con el resto del laboratorio *RoboticsLabURJC*, para este trabajo se ha utilizado la versión 0.9.13.

CARLA ha sido desarrollado desde cero para respaldar el desarrollo, entrenamiento y validación de sistemas de conducción autónoma. Además de código y protocolos de código abierto, *CARLA* proporciona activos digitales abiertos (diseños urbanos, edificios, vehículos) creados con este propósito y que pueden utilizarse libremente. La plataforma de simulación admite la especificación flexible de conjuntos de sensores, condiciones ambientales, control completo de todos los actores estáticos y dinámicos, generación de mapas, etc. En la Figura 11 se pueden observar dos imágenes del *Town10* de *CARLA*.

Figura 11: Imágenes de la ciudad *Town10* de *CARLA*.

Algunos de los elementos más importantes en *CARLA* son los siguientes [29]:

- Gestor del tráfico: Sistema que se encarga de gestionar el comportamiento del tráfico con el objetivo de que sea realista y útil para entrenar distintos algoritmos para la conducción autónoma.

- Sensores: Dispositivos de percepción que se pueden incorporar a los vehículos. Los datos captados por los sensores son fácilmente accesibles para el cliente de *CARLA*.

Algunos ejemplos de estos dispositivos son: radar, LIDAR, detector de colisiones y la cámara, siendo este último el más realista.

- *Recorder*: Herramienta que permite al usuario acceder a cualquier momento específico de una simulación previamente grabada.
- *ROS Bridge* e implementación de Autoware: *CARLA* ofrece la posibilidad de interactuar directamente con el *middleware* ROS, el estándar de facto en robótica de servicio, tal y como ocurre en otros simuladores para robótica, por ejemplo, *Gazebo* [30].
- Activos abiertos: Diferente activos que *CARLA* facilita al usuario con el objetivo de ofrecerle mayor control sobre la simulación para poder adaptar el entorno a su caso específico.
- Ejecutor de escenarios: Diferentes rutas que ofrece *CARLA* con el objetivo de que los usuarios puedan comparar sus algoritmos y competir entre ellos.
- Este simulador cuenta con un motor de renderizado (*Unreal Engine*) y un motor de físicas. Además, es *open source*.

Una de las principales razones por las que se ha utilizado este simulador es por ser **fotorrealista**, siendo, especialmente la cámara, el sensor con más realismo de los ofrecidos por *CARLA*.

3.3. Lenguaje *Python*

Python es uno de los lenguajes de programación más utilizados en la actualidad [31], especialmente en el ámbito de la robótica, además, es uno de los lenguajes más sencillos, ofreciendo un tiempo de prototipado muy rápido.

Por otro lado, *Python* permite interaccionar con CARLA de una manera muy sencilla, lo que ha sido una gran ventaja a lo largo del trabajo. Esto ha proporcionado la oportunidad de dedicar tiempo al modelado de los algoritmos, en lugar de emplearlo en la resolución de los problemas de código que suelen ser habituales en lenguajes más complejos.

Por estos motivos, este proyecto se ha desarrollado íntegramente en *Python*, en su versión 3.8 por ser la última compatible con *CARLA 0.9.13*. Las principales librerías utilizadas serán descritas en las posteriores secciones.

3.4. *Numpy*

Numpy es una de las principales librerías de *Python* utilizadas para el cómputo numérico [32]. Está especialmente optimizada para trabajar con operaciones tensoriales, lo cual la hace óptima para trabajar con imágenes digitales, las cuales, como es sabido, son tensores. Se ha utilizado esta librería en su versión 1.24.3, principalmente para realizar procesamiento sobre la imagen que aporta la cámara a bordo del vehículo.

3.5. *OpenCV*

Open Source Computer Vision es una librería de código abierto ampliamente utilizada en el contexto de la visión por computador [33], ofreciendo una gran cantidad de algoritmos ya implementados y facilitando el procesamiento de las imágenes, lo cual ha sido necesario para la etapa de la percepción.

Una vez más, esta biblioteca es muy sencilla de utilizar ya que, aunque en su mayor parte está implementada en *C++*, es posible utilizarla desde *Python* con el *binding opencv-python*. Por este motivo, las llamadas a los métodos de esta librería son muy eficientes. Concretamente, en este trabajo se ha utilizado dicha librería en su versión 4.7.0.72.

3.6. *PyTorch*

PyTorch es un *framework* de código abierto ampliamente utilizado para entrenar algoritmos de aprendizaje profundo en particular y de aprendizaje automático en general [34]. En este caso, se ha utilizado *PyTorch* 2.0.0 para hacer inferencia con una red previamente entrenada que era capaz de, dada una imagen, decidir qué píxeles eran parte del carril de la carretera y qué píxeles no lo eran.

Cabe destacar que para sacar el máximo provecho a este tipo de marcos de trabajo es conveniente tener una *GPU* de *NVIDIA* con *CUDA* [35] correctamente instalado, lo cual

reducirá en gran medida el tiempo de inferencia.

4. Aplicaciones

En esta sección se van a explicar en detalle las dos aplicaciones de conducción autónoma basada en visión que se han resuelto en este TFM. En primer lugar se introducirá el problema, se hablará de la configuración de los parámetros de entrenamiento, se definirán los estados, la función de recompensa, y, por último, se comentarán los resultados experimentales obtenidos.

4.1. Sigue-línea visual

La primera aplicación resuelta es la que consiste en que un vehículo simulado en *CARLA* sea capaz de, utilizando únicamente información visual, seguir una línea. Este es uno de los problemas más básicos que se pueden resolver en el contexto de la conducción autónoma y que ya ha sido resuelto previamente en el *RoboticsLabURJC* en otros simuladores como *Gazebo*.

4.1.1. Escenario

Como se ha comentado anteriormente en la sección 3.2, en este trabajo se ha utilizado el simulador *CARLA*. El primer paso para resolver este problema fue encontrar un circuito en el que entrenar y probar la solución. Dado que *CARLA* presenta entornos muy realistas, se realizó una búsqueda exhaustiva en todas las poblaciones que ofrece este simulador. Finalmente, para este problema se ha utilizado el siguiente segmento pintado en verde de la ciudad *Town07* de la version 0.9.13 de *CARLA*. La razón de esta elección fue que este circuito ofrecía condiciones favorables en términos de percepción de la línea.

Como se puede observar en la Figura 12, este 'circuito' es uno de los pocos que presentan una línea continua larga que pueda servir de referencia para que el vehículo la siga. Este segmento de la ciudad presenta una curva bastante cerrada, una recta larga que servirá para poner a prueba la capacidad del agente de no hacer *zig-zag*, y, en su mayor parte, consiste en curvas suaves.

Por otro lado, se ha utilizado el *Tesla Model 3* como agente para resolver este problema (Figura 13). Este vehículo simulado tiene, a su vez, una cámara simulada que capta

Figura 12: Circuito de la ciudad *Town07* de *CARLA* en el que fue entrenado el agente de Aprendizaje por Refuerzo.

imágenes de 640×480 en RGB, colocada en la parte delantera y centrada.

Figura 13: *Tesla Model 3* simulado en *CARLA*.

La cámara a bordo del vehículo captura imágenes a una velocidad aproximada de 30 *FPS*. Por otro lado, los actuadores del vehículo simulado reciben órdenes de aceleración (*throttle*), que tendrá un valor en coma flotante entre 0 y 1, y de ángulo de giro (*steer*), el cual tendrá un valor entre -1 y 1.

4.1.2. Aprendizaje por Refuerzo en *CARLA*

Como parte de este trabajo, y siguiendo el paradigma de *Programación Orientada a Objetos*, se han creado dos clases que sirven para manejar la simulación y para implementar el algoritmo de Aprendizaje por Refuerzo utilizado, *Q-learning*. Estas clases han sido implementadas siguiendo la estructura propuesta en el estándar de *OpenAI Gym*, cuyos métodos más importantes son:

- **Método *step***: El método *step* es fundamental en el Aprendizaje por Refuerzo, ya que se encarga de llevar a cabo una acción determinada a en el estado actual s, lo que resulta en una transición al nuevo estado s' y la obtención de una recompensa r. Este método devuelve 4 elementos: el nuevo estado s', la recompensa r', la variable booleana *done* que indica si se debe reiniciar o no la simulación, bien porque se ha llegado al final del circuito o si el vehículo ha pasado a un estado demasiado malo, y por último una variable opcional *info* que podría contener información añadida

para tareas de visualización o de *debugging*.

- **Método *reset***: El método *reset* es, también, uno de los más importantes que se deben implementar para entrenar un agente de este tipo. Se ejecutará bien cuando el vehículo se haya desviado demasiado de la línea que tiene que seguir, o bien cuando se haya alcanzado el final del circuito.

Estos métodos son recogidos dentro de la clase *QlearningAgent*, que a su vez contiene la definición de los estados y de la función de recompensa, ambas serán explicadas posteriormente en este documento.

Por otro lado, la clase *CarlaEnv* se encarga de inicializar correctamente la simulación y de interactuar con ella. Los métodos más destacables son los que siguen:

- **Método *spawn_vehicle***: Este método se encarga de colocar en la simulación, en uno de los lugares posibles dentro del circuito, al vehículo junto con el sensor cámara. A este método se le llama al comienzo de cada época del entrenamiento, es decir, cada vez que se ejecuta el método *reset* explicado anteriormente.
- **Método *control***: Este método se encarga de, dada una acción, traducirla a una orden para los actuadores del vehículo.
- **Método *calc_center***: Este es uno de los métodos principales de todo el proceso de entrenamiento ya que se encarga de leer la imagen de la cámara a bordo del vehículo, procesarla, y devolver el píxel o píxeles centrales de la línea que el coche debe seguir. En posteriores secciones se explicará cómo se procesan dichas imágenes ya que será determinante para decidir en qué estado se encuentra el agente.

A continuación, se muestra en la Figura 14 un diagrama estático de clases conteniendo las clases desarrolladas en este Trabajo de Fin de Master, junto con todos sus métodos y atributos públicos y privados.

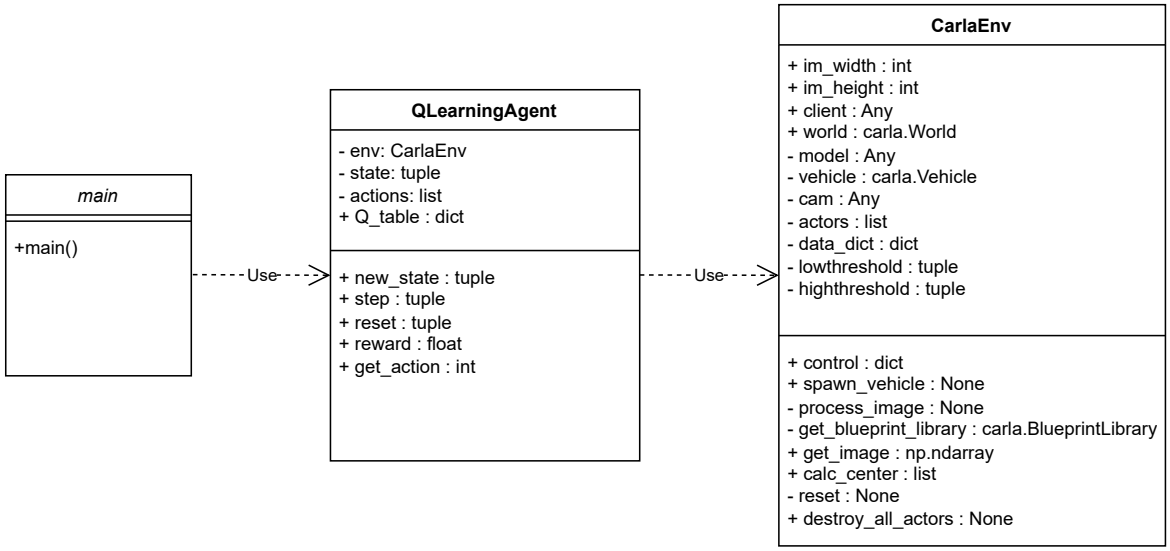

Figura 14: Diagrama estático de clases conteniendo la funcionalidad completa para interactuar con el simulador y llevar a cabo un entrenamiento del algoritmo de Aprendizaje por Refuerzo.

4.1.3. Reinicios Aleatorios

Una propiedad adicional añadida al entrenamiento del agente es la posibilidad de reiniciar aleatoriamente a una de las posiciones seleccionadas previamente en el circuito. El propósito de esta mejora es optimizar el proceso de entrenamiento al exponer al agente a situaciones que normalmente solo se encuentran al completar ciertas secciones del circuito. Esta funcionalidad resulta especialmente beneficiosa en situaciones en las que la convergencia del agente no es inmediata, ya que le permite aprender de manera más eficiente durante las etapas iniciales del entrenamiento, cuando su capacidad de exploración es más activa.

La inclusión de los reinicios aleatorios evita la necesidad de que el agente recorra todo el circuito desde el principio si se encuentra con una situación no explorada al final del recorrido. En cambio, le permite volver a enfrentarse a dicha situación de manera más rápida y efectiva. Esto contribuye a un proceso de entrenamiento más genérico y permite al agente aprender de manera más ágil, mejorando su capacidad para lidiar con desafíos

en diferentes partes del circuito, evitando el *overfitting*.

La Figura 15 muestra una vista cenital de la ciudad utilizada para el entrenamiento. En esta imagen, se destacan los puntos designados del circuito donde el agente tiene la posibilidad de reaparecer, junto con el sentido de circulación establecido.

Figura 15: Vista cenital de la ciudad *Town07* de *CARLA*. Marcados con un círculo y una flecha verde se muestran las posiciones donde el agente puede reaparecer.

4.1.4. Entrenamiento

Una vez se ha diseñado la arquitectura necesaria para llevar a cabo un entrenamiento dentro de *CARLA*, se deben tomar dos decisiones importantes adicionales. En primer lugar, se deben definir los estados, para los cuales se utilizará la información visual captada por la cámara a bordo del vehículo, utilizando técnicas clásicas de visión artificial para procesar las imágenes. En segundo lugar, y como una de las propiedades del algoritmo *Q-learning* son las acciones discretas, se debe decidir el número de acciones que el agente va a ser capaz de realizar.

Estados

Con el fin de calcular en qué estado se encuentra el agente, se debe en primer lugar procesar la imagen. El primer paso llevado a cabo en este aspecto ha sido el de reducir a la mitad la altura de la imagen, reduciendo su tamaño de 640×480 a 640×240, ignorando aquellas partes de la imagen que no aportan ninguna información útil para este propósito y reduciendo así el tiempo de cómputo.

Tras reducir el tamaño de la imagen, se lleva a cabo el procesado de la imagen, siendo este básicamente un filtro de color centrado en el amarillo (Figura 16). Esta funcionalidad se recoge en el método *calc_center* dentro de la clase *CarlaEnv*.

```
def calc_center(self):
    open_cv_image = np.uint8(self._data_dict["image"])
    open_cv_image = open_cv_image[240:480, :, :]
    image_copy = np.copy(open_cv_image)
    hsv_frame = cv2.cvtColor(image_copy, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsv_frame, self._lowthresholds, self.
   _highthresholds)

    positions = []
    for y in [100, 110, 120, 130]:
        x = 620
        while x > 0:
            if mask[y][x] == 255:
                break
```

```
            x -= 1
        z = x
        while z > 0:
            if mask[y][z] != 255:
                break
            z -= 1
        pos = ((x + z) / 2)
        if 610 < x <= 620:
            pos = 0
        positions.append((pos, y+240))

    return positions
```

Fragmento 1: Función encargada de procesar la imagen y calcular los distintos centros de la línea que el vehículo debe seguir.

A alto nivel, este algoritmo sigue los siguientes pasos:

- Reducir el tamaño de la imagen.
- Pasar la imagen del espacio de color *BGR* propio de *OpenCV* al espacio *HSV* con el objetivo de obtener un algoritmo algo más robusto ante cambios de intensidad en la imagen.
- Generar una máscara binaria a partir de un filtro de color centrado en el amarillo.
- Recorrer la imagen resultante de derecha a izquierda a distintas alturas de la imagen hasta encontrar el primer píxel marcado en blanco, x.
- A partir de ese punto, se sigue iterando a lo largo de las columnas hasta encontrar el siguiente píxel en negro, z, para finalmente guardar el punto medio entre x y z.
- Finalmente, se devuelven 4 puntos perceptivos de la imagen: 4 centros de la línea a distintas alturas.

A partir de estos puntos, se definen los estados dividiendo la imagen en $n = 17$ franjas verticales equidistantes. La franja que contiene el punto medio previamente calculado se convierte en el elemento definitorio del estado actual del sistema. Es importante señalar que en el desarrollo de este trabajo se optó por la utilización de tres líneas perceptivas específicas para la definición de los estados tal como se muestra en la Figura 16.

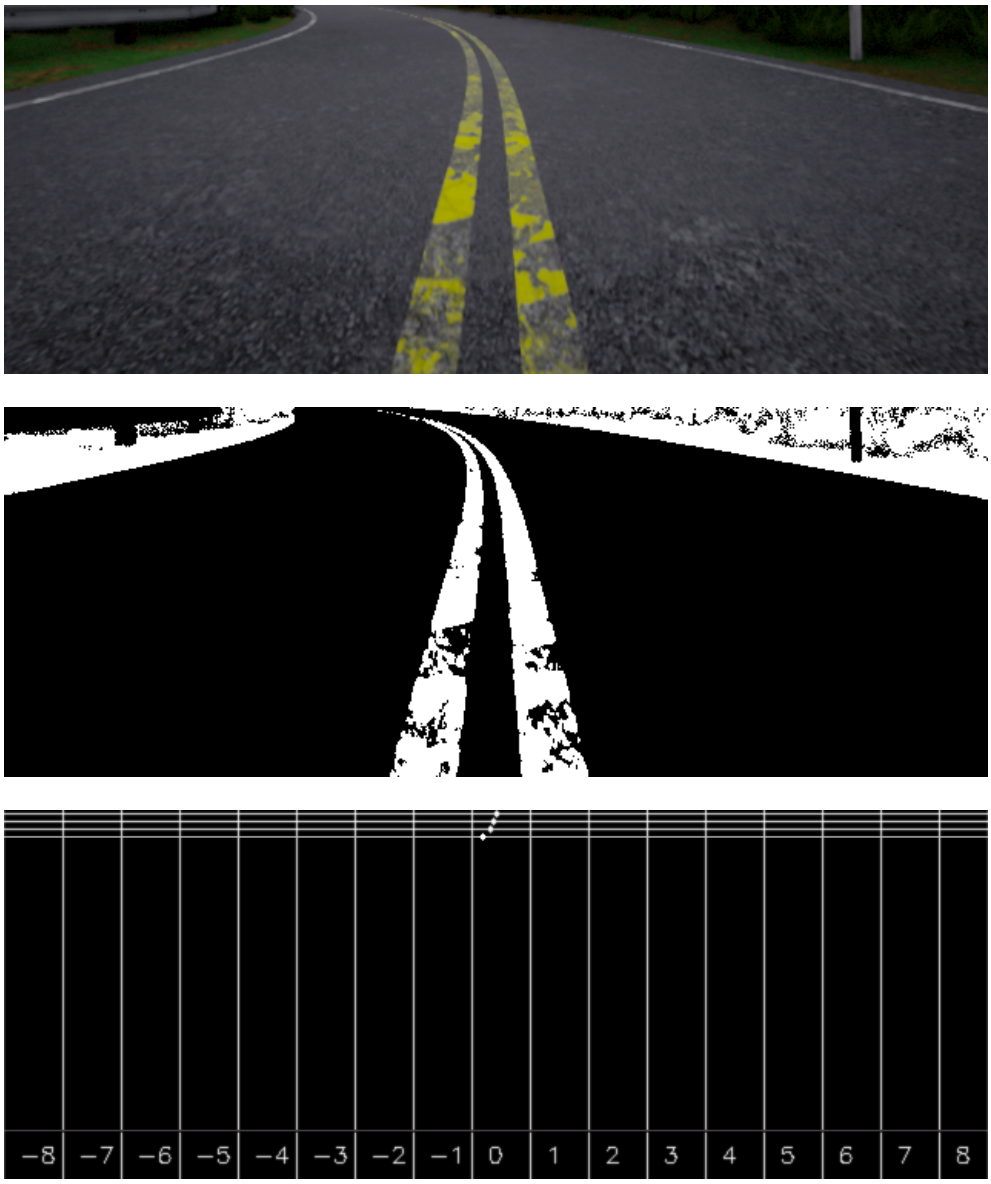

Figura 16: Arriba, un ejemplo de una imagen reducida de tamaño sacada de la cámara a bordo de vehículo simulado en *CARLA*. En el medio, esta misma imagen una vez se ha procesado con el algoritmo propuesto. Por último, un ejemplo de lo que sería un estado $s = 0, 0, 0, 0$.

Acciones

En un entorno realista, las acciones que toma un conductor son continuas, es decir, no están limitadas a un conjunto predefinido. Por ejemplo, cuando una persona conduce un vehículo por una carretera tiene la opción de girar un ángulo $\alpha \in (-900, 900)$. Este sería el caso ideal para los agentes de Aprendizaje por Refuerzo, y, aunque sí es así en algoritmos como por ejemplo el *Deep Deterministic Policy Gradients* (DDPG), no lo es para otros

algoritmos más clásicos, haciendo que sea necesaria la definición de acciones discretas.

En este caso particular y tal como se ha mencionado anteriormente, el algoritmo utilizado en este trabajo tiene como propiedad las acciones discretas. Es por este motivo que se debe decidir cuántas acciones y cuáles va a ser capaz de realizar el agente. En este caso, se han fijado 5 acciones posibles, distribuidas como se muestra en la Tabla 2, las cuales sirven para alcanzar una velocidad media de 20 km/h.

Action	0	1	2	3	4
Throttle	0.32	0.7	0.32	0.2	0.2
Sterring Angle	-0.2	0.0	0.2	-0.4	0.4

Tabla 2: Tabla de acciones para la aplicación *sigue-línea*.

Cabe destacar que este conjunto de acciones se ha seleccionado teniendo en cuenta las características generales del circuito el cual el agente debe completar, con acciones simétricas para los giros y eliminando la opción de estar parado ($v = 0, w = 0$), debido a que ésta siempre obtendría la máxima recompensa, pero al no estar moviéndose, no sirve para la implementación del prototipo. Más adelante, se presentará otro conjunto de acciones diferente, más adaptado a la circulación por autopista.

Función de Recompensa

La función de recompensa es otro de los elementos importantes del Aprendizaje por Refuerzo, siendo la definición de ésta uno de los aspectos más críticos para conseguir el correcto funcionamiento de los algoritmos. El objetivo principal de la función de recompensa en el Aprendizaje por Refuerzo es proporcionar una medida numérica que guíe al agente hacia la toma de decisiones óptimas en un entorno dado.

Previamente, en el Fragmento 1, se han detallado los pasos hasta calcular los distintos centros de la línea. Estos puntos serán utilizados tanto para calcular el estado actual en el que se encuentra el agente como para calcular la recompensa que se le otorga. El objetivo de esta aplicación es que el centro del vehículo esté alineado con el centro de la línea. Por este motivo, cuando estos dos puntos estén alineados o con una desviación muy pequeña,

la recompensa será máxima, y se irá reduciendo progresivamente hasta llegar a un límite considerado como no válido en el cual se reiniciará al agente en una de las posiciones explicadas en 4.1.3.

Formalmente, se define la función de recompensa de la siguiente manera. Sea C_i el centro de la línea a diferentes alturas, con $i \in \{1, ..., 4\}$, y sea $\bar{C} = (C_1, C_2, C_3, C_4)$. En primer lugar, se debe definir una función auxiliar, f, tal que

$$f(C) = \begin{cases} 15 & \text{si } 0 \leq |320 - C| \leq 38 \\ 12 & \text{si } 38 \leq |320 - C| \leq 2 \cdot 38 \\ 10 & \text{si } 2 \cdot 38 \leq |320 - C| \leq 3 \cdot 38 \\ 2 & \text{si } 3 \cdot 38 \leq |320 - C| \leq 5 \cdot 38 \\ 1 & \text{si } 5 \cdot 38 \leq |320 - C| \leq 6 \cdot 38 \\ -100 & \text{si } 6 \cdot 38 \leq |320 - C| \end{cases}$$

Sea v la velocidad del automóvil en m/s, proporcionada por el simulador. Sea $\lambda = 1.02$. Sea α el *steer* de la acción seleccionada. Entonces, la función de recompensa elegida en este TFM está dada por la expresión (1).

$$R(\bar{C}, v, \alpha) = \sum_{j=1}^{4} \frac{1}{4} f(C_j) + \frac{1}{2} v - \lambda \alpha \tag{1}$$

Exploración-explotación

El paradigma *exploración-explotación* es un aspecto clave en el Aprendizaje por Refuerzo y se refiere a la forma en que un agente toma decisiones para maximizar su recompensa a lo largo del tiempo en un entorno desconocido o incierto. Estos dos conceptos representan un equilibrio fundamental en el proceso de aprendizaje y se pueden entender de la siguiente manera:

- **Exploración:** La exploración se refiere a la estrategia de tomar acciones que el agente no ha intentado previamente o que ha utilizado con menos frecuencia. En

otras palabras, el agente explora el entorno para descubrir cómo funcionan diferentes acciones y estados. La exploración es fundamental al principio del aprendizaje cuando el agente tiene poco conocimiento sobre el entorno.

- **Explotación:** La explotación implica tomar las acciones que el agente ha aprendido que conducen a recompensas más altas en función de su experiencia pasada. El agente explota su conocimiento para tomar decisiones que le darán la máxima recompensa inmediata. La explotación es importante a medida que el agente acumula experiencia y confianza en su conocimiento.

El equilibrio entre exploración y explotación es crítico para el éxito del Aprendizaje por Refuerzo. Si un agente se enfoca exclusivamente en la exploración, podría perder oportunidades de obtener recompensas altas utilizando su conocimiento existente. Por otro lado, si se centra demasiado en la explotación, podría quedarse atrapado en un comportamiento subóptimo sin descubrir estrategias mejores.

Existen varias estrategias para lograr este equilibrio, pero en este TFM se ha seleccionado la ***Epsilon-Greedy*** por ser una de las más comunes y simples. Con esta estrategia el agente elige la mejor acción según el conocimiento que ha adquirido hasta el momento durante la mayor parte del tiempo, pero, ocasionalmente, elige una acción al azar (exploración) con una probabilidad ε definida.

Típicamente, y como se ha hecho en este trabajo, en este tipo de problemas se fija un valor para ε muy cercano a 1 al comienzo del entrenamiento, y se va reduciendo a medida que el entrenamiento avanza y el agente va adquiriendo conocimiento sobre su entorno.

Aprendizaje

Hasta ahora se han explicado todas las partes del sistema necesarias para llevar a cabo un entrenamiento con un algoritmo de Aprendizaje por Refuerzo. A continuación, se va a detallar cómo encajan cada uno de los elementos entre ellos y cuáles son los pasos que se siguen en un entrenamiento típico.

Algunos aspectos adicionales que se deben comentar en este momento son los valores de los hiperparámetros del algoritmo *Q-learning*, los cuales se han establecido de manera

experimental, observando el comportamiento del agente durante su entrenamiento con distintos valores. Los valores de los hiperparámetros se observan en la Tabla 3.

Hiperparámetro	**Valor**
α	0.05
γ	0.9
ε	0.99
ϕ	0.9998

Tabla 3: Hiperparámetros establecidos experimentalmente para un entrenamiento utilizando el algoritmo *Q-learning*. ϕ representa el factor de reducción aplicado a ε, el cual es esencial en el paradigma de la '*exploración-explotación*'.

Una vez se han definido los hiperparámetros, se puede comenzar el entrenamiento. En primer lugar, se establece la conexión con el simulador, configurando el clima de la siguiente manera:

```
1    weather = carla.WeatherParameters(cloudiness=0.0, precipitation=0.0,
     sun_altitude_angle=2.5)
```

Fragmento 2: Clima configurado con condiciones favorables para facilitar el entrenamiento del agente.

A continuación, se llama al método *spawn_vehicle* de la clase *CarlaEnv*, se inicializa la Tabla-Q con todos sus valores a 0 y se comienza el bucle de entrenamiento. El proceso iterativo que se sigue hasta finalizar el entrenamiento es el siguiente:

1. **Se llama al método *reset*** para colocar al vehículo en uno de los puntos de partida seleccionados.

2. **Comienza el episodio** que se estará ejecutando mientras la variable *done* sea falsa.

3. **Se selecciona una acción** siguiendo el paradigma *exploración-explotación* explicado anteriormente.

4. **Se llama al método *step* con la acción seleccionada**, el cual traduce la acción para que se pueda aplicar a los motores del vehículo simulado, obtiene una nueva

imagen de la cámara a bordo, calcula el nuevo estado s_{t+1} y retorna al programa principal. Cabe destacar que este método también devuelve la variable booleana *done* que será verdadera si la recompensa ha sido muy mala y, por tanto, reiniciará al agente y se pasará al siguiente episodio.

5. Con el nuevo estado y la recompensa **se actualiza la Tabla-Q** siguiendo la ecuación explicada en (1).

El resultado final del entrenamiento es un archivo en formato *pickle* de *Python* que se puede utilizar para hacer inferencia, llevando al agente a otras ciudades de *CARLA* e incluso en entornos completamente diferentes.

Finalmente, se han explicado todos y cada uno de los aspectos necesarios para llevar a cabo satisfactoriamente un entrenamiento de un algoritmo de Aprendizaje por Refuerzo en *CARLA*. En la siguiente sección, se comentarán y discutirán los resultados obtenidos al finalizar el entrenamiento.

Experimentos

Con el objetivo de asegurar el correcto funcionamiento de los algoritmos, se han extraído una serie de métricas de los entrenamientos realizados. La métrica más importante que se ha utilizado para decidir cuándo se paraba un entrenamiento es la de la recompensa acumulada. Esta métrica consiste en la suma de las recompensas que el agente ha ido obteniendo en un determinado episodio cada vez que da un *paso*, es decir, cada vez que toma una acción en un estado concreto. Idealmente, esta recompensa acumulada por episodio debe ir aumentando a medida que avanza el entrenamiento hasta alcanzar la convergencia.

En la Figura 17 se pueden observar varias gráficas. La primera, muestra cómo ha ido decayendo el factor de exploración a lo largo del entrenamiento. La segunda, muestra la recompensa acumulada por episodio. En la última gráfica se muestran los *pasos* por episodio. Como se puede observar, los *pasos* así como la recompensa acumulada muestran una convergencia. alcanzando ambos un valor fijo. Se debe destacar que, a mitad del entrenamiento, se ha fijado $\varepsilon = 0$ con el objetivo de que el algoritmo termine de converger.

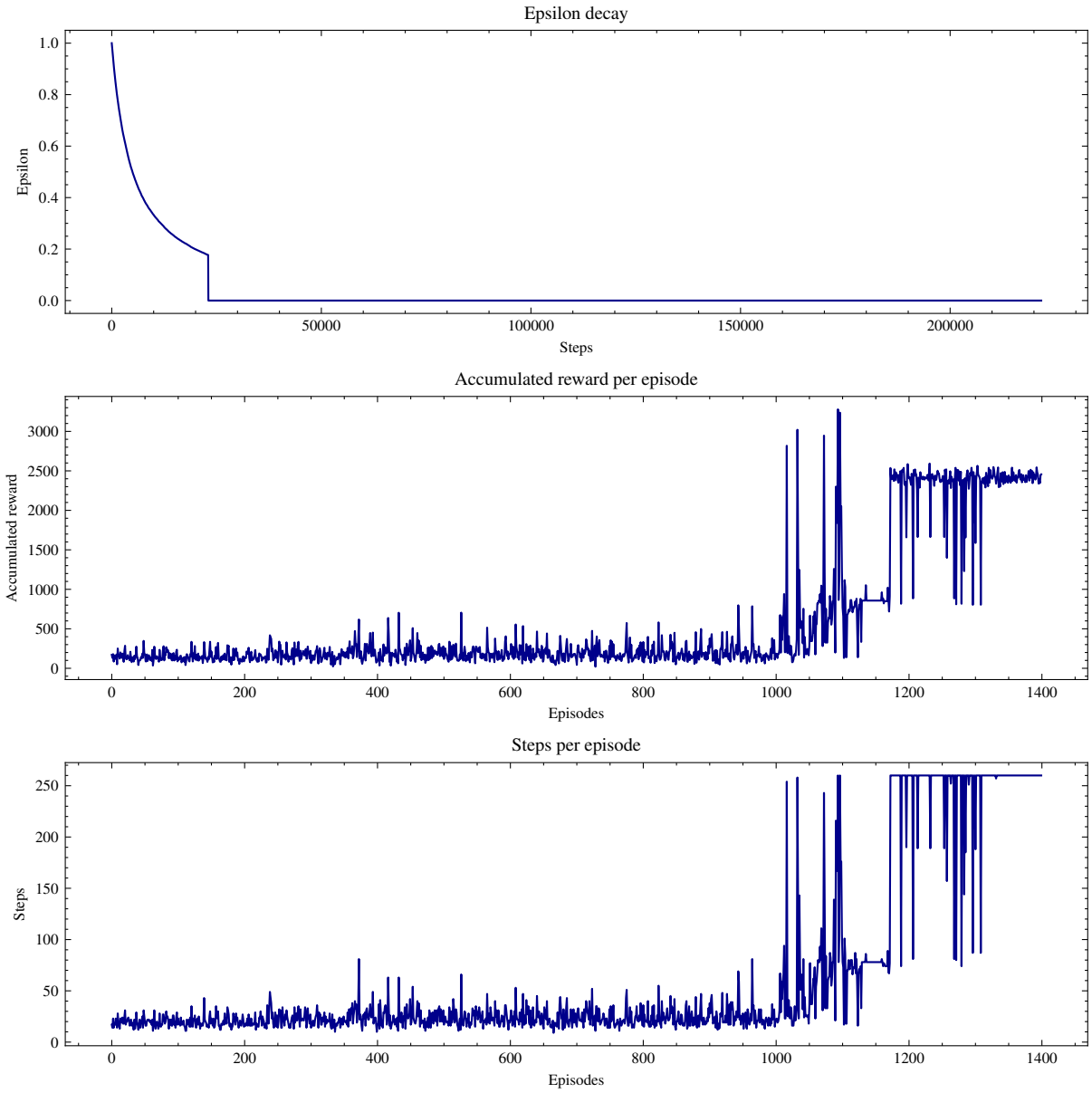

Figura 17: Arriba, gráfica conteniendo el decaimiento del factor de exploración ε. En el medio, la recompensa acumulada que el agente obtiene por episodio. Abajo, los pasos dados en cada episodio del entrenamiento.

Cada entrenamiento consta de un total de 1400 épocas y cada época acaba o bien porque se ha tenido que reiniciar al agente o porque se ha superado un número concreto de *pasos*, en concreto, se considera que se ha terminado el recorrido cuando se han alcanzado un total de 260 *pasos*.

Un aspecto importante que debe destacarse es que, inicialmente, este algoritmo se entrenó sin tener en cuenta la sincronización entre el simulador y el cliente. Esto generaba proble-

mas al calcular la recompensa obtenida al realizar una acción en un estado determinado, ya que se calculaba la recompensa de un estado pasado y no del estado actual. En un primer momento, se intentó resolver este problema modificando los estados, las recompensas y los hiperparámetros. Finalmente, se llegó a la conclusión de que el problema residía en la falta de sincronización en *CARLA*. Al activar el modo síncrono en *CARLA*, los resultados pasaron de un gráfico que no convergía a ningún valor a lo que se muestra en la Figura 17.

Para activar este modo, en primer lugar se intentó seguir la documentación ofrecida por *CARLA* en [36]. Sin embargo, no se consiguió activar este modo correctamente, ya que se obtenían problemas de desbordamiento de memoria y conflictos a la hora de leer información de los sensores de *CARLA*. Por estos motivos, fue necesario realizar una investigación específica, preguntando a otros miembros del *RoboticsLabURJC*, hasta que finalmente se dio con la clave. Concretamente, se ha conseguido activar el modo síncrono en *CARLA* como se muestra en el Fragmento 3:

```
1    self.traffic_manager = self.client.get_trafficmanager(8000)
2    settings = self.world.get_settings()
3    settings.fixed_delta_seconds = 0.05
4    self.traffic_manager.set_synchronous_mode(True)
5    self.world.apply_settings(settings)
```

Fragmento 3: Configuración del modo síncrono en *CARLA* 0.9.13

Con este modo síncrono activado, se asegura que siempre se esté iterando a 20*FPS*.

Validación Experimental

Una vez se ha finalizado el entrenamiento de este agente, se pasa a probar su funcionamiento en inferencia, utilizando la *Tabla-Q* resultado de dicho entrenamiento. El funcionamiento de este agente se ha resumido en un vídeo, accesible en `https://www.youtube.com/watch?v=-yCdYJnClME`.

Como se puede observar, el agente es capaz de realizar la tarea satisfactoriamente, circulando a una velocidad relativamente alta, y aunque presenta ciertas oscilaciones, éstas se pueden deber al conjunto de acciones seleccionado, lo cual podría solucionarse sim-

plemente añadiendo más acciones a dicho conjunto. Se debe destacar que en inferencia, el modo síncrono fue desactivado, alcanzando una media de 35*FPS*. En la Figura 18 se puede observar al agente entrenado siguiendo la línea, tanto en curva como en recta.

Figura 18: Vehículo navegando por el *Town07* en curva (izquierda) y en línea recta (derecha).

4.2. Sigue-carril visual

Una vez se ha llevado a cabo satisfactoriamente el desarrollo de un prototipo de controlador visual para la aplicación de seguir una línea basado en Aprendizaje por Refuerzo, se pasa a un problema más complejo y con más relevancia en la literatura actual.

El problema de seguir un carril se asemeja al de seguir una línea, pero en este caso, en lugar de seguir una línea de un color específico, el agente debe ser capaz de circular por un carril que está delimitado por dos líneas, siendo comúnmente una de ellas discontinua. Aunque a priori ambos problemas parecen similares, en la realidad este último es más complejo, ya que a diferencia del caso anterior, es muy fácil perder la referencia del carril y por tanto calcular mal el estado en el que se encuentra el agente. Por este motivo, en esta aplicación se debe hacer mayor énfasis en la percepción, la cual debe ser muy robusta.

Dado que en la sección anterior se han abordado los aspectos fundamentales del algoritmo, esta sección se centrará en las diferencias más destacadas, esencialmente en la percepción y la cantidad de estados.

4.2.1. Escenario

A diferencia del caso anterior, seleccionar un escenario ha sido una tarea sencilla en esta ocasión, debido a que en *CARLA*, la mayoría de las calles disponen de carriles bien definidos en los que se puede ubicar el vehículo para su entrenamiento. En este contexto, se ha optado por utilizar el segmento marcado en verde en la Figura 19 en la versión 0.9.13 del *Town04* de *CARLA*. Este entorno ofrece condiciones favorables que simplificarán la detección del carril.

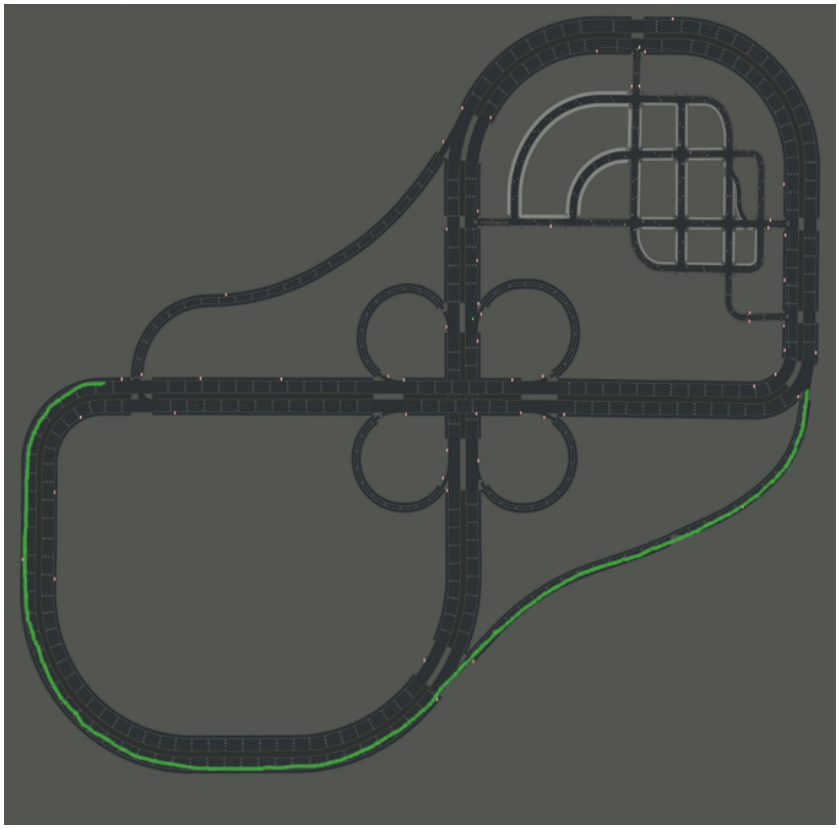

Figura 19: Circuito de la ciudad *Town04* de *CARLA* en el que fue entrenado el agente de Aprendizaje por Refuerzo.

4.2.2. Percepción

Para abordar este problema, se ha dedicado una sección exclusivamente a explicar cómo el agente es capaz de, dada una imagen, detectar el carril y calcular el punto medio de dicho carril, que será el elemento clave con el que se definirán los distintos estados en los que el agente se puede encontrar.

En este caso, se ha utilizado un modelo del Estado del Arte diseñado para realizar segmentación semántica. Este modelo, denominado *MobileNetV3-Small* fue propuesto por investigadores de *Google AI* y *Google Brain* en [37] alcanzando nuevos resultados en clasificación, segmentación y detección. Concretamente, los autores argumentan que el modelo propuesto es un 6.6 % más preciso que su versión anterior.

Uno de los principales motivos por los que se eligió utilizar esta arquitectura es porque es muy liviana y eficiente computacionalmente (siendo capaz de alcanzar hasta los 38 *FPS* en inferencia si se ejecuta en GPU), ya que ha sido pensada para ejecutarse en dispositivos con recursos limitados. En el caso de la conducción autónoma, es bien sabido que se deben tomar decisiones en tiempo real, por lo que es una necesidad esencial que la red pueda hacer inferencia a velocidades superiores a los 20 *FPS*. Este modelo concreto, según los autores del artículo, alcanza el 67.4 % de $mIoU$ [38]. Se puede observar la fórmula de dicha métrica en (2)

$$mIoU = \frac{1}{N} \sum_{i=1}^{N} \frac{TP_i}{TP_i + FP_i + FN_i} \tag{2}$$

siendo N el número de clases, TP_i el número de verdaderos positivos para la clase i, FP_i el número de falsos positivos para la clase i y FN_i el número de falsos negativos para la clase i.

Además, y ya que la necesidad del trabajo en este momento es la de detectar el carril en el que se encuentra el vehículo, este modelo ha sido reentrenado con una base de datos particular publicada en *Kaggle* [39].

Un problema que se presenta al utilizar esta red, es que al devolver una segmentación sobre la imagen de entrada, dicha segmentación no define una línea continua. Esto presenta una serie de inconvenientes, como, por ejemplo, el hecho de que no se puedan establecer las líneas de percepción para calcular los centros del carril a alturas arbitrarias, siendo necesaria la colocación de éstos en una franja muy específica de la imagen.

Por este motivo, se ha decidido aplicar un post-procesado a la salida de la red. Este post-procesado se basa en un ajuste polinómico sobre dicha salida. Ésta devuelve un *frame* en

el que todos los píxeles toman el valor cero salvo aquellos detectados como parte del carril, los cuales valdrán 255, tal y como se puede apreciar, para el lado izquierdo, en la Figura 20. Se destaca, por último, que esta detección se lleva a cabo mediante dos subprocesos independientes, uno por cada una de las líneas que delimitan el carril.

Entonces, tomando la imagen como un dominio bidimensional, se hace un ajuste lineal en tres dimensiones que devuelva la línea que mejor se adapta a esta nube de puntos formada por los píxeles salida de la red. Con esto, el carril queda aproximado por este ajuste (uno por cada una de las líneas delimitadoras del carril) y se asegura el hecho de poder situar el centro del carril a cualquier altura.

4.2.3. Aprendizaje por Refuerzo en *CARLA*

Como se ha señalado anteriormente, la base del módulo desarrollado es muy similar a la que se desarrolló para el *sigue-línea* visual, por lo tanto, en esta sección se van a explicar aquellos métodos adicionales que han sido necesarios para solucionar el *sigue-carril* visual.

En concreto, se han desarrollado dos clases adicionales. Por una lado, la clase denominada *LaneDetector* que será la encargada de recibir una imagen, procesarla, y devolver en forma de lista los distintos centros del carril a distintas alturas de la imagen. Además, se ha desarrollado una clase para encapsular las posibles acciones que puede tomar el agente. Estas clases se integran con el resto como se refleja en la Figura 21.

4.2.4. Reinicios aleatorios

Como se hizo en la sección 4.1.3, se han introducido los reinicios aleatorios en este problema con el objetivo de acelerar el proceso de entrenamiento. Concretamente, se han establecido los puntos que se muestran en la Figura 22 como puntos de reinicio.

4.2.5. Entrenamiento

A continuación, se van a definir los estados, las acciones y la función de recompensa utilizados para resolver esta aplicación.

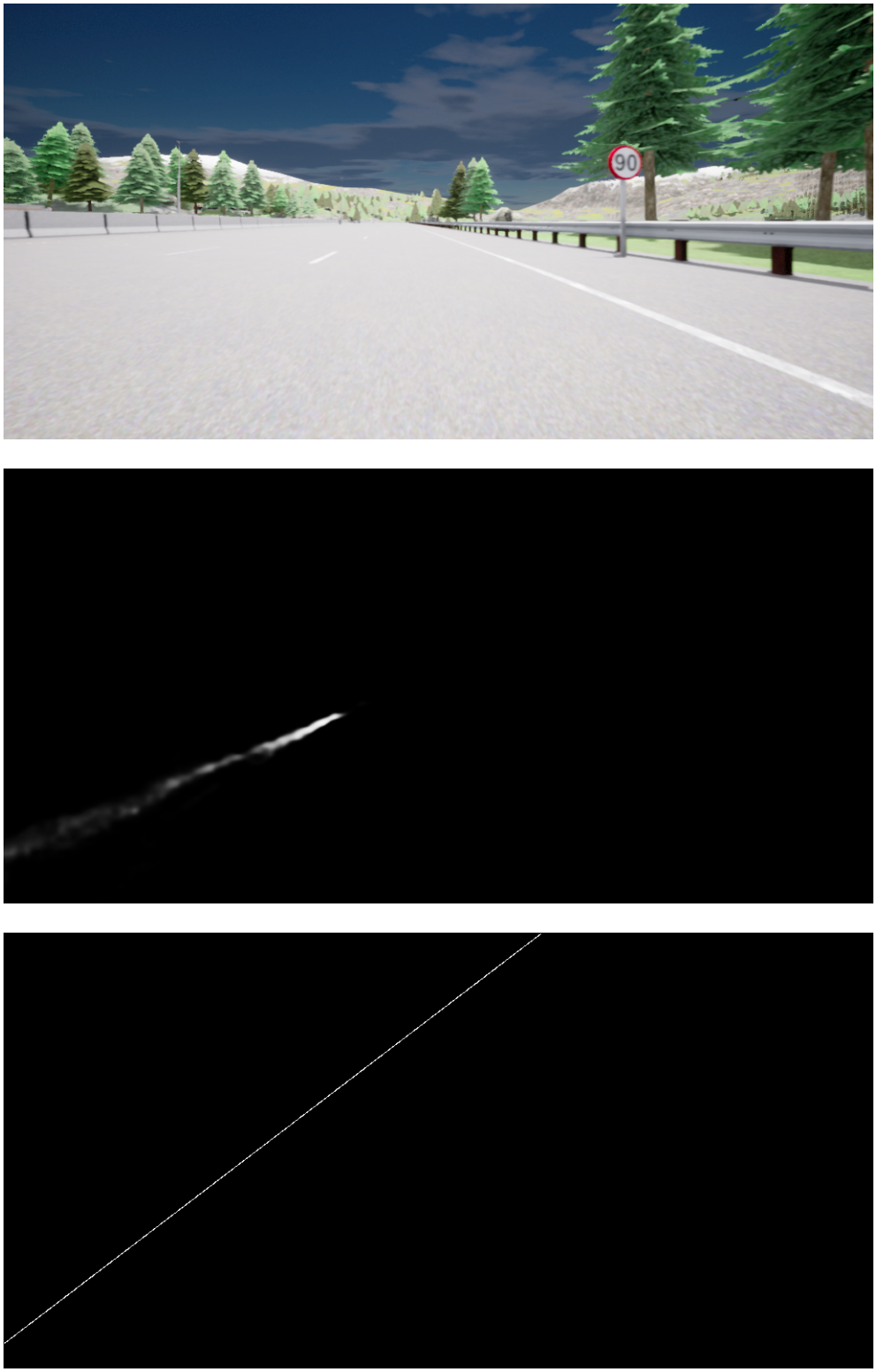

Figura 20: Arriba, imagen de entrada. En el centro, imagen salida de la red de de detección del carril. Abajo, ajuste lineal sobre la imagen salida de la red.

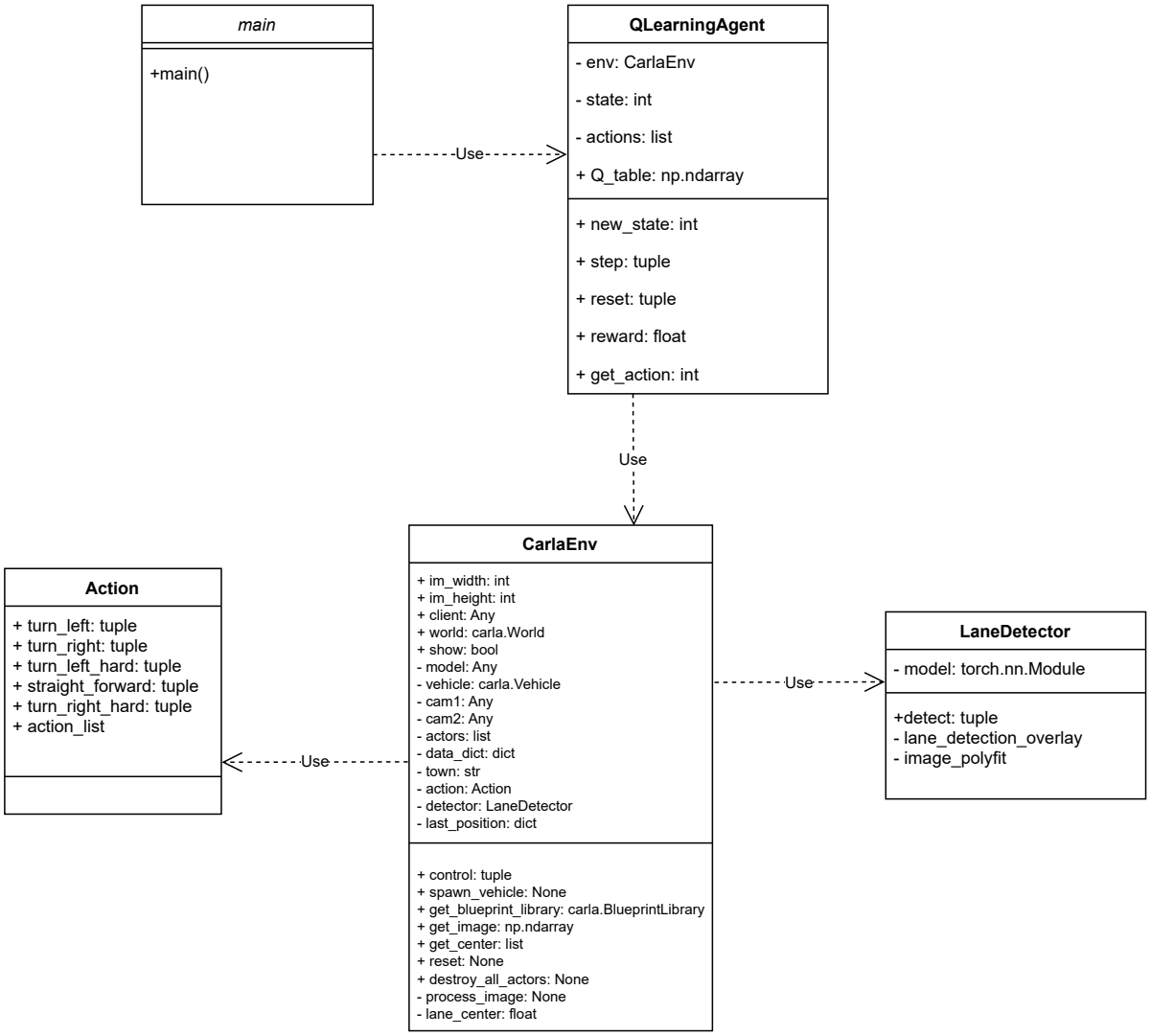

Figura 21: Diagrama estático de clases para el *sigue-carril* que incluye todas las funcionalidades necesarias para interactuar con el simulador y realizar un entrenamiento completo del algoritmo de Aprendizaje por Refuerzo.

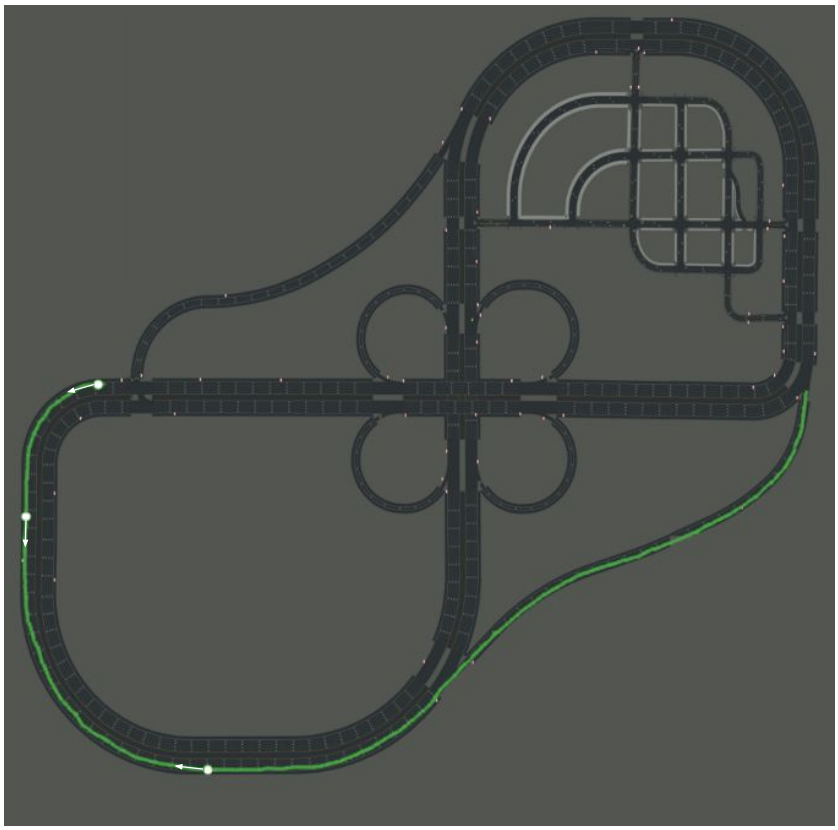

Figura 22: Vista cenital de la ciudad *Town04* de *CARLA*. Marcados con un círculo y una flecha verde se muestran las posiciones donde el agente puede reaparecer.

Estados

Tal como se ha presentado en la aplicación *Sigue-línea*, es necesario definir los estados que se han utilizado para entrenar este agente de Aprendizaje por Refuerzo.

En este caso, y debido a que en este punto del trabajo se disponía de un mayor entendimiento de la problemática que se quería abordar, ha bastado con definir únicamente 5 estados, basados una vez más en la distancia a la que se encuentra el centro del carril con respecto al vehículo. El centro del carril utilizado para calcular el estado se ha definido como la media de la coordenada x de los 4 centros, a 4 alturas distintas de la imagen. Estos estados se ven reflejados en la Figura 23, la cual contiene varios elementos clave:

- Las lineas verdes verticales delimitan los diferentes estados.
- Los cuatro centros de carril indicados como círculos verdes.
- El estado actual en el que se encuentra el agente, señalado en amarillo.
- La salida la red está representada por las líneas azules y rojas, que corresponden a los píxeles de los límites del carril.

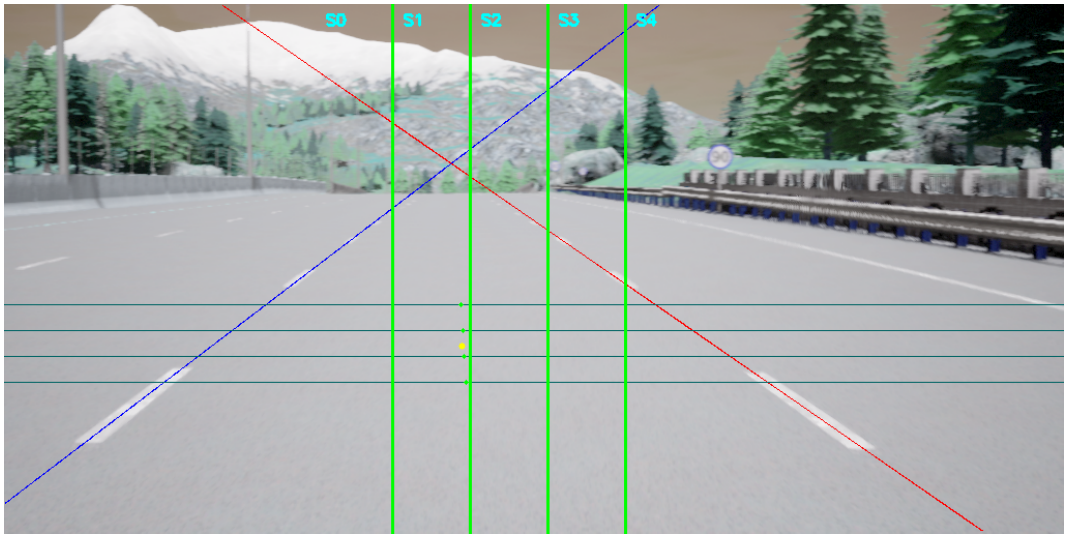

Figura 23: Definición de los estados para la aplicación *sigue-carril*.

Acciones

Se han utilizado 5 acciones distintas que han servido para completar con éxito el circuito establecido. Las acciones se pueden observar en la Tabla 4. Con estas acciones, se alcanza una velocidad media de 25 km/h.

Action	0	1	2	3	4
Throttle	0.35	0.35	0.3	0.6	0.3
Sterring Angle	-0.04	0.04	-0.08	0.0	0.08

Tabla 4: Tabla de acciones para la aplicación *sigue-carril*.

Función de Recompensa

Se describe ahora, matemáticamente, la función de recompensa utilizada en la aplicación del sigue-carril. Para ello, se recomienda observar, en primer lugar, la Figura 24.

Sea C la coordenada x del punto que marca el centro del carril, calculado como la media de los puntos marcados en verde en dicha Figura. Cabe destacar que dichos puntos se obtienen a partir de la salida de la red entrenada para la detección de carril. Se define, entonces, en primer lugar, una función auxiliar, f^*, tal que

$$f^*(C) = \begin{cases} 1 & \text{si } 0 \leq |512 - C| \leq 15 \\ 5 & \text{si } 15 \leq |512 - C| \leq 2 \cdot 15 \\ 2.5 & \text{si } 2 \cdot 15 \leq |512 - C| \leq 3 \cdot 20 \\ 1 & \text{si } 3 \cdot 20 \leq |512 - C| \leq 4 \cdot 20 \\ -10 & \text{si } 4 \cdot 20 \leq |512 - C| \end{cases}$$

Sean, también, p_1, p_2, p_3 y p_4 los puntos identificados en la Figura. De ahora en adelante, se tendrá que $p_1 = (512, 512)$ y $p_2 = (512, 0)$. Es también necesario conocer el ángulo entre una línea vertical de la imagen, definida por el vector $\overrightarrow{p_1 p_2}$ y la línea definida por el vector $\overrightarrow{p_3 p_4}$.

Con esto, se define la función auxiliar g^*, que calcula dicho ángulo, como

$$g^*(p_1, p_2, p_3, p_4) = \arctan\left(\frac{p_{4_y} - p_{3_y}}{p_{4_x} - p_{3_x}}\right) - \arctan\left(\frac{p_{2_y} - p_{1_y}}{p_{2_x} - p_{1_x}}\right)$$

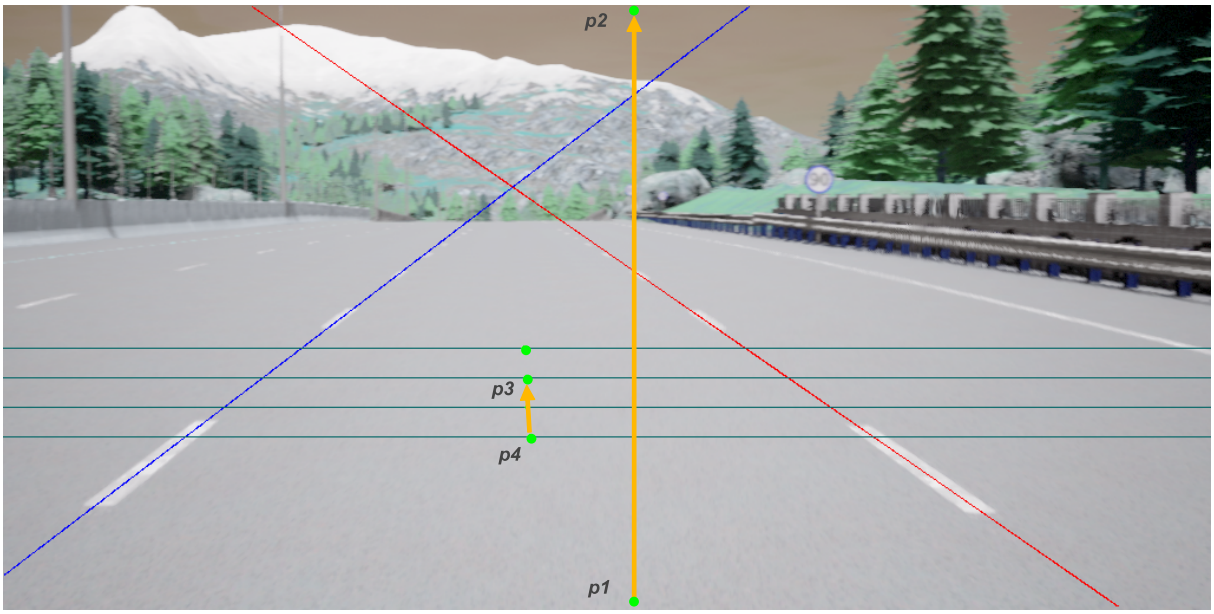

Figura 24: Imagen extraída de *CARLA* que muestra los puntos utilizados para calcular el ángulo del agente con respecto al carril en el que se encuentra. En ella, p_1 y p_2 son puntos fijos, mientras que p_3 y p_4 son puntos hallados a partir de la segmentación realizada por la red. El ángulo que forman entre sí los vectores $\overrightarrow{p_1p_2}$ y $\overrightarrow{p_3p_4}$ será utilizado en la función de recompensa.

En esta función se pueden observar dos términos. El primero de ellos, calcula el ángulo formado por el vector $\overrightarrow{p_3p_4}$ con la horizontal; el segundo, el formado con dicha horizontal por el vector $\overrightarrow{p_1p_2}$. Restando ambos términos, se obtiene el ángulo deseado.

Ahora, sea a la acción seleccionada en un instante concreto, escogida entre las descritas en el apartado anterior. En estas condiciones, se puede definir la función de recompensa buscada, R^*, como

$$R^*(C, a, p_1, p_2, p_3, p_4) = \begin{cases} f^*(C) & \text{if } g^*(p_1, p_2, p_3, p_4) > \zeta \\ 2f^*(C) & \text{if } g^*(p_1, p_2, p_3, p_4) \leq \zeta \wedge a = 3 \\ \dfrac{f^*(C)}{2} & \text{if } g^*(p_1, p_2, p_3, p_4) \leq \zeta \wedge a \neq 3 \end{cases}$$

Como se puede observar, esta función de recompensa otorga una recompensa mayor al agente cuando éste está alineado con el carril, es decir, $g^*(p_1, p_2, p_3, p_4) < \zeta$, y ha seleccionado la acción de ir recto, es decir, $a = 3$.

El parámetro ζ se ha establecido a 6, y, en este caso, ha sido determinado experimentalmente.

Aprendizaje

Hasta este punto, se han abordado todas las componentes del sistema requerido para realizar un proceso de entrenamiento. A continuación, se pasará a describir los hiperparámetros de aprendizaje utilizados en este caso así como la configuración del escenario en *CARLA*.

En primer lugar, se deben señalar cuáles han sido los hiperparámetros establecidos. Éstos se pueden observar en la Tabla 5.

A continuación, se debe configurar el escenario para que sea lo más favorable posible para el agente, pues esto afectará en gran medida a la percepción. En concreto, se ha

Hiperparámetro	Valor
α	0.8
γ	0.9
ε	0.9999
ϕ	0.99998

Tabla 5: Hiperparámetros establecidos experimentalmente para un entrenamiento utilizando el algoritmo *Q-learning*. ϕ representa el factor de reducción aplicado a ε, el cual es esencial en el paradigma de la '*exploración-explotación*'.

establecido el clima como se muestra en el Fragmento 4.

```
1    weather = carla.WeatherParameters(cloudiness=40.0, precipitation
   =0.0, sun_altitude_angle=100.0)
```

Fragmento 4: Clima configurado con condiciones favorables para facilitar el entrenamiento del agente.

Por último, se siguen los mismos pasos de reinicio, selección de acción y actualización de la Tabla-Q que los explicados en la Sección 4.1.4.

A su vez, cuando el entrenamiento finaliza, se obtiene un fichero en formato *pickle* con la Tabla-Q final. Esta tabla se puede cargar después en otro entorno y se puede utilizar para hacer inferencia.

Con esto, se han abordado todos los aspectos esenciales para realizar un entrenamiento utilizando un algoritmo de Aprendizaje por Refuerzo en *CARLA*. En la sección siguiente, se analizarán y debatirán los resultados obtenidos al concluir el entrenamiento.

Experimentos

Una vez más, se ha utilizado principalmente la recompensa acumulada por episodio como métrica para decidir cuándo este algoritmo ha convergido. De la misma manera que se hizo en la Sección 4.1.4, se muestran en la Figura 25 varias gráficas que reflejan el entrenamiento del agente. Igualmente, a mitad del entrenamiento se ha fijado $\varepsilon = 0$. Se

puede observar en esta figura que tras 550 episodios el algoritmo converge, obteniendo la máxima recompensa.

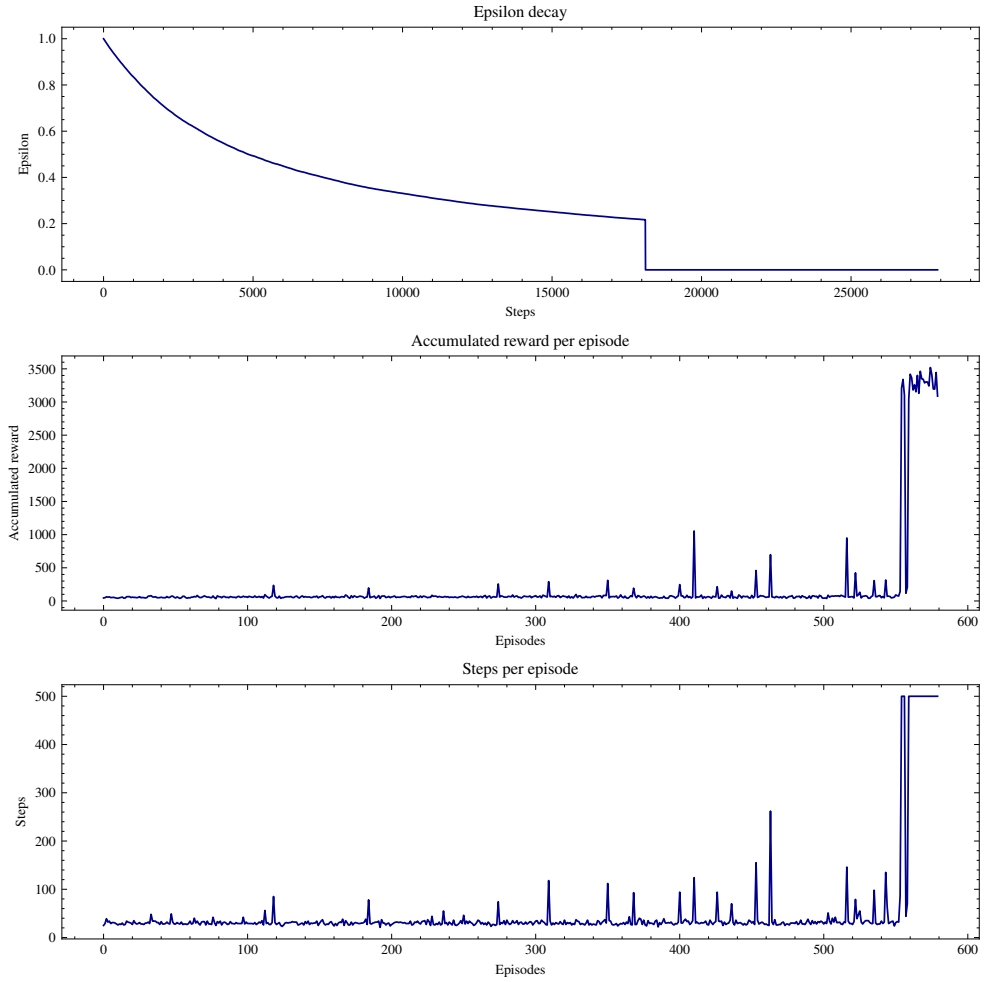

Figura 25: Arriba, gráfica conteniendo el decaimiento del factor de exploración ε. En el centro, la recompensa acumulada que el agente obtiene por episodio. Abajo, los pasos dados en cada episodio del entrenamiento.

Como se ha discutido en la sección 4.1.4, para lograr la convergencia en este problema también se ha necesitado activar el modo síncrono de *CARLA*. La diferencia entre utilizar este modo y no hacerlo se hace evidente en la gráfica de recompensa acumulada, ya que el uso de este modo permite que la recompensa acumulada converja hacia un valor específico. En este caso, se estaba iterando a $50 FPS$.

Validación experimental

Como se hizo en la aplicación del *sigue-línea*, se van a comentar los resultados de utilizar este agente en inferencia. Para ello, se ha editado un vídeo disponible en `https://www.youtube.com/watch?v=_2ma1SqN1MY`. Una vez más, el agente ha aprendido a desempeñar su función correctamente, manteniendo esas pequeñas oscilaciones, e iterando a una media de 25*FPS*. Además, este agente ha sido probado en otros pueblos de *CARLA*, concretamente, en el *Town07* (Figura 12), y se ha comprobado que igualmente puede mantener el vehículo dentro del carril. En la Figura 26, en un segmento del circuito que nunca ha visto durante su entrenamiento.

Figura 26: Vehículo navegando por el *Town04* en curva (izquierda) y en línea recta (derecha).

5. Conclusiones y líneas futuras

En esta sección se van a exponer las conclusiones extraídas al desarrollar este trabajo así como las posibles líneas futuras que se pueden seguir para continuar el proyecto.

5.1. Conclusiones

A lo largo de este TFM se han extraído las siguientes conclusiones:

- Se ha realizado un estudio del Estado del Arte de la robótica con visión en la Sección 2, concretamente de aquellos trabajos que utilizan el Aprendizaje por Refuerzo para llevar a cabo tareas de conducción autónoma (subobjetivo 1).

- Tal y como demuestran las Figuras 17 y 25, se ha conseguido desarrollar un prototipo de las aplicaciones *sigue-línea* y *sigue-carril* con un controlador visual basado en Aprendizaje por Refuerzo con el algoritmo *Q-learning* (subobjetivos 2 y 3) que funcionan satisfactoriamente.

- Gracias al desarrollo que se ha llevado a cabo en este trabajo, se ha logrado adquirir un conocimiento profundo sobre el simulador *CARLA*, lo cual constituía una de las necesidades principales que se debían abordar.

- Se ha utilizado una red neuronal, explicada en la Sección 4.2.2, que se ha utilizado para hacer segmentación de carril, entrenada con una base de datos basada en una versión antigua de *CARLA* y se ha procesado su salida realizando un ajuste polinómico para que ésta sea favorable para el caso de uso presentado en este trabajo.

- Se han diseñado dos funciones de recompensa, explicadas en las Secciones 4.1.4 y 4.2.5, basadas en la posición del vehículo, la velocidad, el ángulo de giro y el ángulo con respecto del carril que han resultado válidas para las aplicaciones de *sigue-línea* y *sigue-carril*.

Finalmente, este trabajo ha puesto en evidencia que la visión artificial junto con el Aprendizaje por Refuerzo son dos herramientas que, juntas, pueden llegar a ofrecer buenos resultados a la hora de abordar ciertas aplicaciones de conducción autónoma. Además,

a diferencia de otros trabajos realizados previamente, este trabajo se basa en un simulador foto-realista, como es *CARLA*, lo cual agrega un valor significativo a la investigación.

Con esto, se han cumplido todos los objetivos planteados al comienzo del trabajo y se da éste por concluido.

5.2. Líneas Futuras

Para concluir esta memoria, se van a presentar las posibles líneas futuras que se pueden seguir para continuar con la investigación:

- Ampliar las funcionalidades que el agente es capaz de realizar. En concreto, la capacidad de circular correctamente por el carril aún cuando hay otros vehículos y objetos presentes, realizar adelantamientos, respetar señales de tráfico, etc.
- Muy ligado al punto anterior, sería interesante abordar este tipo de tareas con algoritmos más complejos y que están ganando relevancia en la literatura actual, como son los algoritmos de Aprendizaje por Refuerzo Profundo (DQN, DDPG, etc.).
- Mejorar los prototipos propuestos, reduciendo la oscilación del vehículo y aumentando la velocidad lineal. Esto podría realizarse introduciendo la velocidad lineal efectiva dentro de la función de recompensa.
- Reentrenar la red neuronal utilizada en el *sigue-carril* con un mayor número de imágenes para aumentar su robustez y permitir que el agente pueda circular por otros pueblos y en otras circunstancias en *CARLA*.
- Diseñar un circuito y migrar la funcionalidad desarrollada a un robot real, por ejemplo el *Rosbot 2R* de *Husarion* (Figura 27).

Figura 27: Robot real *Rosbot 2R* de *Husarion*.

Bibliografía

[1] AI Warehouse. *AI Learns to Walk (deep reinforcement learning)*. 2023. URL: https://www.youtube.com/watch?v=L_4BPjLBF4E&t=71s.

[2] Samuel Arzt. *AI Learns to Park - Deep Reinforcement Learning*. 2019. URL: https://www.youtube.com/watch?v=VMp6pq6_QjI.

[3] Ekim Yurtsever et al. "A Survey of Autonomous Driving: Common Practices and Emerging Technologies". En: *IEEE Access* 8 (2020), págs. 58443-58469. DOI: 10.1109/ACCESS.2020.2983149.

[4] B Ravi Kiran et al. "Deep Reinforcement Learning for Autonomous Driving: A Survey". En: *IEEE Transactions on Intelligent Transportation Systems* 23.6 (2022), págs. 4909-4926. DOI: 10.1109/TITS.2021.3054625.

[5] Society of Automotive Engineers (SAE). *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. Inf. téc. $J3016_{2}01806$. SAE International, 2018. URL: https://www.sae.org/standards/content/j3016_201806/.

[6] Pritam Kore y Suchitra Khoje. "Obstacle Detection for Auto-Driving Using Convolutional Neural Network: ICDECT 2017". En: ene. de 2019, págs. 269-278. ISBN: 978-981-13-1609-8. DOI: 10.1007/978-981-13-1610-4_28.

[7] Jeff Sutherland. *Scrum: The Art of Doing Twice the Work in Half the Time*. Libro. Nueva York, EE. UU., 2014.

[8] Marco Wiering y Martijn Van Otterlo. *Reinforcement Learning: State of the Art*. English. Springer, 2012. ISBN: 978-3-642-27644-6. DOI: 10.1007/978-3-642-27645-3.

[9] Christopher J. C. H. Watkins y Peter Dayan. "Q-Learning". En: *Machine Learning* 8 (1992), págs. 279-292. DOI: 10.1007/BF00992698.

[10] Sheng-Lei Chen et al. "Multi-Step Truncated Q Learning Algorithm". En: *2005 International Conference on Machine Learning and Cybernetics*. Vol. 1. 2005, págs. 194-198. DOI: 10.1109/ICMLC.2005.1526943.

[11] Hado Hasselt. "Double Q-learning". En: *Advances in Neural Information Processing Systems*. Ed. por J. Lafferty et al. Vol. 23. Curran Associates, Inc., 2010. URL: https://proceedings.neurips.cc/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf.

[12] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". En: *Nature* 518 (2015), págs. 529-533.

[13] Hado Van Hasselt, Arthur Guez y David Silver. "Deep reinforcement learning with double Q-learning". En: *Proceedings of the AAAI Conference on Artificial Intelligence* 30.1 (2016). DOI: 10.1609/aaai.v30i1.10295.

[14] Arun Nair et al. "Massively Parallel Methods for Deep Reinforcement Learning". En: (jul. de 2015).

[15] Xiaogang Ruan et al. "Mobile Robot Navigation based on Deep Reinforcement Learning". En: *2019 Chinese Control And Decision Conference (CCDC)*. 2019, págs. 6174-6178. DOI: 10.1109/CCDC.2019.8832393.

[16] Guilherme Cano Lopes et al. "Intelligent Control of a Quadrotor with Proximal Policy Optimization Reinforcement Learning". En: *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*. 2018, págs. 503-508. DOI: 10.1109/LARS/SBR/WRE.2018.00094.

[17] Jemin Hwangbo et al. "Control of a Quadrotor With Reinforcement Learning". En: *IEEE Robotics and Automation Letters* PP (jun. de 2017), págs. 1-1. DOI: 10.1109/LRA.2017.2720851.

[18] Yuansheng Dong y Xingjie Zou. "Mobile Robot Path Planning Based on Improved DDPG Reinforcement Learning Algorithm". En: *2020 IEEE 11th International Conference on Software Engineering and Service Science (ICSESS)*. 2020, págs. 52-56. DOI: 10.1109/ICSESS49938.2020.9237641.

[19] Sepehr Saadatmand et al. "Autonomous Control of a Line Follower Robot Using a Q-Learning Controller". En: *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*. 2020, págs. 0556-0561. DOI: 10.1109/CCWC47524.2020.9031160.

[20] Xi Xiong et al. *Combining deep reinforcement learning and safety based control for autonomous driving*. Dic. de 2016. URL: https://arxiv.org/abs/1612.00147.

[21] Jacob McCalip, Mandil Pradhan y Kecheng Yang. "Reinforcement Learning Approaches for Racing and Object Avoidance on AWS DeepRacer". En: *2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC)*. 2023, págs. 958-961. DOI: 10.1109/COMPSAC57700.2023.00129.

[22] Wenjie Zhu et al. "Application of Reinforcement Learning in the Autonomous Driving Platform of the DeepRacer". En: *2022 41st Chinese Control Conference (CCC)*. 2022, págs. 5345-5352. DOI: 10.23919/CCC55666.2022.9902325.

[23] Junyao Li, Mohamed Abusharkh y Yong Xu. "DeepRacer Model Training for autonomous vehicles on AWS EC2". En: *2022 International Telecommunications Conference (ITC-Egypt)*. 2022, págs. 1-5. DOI: 10.1109/ITC-Egypt55520.2022.9855675.

[24] Amazon Web Services. *Amazon Elastic Compute Cloud (EC2)*. 2023. URL: https://aws.amazon.com/ec2/ (visitado 22-09-2023).

[25] Ahmad Sallab et al. "Deep Reinforcement Learning framework for Autonomous Driving". En: *Electronic Imaging* 2017 (ene. de 2017), págs. 70-76. DOI: 10.2352/ISSN.2470-1173.2017.19.AVM-023.

[26] Praveen Palanisamy. *Multi-Agent Connected Autonomous Driving using Deep Reinforcement Learning*. 2019. arXiv: 1911.04175 [cs.LG].

[27] Shai Shalev-Shwartz, Shaked Shammah y Amnon Shashua. *Safe, multi-agent, reinforcement learning for autonomous driving*. Oct. de 2016. URL: https://arxiv.org/abs/1610.03295.

[28] Alexey Dosovitskiy et al. "CARLA: An Open Urban Driving Simulator". En: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017, págs. 1-16.

[29] CARLA Documentation. *CARLA Introduction*. Consultado el 4 de Septiembre de 2023. URL: https://carla.readthedocs.io/en/latest/start_introduction/.

[30] Gazebo Development Team. *Gazebo Simulator*. Recuperado el 4 de Septiembre de 2023. URL: https://gazebosim.org/home.

[31] Python Software Foundation. *Python*. Recuperado el 4 de Septiembre de 2023. URL: https://www.python.org/.

[32] NumPy Contributors. *NumPy: A fundamental package for scientific computing with Python*. Recuperado el 4 de Septiembre de 2023. URL: https://numpy.org/.

[33] OpenCV Contributors. *OpenCV: Open Source Computer Vision Library*. Recuperado el 4 de Septiembre de 2023. URL: https://opencv.org/.

[34] PyTorch Contributors. *PyTorch: An open source machine learning framework*. Recuperado el 4 de Septiembre de 2023. URL: https://pytorch.org/.

[35] NVIDIA Corporation. *NVIDIA CUDA Toolkit*. Recuperado el 5 de Septiembre de 2023. URL: https://developer.nvidia.com/cuda-toolkit.

[36] *CARLA Documentation - Synchrony and Timestep*. URL: https://carla.readthedocs.io/en/0.9.13/adv_synchrony_timestep/.

[37] Andrew Howard et al. "Searching for MobileNetV3". En: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. Oct. de 2019.

[38] Jonathan Long, Evan Shelhamer y Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". En: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015. URL: https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Long_Fully_Convolutional_Networks_2015_CVPR_paper.pdf.

[39] Thomas Fermi. *Lane Detection for Carla Driving Simulator*. Kaggle Dataset. 2020. URL: https://www.kaggle.com/datasets/thomasfermi/lane-detection-for-carla-driving-simulator?resource=download.