Universidad
Rey Juan Carlos

Máster universitario en Visión Artificial

Conducción autónoma en tráfico usando aprendizaje profundo extremo a extremo

Memoria del Trabajo Fin de Máster

en Visión Artificial

Autor:

Enrique Y. Shinohara Soto

Tutores: Sergio Paniego, José M. Cañas

Julio 2023

Resumen

En este proyecto se presenta un enfoque de aprendizaje profundo basado en visión artificial de extremo a extremo, para que un coche autónomo conduzca autónomamente en condiciones de tráfico. En este contexto, el coche debe tanto mantenerse dentro del carril como mantener una distancia de seguridad con el vehículo delantero, si lo hubiera. Se ha seguido un enfoque de aprendizaje por imitación, creando un conjunto de datos supervisados con el piloto automático del simulador CARLA en diferentes condiciones de tráfico que incluyen vehículos delanteros. Se ha desarrollado una variante del modelo PilotNet, con técnicas de *dropout* adicionales y la velocidad anterior como entrada, y se ha (re)entrenado con dicho conjunto de datos en condiciones de tráfico. Los resultados experimentales muestran que el modelo conduce con éxito el coche en condiciones sin tráfico sin perder rendimiento. También, en condiciones con tráfico, es capaz de mantener la distancia de seguridad con los coches delanteros e incluso generaliza adecuadamente este comportamiento con diferentes vehículos delanteros.

A su vez, se ha creado una métrica para medir la distancia al vehículo delantero y se ha añadido a BehaviorMetrics, una herramienta de evaluación de conducciones autónomas de código abierto.

Abstract

In this proyect an end-to-end vision-based deep learning approach for an autonomous car to drive in traffic is presented. In this context, the car should both keep the lane and also keep a safety distance with the front vehicle, if any. The imitation learning approach has been followed, creating a supervised dataset with the autopilot of CARLA simulator in different traffic conditions which include front vehicles. A variant of the PilotNet model has been developed, with additional dropout and the previous speed as input, and it has been (re)trained with such in-traffic dataset. The experimental results show that the model successfully drives the car in traffic conditions without losing performance in free road conditions. It also keeps safety distance with the front cars and even properly generalizes to several types of front vehicles.

A plugin for measuring the distance to the front vehicle has been created and added to BehaviorMetrics, an open-source autonomous driving assessment tool.

Índice general

Índice de tablas

Índice de figuras

Nomenclatura

API	Application Programming Interfaces
CAGR	Compound annual growth rate
CNN	Convolutional Neural Network
CUDA	Compute Unified Device Architecture
DL	Deep Learning
DRL	Deep Reinforcement Learning
FPS	Frames per second
GPU	Graphics Processing Unit
IP	Internet Protocol
MPC	Model predictive control
MPD	Mean Position Deviation
NHTSA	National Highway Traffic Safety Administration
OA+FLM	Obstacle Avoidance + Follow-Lane Model
PID	Proportional Integral Derivative
POMDP	Partially observable Markov decision process
RGB	Red, Green, Blue
RL	Reinforcement Learning
SAE	Society of Automotive Engineers
TFM	Trabajo Fin de Máster
UAH	Universidad de Alcalá
WiFi	Wireless Fidelity

Capítulo 1

Introducción

Este TFM se encuadra dentro de la conducción autónoma guiada por visión usando tecnologías de aprendizaje automático de extremo a extremo.

En este capítulo introductorio se analizan los desafíos de la conducción autónoma en entornos de tráfico, destacando las mejoras tecnológicas y de seguridad que ofrece. Se comentan posibles enfoques para la programación de este tipo de aplicaciones, acompañados de algún ejemplo. Finalmente, se enumeran los principales objetivos de este trabajo fin de máster.

1.1. Conducción autónoma

Un vehículo autónomo se define como aquel que es capaz de captar, mediante el uso de sensores, su entorno y es capaz de moverse en él sin la ayuda de un ser humano. Entre los diversos sensores utilizados en los vehículos autónomos, la visión artificial desempeña un papel fundamental usando los datos visuales capturados por esos mismos sensores ópticos, como son las cámaras, para analizar y comprender el entorno.

Para proporcionar un mejor contexto sobre los vehículos autónomos, en 2014 la SAE Internacional (Sociedad de Ingenieros de Automoción) publicó una clasificación estándar que fue ampliamente adoptada, incluida por la NHTSA (Administración Nacional de Seguridad del Tráfico en Carreteras). Los niveles de autonomía descritos por SAE se detallan en el estándar J3016 [1], y son los siguientes:

1. Nivel 0 - Sin automatización de la conducción, el conductor controla el coche con sistemas de seguridad.

2. Nivel 1 - Asistencia al conductor, ayuda limitada en el control longitudinal o lateral.

3. Nivel 2 - Automatización parcial de la conducción, control longitudinal y lateral simultáneos, se requiere la supervisión del conductor.

4. Nivel 3 - Automatización condicional de la conducción, el vehículo se encarga de las operaciones rutinarias, el conductor debe estar preparado para intervenir.

5. Nivel 4 - Automatización avanzada de la conducción, el vehículo realiza todas las operaciones con un riesgo mínimo, el conductor es alertado para intervenir si es necesario.

6. Nivel 5 - Automatización completa de la conducción, el vehículo maneja todas las operaciones dinámicas sin necesidad de supervisión por parte del conductor.

En la actualidad, varias empresas están desarrollando e implementando vehículos inteligentes en el mundo real, incluso se pueden encontrar compañías automovilísticas que ya ofrecen servicios de taxis autónomos en diferentes ciudades [2][3][4].

Waymo anunció en 2020 que su vehículo ya llevaba conducidos más de 20 millones de millas por 25 ciudades de Norte América (unos 30 millones de kilómetros), y a su vez, en simulaciones había alcanzado más de 10 billones de millas (10 mil millones de millas) en datos [5]. Aunque estos vehículos se enfocaban en conducir en las mismas ciudades, se afirmó que esta cantidad de datos logró una mejora notable en la capacidad de respuesta de sus algoritmos en diferentes áreas a las usualmente recorridas.

En 2018, Tesla alcanzaba a reunir unos 9 mil millones de kilómetros recorridos en datos, por los vehículos vendidos desde 2014, todo esto con el autopiloto encendido [6]. Siendo su sensor principal, una cámara y mediante el uso de la visión artificial, lo que la convierte en una de las pocas empresas con una red neuronal tan grande y cuya funcionalidad tiene un impacto directo en las vidas humanas.

Y a pesar de la cantidad de investigación y desarrollo acumulados durante años, esta tecnología no se libra de los posibles problemas que acarrean, siendo una prioridad el solventarlos, pues estos errores afectan a la seguridad de los ciudadanos. En 2018, un vehículo autónomo de Uber fue el responsable de la muerte de un peatón en Estados Unidos [7]. De esta forma, se registra el primer caso de un individuo cuya muerte fue causada por un vehículo autónomo directamente.

Pero aun sabiendo la dificultad que conlleva traer esta tecnología y ofrecerla a la sociedad, siguen siendo muchas las razones por las que merece la pena invertir en ellas. Cada vez hay más empresas que se unen al desarrollo de vehículos autónomos, lo cual lleva a invertir más en la investigación y el desarrollo de este tipo de tecnologías. Como se puede ver en la Figura 1, la gráfica muestra que las empresas toman una postura más agresiva con el fin de asegurar sus invenciones por medio de la publicación de patentes.

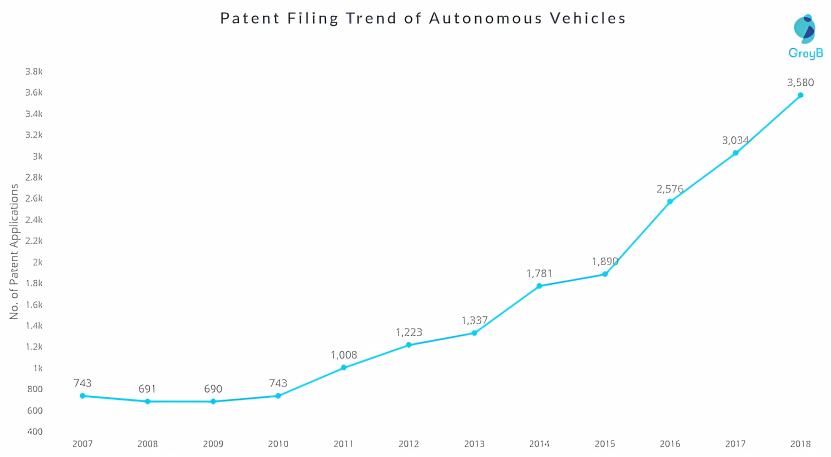


Figura 1: Evolución del número de patentes en el campo de los vehículos autónomos [8].

El desarrollo de las tecnologías de conducción autónoma ha experimentado mejoras significativas y se ha convertido en un foco principal de investigación y desarrollo en los últimos años, hasta tal punto que se prevé una tasa de crecimiento anual compuesto, o CAGR, superior al 40 % desde 2020 hasta 2030 [9] (ver Figura 2). Este campo tan vanguardista se caracteriza por numerosos avances que tienen como objetivo revolucionar el modo de transporte actual, ya sea en términos de localización, navegación, detección, etc. [10], [11]. Los progresos en el desarrollo de vehículos autónomos son altamente demandados a medida que la sociedad avanza.

La creación de este tipo de tecnologías abre la puerta a una forma de viajar mucho más segura, reduciendo no solo el número de accidentes en la carretera, sino también brindando a los usuarios un entorno mucho más cómodo al evitar los problemas de quedar atrapados en el tráfico.

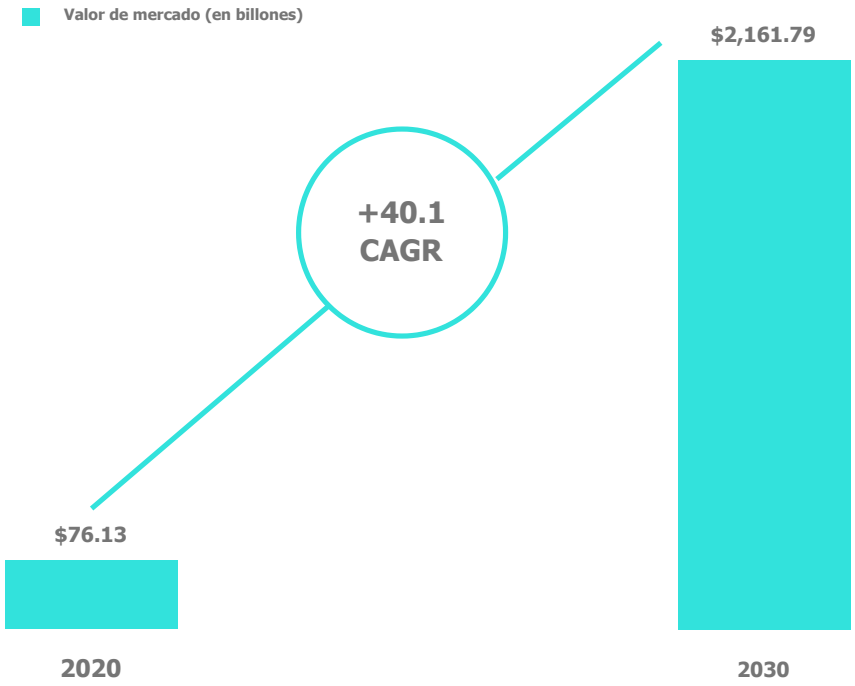


Figura 2: Estadísticas de mercado, en billones (mil millones), de los vehículos autónomos hasta 2030 [8].

En general, es común encontrar dos enfoques distintos para programar las aplicaciones de conducción autónoma. Uno de ellos el *enfoque modular*, que ensambla varios componentes más pequeños e independientes (módulos) en un producto final único. En el contexto de la conducción autónoma, se puede tener un módulo de detección de carriles que garantice el control seguro de los vehículos autónomos. Al detectar y rastrear con precisión los carriles, estos vehículos pueden navegar eficazmente a través de redes viales complejas, minimizando el riesgo de accidentes o colisiones. También se pueden crear módulos para detectar elementos clave en nuestro entorno, como son los semáforos (para detenernos o avanzar según el color de este), peatones u otros vehículos (para priorizar la seguridad tanto de ellos como de nosotros y detenernos a tiempo si se detecta una posible colisión). Además de los módulos de percepción, también sería necesario implementar software de

toma de decisiones como el control del vehículo, la navegación local o la planificación de rutas.

Por otro lado, existe otra aproximación conocida como el *enfoque extremo a extremo*, que asigna la tarea de detección, seguimiento, planificación de rutas, control, etc., a un modelo de aprendizaje automático. Este modelo será el único responsable de manejar todas las entradas, interpretarlas y analizarlas, tomando decisiones informadas para controlar el vehículo de la manera deseada. Para este proyecto en particular, se decidió emplear este enfoque haciendo uso del aprendizaje profundo y el aprendizaje por imitación, el cual se hablará con más detalle en el capítulo 2.

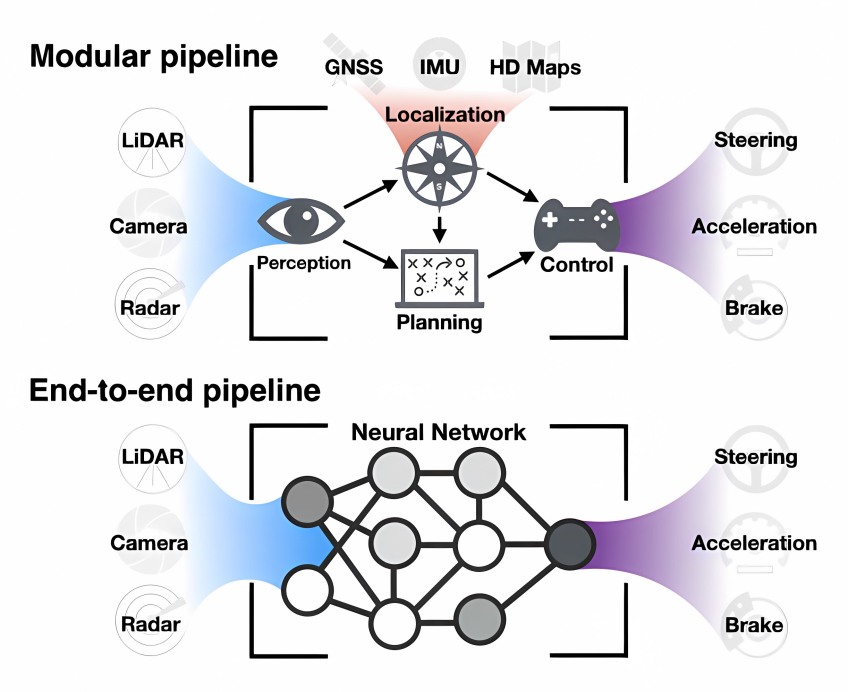


Figura 3: Enfoque modular y extremo a extremo para la navegación autónoma de un vehículo [12]

Al empujar los límites de la tecnología de conducción autónoma, se contribuyen a los avances continuos y al futuro del transporte, siendo el objetivo final que los vehículos autónomos sean una realidad común y brinden beneficios significativos en términos de seguridad y comodidad para las personas.

1.2. Objetivos

El objetivo principal de este trabajo es programar usando aprendizaje profundo a un vehículo autónomo equipado con una cámara, de modo que conduzca respetando los límites de los carriles y que sea capaz de detenerse y proseguir la marcha en caso de que haya otros vehículos en frente suyo. Para ello, se empleará un enfoque de extremo a extremo utilizando el aprendizaje por imitación, con el propósito de generar comandos de control basados en la entrada sensorial capturada por una cámara frontal en tiempo real.

Este objetivo general se ha articulado en cuatro subobjetivos concretos:

1. Configurar el entorno donde mover al vehículo autónomo, evaluando la viabilidad de la creación de un vehículo que siga un recorrido programado. Esto se hará primero dominando el simulador de CARLA para saber cómo moverse en el entorno y aprendiendo el manejo del autopiloto afinando sus respectivos parámetros, llegando también a configurar el tráfico para diferentes ciudades de ejemplo.

2. Generar un conjunto de datos supervisados que reflejen situaciones de manejo con tráfico y sin tráfico utilizando el simulador CARLA. Estos datos deben incluir imágenes capturadas por la cámara a bordo del vehículo, así como los comandos de control generados por un piloto experto automático.

3. Implementar un modelo de aprendizaje profundo que se encargará de la percepción del entorno y de la toma de decisiones basadas en la información sensorial capturada. Se elegirá una arquitectura neuronal, la cual se reentrenará adecuadamente utilizando el conjunto de datos supervisados generado en el subobjetivo anterior, para realizar el seguimiento de carriles en condiciones sin y especialmente con tráfico.

4. Validar experimentalmente el sistema propuesto utilizando el simulador CARLA junto con la herramienta BehaviorMetrics [13]. Se realizarán pruebas exhaustivas en un entorno urbano con y sin tráfico, evaluando el rendimiento del vehículo autónomo para comprobar si está manteniendo la distancia de seguridad, si es capaz de frenar y reanudar la marcha de manera segura en función de lo que esté haciendo el vehículo delantero, y su capacidad para evitar colisiones. Se utilizarán métricas objetivas cuantitativas que nos faciliten medir y evaluar este comportamiento.

1.3. Estructura de la presente memoria

La memoria del trabajo fin de máster se estructura en los siguientes capítulos:

1. Capítulo 1: Introducción - Se proporciona una visión general del proyecto, su motivación y los objetivos que se pretenden lograr.

2. Capítulo 2: Estado del arte - Se sitúa al proyecto en el contexto de trabajos similares existentes en el campo.

3. Capítulo 3: Herramientas utilizadas - Se detallan las herramientas empleadas en el desarrollo del trabajo.

4. Capítulo 4: Aprendizaje por imitación para una conducción en tráfico - Se describe el desarrollo con aprendizaje por imitación de cómo manejarse en situaciones de tráfico.

5. Capítulo 5: Evaluación y experimentos - Se presentan los estudios realizados y los resultados experimentales obtenidos.

6. Capítulo 6: Conclusiones - Se extraen las conclusiones del trabajo realizado hasta el momento y se plantean posibles líneas futuras del proyecto.

Capítulo 2

Estado del arte

En este capítulo se describe el estado actual de las técnicas de aprendizaje automático en el contexto de la conducción autónoma. A su vez, se exploran conceptos basados en tecnologías que abarcan el enfoque de extremo a extremo y modular para la programación de estos sistemas, así como las simulaciones y el conjunto de datos disponibles para resolver el desafío de los vehículos autónomos.

2.1. Sistemas modulares de conducción autónoma

El enfoque de un sistema modular para la conducción autónoma consiste en un diseño y desarrollo en la cual se construye el sistema usando varios módulos o componentes independientes, cada uno encargado de realizar tareas específicas relacionadas con la conducción. Estos módulos se diseñan para que sean flexibles y se comunican entre sí de forma que puedan compartir información y coordinarse para llegar a una solución. Cada módulo puede cubrir funciones como la percepción del entorno, la planificación de rutas, el control del vehículo o el reconocimiento de señales de tráfico, entre otras.

En 2017 empieza a formarse un proyecto bajo el nombre BRAVE [14], liderado por la Universidad de Alcalá. Este proyecto surge de la idea de encontrar dificultades a la hora de introducir vehículos autónomos en la sociedad y aportar posibles soluciones para hacer esto posible. De aquí que se considere este proyecto como un enfoque modular, pues sus investigaciones varían desde análisis de intersecciones, detectores de velocidad basados en visión, predicción de intenciones aplicadas a vehículos y muchas más. El vehículo autónomo desarrollado por la UAH denominado DRIVETIVE [15] hace uso justo de este enfoque modular combinando diferentes técnicas preexistentes para el control, estimación de estado, fusión de datos, comunicación y degradación de datos.

En la Figura 4 se pueden ver algunos de los muchos módulos que se han usado para este vehículo autónomo. La primera imagen en la esquina superior izquierda muestra un controlador MPC que controla la dirección del vehículo para tomar la ruta de la mejor forma posible. La imagen superior derecha está más enfocada a la percepción. Con la imagen captada por la cámara frontal del vehículo, lo que se consigue es detectar usuarios vulnerables en la carretera, para así afrontar la situación de forma más segura y controlada. Por último, la imagen inferior consiste en otro módulo centrado en la percepción, más específicamente en la segmentación. Lo que se consigue es identificar la carretera, es decir, el área donde el vehículo puede y debe conducir.

Figura 4: Ejemplos de algunos de los módulos del vehículo autónomo DRIVETIVE.

En el ámbito de la conducción autónoma, otro tema muy abordado es la planificación de la trayectoria, la cual trata de proveer soluciones óptimas que permitan a los vehículos establecer rutas seguras y eficientes desde un estado inicial hasta un estado final. No lejos de la planificación se pueden encontrar trabajos centrados en la creación de controladores, como es el caso de Sotelo [16]. En este proyecto se desarrolló un controlador de bajo nivel utilizando un POMDP basado en observaciones de WiFi. Se logró implementar un sistema de navegación autónoma robusto en entornos interiores, que permitía recuperar la posición del robot a pesar de la incertidumbre de los sensores.

En otros campos más centrados en la parte de la percepción se encuentran proyectos como el realizado por Lo Bianco et al. [17] el cual proponía un enfoque para la detección conjunta de objetos en la carretera y carriles utilizando una red de segmentación semántica. Combinando diferentes fuentes de anotaciones y el entrenamiento con etiquetas débiles y fuertes, se generaba un nuevo conjunto de datos. El modelo resultante muestra un buen rendimiento general al identificar carriles y clasificar objetos en la carretera. Además, se destaca su capacidad de adaptación al dominio y su comportamiento en tiempo real (ver Figura 5).

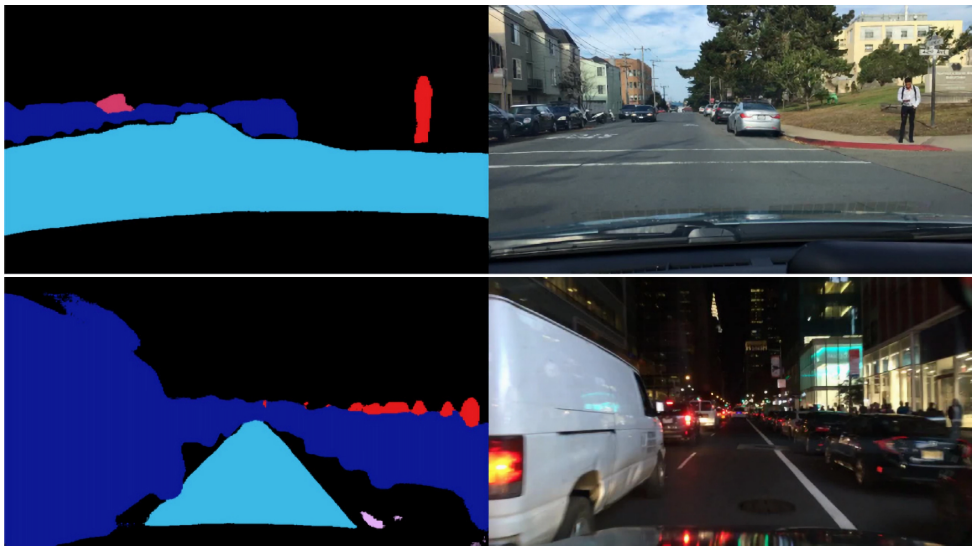


Figura 5: Segmentación del modelo multitarea [17].

Incluso se llegan a ver trabajos de segmentación enfocados al uso las cámaras ojo de pez. Destacadas por su campo de visión más amplio y su capacidad para capturar información de distancia y profundidad de manera más precisa, se han vuelto una herramienta muy útil en la visión artificial. En 2019, Saéz [18] proponía el uso de dos arquitecturas, ERFNet y ERFNetPSP las cuales lograban mejores resultados que los trabajos más avanzados en un conjunto de datos sintéticos. Además, demostraron la capacidad de ERFNetPSP para manejar la distorsión presente en las cámaras ojos de pez (ver Figura 6).

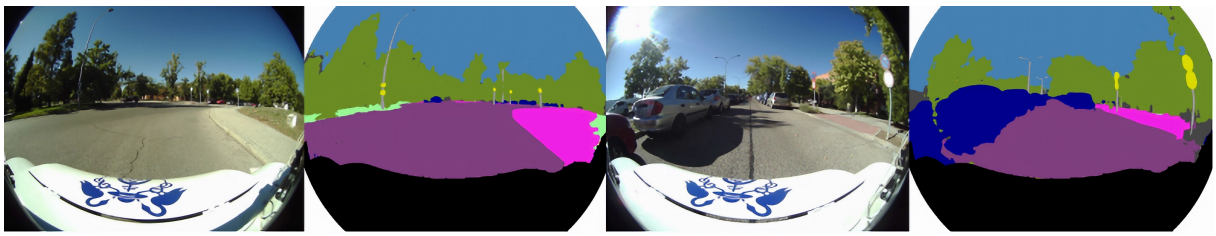


Figura 6: Segmentación de una imagen ojo de pez [18].

2.2. Sistemas extremo a extremo en conducción autonónoma

Un sistema de conducción autónoma extremo a extremo tiene un enfoque más integrado. Un único sistema de aprendizaje automático será el encargado de procesar la información sensorial y tomar decisiones de conducción sin depender de módulos separados. El sistema aprende directamente de los datos recopilados durante la conducción y será el principal responsable de todas las etapas del proceso, desde la percepción hasta la acción (ver Figura 3).

En 2018, Codevilla [19] (ver Figura 7) muestra un ejemplo muy bueno del funcionamiento de un sistema de aprendizaje de extremo a extremo. En este artículo, se entrenó un

vehículo autónomo usando el enfoque extremo a extremo, donde tanto la percepción como la toma de decisiones las hacía una arquitectura de redes neuronales. Lo interesante de la propuesta también era la capacidad de controlar al vehículo cuando esté en la etapa de evaluación, es decir, poder introducir comandos desde fuera y que la red neuronal pueda captar estos comandos y mover al vehículo consecuentemente.

Figura 7: Imágenes de la simulación y la vida real del vehículo entrenado de extremo a extremo [19].

Para que funcione la conducción autónoma en un entorno como son las carreteras siguiendo unas normas de circulación establecidas, necesita de una serie de sistemas que puedan trabajar en tiempo real. Para ello es necesario hacer uso de sensores que localicen tanto al propio coche como a su entorno, y un sistema de control que pueda procesar la información captada por los sensores para controlar al vehículo. Para tratar la gran cantidad de posibles estados a los que se enfrenta el vehículo se emplean algoritmos de aprendizaje automático.

Existen varios enfoques que pueden utilizarse en el planteamiento extremo a extremo de un sistema, cada uno de los cuales desempeña un papel crucial en la definición del alcance y las capacidades de la conducción autónoma. Investigaciones en el aprendizaje por refuerzo y el aprendizaje profundo han logrado enormes avances al superar los límites de sus respectivos campos, y hay muchos proyectos exitosos en estos temas. Todos ellos han mostrado mejoras significativas en el uso de técnicas avanzadas, ya sea creando agentes de juego que puedan competir con humanos o mejorando la conducción autónoma [20] [21]. Algunos de estos ejemplos en los que se basan los artículos anteriores, también se pueden ver en la Figura 8.

Figura 8: Izq.: Un agente entrenado con aprendizaje por refuerzo logró una puntuación superior a la capacidad humana en los 57 juegos de Atari 2600. [22] Dcha.: AVOD, un algoritmo de detección de objetos, fue capaz de obtener uno de los mejores resultados en la prueba de detección de objetos 3D KITTI. [21]

En el contexto de un enfoque de extremo a extremo, algunas de las áreas dentro del aprendizaje automático que destacan son las siguientes:

Aprendizaje por refuerzo (RL): como se ve en la Figura 9, mediante la interacción con el entorno y la retroalimentación en forma de recompensas o penalizaciones, el RL busca enseñar a un agente a tomar decisiones. Aprendiendo de los errores y adaptando los comportamientos en respuesta a las recompensas vistas, el RL puede utilizarse para mejorar las normas de conducción en el tráfico.

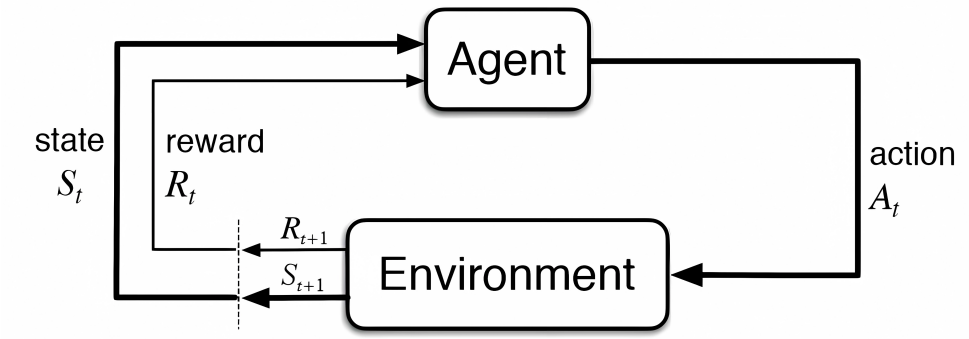


Figura 9: Diagrama base del aprendizaje por refuerzo

Aprendizaje profundo (DL): este método se centra en el entrenamiento de redes neuronales con múltiples capas para aprender y extraer características significativas a partir de datos brutos de sensores. Un ejemplo del enfoque de utilizar un modelo de aprendizaje profundo de extremo a extremo para dirigir un vehículo puede verse en un trabajo realizado por NVIDIA, el modelo PilotNet [23]. En este trabajo pretendían evitar la necesidad de reconocer las características urbanas como los carriles, las carreteras u otros coches.

Desarrollaron, con el uso de una red CNN, un algoritmo capaz de extraer las características útiles de la carretera, tal y como se ve en la Figura 10, para predecir el ángulo de giro correcto para cada situación, sin tener que enseñarlo explícitamente.

Figura 10: Detección de características por el modelo PilotNet

Aprendizaje profundo por refuerzo (DRL): al utilizar redes neuronales profundas como aproximadores de funciones dentro del marco de RL, DRL mezcla las ideas de RL y DL. A través de interacciones con el entorno, el agente puede aprender de principio a fin, mapeando directamente los datos de los sensores de entrada a las acciones de conducción y mejorando el rendimiento. En esta investigación [24], la idea de utilizar un enfoque de aprendizaje de refuerzo profundo se demostró fructífera en situaciones complejas como las intersecciones. Fuera del ámbito de la conducción, un proyecto muy interesante se puede encontrar en el exitoso AlphaGo [25], que fue capaz de vencer a un jugador profesional al juego de mesa Go, haciendo uso de técnicas de DRL.

2.2.1. Aprendizaje por imitación

Dentro del ámbito del aprendizaje por imitación podemos encontrar una amplia variedad de algoritmos, cada uno con sus características y ventajas. La base de este tipo de algoritmos se encuentra en una técnica conocida como *behavioral cloning* o clonación del comportamiento [26]. Este método se emplea en robótica y aprendizaje automático para enseñar a un agente o sistema a imitar un comportamiento deseado aprendiendo de ejemplos. El modelo aprende a imitar las actividades o comportamientos de una demostración experta.

Se pueden ver algunos ejemplos interesantes en este ámbito, como en un juego de disparos conocido como Counter Strike, en el que el agente aprendió a jugar como un jugador ocasional, navegando, identificando, disparando y recargando cuando era necesario [27]. Este experto se entrenó utilizando muchos datos de juego humano y algunos datos de expertos de alta calidad.

El problema del aprendizaje por imitación en conducción autónoma aparece cuando el agente entrenado se encuentra con situaciones que no se le han mostrado durante el entrenamiento. En esas situaciones nunca vistas, es difícil que cree un comportamiento que sea satisfactorio (ver Figura 11).

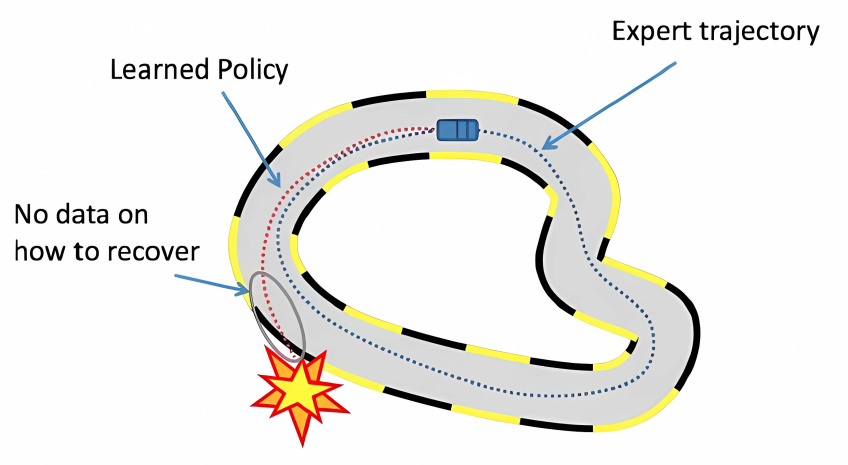


Figura 11: Representación visual de un agente que falla cuando entra en una zona no visitada por el experto. [28]

El algoritmo básico de aprendizaje por imitación, a pesar de estos errores, ofrece una técnica simple y eficiente para muchas de las tareas que se quieran resolver, siempre que estos errores no causen graves consecuencias. Una de las formas para evitar este tipo de comportamientos reside en las demostraciones del experto. Cuanto más completas sean y mejor cubran lo que se quiere enseñar, mejores resultados se pueden obtener, pero rápidamente se puede llegar a la conclusión de la dificultad que conlleva el obtener y hacer uso de una cantidad enorme de datos, y es que a medida que se van creando nuevas propuestas de algoritmos, estos traen consigo mejoras en los algoritmos base haciéndolos más robustos ante este tipo de problemas, quitando la necesidad de tener que disponer de una cantidad inabarcable de datos.

En las tecnologías más avanzadas, es habitual combinar el aprendizaje por imitación [19] con otras técnicas, como el aprendizaje por refuerzo que mencionamos anteriormente, creando sistemas híbridos capaces de manejar tareas complejas. Ejemplos notables de este tipo de algoritmos son:

1. IQ-Learning [29]: originario de la Universidad de Stanford, este algoritmo se centra inicialmente en imitar el comportamiento de un experto. Aprendiendo de las acciones del experto, el agente intenta replicar su comportamiento. Una vez que el agente ha imitado con éxito al experto, el algoritmo emplea principios de aprendizaje por refuerzo. Concede al agente la libertad de explorar e investigar sus propias soluciones a estados previamente no visitados, mejorando así sus capacidades más allá de la mera imitación.

2. Aprendizaje por refuerzo inverso [30]: en este enfoque, el objetivo principal es aprender la función de recompensa con la ayuda de un experto. El experto desempeña un papel fundamental en la optimización de la función de recompensa creada por

el agente basándose en su comportamiento. A través de este proceso iterativo, el aprendizaje del agente es guiado y refinado, lo que le permite adaptarse y mejorar su rendimiento a lo largo del tiempo.

2.3. Simuladores de conducción autónoma

En enfoques de aprendizaje por imitación es realmente importante describir de dónde se recogen los datos de entrenamiento, especialmente en el ámbito de la conducción autónoma. Podemos recogerlos tanto del mundo real como de datos sintéticos recogidos en una simulación. Compañías como Tesla, recogen una cantidad inmensa de datos brutos del mundo real gracias a su flota de pruebas de conducción autónoma completa, pero también es esencial valorar los datos sintéticos generados a partir de simulaciones, ya que no siempre es fácil acceder a un conjunto de datos tan grande. Se pueden encontrar empresas que ya ofrecen generar datos sintéticos [31] o bien ofrecen sus simuladores que permiten recolectar datos sintéticos eligiendo las situaciones a las que queremos que se enfrente nuestro vehículo. Algunos de estos simuladores usados en la conducción autónoma son:

1. SUMO [32], un software de simulación de tráfico de código abierto utilizado para visualizar e imitar situaciones de tráfico realistas. Permite evaluar la eficiencia del tráfico, la congestión y la eficacia de las estrategias de gestión del tráfico, así como crear redes viarias detalladas que tienen en cuenta la demanda de tráfico y el comportamiento de los vehículos. Una de las principales fortalezas de SUMO es su capacidad para simular grandes volúmenes de tráfico de manera eficiente. De esta forma, los usuarios son capaces de evaluar el rendimiento del sistema de transporte en diferentes condiciones, así como probar agentes que empleen una conducción autónoma, lo que resulta valioso para la implementación de políticas de gestión del tráfico más efectivas (ver Figura 12).

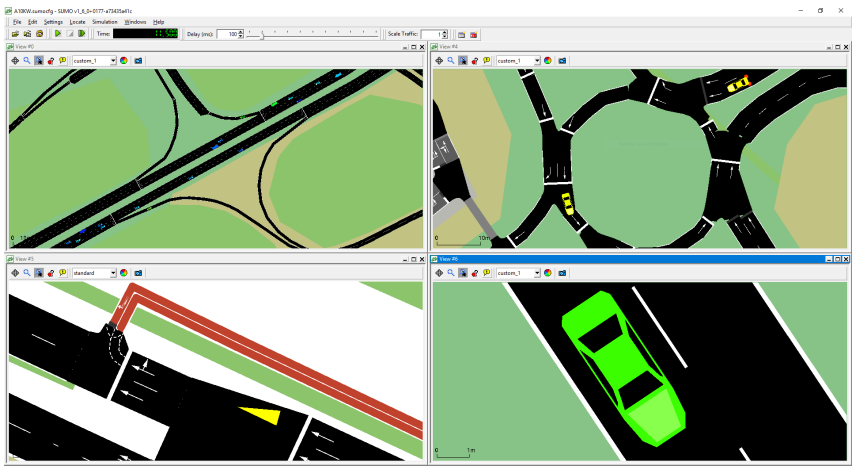


Figura 12: Aplicación del simulador de tráfico, SUMO. [32]

2. Waymo creó su propia herramienta de simulación para recopilar datos sintéticos, llamada SimulationCity [33]. Este simulador destaca por su alta fidelidad y capacidad para recrear escenarios de conducción muy realistas. Con la posibilidad de

generar datos sintéticos en entornos controlados, Waymo puede probar y mejorar el comportamiento de sus vehículos autónomos en una amplia gama de situaciones, desde las habituales hasta las más complicadas y menos frecuentes. Este simulador ofrece pruebas escalables y seguras, permitiendo a Waymo iterar rápidamente y entrenar sus modelos de aprendizaje automático para mejorar la seguridad y eficiencia de sus vehículos autónomos (ver Figura 13).

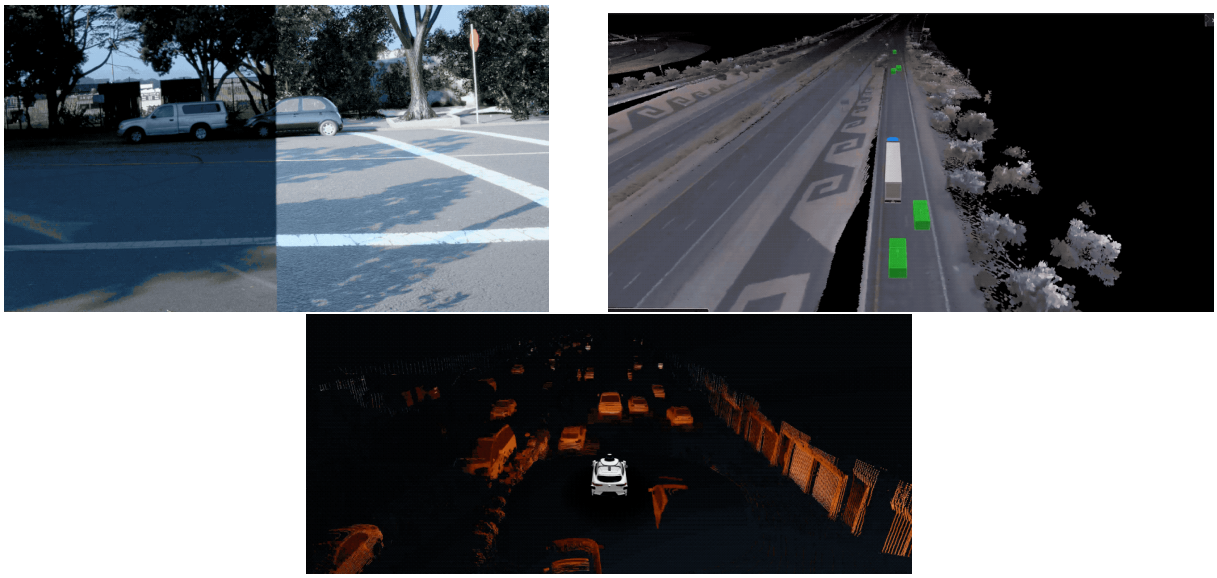


Figura 13: Simulador SimulationCity creado por Waymo. Mostrando el uso de sensores como el LIDAR, situaciones de tráfico con diferentes vehículos y un ejemplo de una cámara RGB frontal (arriba-izquierda) donde el lado izquierdo es el mundo real y el derecho el mundo simulado.

3. Wayve Infinity Simulator [34], un simulador creado por la empresa Wayve cuyo objetivo se centra en conseguir crear vehículos autónomos sin la necesidad de optar por el enfoque modular del que tanto se dependió para llegar a la tecnología actual. Si bien este último enfoque ha mostrado mejoras notables, pues ayudan a resolver problemas específicos en la conducción, Wayve defiende que tienen un problema de escalabilidad [35], esto es, que no son capaces de generalizar a nuevas situaciones nunca vistas. Por ello, optan por usar el enfoque extremo a extremo, con el fin de poder crear vehículos capaces de generalizar ante situaciones complejas, evitando la fragilidad de las reglas creadas por cada módulo. Este simulador se diseñó con la idea de entrenar y evaluar vehículos autónomos que hayan hecho un aprendizaje de extremo a extremo, contando con un entorno realista, diverso, controlable y muy escalable (ver Figura 14).

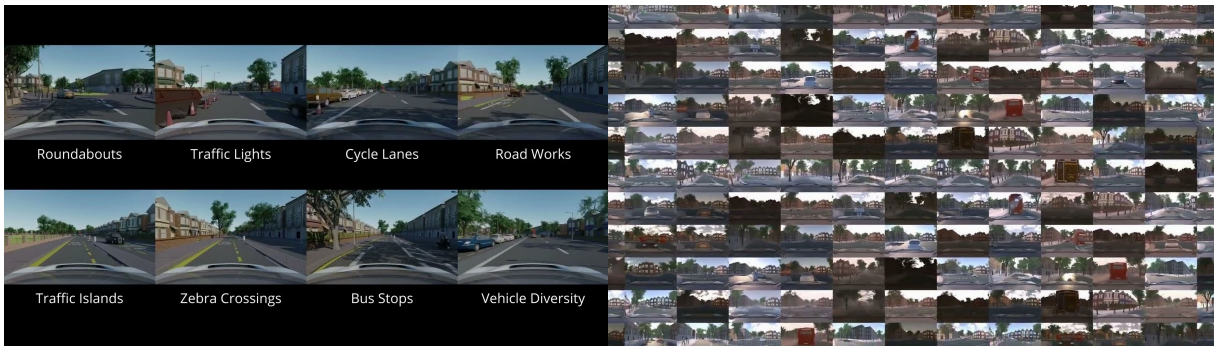


Figura 14: Simulador Wayve Infinity

4. CARLA Simulator [36], es otra herramienta de simulación de código abierto con capacidad para generar entornos virtuales altamente realistas, como escenarios urbanos y rurales, para permitir a los desarrolladores probar y validar algoritmos y sistemas de conducción autónoma en un entorno seguro y controlado. CARLA ofrece una amplia gama de sensores virtuales, como cámaras y LiDAR, y permite la interacción con vehículos, peatones y ciclistas simulados. Además, CARLA a su vez cuenta con una arquitectura modular y flexible que facilita la incorporación de nuevos modelos y algoritmos.

   Los foros de CARLA son una parte muy interesante y activa de la comunidad, pues ofrecen un lugar donde los usuarios y desarrolladores conversen intercambiando ideas, planteando preguntas, recibiendo soporte técnico y compartiendo soluciones. Los foros de CARLA fomentan la colaboración y el aprendizaje mutuo, lo que facilita a un desarrollo más sólido y una mejor comprensión de la conducción autónoma utilizando el simulador CARLA (ver Figura 15).

Figura 15: Entorno simulado de CARLA

2.4. Datasets de conducción autónoma

También es posible encontrar conjuntos de datos de percepción relevantes ya recopilados. Normalmente, cada dataset ofrece su propio conjunto de características y anotaciones específicas, pero en general, todos tienen el objetivo de proporcionar datos los más realistas y completos posibles para avanzar en el campo de la percepción, planificación y control de vehículos autónomos.

Los conjuntos de datos centrados en la planificación y el control del vehículo son fundamentales en el campo de la conducción autónoma, ya que proporcionan ejemplos y datos relevantes para entrenar y evaluar algoritmos de planificación y control en entornos de conducción realistas. Siendo este tipo de datasets los que se usaron en el proyecto, a continuación se muestran ejemplos de datasets ya creados:

1. Waymo Open Dataset [37]: el Waymo Open Dataset proporciona una gran colección de datos obtenidos a partir de sensores, que incluyen nubes de puntos LiDAR, imágenes de cámaras y poses de vehículos, capturados por la flota de vehículos autónomos de Waymo, lo que lo hace adecuado para tareas de planificación y control.

2. ApolloScape [38]: además de ser un dataset conocido por sus datos en percepción, el conjunto de datos de ApolloScape también incluye información valiosa para la planificación y el control. Ofrece mapas de alta definición, datos de sensores y anotaciones para diversas tareas, como las trayectorias, la predicción de comportamientos y la toma de decisiones.

Estos conjuntos de datos, junto con otros, se han convertido en puntos de referencia para investigadores y desarrolladores en el campo de la conducción autónoma. Pero si bien gran parte de los datasets presentados anteriormente están más centrados en el apartado de la planificación y el control del vehículo, también es posible encontrar datasets centrados en ofrecer información de la percepción como la detección de objetos, el seguimiento o la segmentación.

KITTI [39], un dataset ampliamente utilizado y conocido en el campo de la conducción autónoma y la percepción del entorno. Generado por el Karlsruhe Institute of Technology, contiene datos de múltiples sensores, como cámaras RGB, LiDAR y GPS. El dataset incluye imágenes estéreo, datos de LiDAR en 3D y anotaciones detalladas, que van desde la detección y seguimiento de objetos, la segmentación semántica y hasta el flujo óptico. KITTI es un dataset muy completo enfocado en un entorno urbano, lo que lo hace valioso para el desarrollo y la evaluación de algoritmos de conducción autónoma en entornos del mundo real (ver Figura 16).

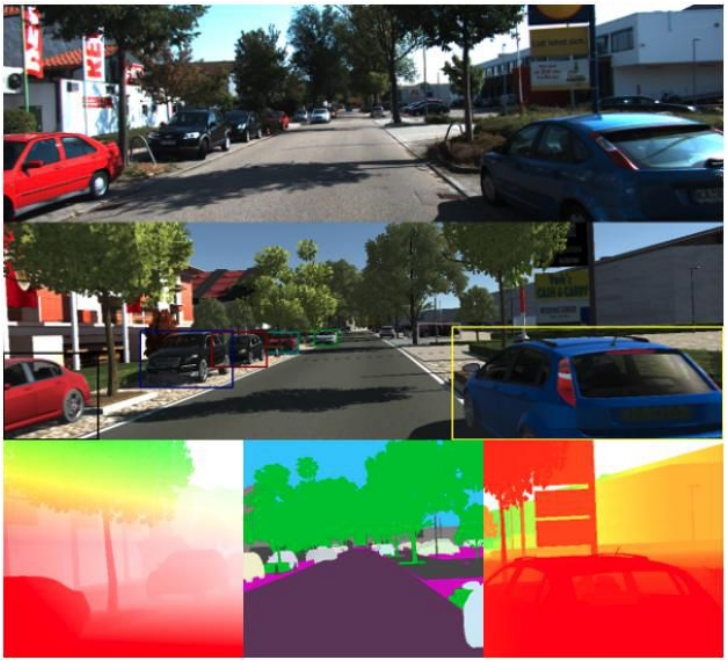


Figura 16: Ejemplos del dataset KITTI [39].

nuScenes [40] es otro dataset muy reconocido, el cual se enfoca en proporcionar datos para la conducción autónoma en entornos urbanos complejos. Recopilado por la empresa nuTonomy (ahora parte de Aptiv), contiene información de múltiples sensores, como cámaras, LiDAR, radares y odometría. Proporciona una amplia variedad de anotaciones, incluyendo detección y seguimiento de objetos, predicción de trayectorias, clasificación semántica y mapas de alta definición. Así como datos de diferentes ciudades y situaciones de tráfico muy variadas (ver Figura 17).

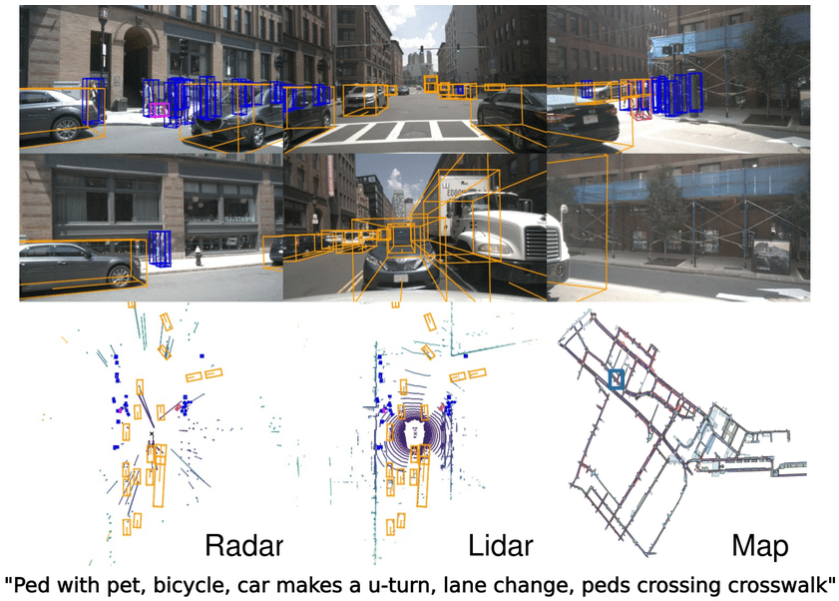


Figura 17: Ejemplos del dataset nuScenes [40].

Cityscapes [41], este dataset se centra en la compresión de imágenes de escenas urbanas por medio de sus imágenes semánticas. Contiene una gran cantidad de imágenes de alta resolución capturadas en diversas ciudades europeas. El dataset está anotado a nivel de píxel, lo que permite la identificación y clasificación precisa de objetos y regiones en la imagen. Además de la segmentación semántica, Cityscapes también proporciona anotaciones de instancias y de bordes (ver Figura 18).

Figura 18: Ejemplo de la anotación semántica del dataset Cityscapes [41].

Capítulo 3

Herramientas utilizadas

En esta sección se comentan los diferentes sistemas operativos usados, así como las herramientas principales para el desarrollo del proyecto.

3.1. Sistemas operativos

El desarrollo de este trabajo se estableció sobre la base de dos sistemas operativos Linux distintos. Inicialmente, el proyecto se comenzó en el sistema operativo Ubuntu en su versión 22.04, donde se utilizó el simulador de conducción autónoma CARLA en su versión 0.9.13. Sin embargo, a medida que avanzaba la investigación, surgieron necesidades adicionales. Era necesario incorporar ROS Noetic, una herramienta crucial para el correcto funcionamiento de la herramienta BehaviorMetrics, la cual tenía como objetivo evaluar los modelos entrenados.

Aquí surgió un problema y es que se descubrió que ROS Noetic tenía soporte oficial únicamente hasta la versión 20.04 del sistema operativo Linux. Ante esta situación, se tomó la decisión de instalar dicha versión en otro disco disponible. Al hacerlo, se pudo instalar ROS Noetic en conjunto con CARLA 0.9.13 en la versión de Ubuntu 20.04 del sistema operativo.

3.2. CARLA

El simulador CARLA es de las herramientas pilares del proyecto, pues es donde se recopilará información del entorno y donde se entrenará y evaluará el vehículo autónomo desarrollado. Para este trabajo de fin de máster se ha decidido usar la versión más reciente en el momento de empezarla, que es la 0.9.13, pues al estar en constante desarrollo, se pueden aprovechar todas las mejoras, nuevas funcionalidades y posibles arreglos ante fallos surgidos en versiones anteriores. CARLA es un simulador de código abierto el cual ofrece una amplia gama de mapas como los que se ven en la Figura 19.

Figura 19: Panorámicas de las ciudades 1, 2 y 4 entre otras, del simulador CARLA

Los mundos son los elementos esenciales de la simulación, y CARLA lleva años trabajando con el fin de convertirlos en entornos vivos, similares a la vida real, pero que, a su vez, dejen una gran variedad de posibles configuraciones, tanto de clima como de objetos o agentes, todo para ofrecer la posibilidad de crear escenarios a gusto del usuario desarrollador.

CARLA cuenta con una API muy flexible, que deja a los usuarios controlar en su totalidad la simulación. La base de este simulador se encuentra en dos conceptos, el cliente y el mundo. El mundo funciona como servidor al cual el cliente se debe conectar para o bien conseguir información del mundo, o bien mandar información para que este se actualice.

En primer lugar, hay que lanzar el servidor, que se hace ejecutando el guion de órdenes, también conocido como *shell script* de CARLA.

```
./CarlaUE4.sh -quality-level=Low
```

Una vez se tiene el servidor corriendo, en el trozo de código que se muestra a continuación se puede ver la simplicidad con la que CARLA ofrece esta conexión entre el cliente y el mundo. Básicamente, bastaría con establecer la conexión entre el cliente, que sería el fichero donde se lance el código, y el servidor. Ofreciendo la IP del servidor (*args.host*) junto con el puerto donde se haya lanzado (*args.port*), el cliente estará conectado con el mundo y podrá interactuar libremente con él.

```
client = carla.Client(args.host, args.port)
client.set_timeout(4.0)
```

Una vez visto como se conecta el cliente con el mundo, quedaría ver cómo se conecta el cerebro del vehículo autónomo, el cual contiene el modelo de aprendizaje automático para la toma de decisiones, con el propio vehículo que se encuentra dentro del simulador. Para ello primero se genera un agente, CARLA ofrece funciones ya implementadas que agilizan

mucho la interacción con el simulador, como se puede ver en el siguiente trozo de código donde se llama a una función *spawn_actor* el cual recibe como parámetros de entrada el plano del vehículo y la posición dentro del mapa donde se generará.

```
self.vehicle = self.world.spawn_actor(blueprint, spawn_point)
```

Ya se ha creado e identificado el vehículo, por lo que es posible establecer comunicación con este para recibir y mandar información. En el caso de este trabajo, tenemos una clase *RLAgent* que se atribuye a la variable *agent* la cual contiene el cerebro del vehículo autónomo. Este cerebro requiere como entrada principal la imagen captada por la cámara delantera y la velocidad actual del vehículo (para más detalle, dirigirse al capítulo Modelo neuronal). La lógica dentro del cerebro del agente se encarga de devolver los comandos necesarios en ese momento para que el vehículo sepa el siguiente paso que dar. Haciendo uso de otra función de CARLA llamada *apply_control*, se mandan los valores del giro, aceleración y frenado necesarios para que el vehículo conduzca de forma segura en el simulador.

```
steer, throttle, brake, image = agent.run_step(world.
    camera_manager.image, current_velocity)
world.vehicle.apply_control(carla.VehicleControl(throttle=float
    (throttle), steer=float(steer), brake=float(brake)))
```

3.3. BehaviorMetrics

BehaviorMetrics [13], cuya arquitectura se puede ver en la Figura 20, es la herramienta que se ha usado y mejorado en este trabajo para evaluar el comportamiento del vehículo autónomo desarrollado. El uso de esta herramienta externa a CARLA es debido a que CARLA no proporciona todas las métricas necesarias para evaluar la correcta funcionalidad de los modelos, y las métricas de entrenamiento estándar, como es la evolución del error cuadrático medio (MSE), no son suficientes para evaluar el desempeño adecuadamente. El uso de BehaviorMetrics permite comprobar entre una variedad de cerebros cuáles son los que ofrecen unos mejores resultados, comparándolos por medio de una serie de métricas cuantitativas.

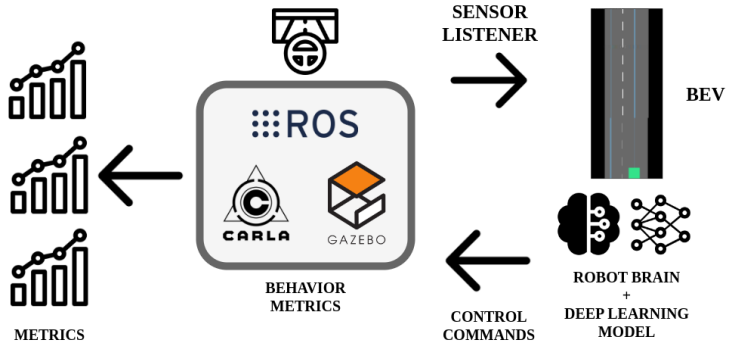


Figura 20: Arquitectura de la herramienta BehaviorMetrics.

Algunas de las métricas que se calculan con esta herramienta son:

1. El tiempo de inferencia de la GPU.
2. La media de iteraciones que hace el cerebro en tiempo real.
3. El tiempo total que duró la ejecución del experimento.
4. La distancia total recorrida por el vehículo durante la ejecución del experimento.
5. La desviación media del vehículo con el centro de su respectivo carril.

Y además, BehaviorMetrics devuelve un mapa, como se puede ver en la Figura 21, del simulador, con información visual de la desviación del vehículo, de su carril, y las zonas donde el vehículo invadió el carril contrario o se salió de la carretera.

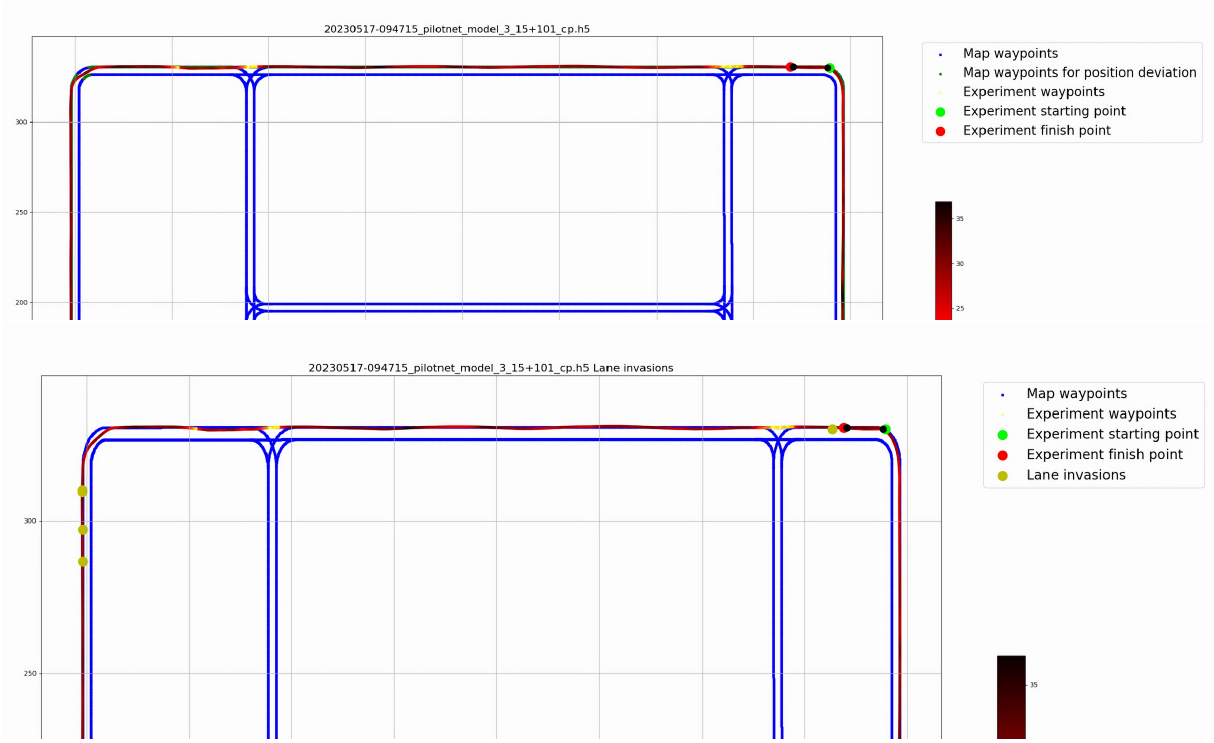


Figura 21: Métricas obtenidas de BehaviorMetrics. (Arriba) Mapa de la desviación del vehículo. (Abajo) Mapa de las invasiones del carril.

3.4. Python

Python [42] es un lenguaje de programación interpretado, lo que quiere decir que no requiere de un proceso de compilación antes de ejecutar el código. El intérprete, a medida que lee las líneas de código, las va ejecutando. Es ampliamente utilizado en diversos campos, desde el desarrollo web y científico hasta la automatización de tareas y el aprendizaje automático. Además, el hecho de que cuente con una amplia biblioteca estándar y una comunidad activa que proporciona una amplia gama de módulos y paquetes, lo que permite a los desarrolladores aprovechar funcionalidades preexistentes.

La razón principal para usar Python residía en las facilidades que ofrece para entrenar modelos de aprendizaje automático. Se instaló la versión 3.7 para todo el proyecto por su

gran uso en el simulador de CARLA y porque, más tarde, la herramienta BehaviorMetrics haría uso de esta misma versión para su correcto funcionamiento.

3.5. Tensorflow y CUDA

Con el fin de aprovechar la gráfica de la que se dispuso (Nvidia RTX 3060 portátil) y puestos que el proyecto giraba en torno al entrenamiento de una inteligencia artificial, la instalación del paquete Tensorflow [43] era más que necesaria.

TensorFlow es una biblioteca de código abierto ampliamente utilizada para el desarrollo y entrenamiento de modelos de aprendizaje automático. Con el fin de aprovechar al máximo la GPU, se usó TensorFlow-GPU, una versión específica de TensorFlow que se creó para utilizar el poder de cálculo de las GPU en tareas de aprendizaje automático y procesamiento de datos. Para este proyecto, se eligió TensorFlow-GPU en su versión 2.10.0. Esta elección se basó en la necesidad de garantizar la compatibilidad con Python 3.7 y con la herramienta BehaviorMetrics. A su vez, se escogió la versión 11.0 de CUDA para evitar conflictos tanto con la versión de TensorFlow-GPU como con la versión de Python utilizada en el proyecto.

La integración de TensorFlow con las GPU se logra gracias a CUDA [44], una plataforma desarrollada por NVIDIA que permite ejecutar cálculos de forma paralela en las GPU. TensorFlow-GPU se diseñó con el fin de aprovechar al máximo las capacidades de CUDA y las GPU compatibles.

El uso de Tensorflow con CUDA permite realizar operaciones matemáticas de manera mucho más eficiente que si se usara solo la CPU, consiguiendo reducir los tiempos de entrenamiento y aumentando los FPS de la simulación, lo cual es muy importante para asegurar un buen rendimiento del sistema.

3.6. OpenCV

OpenCV [45], acrónimo de *Open Computer Vision*, es una biblioteca de código abierto que, como su nombre indica, es muy usada en las áreas de Visión Artificial. Proporciona una gran variedad de funciones y algoritmos para realizar tareas como reconocimiento de objetos, seguimiento de objetos, detección de rostros, calibración de cámaras, entre otras.

Esta librería se instaló con la versión 4.6.0.66, dado que era la versión más nueva cuando se empezó el proyecto y no causaba ningún conflicto con el resto de librerías. OpenCV se usó con dos fines. Primero para la creación de un cerebro capaz de hacer que el vehículo siga la ruta dentro de su carril. Segundo, para procesar las imágenes que recogía la cámara RGB montada en la parte frontal del vehículo, bien para leerlas o bien procesarlas para cumplir los requisitos del sistema.

3.7. Pygame

Pygame [46] es una biblioteca de código abierto para el desarrollo de videojuegos y aplicaciones multimedia en Python que se instaló en su versión 2.3.0. Proporciona una serie de

funciones que permiten la creación de gráficos, sonidos, animaciones y manejo de eventos, facilitando así la creación de juegos y programas interactivos. Para el proyecto, Pygame sirve como ayuda a la visualización de lo que pasa en el simulador CARLA. Gracias a que CARLA ofrece facilidades para conectarse con la librería Pygame, es posible observar, por medio de la cámara del vehículo, el comportamiento del modelo entrenado.

3.8. Numpy

NumPy [47] es una biblioteca de código abierto fundamental en el entorno de programación de Python para el procesamiento numérico y científico. Proporciona estructuras de datos eficientes como matrices multidimensionales, junto con una amplia gama de funciones y operaciones matemáticas muy optimizadas.

Esta librería se instaló en su versión 1.21.6, y en el proyecto se usa específicamente para tratar con las imágenes obtenidas por la cámara RGB del vehículo. Ya sea para dimensionar imágenes, guardarlas o abrirlas.

Capítulo 4

Aprendizaje por imitación para una conducción en tráfico

En esta sección se analizan los detalles principales del proyecto. En primer lugar, se detalla la configuración del entorno experimental junto con sus restricciones. Después se verán los tres componentes principales de los que se compone el proyecto: conjunto de datos, modelo propuesto y entrenamiento del modelo.

4.1. Agente experto programado

El experto es aquel que sirve como referencia o modelo para enseñar al vehículo autónomo cómo comportarse y tomar decisiones. Una primera versión del experto se programó como un controlador PID que permitiera mantener el vehículo en su carril. Este controlador PID se divide principalmente en 3 partes como se ve en la Figura 22, la parte proporcional, la parte integral y la parte derivativa. Este controlador, a través de la realimentación de los valores obtenidos, regula el ángulo de dirección del coche. El controlador proporcional se encarga de que el ángulo de dirección se aproxime a su valor deseado de forma proporcional a su error. La parte integral del controlador hace uso de los errores pasados para proporcionar una fuerza extra en caso de que el valor tarde demasiado en alcanzar el valor deseado. Por último, la parte derivativa detecta si el movimiento desde el valor real al deseado es demasiado brusco, suavizando así la variable si va demasiado rápida hacia su valor deseado.

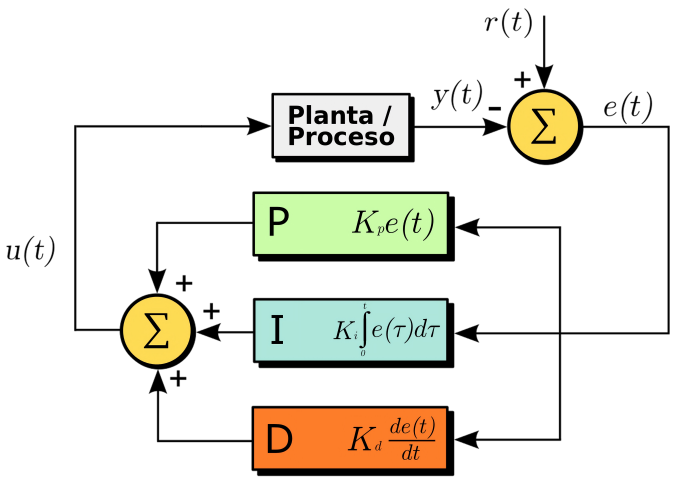


Figura 22: Vista general del funcionamiento de un controlador PID.

Esta primera versión funciona de la siguiente manera. Analizando la Figura 23, se ven dos puntos azules que son los que definen los extremos del carril, además de una línea roja que define el centro del vehículo. La línea verde indica la dirección hacia la cual se dirige el carril, considerando los puntos extremos del carril y el punto medio del vehículo. Esta será la desviación del vehículo respecto a la carretera. Los parámetros del controlador PID se ajustan para mantener la línea verde lo más cercana posible a la línea roja, logrando así que el vehículo siga el carril. Sin embargo, se presenta un problema cuando la línea que marca los dos carriles se interrumpe, lo cual ocurre en algunas zonas.

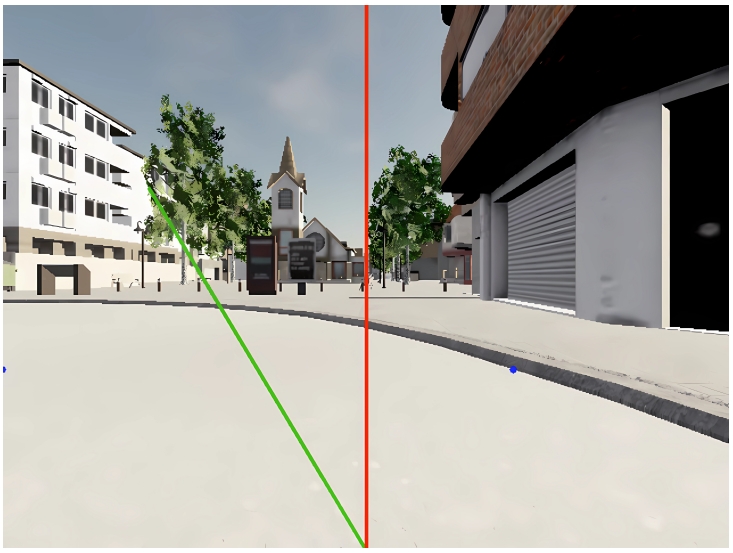


Figura 23: Funcionamiento de la primera versión del experto sigue-carril.

4.1.1. Autopiloto de CARLA como conductor experto

Para resolver los problemas encontrados por el experto usando un controlador PID, se puede hacer uso del enfoque modular, añadiendo nuevos componentes que traten con los

diferentes problemas a medida que vayan surgiendo individualmente, pero con el objetivo de enfocar el proyecto en la parte del entrenamiento de una inteligencia artificial de extremo a extremo. La solución más simple consistió en utilizar el piloto automático incorporado en CARLA como conductor experto.

El autopiloto de CARLA está configurado para seguir una ruta específica y básica para que sea capaz de aprender a permanecer en el carril, girar y detenerse cuando otro vehículo está delante de él. El piloto automático proporcionado por CARLA fue usado como agente experto de donde extraer los datos. Con él, se generaron un conjunto alrededor de 150.000 imágenes junto con datos supervisados. Aproximadamente el 47 % de los datos es bajo condiciones que incluyen otro vehículo en el entorno, mientras que el 53 % restante no.

4.2. Datasets de conducción en tráfico

El conjunto de datos recogido en el entorno urbano Town02 (ver Figura 24) del simulador CARLA [36] se puede dividir en tres conjuntos principales, pues ha ido variando según se progresó el trabajo de fin de máster. Primero, se genera un dataset simulando un entorno urbano sin tráfico donde el vehículo principal simplemente sigue su carril, este será el dataset de "conducción sin tráfico". Segundo, se generó un dataset en un entorno con tráfico donde el vehículo principal sigue su carril como el dataset anterior, pero esta vez va a tener un vehículo delantero que frenará y continuará su marcha a lo largo de su recorrido. Este será el dataset de "conducción con tráfico único". Finalmente, el dataset más completo se genera igual que el segundo en un entorno de tráfico, pero esta vez contando con diferentes vehículos delanteros para así enseñar a cómo generalizar el comportamiento de frenada y reanudación de la marcha ante vehículos delanteros distintos. Este último dataset será el de "conducción con tráfico variado".

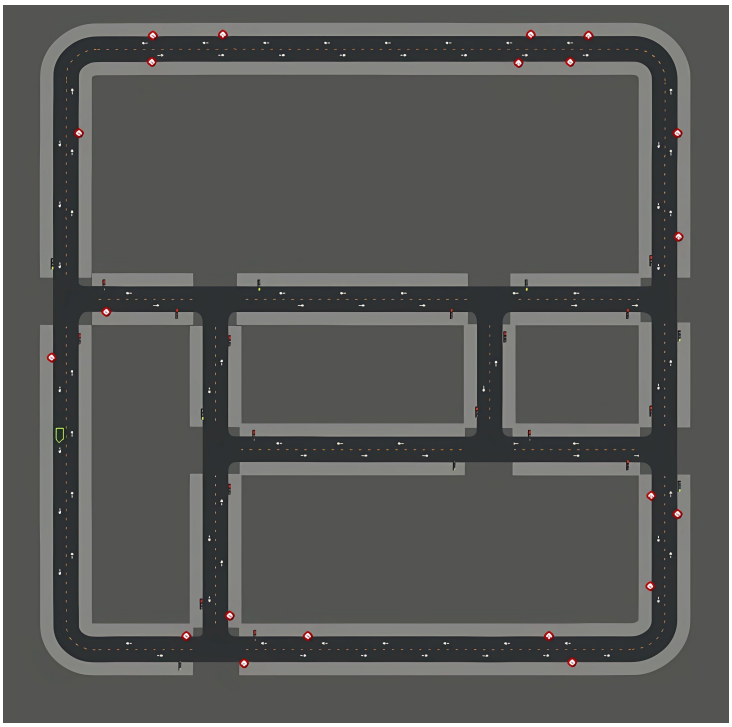


Figura 24: Vista de pájaro del entorno Town02 del simulador CARLA.

Para el entrenamiento y las pruebas, se utiliza una cámara RGB frontal que captura imágenes de 480 píxeles de altura y 650 píxeles de anchura (480 x 650). Esta configuración permite obtener datos de forma sencilla y eficiente, a partir de la cual crear un vehículo autónomo lo suficientemente robusto como para generalizar su manejo en un entorno urbano. El modelo del vehículo usado es un Tesla Model 3 como el que se ve en la Figura 25.

Figura 25: Modelo del vehículo autónomo principal del simulador CARLA.

4.2.1. Gestión del escenario del simulador

La simulación de CARLA puede ejecutarse de dos formas, en alta calidad y en baja calidad. Si bien cuanta más calidad, más se puede acercar la simulación al mundo real, para este trabajo de fin de máster esto no era estrictamente necesario. Si además se tiene en cuenta que es lo que antes se prioriza, si la calidad gráfica de la simulación o las imágenes por segundo, pues la responsividad del simulador aumenta o disminuye en función de la calidad gráfica, se va a priorizar que el simulador sea ágil y responsivo. Por eso mismo, al lanzar el servidor de CARLA, siempre se hará con la opción de calidad gráfica baja (*_quality_level=Low*).

```
./CarlaUE4.sh /Game/Carla/Maps/Town02 -quality-level=Low -ResX
   =640 -ResY=480
```

Otro aspecto importante en la gestión del entorno son los figurantes, es decir, los vehículos que se mueven en el simulador aparte del principal. Estos vehículos se generan delante del vehículo principal lo más cerca posible de este para que se dé, de forma más habitual, las frenadas y reanudaciones de la marcha. CARLA ofrece diferentes modelos de vehículos, desde turismos comunes, motocicletas, furgonetas, ambulancias y otros más. Esta diferencia en la forma y el color de los diferentes vehículos son los que otorgarán al cerebro del vehículo autónomo, la capacidad de generalizar ante cualquier obstáculo presente en su carril.

Finalmente, con el fin de generar datos más relevantes, lo que se hizo fue alterar el clima del simulador. CARLA proporciona formas muy sencillas que cambian por completo como

percibe el vehículo, su entorno como se puede ver en la Figura 26, esto es realmente importante para una conducción autónoma robusta que siga conduciendo de forma segura ante los inevitables cambios climáticos presentes en el mundo real.

Figura 26: Ejemplos de climas variados en el simulador CARLA.

La forma de modificar el clima es bastante sencilla, pues una vez más, CARLA ofrece muchas facilidades para ello. El objeto de la clase Clima (*Weather*) se puede obtener del servidor a partir de la función *get_weather*. Este objeto tiene una gran variedad de atributos que serán los valores que alteren la posición del sol, las nubes, lluvia, niebla, entre otros. Con esto se puede configurar de muchas formas el simulador para conseguir un dataset muy variado.

```
weather = world.world.get_weather

weather.cloudiness = 60.0
weather.precipitation = 40.0
weather.wind_intensity = 40.0
weather.precipitation_deposits = 40
weather.sun_azimuth_angle=275.0
weather.sun_altitude_angle=20.0
weather.fog_density=5.0
weather.fog_distance=0.75
weather.wetness=80.0

world.world.set_weather(weather)
```

4.3. Modelo neuronal

El modelo que se utilizó es una variante del modelo PilotNet [23]. Esta red consta originalmente de 9 capas, las cuales incluyen una capa de normalización, 5 de ellas son capas

convolucionales y 4 capas de redes neuronales totalmente conectadas o densas. En la Figura 27 se puede ver la arquitectura de este modelo, donde como entrada recibe una imagen normalizada de tamaño (66 alto, 200 de ancho, 3 canales RGB), y como salida se genera el ángulo de giro del vehículo.

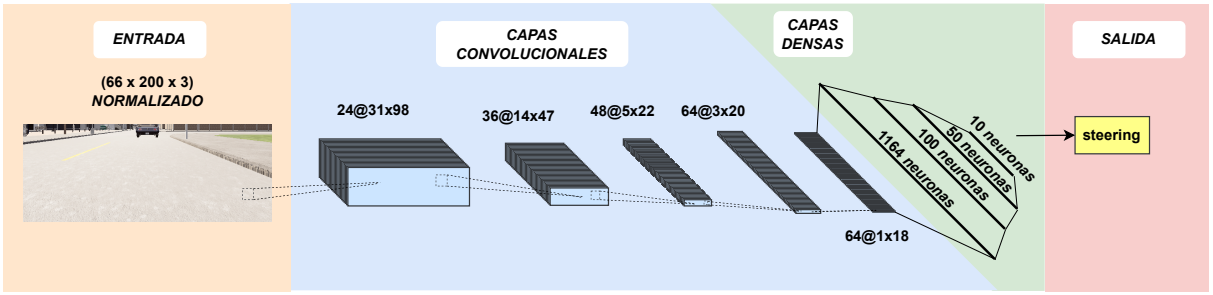


Figura 27: Arquitectura del modelo de PilotNet.

Con el fin de usar este modelo para nuestros propósitos, fue necesario realizar algunos ajustes. Dado que el ámbito de nuestro proyecto incluía la detención siempre que hubiera un obstáculo en la carretera, añadimos dos salidas más para nuestro modelo, dejando al modelo con la tarea de predecir la dirección, el acelerador y el freno. A su vez, una vez establecidas las entradas y salidas y analizando la configuración del modelo, se llegó a la conclusión que hacer uso de una técnica de regularización conocida como *dropout* producía muy buenos resultados globales. Esta *dropout* se incluye en las capas de redes neuronales densas.

Este modelo, como se puede ver en la Figura 28, contaba con una salida extra en comparación al PilotNet original, el valor de aceleración/frenada. Teniendo la arquitectura del modelo diseñada y entrenándolo usando un dataset propio generado con el simulador CARLA, se conseguía un modelo alternativo PilotNet*, capaz de conducir en un entorno urbano.

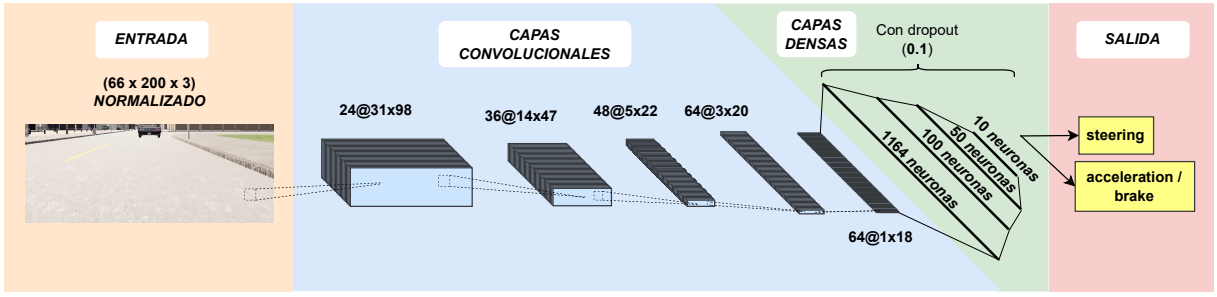


Figura 28: Arquitectura del modelo PilotNet*.

A medida que se va desarrollando más el trabajo, el modelo se fue mejorando hasta llegar a la última versión del modelo: OA+FLM. Con el fin de aumentar la contextualización del mundo en el que el vehículo va a conducir, se añadió la velocidad previa para cada instante de tiempo donde se recogían los datos. La idea principal detrás de esta decisión es que aportando al vehículo más información relevante del mundo en el que está, es posible hacer que este se comporte mejor, en este caso, cuando haya un vehículo obstaculizando

su carril o deje de obstaculizarlo. Esta velocidad previa (pV) se introduce en la entrada del modelo neuronal junto con la imagen. Así, si tenemos una imagen de la forma (66, 200, 3) que representan (alto, ancho, canal), la velocidad previa se añade como un canal extra en la imagen de entrada, dando una imagen de entrada final con dimensiones (66, 200, 4) tal como se puede ver en la Figura 29. En el proceso descrito, tanto la imagen de entrada como la velocidad previa se someten a un proceso de normalización. Los comandos de control se normalizan al crear el dataset, por lo tanto, las salidas de la red neuronal vienen normalizadas, siendo necesario realizar un proceso de desnormalización para ambas salidas.

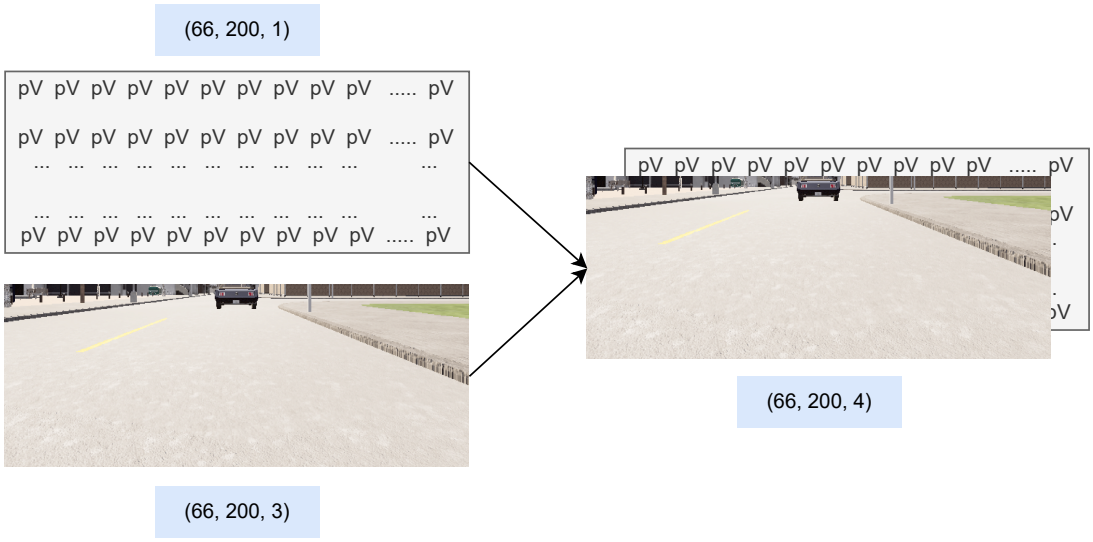


Figura 29: Visualización del canal extra donde se introduce la velocidad previa junto con la imagen de entrada al modelo.

Además, el OA+FLM se entrenó con el dataset final comentado en la sección de *Dataset* de este capítulo y cuenta con las mismas dos salidas del modelo PilotNet*. La arquitectura de este modelo se puede ver en la Figura 30.

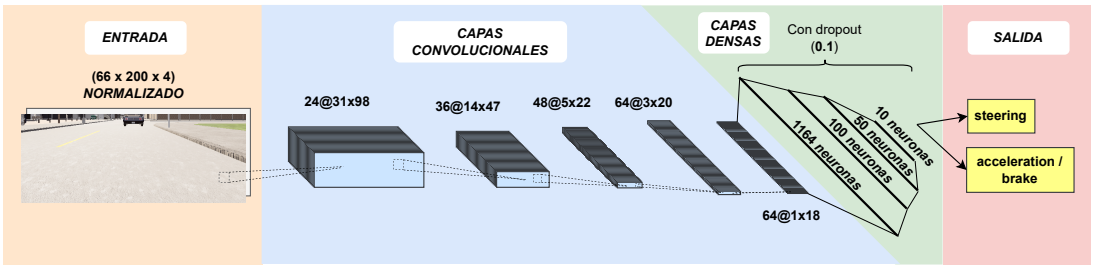


Figura 30: Arquitectura del modelo OA+FLM.

4.4. Entrenamiento

Ya con los datos crudos obtenidos de la simulación de CARLA se puede entrenar un modelo que pueda predecir los comandos para controlar al vehículo autónomo. Sin em-

bargo, antes de iniciar el entrenamiento, es recomendable realizar un preprocesamiento en el conjunto de datos con el fin de mejorar el entrenamiento y, en consecuencia, mejorar el comportamiento final del vehículo. Estas prácticas adicionales garantizan una mayor calidad en el entrenamiento del modelo y contribuyen a un mejor rendimiento del vehículo autónomo.

4.4.1. Imagen de entrada

La imagen de entrada con la que va a tratar el modelo CNN es bastante grande, teniendo en cuenta que tendrá que mapear 307.200 píxeles de una imagen de la que probablemente no se necesiten todos. Es por eso que se puede reducir este número y, al hacerlo, se reduce también el tiempo de entrenamiento. Al recortar la imagen casi a la mitad, se omiten elementos como el cielo o los edificios, pues no aportan información útil para que el vehículo pueda predecir hacia donde girar o si debería parar. Una vez que tenemos la versión recortada de la imagen, podemos reducir aún más el tamaño de la imagen, ya que no es necesario tener una imagen de alta calidad para extraer las características principales. Para ello, redimensionamos la imagen a 66 píxeles de alto y 200 de ancho, quedándonos con unos 13.200 píxeles que mapear, lo que apenas supone un 4 % de la imagen original (ver Figura 31).

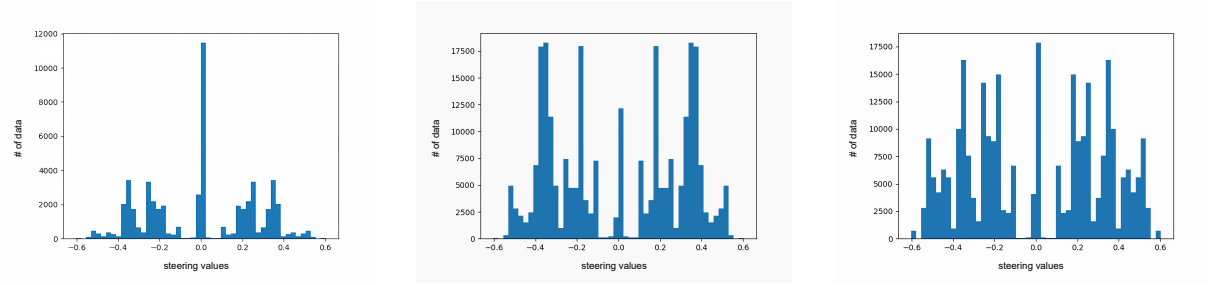
Figura 31: Izq.: Imagen RGB tomada por la cámara frontal del coche. Dcha.: imagen final cortada usada como entrada del modelo.

4.4.2. Balanceo de datasets

Uno de los primeros procesamientos que se hicieron fue el de balancear mejor el dataset y así lograr un comportamiento más sólido y propio para los objetivos del trabajo. Unos datos con sobre representación de las rectas, como es el caso del entorno Town02 donde la ciudad tiene más rectas que curvas, pueden hacer que el vehículo sobreajuste las rectas olvidando cómo se toma una curva correctamente. Y lo mismo si damos más prioridad a las curvas que a las rectas. Es importante ajustar correctamente los datos, balanceándolos de forma que el agente pueda aprender ambas tareas de giro y de mantenerse dentro del carril en las rectas.

En la Figura 32 se puede ver la distribución de los diferentes casos comentados previamente. Sacando que las Figuras 32a y 32b se centran en o bien priorizar las rectas o bien

las curvas respectivamente, estas van a causar inevitablemente que el modelo sobresalga en su capacidad para mantenerse dentro del carril recto o en su capacidad para tomar curvas correctamente. Ahora bien, la Figura 32c muestra unos datos más balanceados, siendo un ejemplo de lo que se quiere conseguir. Unos datos suficientes tanto como para los casos sigue-carril recto como para la toma de curvas.



(a) (b) (c)

Figura 32: Histograma de los datasets de giro. (a) Datos donde la prioridad se la da a las rectas antes que a las curvas. (b) Datos donde la prioridad se la da a las curvas antes que a las rectas. (c) Datos bien balanceados.

Por el otro lado, este proyecto pretende resolver no solo el problema sigue-carril, sino también la capacidad de frenar y acelerar cuando sea necesario, en caso de que otro vehículo esté bloqueando el carril. Es por eso que nuestro dataset también se divide en datos cogidos con el propósito de enseñar al vehículo a seguir su carril, y datos que muestran mejor cómo frenar y acelerar ante estos obstáculos. La Figura 33 muestra un histograma del dataset basado en los valores de velocidad y frenado [0, 1], donde el rango de [0, 0.5] pertenece a los valores de la frenada, mientras que el rango [0.5, 1] a los valores de aceleración.

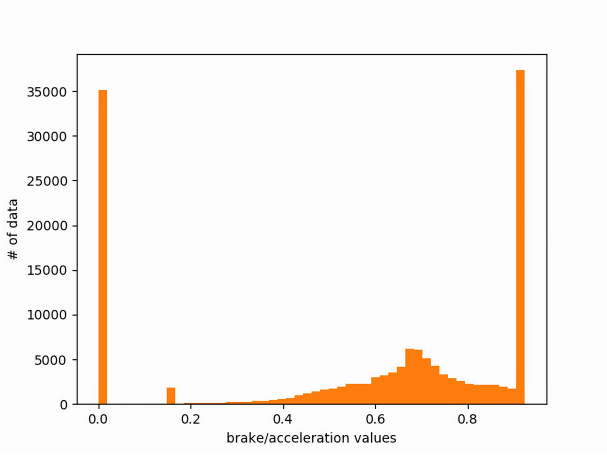


Figura 33: Histograma del dataset de aceleración y frenado

La dificultad se presenta cuando hay que balancear el dataset teniendo en cuenta el giro, por un lado, y la aceleración/frenado, por otro. Si balanceamos los datos para disminuir los valores donde el giro es nulo o casi nulo, es decir, en rectas, se podría estar alterando el balanceo con respecto a los datos de aceleración/frenada, y viceversa. Hay que hacer

una gran cantidad de experimentos y pensar bien cómo balancear ambos casos con el fin de conseguir un dataset útil.

Finalmente, este balanceo de datos, al realizarse por medio de eliminaciones de datos con el fin de o bien priorizar rectas o bien priorizar curvas, nos dejan con un total de 130.000 imágenes con sus respectivos datos supervisados.

4.4.3. Aumentado de datos

La biblioteca *albumentation* [48] se utiliza durante el entrenamiento con el fin de aumentar la variedad del conjunto de datos. Permite cambiar la iluminación, desenfocarla y añadir artificialmente nieve, niebla, lluvia y sombras a nuestras imágenes. Esta variedad ofrece robustez a la hora de entrenar al agente con datos más variados. No obstante, concluimos que al voltear las imágenes, aunque resultaba útil para generar un conjunto de datos más variado, pues se nivelaban mejor los giros a izquierda y derecha, no se distinguía si debía estar en el carril derecho o izquierdo, lo cual conllevaba un comportamiento inadecuado a la hora de conducir como debe.

El aumentado de datos, tal y como se hace en el proyecto, aumenta la variabilidad y la diversidad del dataset sin aumentarlo en cantidad.

4.4.4. Evolución del aprendizaje

En las Figuras 34 y 35 se puede visualizar la evolución del entrenamiento del modelo. La Figura 34 presenta la evolución del error cuadrático medio a lo largo del proceso de entrenamiento, siendo este error la función de pérdida que enseña la diferencia con los valores supervisados incluidos en el dataset. Con un total de 100 épocas, se puede observar una mejora consistente en la métrica de pérdidas a medida que avanza. Sin embargo, después de aproximadamente 40-50 épocas, el error se estabiliza y casi no muestra ninguna mejora. Para evitar un posible sobreajuste, detenemos el proceso de entrenamiento alrededor de esta época. Terminando el entrenamiento en este punto, se consigue un buen equilibrio entre la optimización del rendimiento del modelo y la evitación de un ajuste excesivo que pueda conducir a un sobreajuste.

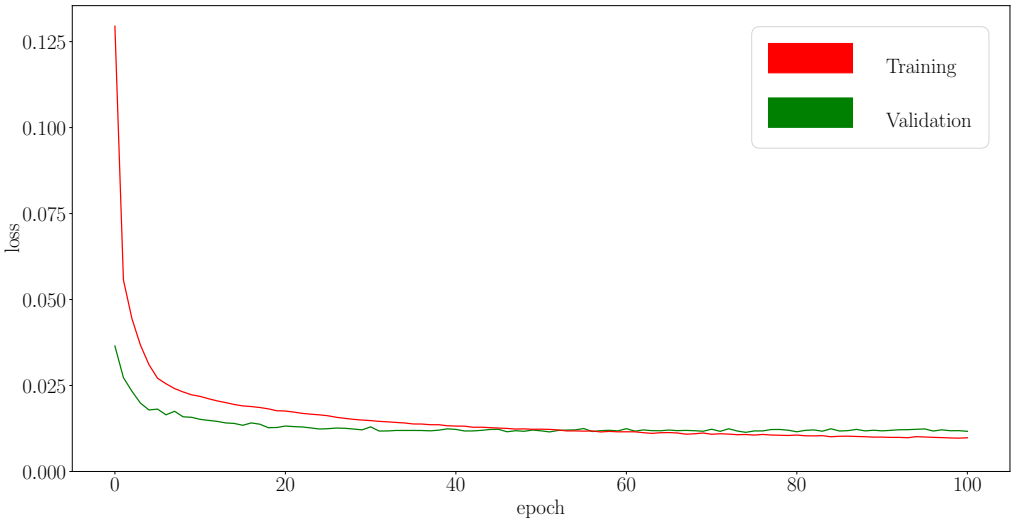


Figura 34: Gráfica de la evolución del error en los datos de entrenamiento y de validación para cada época.

La Figura 35 compara los valores de giro y aceleración del experto (gráfico verde) con las predicciones de giro y aceleración del modelo (gráfico rojo). El eje Y representa los valores del acelerador o el giro, mientras que el eje X indica la marca de tiempo de nuestros datos. Si se analizan, el gráfico superior muestra una mejor superposición comparada con el inferior, lo que indica que el modelo funciona mejor en ciertos aspectos. La red neuronal maneja tanto el control de la dirección como el acelerador/freno, pero lograr un buen rendimiento en ambas tareas es todo un reto. Y es que el conjunto de datos se centra más en trayectos rectos y curvas que en situaciones en las que la carretera está bloqueada por otro vehículo.

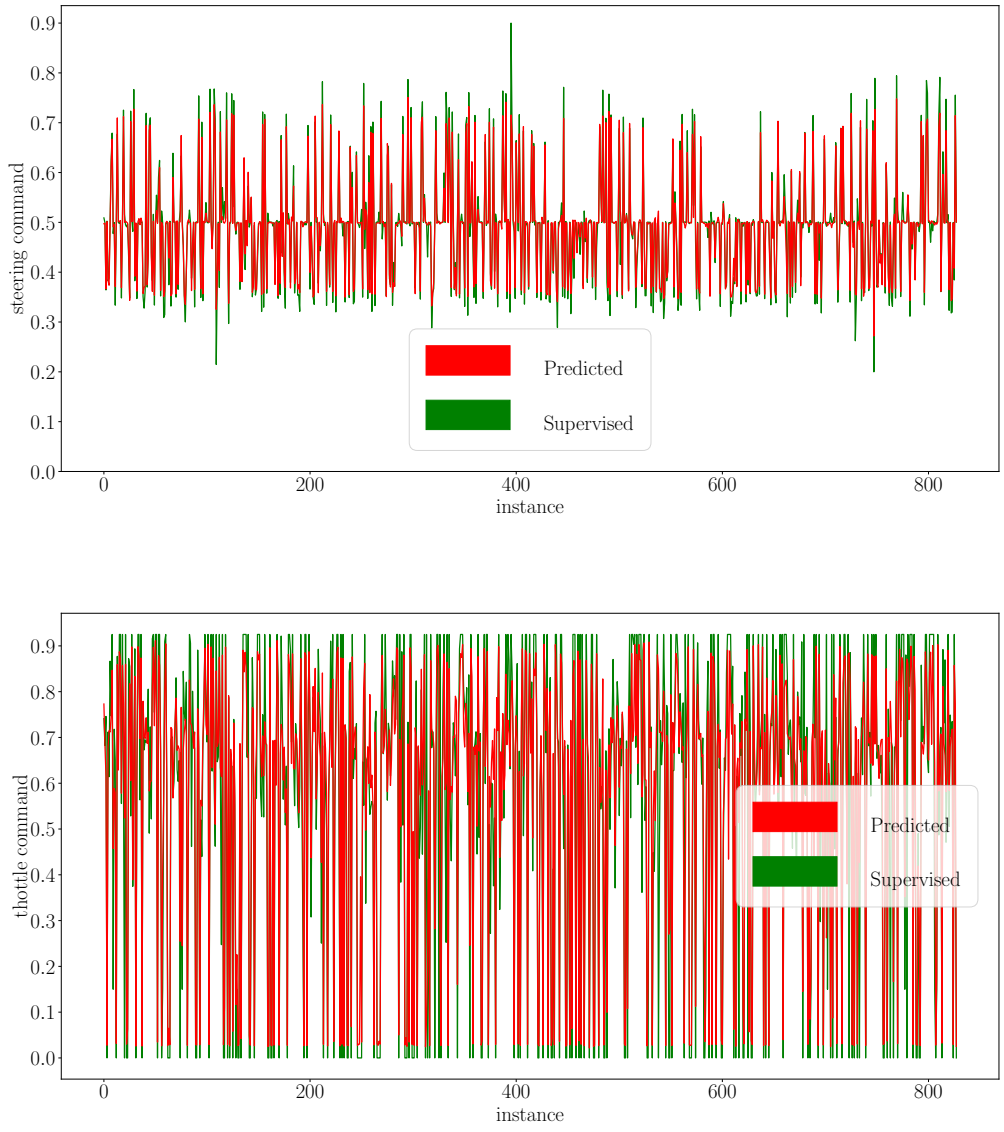


Figura 35: Comparación de los valores supervisados del experto (rojo) de giro (arriba) y de la aceleración/frenada (abajo) con respecto a los valores predichos (verde).

Para comprender mejor el comportamiento de nuestro modelo y detectar con éxito cualquier comportamiento inusual, una idea es observar cómo percibe el mundo la red neuronal. La Figura 36 representa visualmente las características extraídas por las capas CNN del modelo. Se puede ver que reconoce los extremos del carril y destaca las ruedas del coche delantero. La atención prestada a las ruedas es el resultado del entrenamiento con diversos vehículos, lo que permite una generalización eficaz y reduce el riesgo de colisión con cualquier vehículo en carretera.

Figura 36: Visualización del mapa de características extraído de las capas de la CNN.

Capítulo 5

Evaluación y experimentos

En este capítulo se presentan los experimentos realizados y las herramientas usadas para evaluar y medir el desempeño del proyecto. Se analizan la efectividad y viabilidad de la aplicación del aprendizaje profundo en la conducción autónoma en tráfico real, abordando con más atención las capacidades sigue-carril y el comportamiento frente a obstáculos en el carril.

5.1. Evaluación automática y cuantitativa

La evaluación cuantitativa da la capacidad de medir y analizar los datos para evaluar el rendimiento, la eficacia y la calidad del modelo. Y lo que es más importante, esta evaluación es útil para comparar el rendimiento y la calidad de distintos modelos neuronales con el fin de ver si los cambios en el entrenamiento, el conjunto de datos o el modelo producen mejoras en el rendimiento.

5.1.1. Evaluaciones proporcinadas por CARLA

Para evaluar el rendimiento de los modelos, CARLA ofrece mucha información útil del vehículo, lo cual viene bien para diseñar métricas que se necesiten en función de las situaciones que se deseen analizar. Algunas de estas métricas proporcionadas por CARLA son:

1. Colisiones: crea un evento cada vez que se produce una colisión entre el vehículo principal y cualquier actor de la simulación, ya sea otro vehículo, un edificio o un objeto del entorno.

2. Invasiones de carril: crea el evento cuando el vehículo principal cruza una marca de carril.

3. Posición del vehículo: da la información sobre la ubicación absoluta y orientación de cualquier actor en la simulación.

5.1.2. BehaviorMetrics

El acceso a esta información proporcionada por CARLA puede ser insuficiente si queremos analizar más profundamente el rendimiento de nuestros modelos. Para mejorar nuestras capacidades de análisis, se ha usado la herramienta BehaviorMetrics [13]. Este software es una plataforma donde probar varios enfoques de aprendizaje por imitación de extremo a extremo y permite evaluar y valorar comportamientos complejos para diferentes robots autónomos utilizando técnicas de aprendizaje automático y aprendizaje profundo.

Usando la información proporcionada por la simulación CARLA en la herramienta BehaviorMetrics, es posible obtener un conjunto de métricas adicionales interesantes que resultan más útiles para evaluar el comportamiento del vehículo.

Para evaluar el rendimiento de nuestro modelo, BehaviorMetrics proporciona las siguientes métricas:

1. La tasa de éxito: este porcentaje representa la tasa con la que el vehículo completa con éxito una vuelta al circuito de referencia.

2. La desviación media de la posición (MPD): esta métrica indica la desviación media en metros del vehículo respecto al centro del carril designado por km.

3. Invasiones de carril: esta métrica representa el número medio de veces que el vehículo se cruza al carril contrario o se sale de la carretera. BehaviorMetrics genera esta métrica, a diferencia de CARLA, por kilómetro, con la idea de comparar estas métricas entre experimentos que puedan haber sido configurados de forma diferente.

5.1.3. Métrica de distancia al vehículo delantero

Para evaluar las tareas de seguimiento de carril, estas métricas anteriores son más que suficientes, ya que muestran lo bien que el coche se mantiene dentro de su carril correspondiente. Pero si queremos ver cómo se comporta el vehículo principal cuando hay otro vehículo delante, necesitamos otra métrica adicional.

Una buena forma de evaluar el manejo adecuado frente a un obstáculo es mediante el uso de la distancia al vehículo delantero, y esta métrica se desarrolló específicamente en este TFM. En primer lugar, se obtiene la distancia al vehículo más cercano junto con la diferencia de orientaciones entre ambos vehículos. Esto permite verificar si el vehículo más cercano se encuentra realmente adelante del vehículo principal o si simplemente está pasando a su lado en el carril contrario. A continuación, se divide la distancia en cuatro rangos: distancia grande (entre 20 y 50 metros), distancia media (entre 15 y 20 metros), distancia corta (entre 6 y 15 metros) y distancia peligrosa (entre 0 y 6 metros). El objetivo de esta métrica es observar cuánto tiempo el vehículo principal se mantiene a una distancia determinada al acercarse al vehículo delantero. Así podemos verificar si el vehículo está entrando en la distancia peligrosa y en qué medida. También es posible determinar si se está deteniendo progresivamente cuando el vehículo delantero también se está deteniendo o si ni siquiera está intentando evitar un posible accidente.

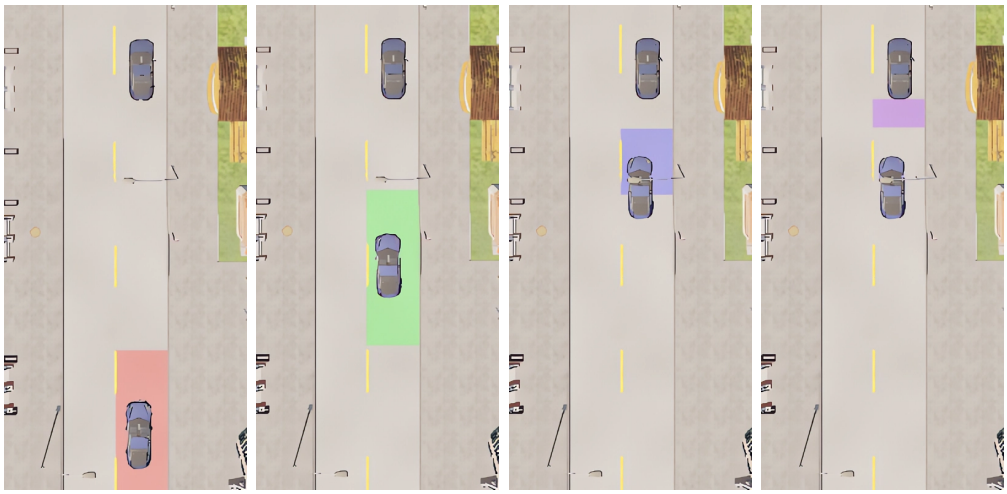


Figura 37: Distancias largas, medias, cortas y peligrosas con respecto al vehículo delantero.

El desarrollo de esta métrica se puede ver con más detalle analizando el código. En primer lugar, gracias a que BehaviorMetrics usa ROS-Noetic, por medio del paquete *carla_ros_bridge* es posible extraer información relevante del simulador CARLA para usarla dentro de BehaviorMetrics.

En este caso, vamos a comprobar la orientación de los dos vehículos que se pueden dividir en dirección, o más conocido en inglés como *yaw*. Más específicamente el que se utilizará será la dirección.

```
yaw_difference = abs(point['pose.pose.orientation.y'] - point_2
   ['pose.pose.orientation.y'])
```

Con esto, se obtiene la diferencia de la dirección entre los dos vehículos, pudiendo comprobar si el vehículo delantero se encuentra realmente delante del vehículo principal o si se trata de otro coche que se dirige a otra dirección.

```
if(-20 < yaw_difference) and (yaw_difference < 20):
```

El resto se puede definir de la siguiente forma. Tal y como se ve en el código a continuación, se extraen dos variables, una llamada *distance_front* que guarda la distancia entre el vehículo principal y el vehículo delantero, y otra llamada *distance* que calcula la distancia recorrida por el vehículo principal. Primero se comprueba cuál es la distancia que separa con el vehículo delantero. En función de esta distancia, caerá en un rango u otro ya preestablecido. El resto sería cuestión de añadir la distancia recorrida por el vehículo principal a la variable que le corresponda según el rango en el que se encuentre: *great_distance*, *medium_distance*, *close_distance*, *dangerous_distance*.

```
distance_front = np.linalg.norm(current_checkpoint -
   current_checkpoint_2)
distance = np.linalg.norm(previous_point - current_checkpoint)

if 20 < distance_front < 50:
    great_distance += distance
elif 15 < distance_front <= 20:
    medium_distance += distance
```

```
elif 6 < distance_front <= 15:
    close_distance += distance
elif distance_front <= 6:
    dangerous_distance += distance
```

5.2. Validación experimental

Se diseñaron tres experimentos para evaluar el comportamiento del coche: dos centrados en ejecuciones típicas con y sin tráfico, los cuales permiten comprobar cómo se maneja el vehículo tanto para seguir el carril como para evitar posibles colisiones con otros vehículos, y otro para probar la capacidad de generalización del modelo al evitar obstáculos. Este último experimento se diseñó de forma que se comprueba solamente la forma en la que el vehículo principal se acerca al vehículo delantero y si éste termina chocando con él o si frena adecuadamente.

En la evaluación, se compararon los dos modelos que se explicaron con más detalle en el capítulo Modelo neuronal: (a) el OA+FLM (Obstacle Avoidance + Follow-Lane Model), el cual fue entrenado con el dataset de conducción con tráfico variado, y (b) el modelo alternativo de PilotNet, PilotNet*, que fue entrenado con el dataset de conducción sin tráfico.

La evaluación tiene lugar en el entorno Town02, donde se recogieron los datos de entrenamiento, y el entorno Town01, que nunca se utilizó para entrenar el modelo, solo para realizar las pruebas de evaluación.

5.2.1. Ejecución típica sin tráfico

Para los experimentos de una ejecución típica, se hizo al modelo dar tres vueltas en el sentido de las agujas del reloj y otras tres vueltas en una ruta en sentido contrario en cada ciudad. En total, se dieron 12 vueltas sin tráfico.

Esta ejecución típica sin tráfico tiene como objetivo principal probar las capacidades de seguimiento de carril de los modelos. Viendo la Tabla 1, podemos observar cómo el comportamiento es bastante parecido del PilotNet* frente al OA+FLM, pues los dos fueron entrenados para poder seguir su recorrido dentro del mismo carril donde se encuentren.

Tabla 1: Métricas para dos ciudades y dos modelos diferentes en una carretera sin tráfico. Tasa de éxito: cuanto más alta, mejor; el resto: cuanto más bajo, mejor.

	Town01		Town02	
	PilotNet*	OA+FLM	PilotNet*	OA+FLM
Tasa de éxito (%)	100	**100**	100	**100**
MPD	1.39	**0.19**	0.75	**0.22**
Invasión de carril	22.47	**4.75**	19.08	**3.42**

Como se ha comentado, el modelo OA+FLM tiene un rendimiento muy similiar al PilotNet* en escenarios sin tráfico. Y es que a pesar de haber entrenado al OA+FLM para

que abarque más tareas, no solo que se mantenga en el carril, sino que tenga que estar atento a otros vehículos que frenen delante de él, este no pierde prestaciones y es capaz de seguir manteniendo correctamente al coche dentro de su respectivo carril. Para poder ver al OA+FLM en movimiento, se proporciona un vídeo ilustrativo[1].

5.2.2. Ejecución típica con tráfico

La ejecución típica *con* tráfico consiste en un total de 12 simulaciones, con la misma configuración que la ejecución típica sin tráfico, pero con un vehículo circulando delante del vehículo principal. En esta ejecución, el vehículo delantero siempre es el mismo modelo y se desplaza haciendo uso del piloto automático de CARLA y circulará lo más lento posible para molestar al vehículo principal. Ahora se evalúa el rendimiento del modelo en una simulación de vuelta completa, centrándonos específicamente en el mantenimiento del carril y la evitación de obstáculos.

La Tabla 2 pone de relieve el salto en el rendimiento del modelo OA+FL respecto al modelo PilotNet*.

Tabla 2: Métricas para dos ciudades y dos modelos diferentes en condiciones con tráfico

	Town01		Town02	
	PilotNet*	OA+FLM	PilotNet*	OA+FLM
Tasa de éxito (%)	0	**81**	0	**100**
MPD	43.12	**0.26**	50.57	**0.32**
Invasión de carril	28.87	**6.54**	69.65	**1.48**

Para cada kilómetro, se puede ver que la tasa de éxito es bastante alta en la ciudad donde se entrenaron los modelos, Town02. Una vez que llegamos a la ciudad de pruebas, Town01, el porcentaje de éxito desciende para ambos modelos, pero el modelo OA+FL sigue llevando la ventaja.

MPD también ofrece información sobre el rendimiento de los modelos. Muestra la desviación media de la posición del vehículo respecto al centro del carril. Como se puede ver en la Tabla 2, el modelo OA+FL supera al modelo PilotNet* con un comportamiento significativamente más controlado y suave a la hora de mantenerse centrado en su carril.

Y es que es importante destacar que el modelo PilotNet* nos da una tasa de éxito del 0 %, pues no es capaz de frenar cuando se encontraba un vehículo delante. En contraste, el modelo OA+FLM logró una tasa de éxito del 81 % en el entorno Town01 y del 100 % en el entorno Town02, pues sí que podía afrontar las frenadas y reanudaciones de la marcha cuando se encontraba un vehículo delante. Estos resultados, visibles en la segunda parte del vídeo[1], vuelven a mostrar la capacidad del modelo OA+FLM para detectar y responder adecuadamente a la presencia de otros vehículos en la vía a la vez que se mantiene dentro de su carril, respetando así las normas de tráfico.

5.2.3. Generalización para diferentes vehículos delanteros

En la evaluación de la generalización no fue necesario hacer que el coche diera una vuelta entera. En su lugar, se probó el vehículo principal con seis vehículos diferentes que se

[1] https://www.youtube.com/watch?v=TxYeBWp0XN8&t=3s

movían a un ritmo significativamente lento, deteniéndose en los semáforos y reanudando la marcha cuando el semáforo se ponía en verde. Se realizaron un total de 16 experimentos, con cuatro coches diferentes situados en cuatro posiciones distintas dentro de la simulación para cada modelo.

La distancia entre el vehículo principal y el vehículo delantero se divide en cuatro secciones, como se observó en la Figura 37: distancia larga (20-50 metros), distancia media (15-20 metros), distancia corta (6-15 metros) y distancia peligrosa (0-6 metros). La distancia peligrosa debe evitarse en la simulación, ya que se considera inaceptable en escenarios reales. Respetar estas distancias es crucial para una conducción segura en condiciones de tráfico.

La Tabla 3 presenta el ratio de distancia, es decir, qué porcentaje del kilómetro total ha recorrido el vehículo principal a diferentes distancias en relación con el coche delantero, lo que permite evaluar el comportamiento del vehículo principal ante la presencia de un vehículo delantero. Observando la Tabla 3, de un primer vistazo se puede apreciar que el OA+FLM hace un mejor trabajo que PilotNet* al estar menos tiempo a una distancia peligrosa del coche delantero.

Además, si se analiza con más detalle, se puede ver de qué forma el vehículo está pasando de una distancia larga a una corta examinando los valores. El OA+FLM muestra ratios progresivos, pasando más tiempo en distancias largas y medias con el coche delantero y reduciendo su distancia a medida que se va acercando, lo que indica un comportamiento de parada correcto. Por el otro lado, PilotNet* muestra un tiempo de permanencia mínimo en la distancia media. Carece de avance progresivo, acercándose probablemente a una velocidad constante sin reducir ni tener en cuenta el obstáculo delantero.

Tabla 3: Métricas para la distancia con respecto al vehículo delantero.

	Town01		Town02	
	PilotNet*	OA+FLM	PilotNet*	OA+FLM
Distancia peligrosa	7 %	**4 %**	17 %	**0 %**
Distancia corta	17 %	21 %	20 %	4 %
Distancia media	9 %	38 %	13 %	22 %
Distancia larga	67 %	37 %	50 %	73 %

Se proporciona un vídeo[2] para enseñar las capacidades de generalización, del OA+FLM, ante diferentes vehículos delanteros.

[2] https://www.youtube.com/watch?v=mVSfxQeWwrQ

Capítulo 6

Conclusiones

En este último capítulo se comentarán los diferentes objetivos conseguidos a lo largo del desarrollo del TFM junto con algunas líneas de trabajo futuras que se proponen.

6.1. Recapitulación de objetivos

En este trabajo se ha presentado una propuesta para una conducción autónoma segura en situaciones *con* tráfico, siguiendo un enfoque extremo a extremo basado en visión con aprendizaje profundo haciendo uso del aprendizaje por imitación. El objetivo general, que se comentó en la sección Objetivos, se ha conseguido al lograr implementar satisfactoriamente un cerebro para un vehículo autónomo que garantice sus capacidades de mantenerse dentro de su respectivo carril y de detenerse y proseguir la marcha en caso de que haya otros vehículos en frente suyo.

En primer lugar, se evaluó y se configuró el entorno del vehículo autónomo usando el simulador de CARLA. Aprendiendo a configurar correctamente el autopiloto de carla, se consiguió tener un experto para que funcionara en diferentes ciudades de ejemplo, véase la sección Agente experto.

Se generaron varios conjuntos de datos supervisados con una gran cantidad de imágenes tomadas por la cámara a bordo y los correspondientes comandos de control ordenados por el experto piloto automático mientras el coche circulaba en carretera vacía y con tráfico dentro del simulador CARLA, tanto con un solo modelo de vehículo delantero como con modelos variados.

Se ha desarrollado un modelo neuronal basado en PilotNet del que se habla en detalle en la sección Modelo neuronal. Este incluye como extra algunas capas de *dropout*, la velocidad previa del coche como entrada junto con la imagen obtenida por la cámara frontal y el ángulo de giro y aceleración/freno como salida. El modelo se entrenó con el conjunto de datos supervisados. Gracias a esto, es capaz de percibir satisfactoriamente su entorno a partir de las imágenes y de realizar una correcta toma de decisiones para mantenerse en el carril en condiciones sin y con tráfico.

Más allá de los bajos valores de la función de pérdida en el conjunto de datos de prueba, visible en la etapa de entrenamiento en la sección Entrenamiento, el sistema ha sido validado experimentalmente en el simulador CARLA, en varias ciudades, y utilizando métricas objetivas, holísticas y cuantitativas, como son por ejemplo la desviación media

de la posición del coche respecto al centro del carril, invasiones de carril y distancia al vehículo delantero.

Los resultados experimentales, enseñados en la sección Validación experimental, muestran que el modelo conduce con éxito y seguridad el coche en condiciones con tráfico, sin perder rendimiento en condiciones donde la carretera está vacía. También es capaz de mantener la distancia de seguridad con los coches delanteros e incluso generaliza adecuadamente a varios tipos de vehículos delanteros sin restringirse a los de una apariencia concreta.

6.2. Futuros trabajos

Como línea de futuro estamos trabajando para ampliar el planteamiento de extremo a extremo para que también se ocupe con éxito de los semáforos y los cruces. El caso de los semáforos, una buena idea sería hacer uso de redes de detección de imágenes, las cuales permitirían detectar explícitamente los semáforos, en caso de que se encuentren en el rango de la cámara, para ver si está en rojo o si está en verde. Con el fin de seguir la línea del TFM, se podría una vez más hacer uso del enfoque extremo a extremo. Este enfoque contaría con otro nuevo dataset donde se enseñaría a la red a detectar los semáforos en las imágenes y según su color, predecir si tiene que frenar o si puede continuar con su marcha.

En el caso de los cruces, ahora mismo el modelo no está tratando los cruces como debería. Si bien no se sale de ellos y trata de mantenerse lo más recto posible hasta volver a encontrar la línea separadora de los carriles, el fin sería aumentar la complejidad a la hora de afrontarlos, siguiendo las reglas de tráfico de cómo tomarlas, respetando posibles vehículos que estén en ella y pudiendo ordenar al vehículo que prosiga su recorrido recto cogiendo una salida concreta del cruce.

Bibliografía

[1] SAE International. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. [Online; accessed 03-July-2023]. 2021. URL: https://www.sae.org/standards/content/j3016_202104/ (vid. pág. 1).

[2] Jon Fingas. *Waymo trials fully driverless rides in San Francisco*. [Online; accessed 03-July-2023]. 2022. URL: https://www.engadget.com/waymo-fully-driverless-rides-san-francisco-183703989.html (vid. pág. 2).

[3] Rob Lenihan. *GM Brings Robotaxis to These Major U.S. Cities*. [Online; accessed 03-July-2023]. 2023. URL: https://www.thestreet.com/electric-vehicles/gm-brings-robotaxis-to-these-major-u-s-cities (vid. pág. 2).

[4] Lei Kang. *Baidu's robotaxi platform Apollo Go gets permit to offer fully driverless rides in Beijing*. [Online; accessed 03-July-2023]. 2023. URL: https://cnevpost.com/2023/03/17/baidus-apollo-go-gets-permit-fully-driverless-rides-beijing/ (vid. pág. 2).

[5] Kyle Wiggers. *Waymo's autonomous cars have driven 20 million miles on public roads*. [Online; accessed 03-July-2023]. 2020. URL: https://venturebeat.com/2020/01/06/waymos-autonomous-cars-have-driven-20-million-miles-on-public-roads/ (vid. pág. 2).

[6] Inc. Tesla. *Tesla Vehicle Safety Report*. [Online; accessed 03-July-2023]. 2018. URL: https://www.tesla.com/es_ES/VehicleSafetyReport (vid. pág. 2).

[7] Ethan Sacks. *Self-driving Uber car involved in fatal accident in Arizona*. [Online; accessed 03-July-2023]. 2018. URL: https://www.nbcnews.com/tech/innovation/self-driving-uber-car-involved-fatal-accident-arizona-n857941 (vid. pág. 2).

[8] GreyB. *Top 30 Self Driving Technology and Car Companies*. [Online; accessed 03-July-2023]. 2021. URL: https://www.greyb.com/autonomous-vehicle-companies/ (vid. págs. 2, 3).

[9] Abhay S , Sonia M. *Autonomous Vehicle Market Statistics 2030*. [Online; accessed 03-July-2023]. 2021. URL: https://www.alliedmarketresearch.com/autonomous-vehicle-market (vid. pág. 3).

[10] Lei Tai, Shaohua Li y Ming Liu. "A deep-network solution towards model-less obstacle avoidance". En: *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2016, págs. 2759-2764 (vid. pág. 3).

[11] Ping Wu et al. "Vision-based robot path planning with deep learning". En: *Computer Vision Systems: 11th International Conference, ICVS 2017, Shenzhen, China, July 10-13, 2017, Revised Selected Papers 11*. Springer. 2017, págs. 101-111 (vid. pág. 3).

[12] New Jersey Institute of Technology. *Autonomous Agents*. [Online; accessed 03-July-2023]. 2020. URL: https://pantelis.github.io/cs677/docs/common/lectures/autonomous-cars/ (vid. pág. 4).

[13] Behavior Metrics contributors. *Behavior Metrics*. [Online; accessed 03-July-2023]. 2022. URL: https://github.com/JdeRobot/BehaviorMetrics (vid. págs. 5, 21, 39).

[14] UAH. *Investigadores de la Universidad de Alcalá desarrollan un sistema para la mejora de la conducción autónoma*. [Online; accessed 03-July-2023]. 2021. URL: https://portalcomunicacion.uah.es/diario-digital/actualidad/investigadores-de-la-universidad-de-alcala-desarrollan-un-sistema-para-la-mejora-de-la-conduccion-autonoma.html (vid. pág. 6).

[15] Ignacio Alonso et al. "The Experience of DRIVERTIVE-DRIVERless cooperaTIve VEhicle-Team in the 2016 GCDC". En: *IEEE Transactions on Intelligent Transportation Systems* PP (oct. de 2017), págs. 1-13. DOI: 10.1109/TITS.2017.2749963 (vid. pág. 6).

[16] MA Sotelo et al. "Low level controller for a POMDP based on WiFi observations". En: *Robotics and Autonomous Systems* 55.2 (2007), págs. 132-145 (vid. pág. 7).

[17] Leonardo Cabrera Lo Bianco et al. "Joint semantic segmentation of road objects and lanes using Convolutional Neural Networks". En: *Robotics and Autonomous Systems* 133 (2020), pág. 103623 (vid. págs. 7, 8).

[18] Álvaro Sáez et al. "Real-time semantic segmentation for fisheye urban driving images based on ERFNet". En: *Sensors* 19.3 (2019), pág. 503 (vid. pág. 8).

[19] Felipe Codevilla et al. "End-to-end driving via conditional imitation learning". En: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, págs. 4693-4700 (vid. págs. 8, 9, 12).

[20] Jiajun Fan y Changnan Xiao. "Generalized data distribution iteration". En: *arXiv preprint arXiv:2206.03192* (2022) (vid. pág. 9).

[21] Jason Ku et al. "Joint 3d proposal generation and object detection from view aggregation". En: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, págs. 1-8 (vid. págs. 9, 10).

[22] Adrià Puigdomènech Badia et al. "Agent57: Outperforming the atari human benchmark". En: *International conference on machine learning*. PMLR. 2020, págs. 507-517 (vid. pág. 10).

[23] Mariusz Bojarski et al. "End to End Learning for Self-Driving Cars". En: *arXiv preprint arXiv:1604.07316* (2016) (vid. págs. 10, 29).

[24] Rodrigo Gutiérrez-Moreno et al. "Reinforcement Learning-Based Autonomous Driving at Intersections in CARLA Simulator". En: *Sensors* 22.21 (2022), pág. 8373 (vid. pág. 11).

[25] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". En: *nature* 529.7587 (2016), págs. 484-489 (vid. pág. 11).

[26] Faraz Torabi, Garrett Warnell y Peter Stone. "Behavioral cloning from observation". En: *arXiv preprint arXiv:1805.01954* (2018) (vid. pág. 11).

[27] Tim Pearce y Jun Zhu. "Counter-Strike Deathmatch with Large-Scale Behavioural Cloning". En: *2022 IEEE Conference on Games (CoG)*. IEEE. 2022, págs. 104-111 (vid. pág. 11).

[28] SmartLab AI. *A brief overview of Imitation Learning*. [Online; accessed 03-July-2023]. 2019. URL: https://smartlabai.medium.com/a-brief-overview-of-imitation-learning-8a8a75c44a9c (vid. pág. 12).

[29] Divyansh Garg et al. "Iq-learn: Inverse soft-q learning for imitation". En: *Advances in Neural Information Processing Systems* 34 (2021), págs. 4028-4039 (vid. pág. 12).

[30] Jonathan Ho y Stefano Ermon. "Generative Adversarial Imitation Learning". En: *Advances in neural information processing systems* 29 (2016) (vid. pág. 12).

[31] Anyverse. *Tesla bets on the physically correct synthetic data at its AI Day*. [Online; accessed 03-July-2023]. 2021. URL: https://anyverse.ai/news/tesla-ai-day-physically-correct-synthetic-data/ (vid. pág. 13).

[32] Pablo Alvarez Lopez et al. "Microscopic Traffic Simulation using SUMO". En: *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. 2018, págs. 2575-2582. DOI: 10.1109/ITSC.2018.8569938 (vid. pág. 13).

[33] Waymo Team. *Simulation City: Introducing Waymo's most advanced simulation system yet for autonomous driving*. [Online; accessed 03-July-2023]. 2021. URL: https://waymo.com/blog/2021/06/SimulationCity.html (vid. pág. 13).

[34] Wayve Team. *Introducing Wayve Infinity Simulator*. [Online; accessed 03-July-2023]. 2022. URL: https://wayve.ai/thinking/introducing-wayve-infinity-simulator/ (vid. pág. 14).

[35] Jeffrey Hawke, Vijay Badrinarayanan, Alex Kendall et al. "Reimagining an autonomous vehicle". En: *arXiv preprint arXiv:2108.05805* (2021) (vid. pág. 14).

[36] Alexey Dosovitskiy et al. "CARLA: An open urban driving simulator". En: *Conference on robot learning*. PMLR. 2017, págs. 1-16 (vid. págs. 15, 27).

[37] Pei Sun et al. "Scalability in perception for autonomous driving: Waymo open dataset". En: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, págs. 2446-2454 (vid. pág. 16).

[38] Peng Wang et al. "The apolloscape open dataset for autonomous driving and its application". En: *IEEE transactions on pattern analysis and machine intelligence* (2019) (vid. pág. 16).

[39] Andreas Geiger et al. "Vision meets Robotics: The KITTI Dataset". En: *International Journal of Robotics Research (IJRR)* (2013) (vid. págs. 16, 17).

[40] Holger Caesar et al. "nuScenes: A multimodal dataset for autonomous driving". En: *CVPR*. 2020 (vid. pág. 17).

[41] Marius Cordts et al. "The cityscapes dataset for semantic urban scene understanding". En: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, págs. 3213-3223 (vid. pág. 18).

[42] Guido Van Rossum y Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697 (vid. pág. 22).

[43] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/ (vid. pág. 23).

[44] NVIDIA, Péter Vingelmann y Frank H.P. Fitzek. *CUDA, release: 10.2.89*. 2020. URL: https://developer.nvidia.com/cuda-toolkit (vid. pág. 23).

[45] G. Bradski. "The OpenCV Library". En: *Dr. Dobb's Journal of Software Tools* (2000) (vid. pág. 23).

[46] Pete Shinners. *PyGame*. 2011. URL: http://pygame.org/ (vid. pág. 23).

[47] Charles R. Harris et al. "Array programming with NumPy". En: *Nature* 585.7825 (sep. de 2020), págs. 357-362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2 (vid. pág. 24).

[48] Alexander Buslaev et al. "Albumentations: fast and flexible image augmentations". En: *Information* 11.2 (2020), pág. 125 (vid. pág. 34).