



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

 etsinf

Escola Tècnica
Superior d'Enginyeria
Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Coche autónomo robótico con reconocimiento de señales de tráfico utilizando aprendizaje profundo.

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Carles Andrés Solanes

Tutor: Carlos Monserrat Aranda

Curso 2023-2024

Resumen

La conducción autónoma es una tecnología actualmente en auge. Ya son muchos los proyectos de coches con sistemas de conducción automática. Aunque la mayoría de estos desarrollos no son coches con una autonomía total, ya hay varias flotas de coches totalmente autónomos, que están a prueba en diversas ciudades del mundo.

Los beneficios para la sociedad que los coches autónomos brindan son muchos, desde la comodidad y accesibilidad a personas con diversas discapacidades, hasta la seguridad vial y la reducción de la tasa de mortalidad en las carreteras. Por eso, es importante que la comunidad de ingenieros siga aportando en este ámbito y ayudando en el crecimiento de esta tecnología.

Este proyecto se centra en el desarrollo de un sistema de detección de señales de tráfico, un módulo esencial en todo sistema de conducción autónoma. Para ello, se utiliza un pequeño coche robótico equipado con una Raspberry Pi y una unidad de procesamiento tensorial. Partiendo de un modelo de detección de objetos preentrenado que ofrece la API de detección de objetos de TensorFlow y usando el entorno de programación Google Colab, se consigue entrenar un modelo de aprendizaje profundo que detecta un conjunto estipulado de señales de tráfico a miniatura.

Finalmente, mediante la ejecución de un programa de conducción Python, se consigue inferir el modelo entrenado y accionar el robot para interpretar las señales de tráfico en un entorno a pequeña escala y controlado. Como resultado de las pruebas realizadas, el robot es capaz de detectar todas las señales de tráfico que se encuentra, consiguiendo interpretarlas correctamente y recorriendo el circuito completamente sin salirse. Estos resultados se ven ligeramente alterados cuando las condiciones de luminosidad son adversas.

Palabras clave: aprendizaje profundo, conducción autónoma, visión artificial, redes neuronales convolucionales

Resum

La conducció autònoma és una tecnologia actualment en auge. Ja són molts els projectes de cotxes amb sistemes de conducció automàtica. Encara que la majoria d'aquests desenvolupaments no són cotxes amb una autonomia total, ja hi ha diverses flotes de cotxes totalment autònoms, que estan a prova a diverses ciutats del món.

Els beneficis per a la societat que els cotxes autònoms brinden són molts, des de la comoditat i accessibilitat a persones amb diverses discapacitats, fins a la seguretat vial i la reducció de la taxa de mortalitat a les carreteres. Per això, és important que la comunitat d'enginyers seguixca aportant en aquest àmbit i ajudant al creixement d'aquesta tecnologia.

Aquest projecte es centra en el desenvolupament d'un sistema de detecció de senyals de trànsit, un mòdul essencial en tot sistema de conducció autònoma. Per tal d'aconseguir-ho, s'utilitza un petit cotxe robòtic equipat amb una Raspberry Pi i una unitat de processament tensorial. Partint d'un model de detecció d'objectes preentrenat que facilita l'API de detecció d'objectes de TensorFlow y utilitzant l'entorn de programació Google Colab, s'aconsegueix entrenar un model d'aprenentatge profund que detecta un conjunt estipulat de senyals de trànsit a miniatura.

Finalment, mitjançant l'execució d'un programa de conducció Python, s'aconsegueix inferir el model entrenat i accionar el robot per tal d'interpretar les senyals de trànsit en un entorn a petita escala i controlat. Fruit de les proves realitzades, el robot és capaç de detectar totes les senyals de trànsit que es troba, aconseguint interpretar-les correctament i recorrent el circuit completament sense eixir-se'n. Aquests resultats es veuen lleugerament alterats quan les condicions de lluminositat son adverses.

Paraules clau: aprenentatge profund, conducció autònoma, visió artificial, xarxes neuronals convolucionals

Abstract

The emergence of autonomous driving has caused the increase of many projects with self-driving systems. Although most of these developments are not fully autonomous cars, there are already several fleets of fully automatic cars being tested in various cities around the world.

This has contributed many benefits to society, as comfort and accessibility to people with various disabilities, to road safety and the reduction of the death rate on roads. For this reason, it is important that the engineering community continues contributing in this area and helping in the growth of this technology.

This project focuses on the development of a traffic signs detection system, an essential module in any autonomous driving system. It uses a small robotic car equipped with a Raspberry Pi and a tensor processing unit. Starting from a pre-trained object detection model provided by the TensorFlow object detection API and using the Google Colab programming environment, a deep learning model is trained to detect a stipulated set of miniature traffic signs.

Finally, by running a Python driver program, the trained model is inferred and the robot is actuated to interpret the traffic signals in a small-scale and controlled environment. As a result of the tests carried out, the robot is capable of detecting all the traffic signs it encounters, interpreting them correctly and traveling the circuit completely without going off. These results are slightly altered when lighting conditions are adverse.

Key words: deep learning, autonomous driving, artificial vision, convolutional neural networks

Índice general

Índice general	IV
Índice de figuras	VI
1 Introducción	1
1.1 Motivación	1
1.2 Objetivos	2
1.3 Estructura de la memoria	3
2 Estado del arte	5
2.1 Coches autónomos en la actualidad	5
2.1.1 Evolución de los vehículos de conducción autónoma	5
2.1.2 Tecnologías detrás de los autopilotos	9
2.1.3 Principales obstáculos y complicaciones	11
2.2 La visión artificial	12
2.2.1 Definición y evolución	12
2.2.2 Principales técnicas y algoritmos	14
2.3 Reconocimiento de señales de tráfico	20
2.3.1 Contexto actual de las tecnologías de identificación de señales . .	20
2.3.2 Algoritmos y modelos de aprendizaje profundo	21
3 Diseño de la solución	23
3.1 Análisis del problema	23
3.2 Materiales requeridos para la implementación del proyecto	24
3.2.1 Señales de tráfico	25
3.2.2 <i>Hardware</i> utilizado	27
3.2.3 <i>Software</i> utilizado	30
4 Desarrollo de la solución	34
4.1 Puesta en funcionamiento y configuración del robot	34
4.2 Construcción del <i>dataset</i>	37
4.2.1 Recolección de imágenes	37
4.2.2 Etiquetado y anotación de los datos	39
4.2.3 Aumento de datos	41
4.3 Entrenamiento del modelo de detección de señales	43

4.3.1	Configuración del entorno	43
4.3.2	Tratamiento de los datos	43
4.3.3	Arquitectura del modelo y ajuste del proceso de entrenamiento . .	45
4.3.4	Pruebas con el conjunto de test y exportación del modelo	48
4.4	Elaboración del programa de conducción autónoma	50
4.4.1	Importación y coexistencia de ambos modelos	50
4.4.2	Programación del comportamiento del robot	51
5	Implantación y pruebas	54
5.1	Puesta en marcha en distintos circuitos	54
5.2	Ánalisis de los resultados obtenidos	55
6	Conclusiones	58
	Bibliografía	61

Índice de figuras

2.1	Niveles de conducción autónoma reconocidos por la SAE.	6
2.2	El automóvil radio controlado, American Wonder, 1925.	7
2.3	Vehículo autónomo que participó en DARPA Grand Challenge.	8
2.4	Taxi autónomo total de Waymo.	9
2.5	Esquema de los principales sensores de un coche Waymo.	10
2.6	Unidad de procesamiento tensorial (TPU) usada en este proyecto.	13
2.7	Eliminación de ruido mediante erosión y dilatación.	16
2.8	Extracción de descriptores HOG.	18
2.9	Extracción de textura con LBP.	19
3.1	Kit de 17 piezas de señales de tráfico de Lena.	25
3.2	Las 11 señales de tráfico utilizadas en este proyecto.	27
3.3	El coche robótico PiCar utilizado para este trabajo.	28
3.4	BlinkStick Strip Mini.	29
3.5	Make Sense AI, herramienta utilizada para la anotación de las imágenes de entrenamiento.	32
4.1	Configuración del escritorio remoto.	35
4.2	Configuración del sistema de archivos compartidos.	36
4.3	Ejemplos de <i>frames</i> capturados con distintos fondos e iluminaciones. . . .	39
4.4	Anotación de <i>frame</i> de entrenamiento usando Make Sense.	40
4.5	Otro ejemplo de anotación, con distinto fondo.	40
4.6	Volteado horizontal de la señal de sentido obligatorio derecho.	41
4.7	Ejemplos de <i>frames</i> generados para aumentar el <i>dataset</i>	42
4.8	Estructura de directorios para el manejo de datos.	44
4.9	Diagramas de alto nivel de la arquitectura del modelo SSD MobileNet V2.	45
4.10	Tabla elección hiperparámetros.	47
4.11	Gráfico del valor de <i>learning rate</i>	47
4.12	Gráficos de las funciones de pérdida.	48
4.13	Pruebas con el conjunto de test del modelo de detección de señales entrenado.	49
4.14	Pruebas con luz artificial del modelo de detección de señales entrenado. .	49

5.1 Los dos tipos de circuitos utilizados para poner a prueba la solución desarrollada.	54
5.2 Ejemplos de algunos circuitos usados en las pruebas con distintos suelos e iluminaciones.	55
5.3 El coche autónomo detecta y reacciona ante la señal de encendido de luces.	56
5.4 El coche autónomo se detiene al detectar la señal de <i>STOP</i>	56
5.5 El coche autónomo detecta y reacciona ante la señal de sentido obligatorio izquierda.	57

CAPÍTULO 1

Introducción

1.1 Motivación

Los coches autónomos están cada vez más a la orden del día. Y es que, son más y más las marcas, empresas automovilísticas y tecnológicas que se unen a la investigación, creación e implantación de coches que conducen solos. Estos automóviles detectan las líneas de la carretera, reaccionan ante multitud de objetos, señales de tráfico y, en definitiva, un sinfín de situaciones a las que se enfrenta un conductor.

No es de extrañar el éxito rotundo y mediático que estos vehículos tienen. Solo con ver los numerosos beneficios que brindan a la humanidad, nos podemos hacer una idea. Entre ellos destacan: la reducción de la tasa de mortalidad por accidentes de tráfico, causa de muerte muy frecuente en países desarrollados; la comodidad de los conductores; la accesibilidad en la conducción de personas con discapacidades físicas y, sobre todo, la seguridad que estos sistemas de conducción autónoma ofrecen.

Pero no todo son buenas noticias, a pesar de que la conducción autónoma y todas las técnicas que la engloban estén en pleno auge, no deja de ser un ámbito en constante investigación y evolución. Y es que, todavía no es una tecnología al alcance de todo el mundo. Por desgracia, hay muchas personas que no pueden desembolsar la cantidad de dinero que un coche autónomo cuesta. Otro problema es la regulación y actualización de las leyes en muchos países. Sin ir más lejos, en España solo se permite la circulación de vehículos con conducción autónoma de nivel 2, es decir, el conductor debe mantener las manos en el volante en todo momento. Según varios artículos, 2024 podría ser el año en que se permita la circulación de coches con conducción autónoma total [1].

El reconocimiento de objetos y, en concreto, el reconocimiento de señales de tráfico son técnicas que tienen un papel fundamental en todo sistema de conducción autónoma. Las señales de tráfico son los principales elementos en la vía pública que regulan el tránsito. Ayudan a los conductores, otorgándoles la información necesaria e instantánea, a saber

qué decisiones pueden o deben tomar para una correcta y segura circulación. Saber detectar una señal tan trivial como la de *STOP*, permite que dos vehículos no colisionen en un cruce. Esto quiere decir que desarrollar un sistema autónomo de conducción conlleva una gran responsabilidad, es primordial que este perciba e interprete correctamente todas las señales garantizando la seguridad vial.

Por todo lo mencionado en los anteriores párrafos, la motivación de este trabajo se encuentra en el afán por seguir investigando y desarrollando en este campo de los coches autónomos, más concretamente en la detección de señales de tráfico. Aunque los recursos son más limitados y modestos en comparación con los de las grandes multinacionales que aportan a este ámbito, el propósito es contribuir recatadamente al avance de esta tecnología tan puntera y prometedora.

1.2 Objetivos

La idea de realizar este proyecto parte de la propuesta de continuación de otro. El proyecto inicial en cuestión se titula “Construcción de un coche de conducción autónomo robótico” y fue desarrollado por Andrés Martínez Martínez en 2022, bajo la tutorización de Carlos Monserrat Aranda también. En este, se consiguió montar un pequeño coche con una Raspberry Pi 3 Modelo B+, un acelerador Coral Edge TPU de Google y un kit de ruedas, servomotores, placas y cámara. Además se logró entrenar un modelo que permitía al coche dotarlo de una autonomía parcial, haciendo que este pudiese conducir por cualquier circuito nunca antes visto sin salirse del carril.

Dicho esto y explicada en el apartado anterior la motivación de este trabajo, el **objetivo principal** es conseguir que el coche robótico pueda reconocer señales de tráfico y actuar en consecuencia, todo esto en un escenario a pequeña escala que se asemeje a la realidad. El circuito esta constituido por un único carril con curvas delimitado con dos cintas pegadas al suelo y por un conjunto de señales de tráfico a medida, las cuales se detallarán más adelante en el capítulo del diseño de la solución.

Muchas de las señales utilizadas en un entorno real, por razones obvias, no van a tener mucho sentido puestas en el entorno a escala reducida. Por ejemplo, la señal de prioridad en sentido contrario (R-5) o la de prohibido adelantar (R-305), no se pueden interpretar correctamente en un entorno donde solo hay un único carril con una dirección. Por otra parte, el número de señales existentes es muy elevado.

Teniendo en cuenta las limitaciones anteriores, se dará por alcanzado el objetivo si se cumple que:

1. El coche detecte un conjunto establecido de señales.

2. El coche sea capaz de actuar ante cualquiera de ellas de forma automática.
3. Toda la ejecución del programa principal, así como la inferencia de todos los modelos utilizados, se lleve a cabo en los equipos que conforman el coche.

Con el fin de conseguir lo mencionado y de estructurar y marcar los pasos a seguir, se ha decidido dividir el proyecto en diferentes **subobjetivos**:

1. Implementar un programa para la recolecta de *frames* y posterior creación de una base de datos de las señales de tráfico a entrenar.
2. Entrenar un modelo de inteligencia artificial usando aprendizaje profundo para la detección de señales de tráfico.
3. Desarrollar un programa principal para la conducción autónoma del robot.
4. Ejecutar la solución elaborada y realizar pruebas.

En resumen, se pretende mejorar la autonomía parcial que dispone el coche, dotándolo con el reconocimiento de algunas señales. Por tanto, se asumirán los objetivos conseguidos cuando el coche robótico sea capaz de transitar por un recorrido sin salirse del mismo e interpretando todas las señales que se encuentre.

1.3 Estructura de la memoria

En esta sección se pretende desglosar el presente documento y esclarecer resumidamente el contenido de cada capítulo de la memoria.

Primeramente, cabe resaltar que este trabajo se puede dividir en tres partes. La primera, de introducción y situación actual de la tecnología y tema, la componen los primeros dos capítulos. Seguidamente, la parte de diseño y desarrollo de la solución, formada por los capítulos tercero y cuarto. Y, finalmente, una última parte donde se pone en funcionamiento la solución, se realizan pruebas, se analizan los resultados y se concluye el proyecto. Esta última parte abarca el quinto y sexto capítulo.

Hecha esta división, veamos el contenido de cada capítulo. Empezando por el capítulo 2, en este se realiza una investigación del estado del arte tanto de la conducción autónoma como de la visión artificial. Detallando las principales tecnologías y técnicas usadas en estos ámbitos y exponiendo las limitaciones y problemáticas actuales. El final de esta sección se enfoca en el reconocimiento de objetos, haciendo hincapié en el reconocimiento de señales de tráfico, exponiendo las herramientas más usadas y las aplicaciones más punteras.

A continuación, en el capítulo 3, se analiza la problemática del proyecto en profundidad y se obtienen los desafíos del mismo. Posteriormente, se va explicando la forma que la solución va adquiriendo, justificando qué problema solventa cada decisión de diseño. Una vez se tiene el esqueleto de la solución, en el capítulo 4, se describe detalladamente todo el proceso realizado hasta llegar a obtenerla; exponiendo como se ha resuelto todos los contratiempos encontrados. Entre otras cosas, en este apartado se explicará cómo se ha construido la base de datos, cómo se ha logrado entrenar el modelo y cómo se ha conseguido la implementación del programa principal de conducción, aportando la información necesaria que justifique el desarrollo de todo lo mencionado.

En el capítulo 5 se documenta la puesta en ejecución de la aplicación y la realización de distintas pruebas en diferentes circuitos, analizando los resultados obtenidos. Finalmente, en el capítulo 6, se exponen las conclusiones de todo el trabajo realizado, así como de la solución obtenida y los objetivos alcanzados. También se proponen futuras ideas de mejora y flancos abiertos pendientes de futuras continuaciones del proyecto.

CAPÍTULO 2

Estado del arte

2.1 Coches autónomos en la actualidad

2.1.1. Evolución de los vehículos de conducción autónoma

El campo de los coches autónomos, al igual que otros muchos inventos de la ingeniería, ha evolucionado en función de las nuevas técnicas y conocimientos que han ido surgiendo y progresando a lo largo de la historia. Para adquirir una buena visión acerca de los vehículos de conducción autónoma tal y como los conocemos hoy en día, es necesario investigar un poco sobre los orígenes de esta tecnología. De esta forma podremos entender mejor cómo han ido evolucionando.

Pero antes de hacer este repaso hasta la actualidad, es importante aclarar cómo se clasifican internacionalmente los coches con funcionalidades autónomas. La Sociedad de Ingenieros de la Automoción (SAE) distingue en 5 niveles la capacidad autónoma de los vehículos. Siendo nivel 0 un vehículo sin ningún tipo de automatización en la conducción [8].

- **Nivel 1: asistencia al conductor.** Vehículos que disponen de algún sistema que controle el movimiento lateral (giro del volante) o longitudinal (frenado o aceleración), pero nunca los dos a la vez. Por ejemplo, un sistema de aparcamiento automático donde se controle el volante, pero el conductor dirige los pedales.
- **Nivel 2: automatización parcial de la conducción.** Se controla el movimiento lateral y el longitudinal. El conductor es el responsable de la conducción y los coches no están preparados para tomar decisiones en la detección de objetos. En este lugar de la clasificación entran los coches Tesla.
- **Nivel 3: automatización condicional de la conducción.** El vehículo es capaz de realizar todas las funciones que la conducción requiere. Aunque tiene algunas limi-

taciones, como por ejemplo, que no sepa reaccionar ante situaciones de riesgo. El conductor no tiene que estar atento de la conducción aunque sí cuando el sistema le avise para retomar la conducción ante una situación adversa.

- **Nivel 4: alta automatización de la conducción.** Al igual que en el nivel 3, el vehículo puede conducir perfectamente solo, pero ahora sin esperar que el conductor reaccione ante una petición de retorno del mando. El conductor voluntariamente podrá elegir si se activa o desactiva la conducción autónoma.
- **Nivel 5: automatización total de la conducción.** Se puede prescindir del conductor, el coche realiza todas las acciones que un humano desempeña al conducir.

A modo de resumen y esquema visual, a continuación se proporciona una imagen que reúne todo lo mencionado anteriormente.

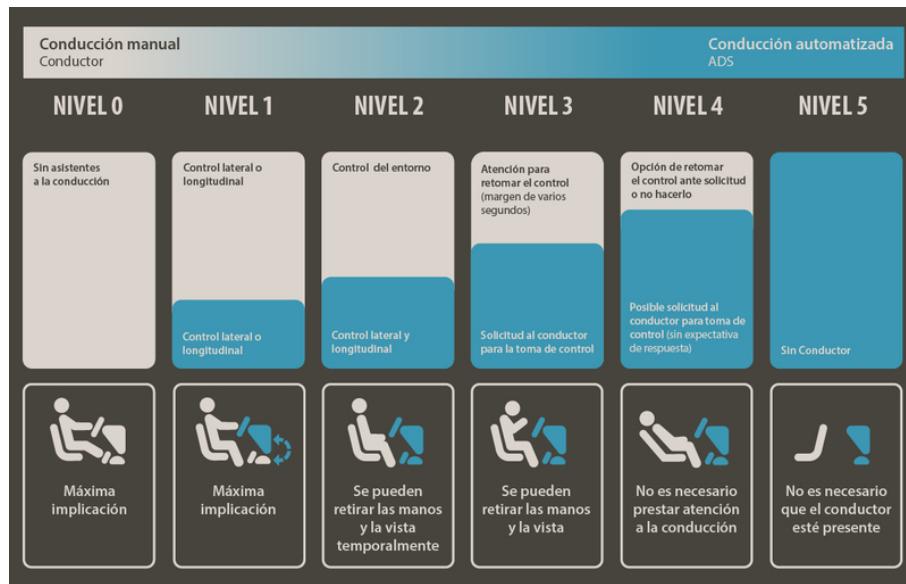


Figura 2.1: Niveles de conducción autónoma reconocidos por la SAE.

Fuente: [8].

Fue en el año 1925 que se fabricó el primer automóvil sin conductor. Un ingeniero eléctrico del ejército estadounidense, el neoyorquino Francis Houdina, desarrolló un coche que se podía conducir de forma remota mediante señales de radio. El vehículo se controlaba desde otro automóvil que iba detrás escoltándolo, como si de un coche de juguete teledirigido se tratase. Era un sedán Chandler que estaba equipado con una antena y unos pequeños motores eléctricos que cuando actuaban podían controlar la dirección del vehículo.

El coche fue bautizado con el nombre de American Wonder y se puso a prueba en varias ciudades en distintas ocasiones. Incluso llegó a chocar contra otro sedán cuando se conducía de manera remota por la Quinta Avenida, Nueva York.

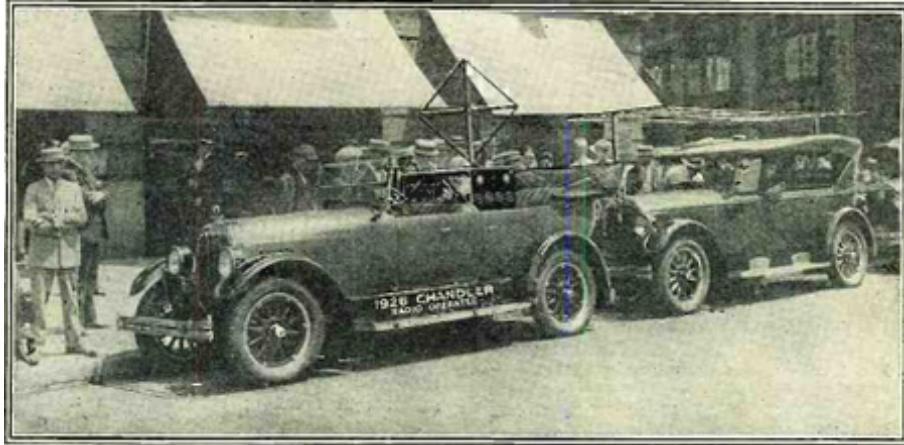


Figura 2.2: El automóvil radio controlado, American Wonder, 1925.

Fuente: [2].

Aunque esta tecnología de conducción por ondas de radio no es utilizada en los automóviles autónomos actuales, fue la que impulsó y motivó a las empresas e ingenieros de la época a interesarse e investigar en la conducción sin piloto. Posteriormente se desarrollaron más vehículos con el mismo sistema de radiocontrol, ya que era la técnica más avanzada y popular que había en esa época. Finalmente se quedó en el olvido, la gente dejó de verlo como un invento útil, más bien era una manera de entretenimiento. El sistema dependía mayoritariamente de la intervención humana.

En los años 60, el afán por continuar investigando en los coches sin conductor seguía vivo. Los ingenieros de la época dejaron de lado el radiocontrol y se interesaron en los sistemas eléctricos. Se empezó a estudiar la conducción autónoma enfocándose en la visión por ordenador. Pero no fue hasta las décadas de los 80 y 90 que esta tecnología empezó a prosperar.

Uno de los proyectos más relevantes en este campo fue desarrollado por la organización internacional europea Eureka. El proyecto se tituló Prometheus y fue financiado por los estados miembros de dicha organización. Tuvo lugar entre 1987 y 1995, culminando con varios vehículos prototípicos que recorrieron miles de quilómetros por París, Múnich y Copenhague. Estos modelos eran capaces de detectar carriles y otros coches mediante visión por computadora. También contaban con múltiples sensores que les permitía medir el frenado, la temperatura, el ángulo de giro y la aceleración [3]. Este proyecto demostró el gran potencial que tenía esta técnica de visión por computadora.

En los Estados Unidos también se interesaron por la visión computarizada como técnica base en el desarrollo de coches sin piloto. Llegando a sacar leyes para la investigación e inversión de millones de dólares en la tecnología.

Entre los diferentes avances en estudios destaca el proyecto Navlab, perteneciente a la universidad Carnegie Mellon. En este se consiguió construir un coche semiautónomo

funcional. También despunta el concurso que la Agencia de Proyectos de Investigación Avanzados de Defensa (DARPA), Darpa Grand Challenge, impulsó [4]. Esta competición consistía en una carrera de vehículos autónomos que tenían que ir de un lugar a otro de los EE. UU. pasando por puntos intermedios. En 2004, se celebró la primera edición y nadie consiguió llegar a la meta. No obstante, en 2005, algunos coches consiguieron llegar al final del recorrido [4]. Estos proyectos contribuyeron en gran parte al avance de los coches autónomos con visión por computador.



Figura 2.3: Vehículo autónomo que participó en DARPA Grand Challenge.
Fuente: [5].

En la actualidad, como bien se ha expuesto en el anterior capítulo, hay muchas organizaciones y grandes multinacionales que desarrollan coches autónomos. Entre las cuales destacan Tesla, Waymo, Toyota, Cruise de General Motors, Ford, entre otros [6].

Los vehículos de Tesla son los más populares y vendidos del mercado actual con tecnología autónoma. Ofrecen un sistema de piloto automático que asiste al conductor de una manera avanzada proporcionando seguridad y comodidad al volante. También cuenta con un sistema de piloto automático avanzado, el cual se puede obtener pagando un extra y habilitando el paquete, como si de una actualización *software* se tratase. Finalmente también ofrece otro paquete llamado capacidad de conducción autónoma total, no obstante, el nombre no le hace justicia, ya que solamente incorpora algunas funcionalidades adicionales como el control de semáforos y señales de *STOP*, que están en fase beta. Aún queda trabajo pendiente para conseguir un nivel de autonomía total (nivel 5).

Waymo, empresa filial de Alphabet, empezó siendo un proyecto de coche autónomo de Google. Actualmente, desde marzo de 2024, ha comenzado a desplegar en distintas ciudades de Estados Unidos una flota de taxis que tienen un nivel de autonomía total, es decir, sin conductor. Primeramente, el servicio que se ofrecía era gratis, a modo de promoción y presentación de la tecnología, aunque a partir de abril pasó a ser de pago.

La población puede llamar a estos taxis mediante una aplicación móvil. El coche pasa a buscar al cliente y lo lleva a su destino calculando las rutas más óptimas [7].

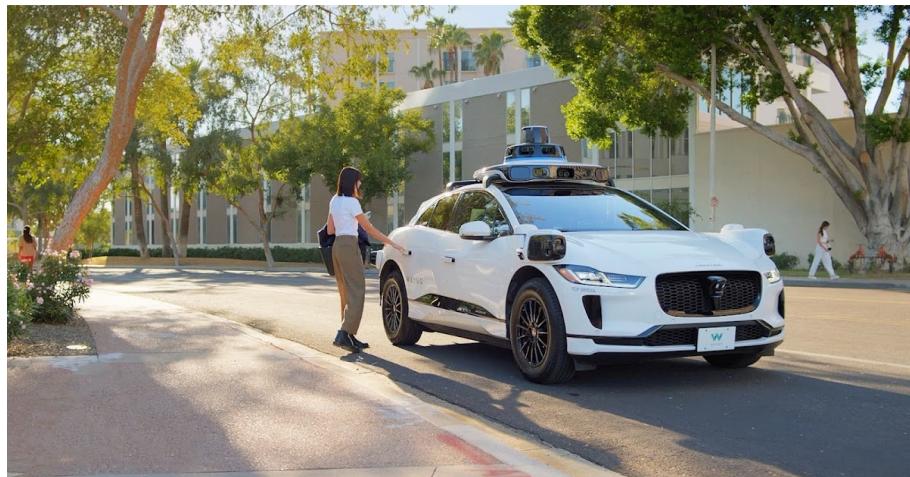


Figura 2.4: Taxi autónomo total de Waymo.

Fuente: Waymo.

AutoX y Baidu son dos grandes empresas del sector que también ofrecen el mismo servicio de taxi autónomo pero en China. No obstante, a estos vehículos se les reconoce un nivel 4 de autonomía.

Como se puede ver, ya se ha llegado a desarrollar la tecnología hasta tal punto de conseguir el máximo nivel de conducción autónoma. Aunque, por el momento, es solo una red de taxis en unas ciudades específicas, en un futuro no muy lejano, se empezarán a ver coches particulares totalmente autónomos comercializándose por todo el mundo. Y es que, la movilidad, tal y como la conocemos, está a punto de cambiar por completo. Como se ha mencionado en la introducción, en España se espera que para este año 2024 se publique el real decreto que permita la conducción de vehículos autónomos nivel 4 y 5. Por el momento, se seguirá investigando y mejorando la tecnología para que esto ocurra.

2.1.2. Tecnologías detrás de los autopilotos

Una vez hecho el repaso de cómo se ha evolucionado hasta los actuales coches autónomos, toca focalizarse en la tecnología que estos equipan. Para ello debemos distinguir entre los equipos *hardware* y los componentes *software*.

En cuanto a los componentes físicos, estos coches destacan por la cantidad de sensores de todo tipo que incorporan. Estos dispositivos tienen un papel fundamental en la percepción del entorno real de conducción y la detección de situaciones peligrosas. Es primordial que el sistema obtenga estos datos en tiempo real y de manera precisa, ya que son importantes a la hora de tomar las decisiones que la conducción requiere.

El **sistema de cámaras**, son los “ojos” principales de los coches autónomos. Hoy en día esta tecnología está muy desarrollada, por tanto, los principales retos son decidir el número y el lugar estratégico donde ponerlas y el procesamiento *software* de las imágenes para poder interpretarlas correctamente.

Unos dispositivos muy utilizados, sobre todo en el desarrollo de vehículos con piloto autónomo de nivel 4 y 5, son los **sensores LIDAR**. Basado en tecnología láser, estos emiten haces láser, de tal forma que para cada pulso emitido se calcula el tiempo de retraso entre la señal emitida y la reflejada. De esta manera se puede detectar la distancia a objetos lejanos y poder generar uno mapas muy detallados del entorno.

El uso de **radares** permite dotar a estos vehículos de sistemas de detección y seguimiento de objetos muy precisos. Con esta tecnología basada en ondas electromagnéticas el coche es capaz de detectar otros vehículos u objetos, incluso la dirección y velocidad que estos llevan. Son dispositivos baratos y precisos en condiciones de difícil visibilidad.

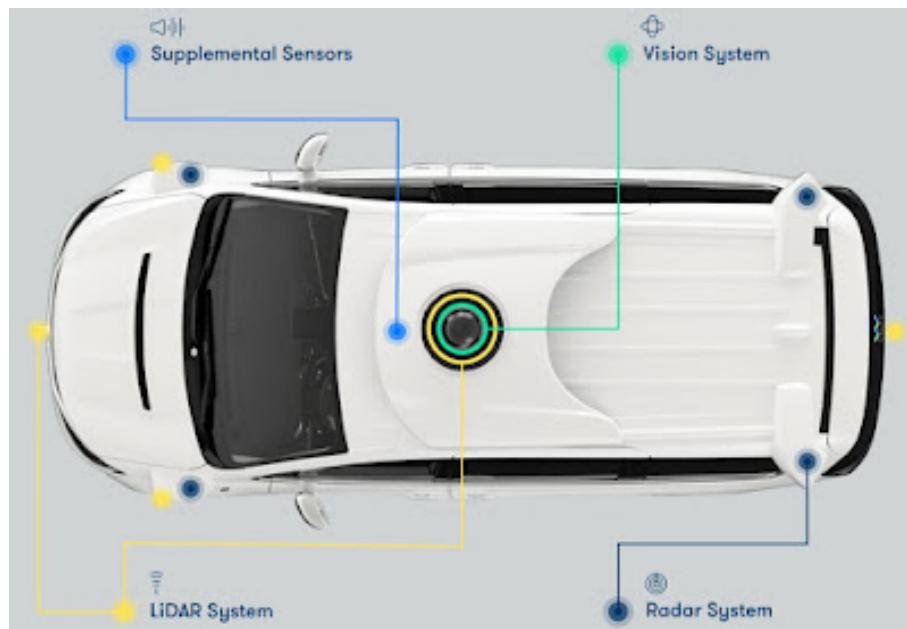


Figura 2.5: Esquema de los principales sensores de un coche Waymo.
Fuente: Waymo.

Los **sensores ultrasónicos**, encargados de detectar objetos cercanos al coche, emiten unas ondas sonoras imperceptibles que rebotan con los elementos del entorno y regresan. Calculando el tiempo de retorno se averigua la distancia de dicho elemento. Estos sensores son muy utilizados en los coches con sistemas de aparcamiento autónomo.

También se encuentran **sensores de movimiento y giroscopio**, que estiman la posición, dirección y velocidad del vehículo. **Sensores infrarrojos** para la detección de objetos en condiciones de baja luminosidad o adversas para las cámaras.

Por último, los **sistemas de posicionamiento y navegación por satélite** también tienen un papel muy importante en la conducción automática. Actualmente se utilizan varios, unos más precisos que otros, dependiendo de la finalidad. Estos son **GPS** (Global Positioning System), **GLONASS** (Global Navigation Satellite System) y **Galileo**, siendo este último 5 veces más preciso que los anteriores.

Pero todos los datos que estos equipos físicos captan necesitan herramientas informáticas que los interpreten. El progreso de tecnologías como la **inteligencia artificial** ha mejorado enormemente el nivel de autonomía de estos coches. Todo este avance de los últimos años ha sido posible gracias a la **IA y el aprendizaje automático**, tecnologías que se consideran fundamentales en este campo. Sin ir muy lejos, el sistema de detección de peatones de los taxis de Waymo mencionados anteriormente, se vio mejorado de manera sobresaliente consiguiendo reducir cien veces la tasa de error [3].

2.1.3. Principales obstáculos y complicaciones

Como ya se ha mencionado anteriormente, aún queda mucho que mejorar para que estos coches automáticos se produzcan a otro nivel y, por tanto, se comercialicen masivamente por todo el mundo, llegando incluso a reemplazar los vehículos actuales. Existen una serie de desafíos pendientes de mejorar en este campo.

En primer lugar, complicaciones que tienen que ver con aspectos tecnológicos. Como la **percepción en situación de condiciones meteorológicas difíciles**, como cuando hay mucha niebla, llueve fuertemente o incluso cuando nieva y parte de la señalización del pavimento no es visible. Se tiene que seguir mejorando la fusión sensorial para desarrollar modelos más precisos. Otro gran obstáculo técnico son las **situaciones complejas o de peligro en la conducción**, y más si se trata de una decisión que se tiene que tomar en cuestión de milisegundos. Para eso hay que continuar investigando y contribuir al avance de algoritmos de decisión.

En cuanto a retos relacionados con la legislación, el principal es la **necesidad de un nuevo marco legal que regule la operatividad y responsabilidad de estos coches**. ¿Qué pasa cuando un vehículo autónomo tiene un accidente? ¿Qué acciones debe tomar? ¿Quién tiene la culpa? Todas estas cuestiones y muchas más necesitan ser aclaradas en términos de la legalidad. También hace falta una **regulación de seguridad específica para vehículos sin conductor**.

Otros obstáculos que preocupan a los ingenieros son los **dilemas éticos y de moralidad**. ¿Qué pasa cuando se tenga que tomar una decisión poco ética, o cuando se tenga que elegir entre dos malas situaciones? El **uso con malas intenciones** o las **brechas de seguridad** de esta tecnología, son problemas que también inquietan.

Finalmente, la mejora de la **coexistencia con los conductores y peatones humanos** es fundamental para poder encaminar la transición a los coches autónomos. Así como la **necesidad de una adaptación global de la infraestructura** actual, mejorando las vías en malas condiciones, sobre todo las vías rurales, y la **adaptación de las señales de tráfico** para facilitar la correcta detección de los sensores.

2.2 La visión artificial

2.2.1. Definición y evolución

La visión artificial, conocida también como visión por computadora, es un campo de la inteligencia artificial que permite a las máquinas interpretar y comprender el mundo visual a través de imágenes y videos. Este campo ha experimentado un crecimiento significativo en las últimas décadas debido al **desarrollo de algoritmos avanzados** y al **aumento de la capacidad de procesamiento de los computadores**. Los avances en la visión artificial han permitido su aplicación en una amplia variedad de campos, desde la robótica y la medicina hasta la seguridad y la industria automotriz. La visión artificial se basa en principios de procesamiento de imágenes, donde las computadoras analizan datos visuales para identificar patrones y realizar tareas específicas, como el reconocimiento de objetos, la detección de anomalías y la navegación autónoma.

El **objetivo principal de la visión artificial** es dotar a las máquinas de la **capacidad de ver y entender su entorno** de manera similar a **como lo hace el ser humano**. Para lograr esto, se utilizan técnicas de procesamiento de imágenes que transforman los datos visuales en información procesable. Los algoritmos de aprendizaje profundo, especialmente las redes neuronales convolucionales (CNN), juegan un papel crucial en este proceso. Las CNN han demostrado ser extremadamente eficaces para tareas como la clasificación de imágenes y el reconocimiento de objetos, debido a su capacidad para aprender características jerárquicas a partir de los datos de entrada. Estas redes neuronales se entrenan utilizando grandes conjuntos de datos etiquetados, lo que les permite identificar y clasificar patrones complejos con alta precisión.

La **incorporación de algoritmos de aprendizaje automático y aprendizaje profundo** ha permitido mejorar significativamente la precisión y velocidad del procesamiento de imágenes. Por ejemplo, modelos como YOLO (*You Only Look Once*) y SSD (*Single Shot MultiBox Detector*) han revolucionado la detección y el reconocimiento de objetos en tiempo real, proporcionando resultados rápidos y precisos incluso en entornos complejos y dinámicos. Estos avances han permitido que la visión artificial se utilice en aplicaciones

críticas, como la conducción autónoma, donde la capacidad de detectar y reaccionar a los obstáculos en tiempo real es esencial para la seguridad.

Además de las mejoras en los algoritmos, la visión artificial también se ha beneficiado del **desarrollo de hardware especializado**. Los procesadores gráficos (GPU) y las unidades de procesamiento tensorial (TPU) han mejorado significativamente la capacidad de las máquinas para procesar grandes volúmenes de datos visuales rápidamente. Esto ha facilitado el entrenamiento de modelos más complejos y precisos, lo que a su vez ha ampliado el alcance de las aplicaciones de la visión artificial.

Otro avance notable en la visión artificial es la **integración de la inteligencia artificial en dispositivos móviles y sistemas embebidos**. La capacidad de ejecutar modelos de aprendizaje profundo en dispositivos con recursos limitados ha abierto nuevas posibilidades para la implementación de soluciones de visión artificial en tiempo real. Esta tendencia ha sido facilitada también por el desarrollo de hardware especializado anteriormente mencionado, que proporcionan la potencia necesaria para ejecutar estos modelos de manera eficiente. Esta evolución ha permitido que tecnologías como la realidad aumentada y la realidad virtual se beneficien de capacidades avanzadas de reconocimiento y procesamiento de imágenes, mejorando la experiencia del usuario y ofreciendo nuevas funcionalidades.



Figura 2.6: Unidad de procesamiento tensorial (TPU) usada en este proyecto.

Fuente: coral.ai.

Además, la **combinación de técnicas de visión artificial con otros campos de la inteligencia artificial**, como el **procesamiento del lenguaje natural y la robótica**, ha resultado en sistemas más integrales y capaces. Por ejemplo, los sistemas de conducción autónoma ahora pueden interpretar señales de tráfico, reconocer peatones y otros vehículos, y tomar decisiones en tiempo real para evitar accidentes. Esta capacidad de integración es crucial para aplicaciones donde la seguridad y la precisión son esenciales,

y ha sido un área de enfoque importante para los investigadores y desarrolladores en los últimos años.

El campo de la visión artificial también ha visto **avances en la robustez y adaptabilidad de los modelos**. Los algoritmos modernos son ahora capaces de manejar variaciones significativas en las condiciones de iluminación y el entorno, lo que mejora su aplicabilidad en escenarios del mundo real. Por ejemplo, los sistemas de vigilancia y monitoreo ahora pueden operar de manera efectiva en una variedad de condiciones climáticas y de iluminación, lo que aumenta su utilidad en aplicaciones de seguridad. Estos avances no solo mejoran la precisión y eficiencia de los sistemas, sino que también amplían el alcance de las aplicaciones posibles, permitiendo que la visión artificial se utilice en nuevos contextos y sectores.

La visión artificial no solo se limita a la **detección y reconocimiento de objetos**, sino que también incluye tareas como la **segmentación de imágenes y el rastreo de objetos**. La segmentación de imágenes divide una imagen en múltiples segmentos, facilitando su análisis y procesamiento. Modelos como U-Net y Mask R-CNN son populares en esta área debido a su capacidad para segmentar con precisión los diferentes elementos de una imagen. Por otro lado, el rastreo de objetos implica seguir el movimiento de uno o varios objetos a través de una secuencia de imágenes o videos, lo que es crucial en aplicaciones como la vigilancia y el monitoreo de tráfico.

La visión artificial sigue siendo un campo de investigación activo y en constante evolución. Los desafíos actuales incluyen mejorar la robustez y la precisión de los algoritmos en condiciones variadas y adversas, así como la integración de estos sistemas en aplicaciones del mundo real. La colaboración entre instituciones académicas y la industria ha sido fundamental para el rápido avance de la visión artificial. La inversión en investigación y desarrollo, junto con la disponibilidad de grandes conjuntos de datos para el entrenamiento de modelos, ha acelerado el progreso en este campo. Proyectos colaborativos entre universidades, empresas tecnológicas y centros de investigación han resultado en innovaciones que no solo mejoran las capacidades técnicas, sino que también abordan desafíos prácticos en la implementación de soluciones de visión artificial. Esta sinergia entre teoría y práctica es esencial para continuar avanzando en el desarrollo de tecnologías que puedan resolver problemas complejos de manera efectiva y eficiente.

2.2.2. Principales técnicas y algoritmos

A lo largo del desarrollo de esta tecnología se han ido confeccionando una serie de técnicas y algoritmos que permiten el procesamiento y la extracción de características de una imagen.

Una tarea muy común antes de empezar a extraer características o detectar formas u objetos dado un *frame*, es el preprocessado del mismo. Para ello, existe un gran abanico de **operaciones de preprocessado** que se pueden realizar, como convertir a escala de grises, la binarización por umbral (*threshold binarization*), la sustracción del fondo (*background subtraction*) y otras operaciones morfológicas como la mediana, erosión y dilatación [9].

La **conversión a escala de grises** consiste en transformar el *frame* en una imagen de tonos de grises, de esta forma se consigue trabajar con 2/3 menos de píxeles, ya que en vez de tratar con una imagen de 3 canales de color R (*red*), G (*green*), B (*blue*); se trabaja con una única dimensión de color L (*luminancia*). Cabe aclarar que esta operación se debe hacer servir únicamente cuando el color no sea necesario a la hora de detectar cualquier característica o no sea crítico en la tarea de visión artificial que se esté realizando. Esta técnica ayudará a mejorar los tiempos de cómputo, haciendo más eficiente la aplicación de visión por computadora o procesamiento de la imagen.

La **binarización por umbral**, también conocida como *threshold binarization*, es una técnica u algoritmo que convierte una imagen en tonos de grises en otra binaria. Esto es posible porque se define un valor umbral de intensidad de luz. De este modo, dado un píxel, si el valor de intensidad de luz es inferior al del umbral, el píxel quedará negro, por otro lado, si la intensidad de luz es superior al umbral definido, el píxel quedará blanco. De igual forma, esta técnica nos permite aligerar aún más la imagen y, por tanto, mejorar notablemente el tiempo de cómputo.

La **sustracción de fondo** o *background subtraction*, consiste en “restar” al *frame* actual una imagen con el fondo de la escena. De esta forma el resultado será una imagen únicamente con los objetos de la escena que han cambiado respecto de la imagen de fondo. Esta técnica es muy útil para la detección de objetos en movimiento en una escena.

La **mediana** es un algoritmo de limpieza de imagen, que consigue reducir el ruido que puede aparecer caóticamente al realizar operaciones como la binarización. El algoritmo se aplica sobre una imagen en tonos de grises y consiste en dividir la misma usando un radio. Seguidamente, se calcula la media de luminosidad de los píxeles de cada división y, finalmente, se sustituye cada uno de los píxeles de una misma división por el mismo gris con el valor de luminosidad medio calculado. De esta forma, resulta una imagen suavizada que es muy útil para simplificar contornos y áreas irregulares, disminuyendo la aparición de ruido a la hora de realizar, por ejemplo, la binarización.

La **erosión** y la **dilatación** son dos operaciones que normalmente se suelen aplicar combinadas y también sirve para eliminar ruido de la imagen original. La erosión consiste en tomar todos los píxeles pertenecientes al objeto (valor a 1) que tengan una conectividad C con los píxeles del fondo (valor 0) y ponerlos a 0. Es decir, poner a 0 todos aquellos píxeles del objeto que sean vecinos al fondo. Por otra parte, la dilatación, es justo el proceso

inverso, tomar cada uno de los píxeles del objeto con valor 1 y poner a 1 todos aquellos píxeles del fondo que tengan una conectividad C con el píxel del objeto. Es decir, poner a 1 los píxeles del fondo que sean vecinos de los píxeles del objeto [10]. Normalmente, al aplicar erosión seguido de dilatación sobre una imagen con ruido, se consigue eliminar el ruido, resultando una imagen más óptima o adecuada para la posterior extracción de características.



Figura 2.7: Eliminación de ruido mediante erosión y dilatación.

Fuente: [9].

Aunque la dilatación y la erosión son las operaciones más fundamentales, ya que la mayoría de algoritmos morfológicos se basan en ellas, también existen otras operaciones comúnmente usadas como apertura, clausura, transformación por ganancia o pérdida, extracción de frontera, afinado o adelgazamiento, engrosamiento, relleno de región, esqueleto, poda, translación y reflexión [10].

Una vez preprocesada la imagen, se pasa al análisis de la misma. Este análisis se puede dividir en tres principales procesos: la segmentación, la descripción y el reconocimiento o clasificación [11].

La **segmentación** es el método que obtiene las características necesarias para diferenciar entre un tipo de objeto y otro. Este proceso divide la imagen en regiones o segmentos agrupando píxeles por características similares. Estas características pueden ser la forma, el perímetro, los ejes (mayor y menor), las texturas (liso, áspero), el color, y muchas más. El resultado es una imagen segmentada por regiones, donde cada región puede corresponder a un objeto a detectar.

Entre las principales técnicas y algoritmos de segmentación, destacan:

- **Segmentación basada en umbral.** Se fija un umbral para separar en dos grupos los píxeles, los que están por encima del umbral y los que se encuentran por debajo. Este umbral puede ser global o local, el global es útil para imágenes con diferencias de intensidad claras, por otra parte, la segmentación por umbral local, fija distintos umbrales para distintas regiones de la imagen, útil en los casos de iluminación poco uniforme.

- **Segmentación basada en bordes.** Primero, se detectan los bordes utilizando operadores como Sobel, Canny o Laplaciano, identificando los cambios bruscos de intensidad, que suelen pertenecer a los bordes de objetos. Seguidamente se agrupan los bordes, juntando los píxeles que son parte de un borde continuo, acotando así los perímetros de los objetos.
- **Segmentación basada en regiones.** El proceso empieza en un píxel semilla y se van agrupando píxeles vecinos con características (color, textura, etc.) similares hasta formar una región. El proceso de división y fusión de regiones va dividiendo recursivamente la imagen en subregiones y se van fusionando si se cumple con los criterios de uniformidad o igualdad.
- **Segmentación basada en clustering.** El algoritmo K-Means agrupa píxeles según las características en K clústeres, los píxeles son asignados al clúster de cuyo centroide es más cercano. Por otra parte, el algoritmo de medias desplazadas, crea clústeres naturales agrupando los píxeles desplazando una ventana de búsqueda por la región con mayor concentración de píxeles.
- **Segmentación basada en modelos estadísticos.** Los modelos de mezcla de gaussianas (GMM) supone que los píxeles han sido generados por una mezcla de distintas distribuciones gaussianas y utiliza el algoritmo Esperanza-Maximización (EM) para aproximar los valores de los parámetros de las distribuciones. Por otro lado, los campos aleatorios de Markov, también conocidos como *Markov Random Fields* (MRF), usando grafos probabilísticos, se modelan las relaciones espaciales entre píxeles, permitiendo segmentar no solo considerando las características individuales de cada píxel, sino también las relaciones de contexto.
- **Segmentación basada en redes neuronales.** Destacan las arquitecturas U-Net, diseñada específicamente para la segmentación semántica; Mask R-CNN, para la detección de objetos, generando máscaras de segmentación por cada objeto detectado; *Fully Convolutional Network* (FCN), una de las primeras arquitecturas totalmente convolucional también ideada para segmentación semántica; SegNet, similar a U-net; DeepLab, desarrollada por Google, entre otras.
- **Segmentación de superpíxeles.** El algoritmo *Simple Linear Iterative Clustering* (SLIC), segmenta la imagen en superpíxeles, estos son una pequeña agrupación de píxeles de igual características, haciendo más fino y preciso el segmentado.

La **descripción** es el siguiente proceso de la visión artificial y consiste en extraer y representar las características importantes o significativas de una imagen para permitir la clasificación. Es una fase esencial en muchos sistemas de visión por computadora, ya que

constituye la base para tareas como el reconocimiento de rostros, la detección de objetos, y más.

Entre los principales descriptores de características en la visión artificial, destacan:

- **Descriptores de puntos clave.** El algoritmo *Scale-Invariant Feature Transform* (SIFT) es uno de los descriptores más populares y robustos, ya que detecta los puntos claves de una imagen, mediante diferencias de gaussianas en diversas orientaciones y escalas, y los describe con un vector de características invariante a la escala y rotación. Además es potente frente a cambios de iluminación. *Speeded-Up Robust Features* (SURF) es una versión optimizada de SIFT, es más rápido sin perder mucha precisión.
- **Descriptores basados en regiones.** Destacan *Histogram of Oriented Gradients* (HOG) y *Maximally Stable Extremal Regions* (MSER). En HOG la imagen es dividida en subbloques, cada uno de ellos es subdividido en celdas y sobre ellas se calcula la magnitud y orientación de los gradientes de cada píxel. Sobre cada celda se calcula el histograma de los gradientes orientados promediado por un peso gaussiano y se almacena en el vector de características de la imagen [12].

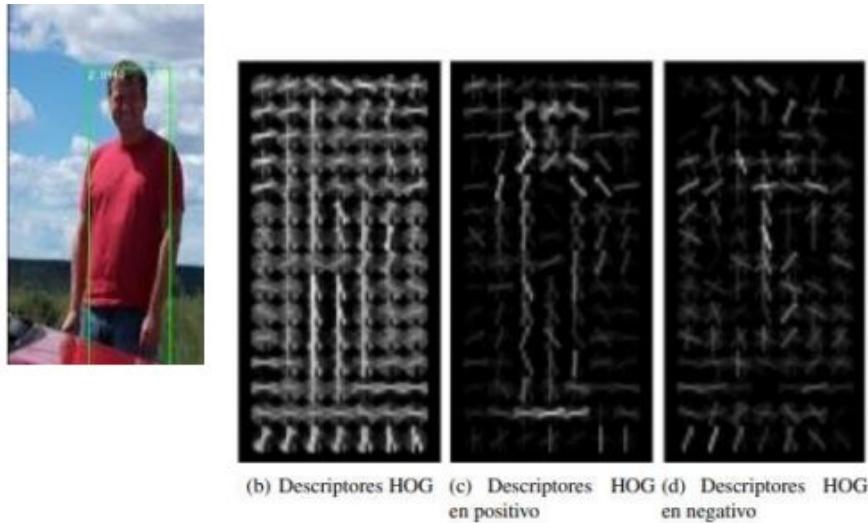


Figura 2.8: Extracción de descriptores HOG.

Fuente: [12].

Por otro lado, MSER detecta regiones extremales, es decir, conjunto de píxeles más brillantes o más oscuros en comparación con sus vecinos, y las agrupa en regiones estables, es decir, que permanecen consistentes a través de un rango de múltiples niveles de umbrales de intensidad. Útil para aplicaciones de reconocimiento de texto y objetos en difíciles condiciones.

- **Descriptores basados en textura.** *Local Binary Patterns* (LBP) es un algoritmo que crea un histograma de texturas. Dada una imagen y para cada píxel, se compara con sus vecinos inmediatos si son mayores (valor 1) o menores (valor 0) en intensidad y se asigna un valor binario de 8 bits, correspondiente a cada uno de los 8 vecinos adyacentes, que posteriormente se convierte a decimal. Se trata de una técnica de visión artificial simple pero muy efectiva para la detección de objetos.

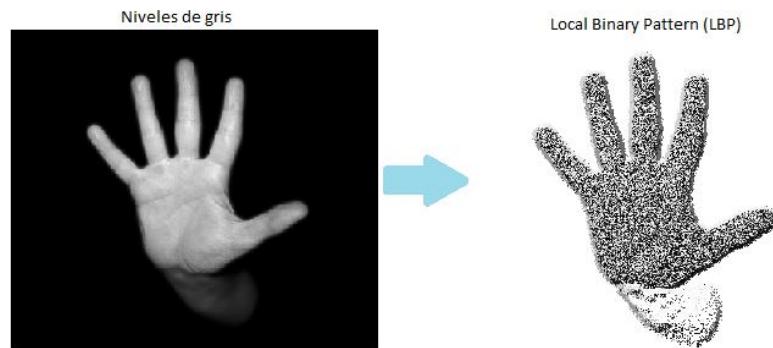


Figura 2.9: Extracción de textura con LBP.

Fuente: [13].

- **Descriptores basados en formas.** Los *Fourier Descriptors* usan la transformada de Fourier para describir la forma de los contornos de objetos.
- **Descriptores de características profundas.** Tras el avance en redes neuronales profundas, se ha popularizado usar las salidas de las capas intermedias de las redes convolucionales (CNNs) entrenadas como descriptores de características. Las redes convolucionales preentrenadas son capaces de extraer los mapas de características de una imagen, captando características más enrevesadas como formas, texturas y patrones. Usan estos mapas como descriptores para tareas de detección de objetos y clasificación.

El **reconocimiento o clasificación** es el proceso que clasifica la imagen o los objetos de una imagen en diferentes categorías tomando como entrada la salida de los descriptores, es decir, las características extraídas [11].

Dentro de las múltiples técnicas de clasificación se pueden distinguir los **clásificadores tradicionales** como *k-Nearest Neighbors* (k-NN), máquinas de soporte vectorial (SVM), árboles de decisión y *random forest*, de los **clásificadores basados en redes neuronales**, como el preceptrón multicapa (MLP), redes convolucionales (CNNs), redes neuronales recurrentes (RNNs) y *transfer learning*. De nuevo, con el avance de las redes neuronales, se ha ganado mucha fama el uso de redes convolucionales o hacer *transfer*

learning, ya que estas técnicas permiten aprender representaciones muy complejas directamente de los datos.

2.3 Reconocimiento de señales de tráfico

2.3.1. Contexto actual de las tecnologías de identificación de señales

El reconocimiento de señales de tráfico es una aplicación crítica en el ámbito de la conducción autónoma y la seguridad vial. Las tecnologías actuales han avanzado significativamente gracias a los desarrollos en aprendizaje profundo y visión artificial, permitiendo que los sistemas de vehículos autónomos detecten, clasifiquen e interpreten señales de tráfico con una precisión y rapidez impresionantes. Estos sistemas utilizan principalmente cámaras y sensores para capturar imágenes del entorno vial, las cuales son luego procesadas por modelos de redes neuronales convolucionales (CNN) para identificar las señales de tráfico presentes.

En el contexto actual, los modelos de detección de objetos como YOLO (You Only Look Once) y SSD (Single Shot MultiBox Detector) son ampliamente utilizados debido a su capacidad para realizar detecciones en tiempo real. Estos modelos han sido adaptados y mejorados continuamente para manejar las variaciones en las condiciones de iluminación y el ángulo de visión, que son comunes en entornos viales.

Además de los algoritmos de detección, la segmentación de imágenes también juega un papel crucial en el reconocimiento de señales de tráfico. Técnicas como Mask R-CNN no solo detectan las señales, sino que también segmentan cada señal de su fondo, lo que mejora la precisión en entornos complejos donde las señales pueden estar parcialmente obstruidas o rodeadas de otros objetos visuales. La segmentación precisa es esencial para aplicaciones donde se requiere una alta fidelidad en la interpretación de la señal, como en la identificación de señales de límite de velocidad y señales de advertencia.

Otra tendencia importante en el reconocimiento de señales de tráfico es la fusión de sensores. Al combinar datos de múltiples sensores como cámaras, LIDAR y radares, los sistemas pueden mejorar la robustez y precisión de la detección. Esta fusión de sensores permite compensar las limitaciones individuales de cada tipo de sensor, como la sensibilidad de las cámaras a las condiciones de iluminación o la capacidad limitada de los LIDAR para identificar colores y textos. La fusión de datos proporciona una visión más completa y fiable del entorno vial, lo cual es crucial para la toma de decisiones en tiempo real.

Finalmente, la evolución de los conjuntos de datos para el entrenamiento de modelos de reconocimiento de señales de tráfico ha sido crucial. Conjuntos de datos como GTSRB

(German Traffic Sign Recognition Benchmark) y LISA (Laboratory for Intelligent and Safe Automobiles) han proporcionado vastas cantidades de imágenes etiquetadas que son esenciales para entrenar y validar los modelos [14]. Estos conjuntos de datos han permitido a los investigadores desarrollar y refinar algoritmos que pueden manejar la diversidad y complejidad del entorno vial real, asegurando que los sistemas sean robustos y fiables en diversas condiciones.

Los avances en el reconocimiento de señales de tráfico continúan mejorando la seguridad y eficiencia de los sistemas de conducción autónoma, integrando de manera efectiva la inteligencia artificial y la tecnología de sensores para crear vehículos más inteligentes y seguros.

2.3.2. Algoritmos y modelos de aprendizaje profundo

El campo del reconocimiento de señales de tráfico ha avanzado notablemente gracias al desarrollo de algoritmos y modelos de aprendizaje profundo, que han transformado la capacidad de los sistemas de visión artificial para interpretar el entorno vial de manera precisa y eficiente. Uno de los modelos más destacados es **YOLO** (You Only Look Once), conocido por su capacidad para realizar detecciones en tiempo real con alta precisión. YOLO se distingue por su enfoque de detección unificado, que permite predecir múltiples clases de objetos y sus ubicaciones dentro de una imagen con una sola pasada a través de la red. Este enfoque no solo mejora la velocidad de procesamiento, sino que también reduce la complejidad computacional, lo que lo hace ideal para aplicaciones en vehículos autónomos donde la rapidez de respuesta es crucial.

Otro modelo importante en este campo es el **SSD** (Single Shot MultiBox Detector), que utiliza una combinación de mapas de características de diferentes resoluciones para detectar objetos de varias escalas. SSD es capaz de identificar y clasificar objetos con alta precisión, incluso cuando están en movimiento o parcialmente ocultos. Esta capacidad es particularmente útil en entornos urbanos densos, donde la variabilidad de las condiciones puede ser alta. La integración de SSD en sistemas de conducción autónoma ha demostrado ser efectiva para mejorar la seguridad y la fiabilidad de estos vehículos, permitiéndoles reaccionar rápidamente a los cambios en el entorno vial.

Además de YOLO y SSD, modelos como **Faster R-CNN** y **Mask R-CNN** también juegan un papel crucial en el reconocimiento de señales de tráfico. Faster R-CNN combina la detección y clasificación de objetos en un solo paso, lo que mejora la eficiencia y precisión de los sistemas de visión artificial. Por su parte, Mask R-CNN extiende las capacidades de Faster R-CNN al incluir la segmentación de objetos, lo que permite una identificación más detallada y precisa de las señales de tráfico, incluso en entornos com-

plejos. Estas capacidades avanzadas de segmentación son esenciales para aplicaciones donde la claridad y la exactitud de la detección son fundamentales.

El modelo **EfficientDet** ha emergido como una solución eficiente y de alto rendimiento para la detección de objetos. Este modelo optimiza la relación entre precisión y eficiencia mediante la combinación de técnicas avanzadas de aprendizaje profundo y optimización de hardware. EfficientDet es especialmente valioso en aplicaciones donde los recursos computacionales son limitados pero se requiere una alta precisión en la detección de objetos. Su diseño escalable permite su adaptación a diferentes niveles de complejidad, desde dispositivos móviles hasta sistemas de alta capacidad en vehículos autónomos.

CAPÍTULO 3

Diseño de la solución

3.1 Análisis del problema

Tal y como se ha mencionado en el apartado de los objetivos, este proyecto nace de la necesidad de continuación y mejora de un coche robótico que ya infiere un modelo de seguimiento de carriles. Es decir, se pretende mejorar este coche dotándolo con un sistema de reconocimiento de señales de tráfico. Además, el pequeño robot debe ser capaz de comportarse en consecuencia con las señales que reconozca.

Para ello, se tienen que distinguir las principales tareas a realizar. Se empieza por la creación de un conjunto de datos de entrenamiento correctamente etiquetados. Seguidamente, se tiene que entrenar un nuevo modelo de detección de objetos y conseguir desplegarlo en el *hardware* del robot. Finalmente, hay que desarrollar un programa que infiera el modelo entrenado junto con el modelo ya existente y programar el nuevo comportamiento del coche al detectar las señales estipuladas.

En cuanto a la primera tarea, se tiene que ingeniar un método de recolección de datos y desarrollar un programa que permita capturar las imágenes de las señales desde la cámara del robot y el entorno de conducción. La cámara que equipa el coche es una limitación para esta tarea, ya que no proporciona imágenes de muy buena calidad, sobre todo en situaciones de poca luminosidad. Pero, en este caso, no se considera cambiarla porque podría perder eficiencia en la salida del modelo ya entrenado de seguimiento de carriles.

Respecto al entrenamiento del nuevo modelo, se tiene que identificar el tipo de problema de aprendizaje automático y elegir la arquitectura de capas que conformará nuestra red neuronal convolucional. Los modelos de detección de objetos no solo retornan a la salida etiquetas pertenecientes a la clasificación de los objetos detectados, sino que también devuelven unos valores numéricos continuos que describen las coordenadas de las cajas que los delimitan (*bounding boxes*). Por tanto, se trata de un problema fundamentado en el **aprendizaje supervisado**, ya que requiere de un conjunto de datos correctamente

etiquetados, donde se realiza una tarea de predicción de las *bounding boxes*, aplicando funciones de **regresión**, y una tarea de **clasificación**, aplicando funciones discretas como puede ser la *softmax* o la *sigmoid*.

En cuanto a la arquitectura, las tres más utilizadas para modelos convolucionales de detección de objetos son R-CNN (*Region-based Convolutional Neural Network*) y sus mejoras Fast R-CNN y Faster R-CNN, YOLO (*You Only Look Once*) y SSD (*Single Shot MultiBox Detector*). R-CNN propone regiones que podrían contener objetos, cada una de estas regiones es pasada por una red neuronal convolucional que se encarga de extraer las características, es la más precisa pero, a pesar de sus mejoras, sigue siendo la más lenta. En cambio, YOLO y SSD predicen de una sola pasada las propuestas de cajas que contienen objetos, ambas son rápidas aunque pierden un poco de precisión respecto a R-CNN. Para este proyecto, dado que un gran limitante es la potencia de cómputo que ofrece la Raspberry y el acelerador Coral Edge TPU, se ha decidido usar la arquitectura SSD, ya que es la más ligera e idónea para dispositivos con bajos recursos y aplicaciones en tiempo real.

Finalmente, el programa desarrollado que interprete el coche debe ser capaz de capturar las imágenes o *frames* de vídeo mientras el vehículo esté en movimiento. Estos datos capturados serán los que se pasen como entrada a la parte de código que ejecute la inferencia de ambos modelos. Por tanto, este bloque debe poder devolver el ángulo de giro de las ruedas del coche que el primer modelo ya retornaba y un número concreto de objetos detectados por *frame*. Es decir, la clase a la que pertenece cada objeto detectado, la puntuación por cada clasificación y los cuatro valores numéricos que indican la región de la imagen donde se encuentra el objeto. De esta forma, el algoritmo podrá interpretar estos datos y hacer que el robot se comporte debidamente. Algo a tener en cuenta para esta tarea es que, en un entorno casero, muchas de las acciones en la conducción real no se pueden reproducir, por eso es muy importante elegir bien las señales que se van a utilizar para el entrenamiento del modelo y la interpretación de las mismas.

3.2 Materiales requeridos para la implementación del proyecto

Ahora que ya se ha analizado el problema detenidamente, es conveniente plantear todo el equipamiento necesario para poder desarrollar la solución. Desde dispositivos *hardware* y herramientas *software* hasta accesorios.

3.2.1. Señales de tráfico

Unos elementos sencillos pero que son esenciales para el desarrollo de este proyecto son las señales de tráfico. Por eso, cuando se empezó a diseñar la solución se dedicó un tiempo prudente para decidir qué señales iban a formar parte del conjunto de datos de entrenamiento del modelo y qué comportamiento podría aplicarse con el robot.

La primera que se planteó fue la de *STOP*, ya que es la más trivial y fácil de implementar usando la librería de control del robot. De hecho, las primeras pruebas en la creación de la base de datos, en la anotación de los mismos, en los primeros intentos de entrenamiento y de despliegue con el robot fueron con este elemento. Más adelante, en el cuarto capítulo del desarrollo de la solución se detallarán estos primeros avances.

Una vez se tuvo claro qué señales se iban a entrenar para su posterior reconocimiento, se empezó a indagar por comercios web hasta encontrar unas señales que se adaptasen al entorno a pequeña escala del que se dispone. El paquete en cuestión contiene 17 piezas, de las cuales no todas han sido utilizadas. El kit de señales es de Lena, una marca conocida de juguetes, y se compró en Amazon.



Figura 3.1: Kit de 17 piezas de señales de tráfico de Lena.

Fuente: [amazon.es](https://www.amazon.es).

Como se puede comprobar en la ilustración, aunque el paquete no contiene una gran variedad de señales, este conjunto de piezas en concreto es ideal porque tiene la peculiaridad de tener las señales en blanco. Pudiendo así escoger a nuestro criterio el conjunto de señales a utilizar, simplemente imprimiendo nuevas señales en papel de pegatina. Finalmente, se tuvo que comprar otro paquete igual, ya que el número de señales redondas era de tan solo 3.

Primeramente, solo se pensó en la funcionalidad que estas señales podían dar, pero después se consideró también la futura creación de la base de datos. Y por tanto, para poder realizar un aumento del *dataset*, se pensó en introducir señales que pudieran ser transformadas mediante técnicas, por ejemplo, de volteado horizontal (*flip*). Aunque no todas se pueden voltear, señales esenciales como las limitadoras de velocidad o la de

STOP si se voltean cambian por completo las características que se extraen de la rotulación. En cambio, otras como la de obligación de ir a la izquierda o a la derecha, se pueden voltear intercambiando después las etiquetas de clasificación.

Por tanto, a continuación se listan las señales que forman parte del conjunto de datos de entrenamiento de este proyecto, proporcionando una breve explicación del comportamiento que tendrá el robot al detectarlas.

Empezando por las que pueden ser volteadas tenemos:

- Señal de **ceda el paso** (R-1): el robot hará una pequeña parada de 1 segundo, seguidamente estará durante 3 segundos conduciendo a velocidad mínima y finalmente alcanzará la velocidad que llevaba antes de detectar la señal.
- Señal de sentido obligatorio **recto** (R-400 c): el coche conducirá recto durante una distancia o tiempo determinado, es decir con un ángulo de las ruedas de 90 grados.
- Señal de sentido obligatorio **izquierda** (R-400 b): el vehículo girará las ruedas con un ángulo de 45 grados durante un tiempo determinado. También encenderá de manera intermitente el LED izquierdo de color naranja.
- Señal de sentido obligatorio **derecha** (R-400 a): de igual forma que la señal de izquierda, pero esta vez girando las ruedas delanteras a 135 grados respecto del eje horizontal y encendiendo el led derecho.
- **Semáforo en rojo**: el coche se detendrá hasta que no reconozca el semáforo verde.
- **Semáforo verde**: el vehículo seguirá conduciendo con la misma velocidad y ángulo de giro de las ruedas.

Y en cuanto a las señales que no pueden ser volteadas, aunque sí se pueden hacer transformaciones como introducción de ruido o cambio de iluminación para el aumento de datos, tenemos:

- Señal de **STOP** (R-2): el robot hará una primera parada de 2 segundos, seguidamente estará durante 1 segundo conduciendo a velocidad mínima, hará una segunda parada de 2 segundos y finalmente alcanzará la velocidad que llevaba antes de detectar la señal.
- Señal de prohibición **velocidad máxima a 30** (R-301 30): el coche estipulará su velocidad actual a 30, hasta que detecte otra señal que cambie su velocidad.
- Señal de prohibición **velocidad máxima a 60** (R-301 60): el vehículo se comportará de igual forma que en la señal de 30, pero pondrá la velocidad actual a 60.

- Señal de **alumbrado de corto alcance** (R-413): el robot encenderá las 4 luces LED de color blanco durante un tiempo determinado.
- Señal de **peligro por obras** (TP-18): el coche conducirá de manera prudente a una velocidad de 20 durante unos segundos, luego retomará la velocidad que llevaba.



Figura 3.2: Las 11 señales de tráfico utilizadas en este proyecto.

Fuente: Elaboración propia.

3.2.2. *Hardware utilizado*

Entre los diferentes dispositivos electrónicos y material físico que se necesitan, los primordiales son los que forman el **coche robótico** que, como ya se ha mencionado anteriormente, ya tenemos montado. Este robot está controlado principalmente por una pequeña computadora, una **Raspberry Pi 3 Modelo B+**, la cual contiene la unidad central de procesamiento (CPU), encargada de ejecutar el programa principal de conducción. Este ordenador cuenta con todo lo que puede tener una computadora personal, solo que a una escala reducida y, por tanto, con unos recursos más limitados. En concreto está equipado con un procesador con arquitectura ARM de 64 bits de 4 núcleos a una frecuencia de reloj de 1,4GHz. También cuenta con 1GB de memoria RAM, ranura para tarjeta SD, puerto HDMI para la salida de vídeo y conectividad Ethernet y Wi-Fi. Se puede comprar a través de internet y su precio actual es de 42 euros.

La Raspberry tiene instalada el **kit PiCar-V V2.0 de la marca SunFounder**, un paquete que contiene todo lo necesario para poder convertir nuestra computadora, en un

pequeño coche robótico. Actualmente se puede comprar a través de internet por un precio que ronda los 100 euros. El lote en cuestión contiene:

- **Cuatro ruedas** de goma pequeñas que permiten al robot poder moverse.
- **Dos motores eléctricos** para la transmisión de las ruedas traseras.
- **Una cámara** pequeña que se conecta vía USB, con una resolución máxima de 480x360 píxeles y un ángulo de visión de 120 grados.
- **Tres servomotores**, dos para dotar a la cámara de dos grados de libertad y uno para conseguir direccionar la ruedas delanteras.
- Un soporte para alimentación portátil (**baterías** de iones de litio, dos pilas).
- Una serie de **placas** que se conectan a la Raspberry mediante una conexión de 40 pines GPIO (Entrada/Salida de Propósito General) estándar. Esta placas permiten interconectar todos los elementos que componen el kit.
- **Un chasis** donde poder sujetar todos los componentes.

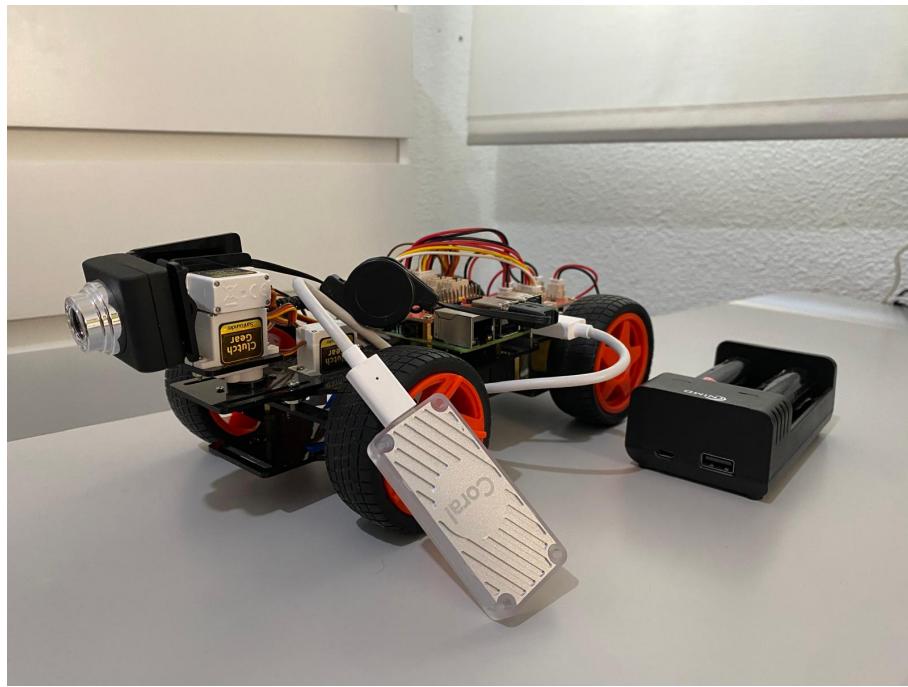


Figura 3.3: El coche robótico PiCar utilizado para este trabajo.

Fuente: Elaboración propia.

Otro dispositivo clave en el desarrollo de este proyecto es el **acelerador Colar Edge TPU** (Unidad de Procesamiento Tensorial), que como se puede observar en la imagen anterior, se conecta a la Raspberry mediante USB. Esta unidad permite inferir los modelos entrenados de un manera más rápida y eficiente, ya que está optimizada para operar

con tensores. Los modelos deben estar convertidos en un formato que esta unidad acepte (formato tflite).

También se ha decidido equipar al robot con una **tira de luces LED**, para poder simular los intermitentes de un coche cuando detecte las señales de izquierda y derecha, y poder encender las luces cuando se detecte la señal de alumbrado de corto alcance. Esta tira contiene cuatro LEDs, cada uno de ellos programables individualmente, que la hace perfecta para este proyecto ya que es versátil y ofrece funcionalidad. La marca de la tira LED es BlinkStick y el modelo concreto es Strip Mini, con un precio de 17 euros. Cabe destacar que la marca ofrece una librería Python para poder controlarla.

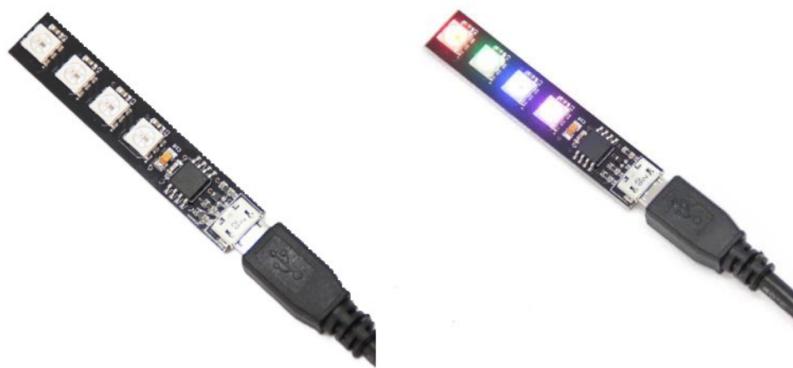


Figura 3.4: BlinkStick Strip Mini.

Fuente: shop.blinkstick.com.

Otro componente físico utilizados han sido la **tarjeta micro SD** de Samnsug modelo EVO Plus, de 128 GB de capacidad, la cual lleva instalado el sistema operativo Raspberry Pi OS. Hay que señalar, que se ha tenido que comprar otra igual para poder hacer copias de seguridad, ya que muchas veces la tarjeta falla y la Raspberry deja de cargar el sistema operativo. También se ha necesitado un **cargador para las baterías** y dos pilas más para poder hacer pruebas mientras las otras se cargan y aumentar de esta forma la productividad.

El **ordenador personal** también ha sido un equipo informático vital para la construcción de la solución, ya que es la principal herramienta de trabajo a la hora de crear el set de datos, el entrenamiento del modelo y la elaboración del algoritmo del programa principal de conducción. Este PC en cuestión equipa un procesador Intel Core i5-4570 de 4 núcleos a una frecuencia de reloj de 3,2 GHz, 8 GB de memoria RAM y un disco de estado sólido de 256 GB. Como se puede notar, no es necesario que sea un ordenador muy potente, ya que el entrenamiento de la red neuronal convolucional se realiza en un entorno de computo en la nube, Google Colab, el cual se detalla más adelante en el apartado de *software* utilizado. También se requiere al menos de un **monitor**, un **teclado** y un **ratón**, para la configuración inicial de la Raspberry.

Finalmente, un accesorio básico pero muy necesario es la **cinta adhesiva rosa** que ya sabe clasificar el modelo actual de seguimiento del carril, utilizada para las diferentes pruebas en distintos carriles y entornos creados.

3.2.3. *Software utilizado*

A lo largo de todo el proceso de desarrollo de este proyecto se han ido utilizando diferentes programas, aplicaciones, sistemas operativos, librerías, lenguajes y entornos de programación. A continuación se expone todo el *software* que ha sido necesario, explicando la importancia y función de cada herramienta usada.

Empezando por la base de muchas herramientas *software*: el **sistema operativo**. En este caso, para el ordenador del robot, se ha estado utilizando el sistema operativo que el fabricante del mismo ha desarrollado y recomienda, **Raspberry Pi OS**, ya que está optimizado para la placa de la Raspberry. Este sistema está basado en la distribución de Linux Debian, una distribución en la que se basan muchos de los sistemas operativos más populares de Linux, como Ubuntu y Linux Mint. Raspberry Pi OS ha sido especialmente útil a la hora de configurar el robot, navegar entre los diferentes directorios, visualizar imágenes capturadas por la cámara del coche y ejecutar programas Python, todo esto gracias a la interfaz gráfica tan simple e intuitiva que ofrece. Por otra parte, en la computadora personal donde se ha elaborado la solución, se ha trabajado con el sistema operativo **Windows**.

En cuanto al **lenguaje de programación**, se ha utilizado **Python**. Un lenguaje de alto nivel e interpretado, es decir, el código se va ejecutando línea a línea y se detiene si se encuentra algún error. Destaca por su sintaxis simple, clara y legible, lo que supone una fácil y rápida comprensión del código. También ofrece un extenso catálogo de módulos y librerías para una amplia variedad de tareas. Además es el lenguaje más popular en el ámbito del aprendizaje automático, ya que tiene las bibliotecas y *frameworks* más populares y usados en este campo, entre los cuales destacan NumPy, Pandas, Scikit-learn, TensorFlow, Keras y PyTorch, que proporcionan herramientas potentes para poder llevar a cabo el tratamiento de datos, la construcción y entrenamiento de modelos de inteligencia artificial. También tiene una de las comunidades más grandes y en constante actividad del mundo de los lenguajes de programación, lo que supone la disposición de muchos recursos y una amplia documentación casi para cualquier tarea o problema que se abarque.

De entre todas las **librerías utilizadas**, cabe mencionar OpenCV, Pandas, Matplotlib, Numpy y TensorFlow y su API para modelos de detección de objetos, que han sido claves para el visualizado y preprocesado de las imágenes, para el tratamiento de los datos y creación de las estructuras de datos de entrada y el posterior entrenamiento del modelo.

OpenCV (*Open Source Computer Vision Library*) es una librería de visión por computadora en tiempo real y de código abierto, desarrollada por Intel. Entre otras cosas, permite el acceso a la cámara, procesar vídeo y manipular imágenes.

Pandas es otra de las librerías usadas y también sirve para el análisis y manipulación de los datos. El uso de los *DataFrames* de Pandas, estructura de datos bidimensional, ha sido especialmente útil para manipulación y creación de los datos de entrada al modelo. Estos *DataFrames* pueden ser escritos o leídos desde archivos como CSV, EScel y SQL.

Matplotlib es la librería de Python que se ha utilizado para la visualización de datos, en concreto, para visualización de las imágenes y los datos de salida del modelo. Facilitando de esta forma, la visualización de los resultados obtenidos. También es muy utilizada para graficar las métricas del entrenamiento a lo largo del mismo y, de esta forma, poder ver si el modelo generaliza bien o se sobre ajusta.

Numpy es la librería por excelencia para el cómputo en Python. Permite trabajar con unas estructuras de datos numéricos como *arrays* multidimensionales y matrices, proporcionando herramientas para trabajar con estas y haciendo más eficiente el cálculo. Numpy es fundamental a la hora de realizar las operaciones matemáticas complejas que el aprendizaje profundo requiere.

TensorFlow es un *framework* de código abierto, desarrollado por Google, que permite la construcción de modelos de aprendizaje automático de una manera fácil. Es ampliamente utilizado en el diseño y entrenamiento de redes neuronales profundas de diferentes arquitecturas, incluyendo redes convolucionales, recurrentes y generativas adversarias. También permite el manejo y procesamiento de grandes volúmenes de datos, pudiendo distribuir el procesamiento de datos en diferentes unidades de procesamiento. Destaca por su portabilidad, ya que los modelos entrenados en TensorFlow pueden ser exportados a distintas plataformas, como es el caso de este proyecto, haciendo uso de TensorFlow Lite. Además, se ha usado la **API de detección de objetos del ecosistema de TensorFlow** que ofrece, entre otras cosas, modelos preentrenados y la posibilidad de realizar entrenamiento personalizados. Esto quiere decir que el desarrollador puede entrenar los modelos para una tarea específica, usando la arquitectura base y los pesos de un modelo preentrenado y realizar el entrenamiento solamente de las últimas capas para poder ajustarse a la tarea.

Hasta ahora se han detallado las librerías que han sido utilizadas para el desarrollo del modelo, pero también cabe mencionar las que han hecho posible el manejo del coche robótico.

Empezando por la **API PyCoral** que, aunque su uso no es exactamente para el controlado del coche, ha sido muy importante para poder desplegar el modelo en el acelerador

Coral Edge TPU y poder, de esta forma, inferir el modelo en esta unidad de procesamiento.

La **API Picar** que ofrece la marca SunFounder para controlar el kit electrónico que conforma el coche robótico. Esta está diseñada para poder controlar los motores y servomotores de manera sencilla. Permitiendo desarrollar programas Python para controlar el movimiento e interacción del robot.

Finalmente también se ha utilizado la **API Blinkstick** para poder programar las animaciones con la tira de luces LED y poder simular así los intermitentes y las luces del coche robótico.

En cuanto a las aplicaciones y entornos de programación que se han usado, destacan Make Sense AI, Google Colab y Visual Studio Code.

Make Sense AI es una aplicación web que permite el etiquetado o anotación de imágenes para tareas de detección de objetos o reconocimiento de imágenes. Es una herramienta potente que genera las anotaciones en el formato que se especifique, el desarrollador solo tiene que preocuparse de, mediante la interfaz gráfica, seleccionar correctamente el área de imagen donde se encuentra el objeto a anotar y la correspondiente clase a la que pertenece. Es muy utilizada en proyectos de inteligencia artificial y aprendizaje automático. También existen otras herramientas, como LabelImg para Python, que hace lo mismo. En este caso se ha decidido usar la primera porque tiene una interfaz más clara y simple.

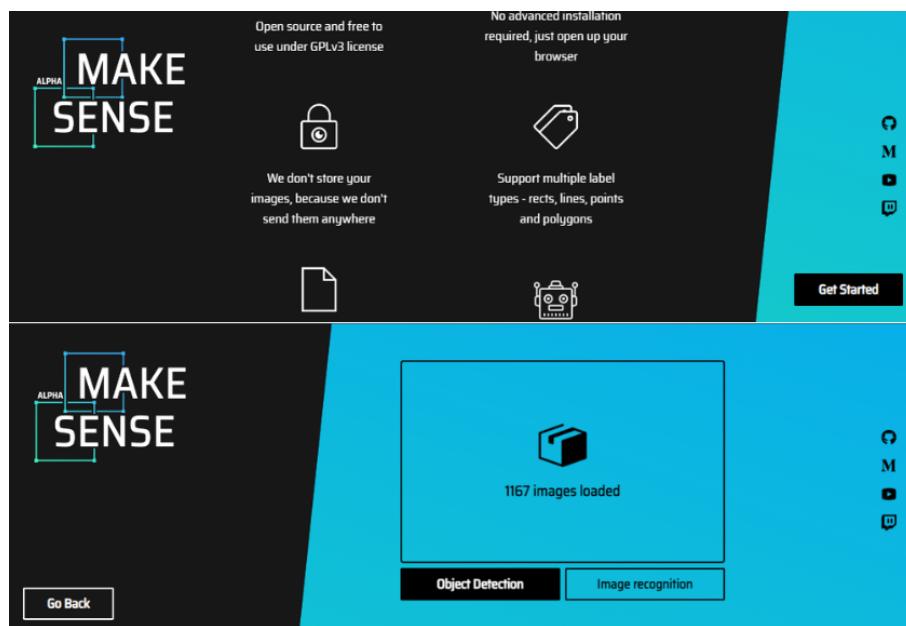


Figura 3.5: Make Sense AI, herramienta utilizada para la anotación de las imágenes de entrenamiento.

Fuente: Elaboración propia, capturas de la web makesense.ai.

Google Colab es una plataforma de Google que ofrece un entorno de **cuadernos Jupyter** en la nube y permite codificar y ejecutar programas Python usando recursos del

propio entorno. Es muy útil en el campo de aprendizaje automático porque ofrece una solución para aquellos desarrolladores que no disponen de tarjetas gráficas para poder entrenar modelos. En este proyecto se ha desarrollado un cuaderno Jupyter que ejecuta el tratamiento previo de datos, el entrenamiento, la visualización posterior del comportamiento del modelo y la extracción del mismo.

Por último, otro entorno de programación que se ha hecho servir es **Visual Studio Code**, un editor de código que ofrece herramientas de todo tipo adaptadas al lenguaje de programación que se use. Ha sido especialmente útil para la programación local y ejecución de pequeños *scripts* Python, para la transformación de las imágenes, renombrar archivos, editar los archivos de anotaciones, etc. También se ha usado para el desarrollo del programa principal de conducción autónoma.

CAPÍTULO 4

Desarrollo de la solución

4.1 Puesta en funcionamiento y configuración del robot

El primer paso a seguir, es conseguir configurar el robot ya montado del que se dispone y poder hacer unas primeras pruebas de funcionamiento. Para ello se requiere poder configurar la conexión inalámbrica a internet, WiFi y la conexión vía SSH de nuestro equipo que conforma el coche con la computadora local que disponemos. De esta forma y, gracias a unas baterías, se puede lograr ejecutar el programa de conducción autónoma del que se parte de manera totalmente remota y ver si efectivamente funciona todo bien.

Empezando por la configuración, se puede apreciar que, con el robot completamente montado, no se puede conectar el cable de salida de vídeo, HDMI, por falta de espacio físico. Dado este problema, se tiene que desmontar el robot parcialmente, desatornillando la Raspberry y una placa que está conectada justo encima de esta, la placa principal del kit de SunFounder, que recibe la alimentación de la batería y permite el paso de corriente a las diferentes placas y elementos que conforman el kit.

Una vez desmontado parcialmente el robot, ahora ya se puede conectar cómodamente todos los periféricos necesarios para la correcta configuración. Por tanto, se conecta el cable HDMI, el ratón, el teclado y el cable de alimentación micro USB y se procede a encender la Raspberry.

Al arrancar la máquina, se presentó un problema, la Raspberry no conseguía arrancar el sistema operativo, se quedaba atascada en una pantalla de carga colorida. Se investigó por distintos foros de internet hasta averiguar de que problema se trataba. Era un problema de la tarjeta micro SD, esta no tenía correctamente cargada la imagen del sistema operativo, estaba corrompida o podía haber un error en el archivo “*bootcode.bin*”, ya que la luz LED roja sí se encendía pero la amarilla no. Finalmente se consiguió solucionar el problema, tan solo quitando la tarjeta micro SD de la Raspberry y conectándola a otra

máquina, en este caso se conectó a una que tenía una instalación de Ubuntu. Se pudo acceder a los archivos de la misma, se ejecutó el comando “`sudo fdisk -l /dev/sdc1`” y, finalmente, se expulsó con seguridad la tarjeta. Cabe resaltar que este problema ocurre con frecuencia cuando realizas el apagado de la Raspberry, y puede ser que la próxima vez que se encienda el sistema no cargue. No se sabe con certeza de que problema se trata, pero repitiendo el anterior proceso se soluciona. De todas formas, se realizó una copia de seguridad de la tarjeta por si acaso; de hecho, se recomienda hacerla.

Una vez solucionado el problema de la tarjeta, ya se puede acceder al sistema operativo Raspberry Pi OS y lo primero que se hace es configurar la conexión WiFi local. Tras este paso, se continua configurando la conexión de escritorio remoto vía SSH, se confirma que la conexión SSH está activa en el apartado “*Menu > Preferences > Raspberry Pi Configuration > Interfaces*”. Mediante un comando por consola, “`ifconfig`” o “`hostname -I`”, se puede averiguar la dirección IP de la máquina. Con todo esto, ya se puede acceder de forma remota a la interfaz de nuestra Raspberry. Solamente hay que ir a la aplicación de escritorio de nuestro equipo Windows, Conexión a Escritorio Remoto, y configurar la conexión introduciendo la dirección IP y el correspondiente usuario y contraseña.

Otro problema que surgió, fue el desconocimiento de la contraseña del usuario “pi” pero esto tuvo solución rápida mediante la ejecución del comando por consola “`sudo passwd pi`”. Este comando, permitió cambiar la contraseña fácilmente por “picar2024”, se deja constancia para futuras ampliaciones y trabajos en relación con este robot.

Una vez configurado el escritorio remoto, ya se puede proceder a volver a montar el robot, atornillando las placas anteriormente desmontadas. Ahora, ya no será necesario usar el cable HDMI, ni el ratón ni el teclado. En cambio, se usará la computadora personal.

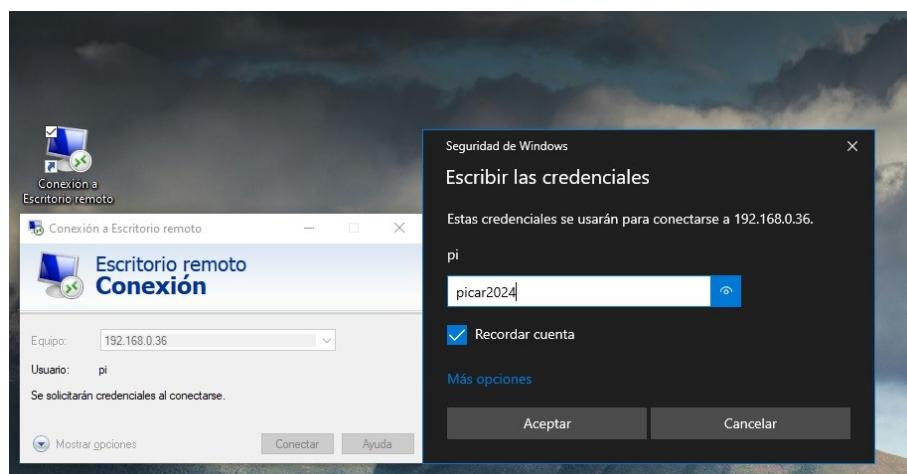


Figura 4.1: Configuración del escritorio remoto.

Fuente: Elaboración propia.

Por último y para acabar con la configuración, es necesario configurar en nuestro ordenador personal, la conexión al sistema remoto de archivos compartidos mediante Samba.

Esto, nos permite poder desarrollar código de una manera más cómoda usando el entorno de programación Visual Studio Code. En la instalación que hay en la tarjeta, ya viene configurado el fichero “*smb.conf*”, que se encuentra en la ruta “*/etc/samba*”. Solo se tiene que acceder al explorador de archivos de nuestro equipo Windows y conectarse a una nueva unidad de red. Se introduce la dirección IP seguido del nombre del recurso compartido, especificado en el archivo de configuración anteriormente nombrado (*\\\192.168.0.36\HOMEP*). Al acceder se piden las credenciales, nombre de usuario y contraseña.

De nuevo, no se disponía de la contraseña del usuario de Samba. Y, se tuvo que cambiar usando el comando “*sudo smbpasswd -a pi*”. Se eligió la misma contraseña que en el anterior caso, “picar2024”.

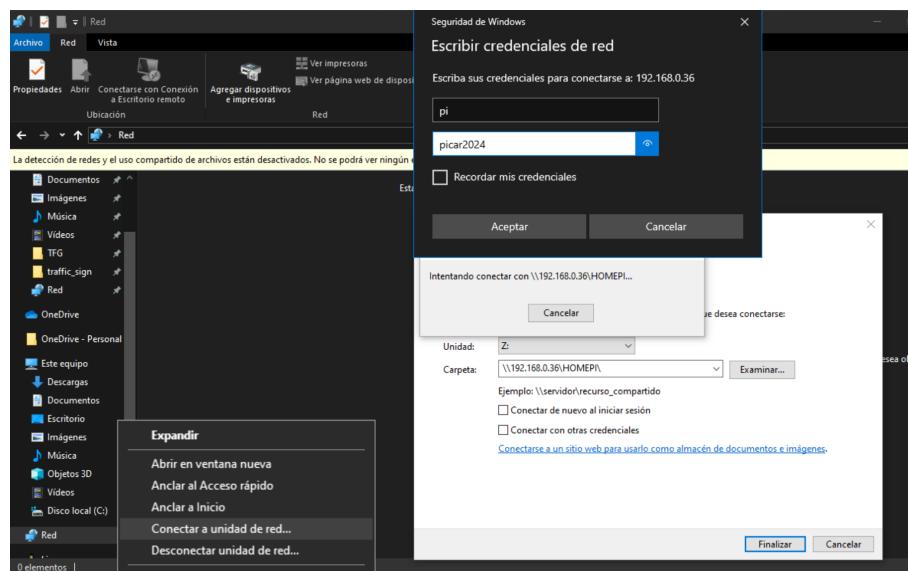


Figura 4.2: Configuración del sistema de archivos compartidos.

Fuente: Elaboración propia.

Ahora ya se podía poner en marcha el programa de conducción autónoma antiguo de manera inalámbrica. Para ello, se construyó un pequeño carril recto con una ligera curva y se ejecutó el programa de seguimiento de carril en modo automático. El robot se comportó correctamente hasta llegar a la curva, que se la saltó. Esto se atribuye a que existía un reflejo de luz natural en el suelo, por lo que se procedió a cerrar la puerta de la habitación donde estaba instalado el entorno de conducción y se volvió a repetir la prueba. Esta vez sí se realizó correctamente la curva, pudiendo comprobar que todos los elementos del robot y el modelo de seguimiento de carriles funcionaban perfectamente.

Más tarde, se realizaron más pruebas con los demás modos de conducción que ofrecía el programa antiguo:

- **Manual:** la dirección del robot se controla mediante teclado.

- **Entrenamiento manual:** se controla el robot manualmente y se van capturando *frames* de vídeo automáticamente para la creación del conjunto de datos de entrenamiento.
- **Automático:** el robot conduce de forma autónoma siguiendo el carril infiriendo el modelo.
- **Entrenamiento automático:** conducción autónoma capturando imágenes para el *dataset* de entrenamiento.

Se detectaron diferentes errores en el fragmento de código que manejaba la conducción manual y algunas inconsistencias en la forma de elegir el modo de conducción, ya que los argumentos a introducir por terminal no coincidían con los que se especificaban.

Para no sobrescribir archivos de código antiguos, se ha creado una nueva versión de los ficheros solucionando los anteriores errores. Estos nuevos archivos están disponibles en el repositorio de GitHub <https://github.com/candsol/traffic-signs-detection-tfg> y, a medida que se vaya desarrollando la solución, serán modificados completamente, incluso los modos de conducción, adaptándose al problema a solucionar.

4.2 Construcción del *dataset*

4.2.1. Recolección de imágenes

Ahora que ya se tiene el coche robótico totalmente configurado y funcional, ya se puede programar y, por tanto, controlar de manera inalámbrica. Es momento de empezar a construir el conjunto de imágenes etiquetadas que van a servir como datos de entrada al entrenamiento del nuevo modelo. De esta forma, podremos conseguir dotar al coche de un sistema de reconocimiento de señales de tráfico. Cabe remarcar que, cuanto mayor sea el número de este conjunto, la calidad de los datos y la precisión de las anotaciones, mejor será el resultado obtenido.

Para abarcar esta tarea, primeramente, se tiene que estipular como serán estos datos. En concreto serán imágenes con una resolución de 320 por 320 píxeles (anchura por altura), que contendrán una o más señales de tráfico de entre las anteriormente expuestas en el apartado de diseño de la solución. Tanto el número de señales como las distancias focales irán variando. Para conseguir esto, se han planteado distintos enfoques o métodos de recolección, realizando distintas sesiones de obtención de imágenes capturadas por la cámara del robot. Cada una de ellas con una metodología diferente, desde la captura de *frames* con el robot en marcha hasta la recolección de estos únicamente usando la cámara.

A continuación se listan las **diferentes técnicas de recopilación** que se pusieron en práctica:

- Recolecta de *frames* con el robot **conduciendo de manera autónoma**.
- Recogida de imágenes con el **coche siendo controlado mediante teclado**.
- Obtención de datos de **forma manual usando únicamente la cámara del robot**.
- Recaudación de *frames* **cambiando los fondos**.
- Compilación de imágenes **variando la luminosidad del entorno**.

La primera idea que se puso en práctica para la recolección de *frames*, consistía en ejecutar el programa principal en modo “entrenamiento automático” para ir capturando las primeras imágenes del *dataset*. Enseguida se planteó cambiar ligeramente la manera de capturar los *frames*, ya que el programa original lo hacía de manera automática cada cierto tiempo. Por tanto, para adaptar la toma de datos a nuestro propósito, se modificó el programa añadiendo la funcionalidad de captura mediante teclado, pulsando la tecla “t”.

Una vez se obtuvieron las primeras imágenes, rápidamente, se quiso mejorar y optimizar la recolecta, ya que la anterior técnica, a pesar de que funcionaba, era muy tediosa. Se tenían que reubicar constantemente las señales de tráfico a miniatura, mientras el robot iba moviéndose de forma autónoma por el circuito y a la vez que se manejaba el teclado, que estaba fuera de escena, para capturar los fotogramas. Entonces, se pasó a recoger imágenes usando el modo “entrenamiento manual”, permitiendo el control total del coche desde el teclado, pudiéndolo parar pulsando la tecla “p” y retomando la marcha pulsando la tecla “g”. Esto permitía poder recolocar adecuadamente las señales de tráfico en el circuito, sin tener que esperar a que el coche volviese a pasar de nuevo por el mismo punto conduciendo de forma automática. También permitía poder tomar distintas capturas de las señales en la misma posición, pero tomadas desde diferentes ángulos.

Finalmente, se llegó a capturar más fotogramas pausando totalmente la conducción del coche robótico y, únicamente, manipulándolo físicamente, se consiguió capturar muchas más imágenes, incluso controlando solamente la cámara. Por lo que se decidió crear un pequeño *script* Python que permitiese la captura de fotogramas de vídeo de la cámara. Dejando de lado el programa principal que controlaba los demás componentes del robot, ya que se decidió que era la manera más óptima para la construcción del *dataset* y se ajustaba a las necesidades del problema.

Con el objetivo de aumentar aún más la cantidad de datos, se realizaron más sesiones de recolecta en distintas franjas horarias del día para poder variar la iluminación, incluso usando iluminación artificial. También se cambiaron los fondos de las imágenes capturadas.



Figura 4.3: Ejemplos de *frames* capturados con distintos fondos e iluminaciones.
Fuente: Elaboración propia.

4.2.2. Etiquetado y anotación de los datos

Una vez se tenían todas las imágenes de entrenamiento recolectadas, era el momento de etiquetarlas correctamente. Para ello, dado que se trata de un problema de detección de objetos, se tienen que etiquetar en un formato establecido. Debido a que cada *frame* puede contener uno o más objetos, de igual tipo o distinto, se debe etiquetar cada objeto con su tipo o *label* y guardar las coordenadas correspondientes a la caja que envuelve el objeto (*bounding boxes*).

Dado que se trata de un proceso laborioso tener que delimitar todos los objetos de todas las imágenes, es importante trabajar con una herramienta que sea simple e intuitiva. Como bien se ha expuesto en el apartado del diseño de la solución (capítulo 3), se ha utilizado la aplicación web Make Sense, que permite poder realizar las anotaciones de cada imagen de manera gráfica.

Este proceso supuso mucha horas de anotación de los datos, ya que se tenía que hacer lo más preciso posible para obtener los mejores resultados. Por tanto, se repitió varias veces el procedimiento, a medida que el *dataset* iba aumentando.

A continuación, se muestran algunos ejemplos de lo que fue la tarea de anotación del conjunto de datos de entrenamiento usando Make Sense:



Figura 4.4: Anotación de *frame* de entrenamiento usando Make Sense.
Fuente: Elaboración propia.



Figura 4.5: Otro ejemplo de anotación, con distinto fondo.
Fuente: Elaboración propia.

Finalmente, como resultado de este proceso, se obtuvieron y descargaron unos ficheros XML (*extensible markup language*), en concreto un fichero por cada imagen. Estos archivos contienen la información necesaria para poder tratar los datos más adelante en el entrenamiento del futuro modelo. Entre otras cosas incorporan datos como el **nombre de la imagen**, la **ruta**, el tamaño (*width, height*) y, por cada objeto que haya en la imagen, el **tipo de objeto** y las **coordenadas** (*xmin, ymin, xmax, ymax*) correspondientes de la **bounding box** que acota dicho objeto.

4.2.3. Aumento de datos

Una técnica que se utiliza mucho a la hora de construir una base de datos para el entrenamiento de un modelo es la ampliación de nuevos datos o ***data augmentation***, en este caso generando nuevas imágenes a partir de las que se tienen. Esto es posible aplicando **transformaciones geométricas** como la rotación, el escalado, el recorte, la translación y el espejado o volteo (*flip*) horizontal o vertical. Otra manera de aumentar los datos es realizando **transformaciones de color y brillo** como la alteración de intensidad de brillo, modificación de contraste, la saturación de los colores y el desenfoque (*blur*). Otro tipo de transformación muy utilizado es la **introducción de ruido** de manera aleatoria por la imagen. Cabe decir que existen muchos más tipos de operaciones que se pueden aplicar a imágenes. Todas estas transformaciones, si se realizan correctamente y se ajustan a las necesidades del problema, contribuyen al aumento del *dataset* y hacen más robusto el modelo entrenado.

En este caso, para la tarea de detectar señales de tráfico, se han desestimado muchas de las opciones de transformación. Por ejemplo, no tenía sentido realizar el volteado vertical, porque las características de las señales cambiaban totalmente. Como bien se explica en el apartado 3.2, algunas señales sí se pueden voltear horizontalmente sin cambiar las características de las mismas. Por tanto, en aquellos *frames* que solamente aparecían señales que se les podía aplicar el *flip* horizontal, se realizó la transformación. Para ello, fue necesario escribir un *script Python* para poder modificar los ficheros XML de las señales de izquierda y derecha y cambiar la etiqueta, ya que, al aplicar la transformación, el significado de estas señales se intercambiaba.

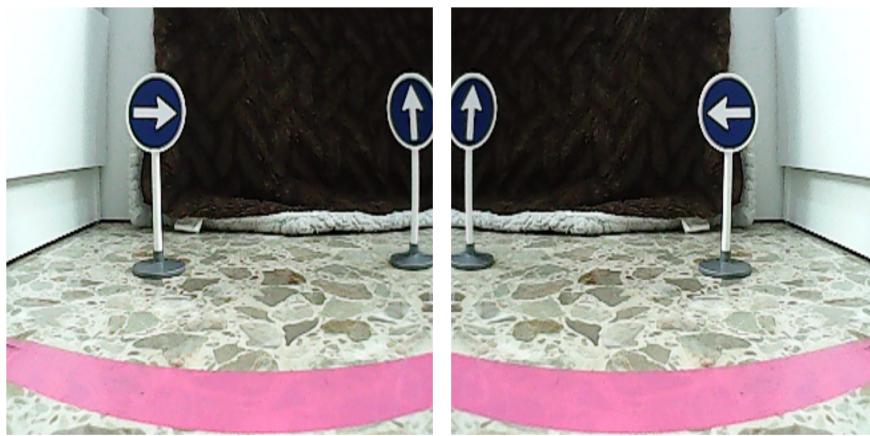


Figura 4.6: Volteado horizontal de la señal de sentido obligatorio derecho.
Fuente: Elaboración propia.

Otro problema que se encontró fueron las coordenadas de las *bounding boxes*, dado que, a menos que la señal estuviera completamente centrada en la imagen, estas cambiaban, como se puede observar en la anterior ilustración. La solución fue recalcularlas

mediante dos fórmulas matemáticas triviales. Dado que se sabía el tamaño de la imagen y las coordenadas de la altura no cambiaban (y_{min} , y_{max}), la nueva $x_{min}' = 320 - x_{max}$ y la nueva $x_{max}' = 320 - x_{min}$. Por tanto, se mejoró el anterior *script* para que hiciese estos cálculos y modificase en los correspondientes archivos XML, las etiquetas, y en caso de ser necesario, las coordenadas.

También se realizaron ligeras rotaciones de 15 grados, ya que si eran mayores, algunas señales como las que tienen números, letras o flechas, cambiaban por completo las características. De nuevo, hizo falta recalcular las coordenadas de las cajas delimitadoras, calculando la matriz de rotación y aplicando la siguiente fórmula:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R \begin{pmatrix} x - cx \\ y - cy \end{pmatrix} + \begin{pmatrix} cx \\ cy \end{pmatrix}$$

Donde ' cx ' y ' cy ' son las coordenadas del centro de la imagen. En este caso 160 ambas, ya que la imagen es de 320x320.

Finalmente, se generaron nuevas imágenes, agregando ruido y cambiando la intensidad de color e iluminación. Estas transformaciones no suponían cambio de las coordenadas de la *bounding boxes*. A continuación se muestran unos ejemplos:



Figura 4.7: Ejemplos de *frames* generados para aumentar el *dataset*.
Fuente: Elaboración propia.

4.3 Entrenamiento del modelo de detección de señales

4.3.1. Configuración del entorno

Una vez se tiene listo el conjunto de datos de entrenamiento, es momento de preparar y configurar el entorno de programación donde se realizará el entrenamiento. Tal y como se ha especificado en el capítulo 3 del diseño de la solución, concretamente en la sección de “*software utilizado*”, se ha utilizado la plataforma Google Colab y los cuadernos Jupyter que esta ofrece para el desarrollo del entrenamiento. Colab permite trabajar y ejecutar programas Python, el lenguaje de programación por excelencia para el entrenamiento de redes neuronales; además, permite el uso de unidades de computación como GPUs y TPUs, esenciales para esta tarea.

Así pues, para configurar el entorno, se tenía que empezar por instalar la **API de detección de objetos** que ofrece el *framework TensorFlow*. Esta tarea puede parecer trivial, pero requirió de varios intentos fallidos, puesto que, para instalar correctamente la API, había que instalar una versión de TensorFlow menor, concretamente la 2.13.1. Esto fue debido a un conflicto de dependencias de versiones entre los diferentes paquetes instalados. Después de una búsqueda entre múltiples foros para resolver este problema, se encontró la forma de instalar correctamente la API siguiendo los pasos de Evan Juras y su cuaderno Jupyter “Train_TFLite2_Object_Detection_Model.ipynb”, disponible en su GitHub <https://github.com/EdjeElectronics> en el repositorio “TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi”.

Finalmente, se clonó el repositorio oficial de TensorFlow y se instaló la API de detección de objetos. Además, se tuvo que instalar la versión 11 de CUDA para poder mantener la compatibilidad de la GPU con la versión de TensorFlow instalada. Seguidamente, se ejecutó con éxito un programa que proporciona la API, “model_builder_tf2_test.py”, que testeaba que todo funcionase correctamente. Todos estos procedimientos se pueden ver en el cuaderno Jupyter que se encuentra en el repositorio GitHub del proyecto.

Ahora ya se tenía el entorno de entrenamiento completamente instalado y funcional y, por tanto, ya se podía empezar a tratar los datos y configurar el entrenamiento.

4.3.2. Tratamiento de los datos

Para la carga y manejo de los datos o imágenes de entrenamiento, primeramente, se tenía que montar la ruta de Google Drive donde previamente se habían subido todas las

imágenes y archivos XML correspondientes a las anotaciones en un archivo ZIP que tenía la siguiente estructura:

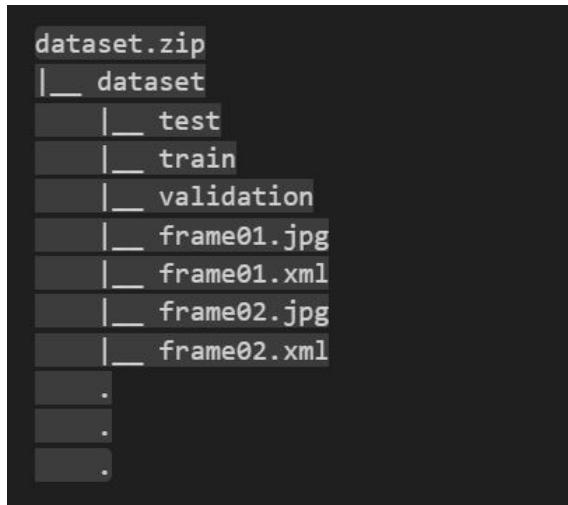


Figura 4.8: Estructura de directorios para el manejo de datos.

Fuente: Elaboración propia.

Montada la ruta de Google Drive, ya se podía copiar y descomprimir el archivo comprimido “datastet.zip” en el entorno local de Colab. El siguiente paso era separar las diferentes imágenes y anotaciones de manera aleatoria en las carpetas de “***train***”, “***test***” y “***validation***”. La carpeta de entrenamiento contenía un 80 % del *dataset* completo e iba destinada a la tarea de entrenamiento, es decir, al reconocimiento de los patrones y características de dichas imágenes. La partición de validación incluía el 15 % del total del conjunto de datos y estos eran usados durante el entrenamiento para validar el rendimiento del modelo, ya que eran datos que no se habían visto durante el entrenamiento. La validación es crucial para la detección de sobreajuste o *overfitting* y ajusta los hiperparámetros del modelo. Finalmente, el *split* de test con el 15 % restante de los datos, útil para la comprobación final del modelo después de completar el entrenamiento.

Seguidamente, se manipularon los archivos XML para extraer las anotaciones y convertirlas a CSV usando la librería Pandas. Este paso fue necesario para poder generar los archivos **TFRecord**. TensorFlow ofrece un formato de archivos binarios llamados TFRecord, que son utilizados para almacenar cantidades grandes de datos de una forma eficiente. Estos permiten el manejo óptimo de datos durante el entrenamiento, especialmente cuando se trabaja con imágenes. En concreto, se escribieron dos archivos TFRecord, uno con los datos de entrenamiento y otro con los de validación. Estos archivos contenían la altura y anchura de la imágenes, el nombre de los archivos JPG, los *frames* codificados en *bytes*, el formato, las coordenadas de los *bounding boxes* y las etiquetas.

Finalmente, se escribió un archivo llamado “***labelmap***”, necesario para el entrenamiento y requerido por la API de detección de objetos para el mapeo de las etiquetas de clasificación con su correspondiente valor numérico. Este es un archivo con extensión

“.pbtxt” . Con todo esto explicado y hecho, ahora ya se podía empezar a configurar el *pipeline* de entrenamiento.

4.3.3. Arquitectura del modelo y ajuste del proceso de entrenamiento

Llegado a este punto, se tenía que elegir cual sería la arquitectura idónea para la tarea y el *dataset* en cuestión. Y, por tanto, saber qué modelo se iba a usar como base del entrenamiento. Como bien se ha especificado en el apartado 3.1, se decidió usar la **arquitectura SSD** (*Single Shot MultiBox Detector*), ya que es la más óptima para los recursos de los que se disponen.

Se usó el modelo “**ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8**”, un modelo preentrenado que ofrece la API de detección de objetos de TensorFlow. Se eligió este porque, de entre todos los que la API ofrece, es el que trabaja con el tamaño de imagen de entrada más pequeño y se ajusta al tamaño de imagen del *dataset* creado.

Estos son unos diagramas que describen la arquitectura de este modelo a alto nivel:

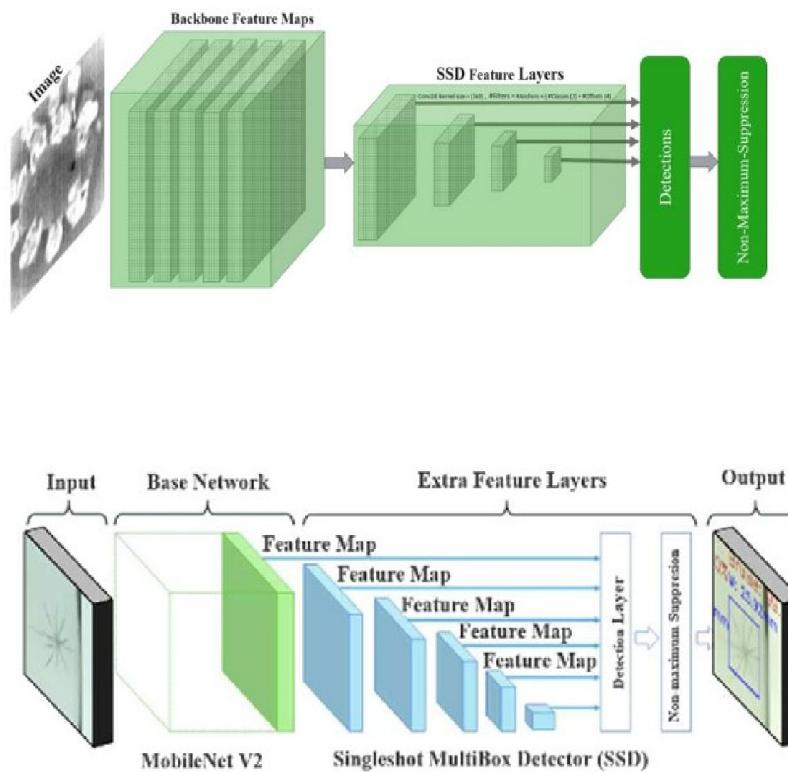


Figura 4.9: Diagramas de alto nivel de la arquitectura del modelo SSD MobileNet V2.

Fuente: [15], [16]

Elegido el modelo, se descargó el **checkpoint** o modelo preentrenado y el correspondiente **archivo de configuración**. Este archivo sirve para configurar el *pipeline* de entrenamiento, es decir, los pasos, procesos y parámetros de la tarea de entrenamiento. En dicho archivo, se especificaron la ruta del modelo base, la ruta del archivo TFRecord de entrenamiento y de validación, la ruta del archivo de mapeo de etiquetas o *label map*, el tamaño del *batch*, el número de pasos de entrenamiento, la tasa de aprendizaje, entre otros parámetros.

Una vez se tuvo todo el entrenamiento configurado, era la hora de empezar el entrenamiento, usando el script “*model_train_tf2.py*” que la API proporciona y especificando todas las rutas correspondientes al fichero de configuración, el directorio de entrenamiento donde se iba a guardar la versión entrenada, el número de *steps* o pasos de entrenamiento y el número de pasos de evaluación.

```
1 # Empezar entrenamiento
2 !python /content/models/research/object_detection/model_main_tf2.py \
3   --pipeline_config_path={custom_config_path} \
4   --model_dir={model_training_path} \
5   --alsologtostderr \
6   --num_train_steps={steps} \
7   --num_eval_steps=50
```

Código 4.1: Comando de ejecución del entrenamiento.

Para llegar a obtener el modelo definitivo, es decir, el de mejor precisión, se tuvieron que realizar muchos entrenamientos cambiando parámetros como la tasa de aprendizaje o el tamaño de *batch* y mejorando y ampliando el *dataset*. Por lo que se dedicó bastante tiempo a la anterior y actual sección; es decir, a la construcción del conjunto de datos y al entrenamiento. En el repositorio del proyecto se encuentra también el nuevo modelo entrenado y su correspondiente archivo de configuración.

Para la elección de los hiperparámetros a utilizar en el entrenamiento del modelo definitivo, se hicieron diferentes pruebas de entrenamiento con un conjunto reducido del *dataset*. Se alternó entre diferentes tamaños de *batch* y valores de *learning rate*. La siguiente tabla muestra algunos valores probados y justifica la elección final de los parámetros:

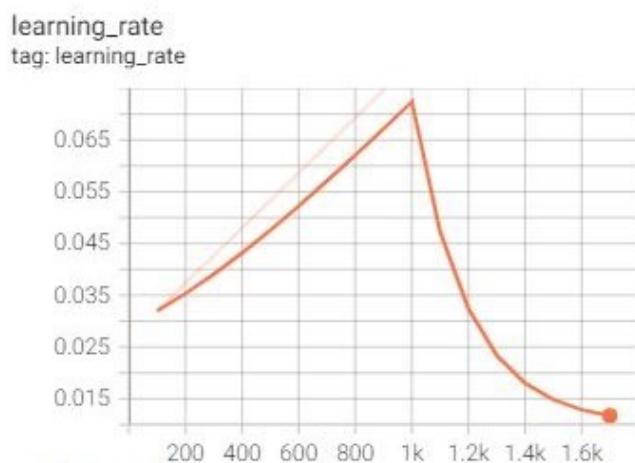
Batch Size	Learning Rate	Resultados obtenidos
12	0.08	Tardaba más en converger, pero se obtenían resultados aceptables. Faltaba precisión en las pruebas de con el conjunto de test, daba bastantes falsos positivos.
16	0.08	Se obtenían mejores resultados, aunque seguía faltando precisión.
32	0.08	Con un valor tan grande de lote, la memoria RAM del entorno Google Colab era un limitante, además, el modelo convergía más rápido sin llegar a generalizar correctamente.
16	0.01	Fue la mejor combinación probada, el modelo convergió más lento, pero al bajar la tasa de aprendizaje, el modelo se ajustó más al mínimo óptimo de la función de pérdida y ganó más precisión.

Figura 4.10: Tabla elección hiperparámetros.

Fuente: Elaboración propia.

El número de pasos de entrenamiento final fue de 1800. Aunque, en este caso, este valor no era relevante configurarlo, dado que, haciendo uso del *script* de entrenamiento que proporciona la API de detección de objetos y viendo el progreso de las gráficas de las funciones de pérdida usando TensorBoard, se pudo decidir cuando parar el entrenamiento manualmente. El programa de entrenamiento va guardando el modelo cada vez que ajusta los pesos, por tanto, cuando se para la ejecución de la celda del cuaderno Jupyter, se para el entrenamiento y se queda guardada en el directorio indicado la última versión del modelo.

También, se utilizó un ajuste dinámico de la tasa de aprendizaje, conocido como *learning rate Warm-Up*. Se comienza con una tasa baja y se va incrementando el valor gradualmente durante los primeros pasos de entrenamiento, luego se mantiene o se va disminuyendo el valor. Seguidamente, se muestra el gráfico que describe el *learning rate* del entrenamiento definitivo:

**Figura 4.11:** Gráfico del valor de *learning rate*.

Fuente: Elaboración propia.

Para finalizar con este apartado, a continuación se muestran las gráficas de las funciones de pérdida del entrenamiento del modelo definitivo:

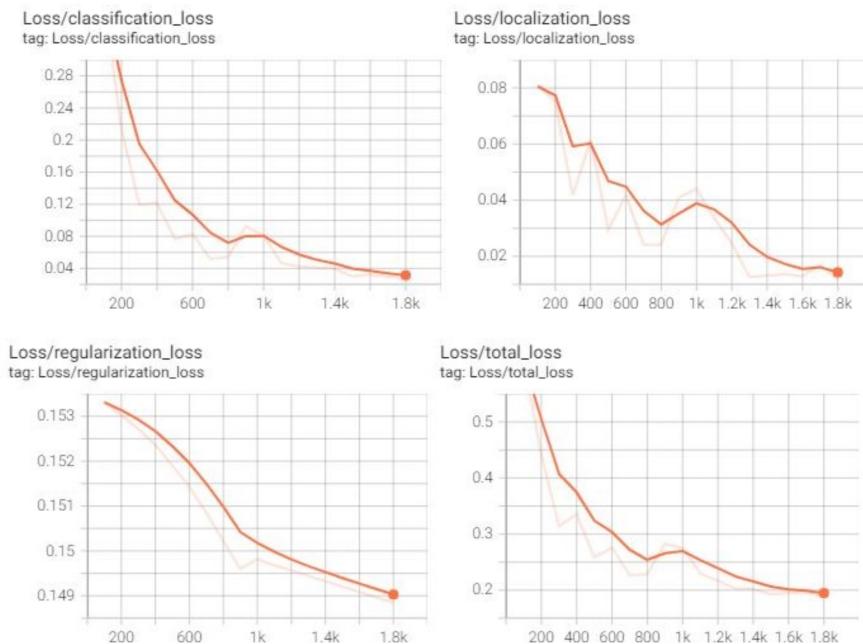


Figura 4.12: Gráficos de las funciones de pérdida.

Fuente: Elaboración propia.

4.3.4. Pruebas con el conjunto de test y exportación del modelo

Una vez se obtuvo el modelo entrenado y listo para detectar señales de tráfico, era momento de hacer unas pruebas en el entorno de Google Colab, antes de testear el modelo en el propio coche robótico. Para ello, se utilizó el conjunto de test previamente creado, un conjunto de imágenes representativo del *dataset* completo, que no se había utilizado durante el proceso de entrenamiento.

Primero, se convirtió el modelo entrenado en un formato que pudiese ser interpretado por la TPU Coral Edge, concretamente TensorFlow Lite. El resultado fue un modelo con la extensión “*.tflite*”, una versión más ligera de TensorFlow utilizada en dispositivos de bajos recursos. Para ello, primeramente se tenía que extraer el grafo del modelo, un grafo que contiene información como la arquitectura y los pesos, usando el *script* Python “*export_tflite_graph_tf2.py*” que la propia API ofrece. Finalmente, utilizando el grafo exportado y el “*TFLiteConverter*”, se convirtió el modelo en dicho formato.

Seguidamente, se realizaron pruebas con el conjunto de test y con otro conjunto nuevo de imágenes, con iluminación artificial y otros fondos, que no habían sido utilizadas en el entrenamiento. El código y las pruebas realizadas se encuentran, de nuevo, en el cuaderno Jupyter del repositorio del proyecto.

A continuación se muestran algunos ejemplos de las pruebas realizadas:



Figura 4.13: Pruebas con el conjunto de test del modelo de detección de señales entrenado.

Fuente: Elaboración propia.

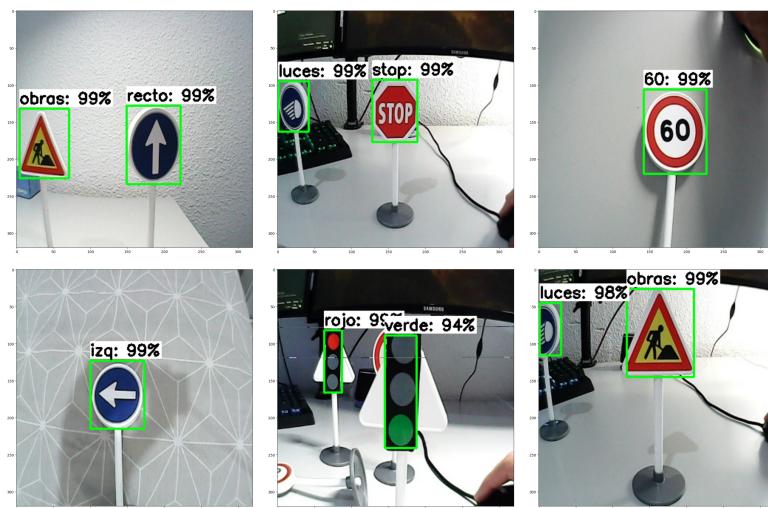


Figura 4.14: Pruebas con luz artificial del modelo de detección de señales entrenado.

Fuente: Elaboración propia.

Finalmente, se obtuvo un 97,3 % de detecciones correctas en el conjunto de test, ya que habían 76 señales a detectar y solo 2 no fueron detectadas, todo esto, ajustando el *threshold* a 0.5.

4.4 Elaboración del programa de conducción autónoma

4.4.1. Importación y coexistencia de ambos modelos

Una vez exportado el modelo entrenado, se probó en el coche robótico. Se empezó creando una nueva clase en el programa “*autonomus_driver.py*” llamada “*TrafficSign-Detector*”, esta clase inicializa el intérprete con el nuevo modelo y contiene el método “*detect_signal*”, que preprocesa el *frame* actual, infiere el modelo, analiza el *output* del modelo e invoca la acción de la señal detectada. Las nuevas versiones de los tres archivos Python que conforman el programa de conducción, “*smart_pi_car_2024.py*”, “*autonomus_driver_2024.py*” y “*signals.py*”, se encuentran en el repositorio GitHub mencionado anteriormente en la sección 4.1.

El primer problema que se detectó al ejecutar el nuevo modelo, fue el tiempo de inferencia, este era muy superior en comparación con el modelo de clasificación para el seguimiento de carriles del que ya se disponía. En concreto, uno tardaba 0.1 segundos y el otro 0.7. Por lo que hacía cuello de botella y retardaba la tarea de seguimiento del carril, que necesita una mayor tasa de refresco. El coche se salía del carril al llegar a la primera curva, ya que se iban acumulando *frames* por procesar, y cuando este llegaba, los ángulos de giro aún eran los de *frames* de cuando el coche aún estaba en la recta. La solución más efectiva para poder inferir ambos modelos y que el coche siguiera correctamente el carril fue trabajar con hilos. El hilo principal del programa se encargaba de procesar el modelo de seguimiento de carril, mientras la tarea más pesada de inferir el modelo de detección de objetos, la hacía otro hilo. Esto no arregló del todo el problema, ya que, otro limitante era la TPU, que solo había una. Seguía ralentizando la tarea de seguimiento de carril la inferencia del nuevo modelo. Finalmente, se decidió bajar la frecuencia con la que se infería el nuevo modelo, ejecutando la tarea de detección de señales cada dos segundos.

```
1 def traffic_sign_detection_task(self):
2     while self.keep_detecting:
3         if (self.actual_frame is not None):
4             _ = self.traffic_sign_detector.detect_signal(self.
5                 actual_frame)
6             time.sleep(2)
7
8 def start_detection_task(self):
9     threading.Thread(target=self.traffic_sign_detection_task).start()
10
11 def drive(self, mode='auto', speed=20):
```

```

11     self.back_wheels.speed = speed
12     if mode == "auto":
13         self.start_detection_task()
14         while self.camera.isOpened():
15             _, self.actual_frame = self.camera.read()
16             if(self.keep_following):
17                 image_lane = self.lane_follower.follow_lane(self.
18                     actual_frame)

```

Código 4.2: Ejemplo del código que hace posible la coexistencia de ambos modelos.

4.4.2. Programación del comportamiento del robot

Para la programación de las acciones que el pequeño coche autónomo tenía que realizar al detectar alguna de las señales descritas anteriormente, se creó una clase para cada señal en el archivo Python “*signals.py*”. Estas clases heredaban de la clase “*Signal*”, así, para cada señal, se podía sobrescribir el método “*play*” y, de esta forma, programar cada acción del robot para cada caso.

Un problema que se detectó al poner en práctica las primeras pruebas de comportamiento, fue la temprana reacción del robot al detectar una señal. Por ejemplo, el coche se detenía un metro antes de la señal de *STOP*. Esto se solucionó fácilmente definiendo y programando el método estático “*esta_cerca*” de la clase “*Signal*”. Este método calcula la altura de la señal, devolviendo *True* o *False* si es lo suficientemente grande como para considerarla cerca o no.

```

1 class Signal(object):
2     def play(self, car):
3         pass
4
5     @staticmethod
6     def esta_cerca(signal_detected):
7         signal_detected_width = (signal_detected[2][3] -
8             signal_detected[2][1]) * 100
9         return signal_detected_width / 320 > 0.1

```

Código 4.3: Ejemplo del código de la clase *Signal*.

Por tanto, en el método ”*detect_signal*”, se inicializa la clase de la señal detectada y se hace la comprobación para saber si está lo suficientemente cerca; si lo está, se ejecuta la acción llamando al método “*play*” y si no, se obvia la interpretación de dicha señal hasta que el coche avance y vuelva a detectar la señal más cerca.

Para la programación del comportamiento en cada señal se sobrescribió el método “*play*” de las siguientes formas:

- **Señal de STOP.** Para esta señal no hubo muchas complicaciones. Simplemente, se guardó en una variable la velocidad actual del coche, haciendo uso de la librería de control del coche y, mediante la misma, se paró el coche durante 3 segundos, usando el método “*sleep*” del módulo “*time*”, para luego retomar la marcha con la misma velocidad guardada.
- **Señales de limitación de velocidad.** Se creó un atributo ”*vel*” en el constructor de la clase “*Velocidad*” que se pasa como argumento al instanciarla, dependiendo de la señal detectada puede tomar el valor 30 o 60. Este atributo se utiliza en el método “*play*” para ajustar la velocidad del robot.
- **Señal de ceda el paso.** Fue muy trivial, se ajustó la velocidad a 20, ya que si se ponía a 10 el coche directamente no se movía, durante 3 segundos y luego se ajustó a 30. De esta forma se consiguió simular de manera aproximada el comportamiento real ante un ceda.
- **Señal de luces.** Mediante el uso de la librería “*blinkstick*”, que ofrece el fabricante de las luces LED usadas en el proyecto, se localiza el dispositivo con la llamada al método “*find_first()*” y se ponen los cuatro LEDs en blanco. Para dar un poco de juego a las señales, se decidió que estas luces se quedarían fijas hasta que se volviese a detectar la misma señal, esta vez se apagarían. De esta forma, se podían encender y apagar las luces únicamente con la misma señal. Para conseguir esto, hizo falta crear el atributo “*lights*” de la clase principal del programa de conducción. Este atributo es un booleano que actúa como *flag* para encender o apagar las luces.
- **Señal de obras.** Para simular el comportamiento precavido ante una señal de obras, se ajustó la velocidad a 20 igual que el ceda, pero esta vez la acción dura más.
- **Señal de ir recto.** Para forzar al coche que vaya recto, sin tener en cuenta la salida del modelo de seguimiento de carriles, se paró la inferencia de dicho modelo ajustando a *False* el *flag* ”*keep_following*”. También, calculando el tiempo que se necesita para recorrer un metro real yendo recto y, consultando la velocidad actual del robot, se puede saber el tiempo que dura la acción. Pasado ese tiempo, el coche vuelve a inferir el modelo de seguimiento de carriles.

```
1 class Recto(Signal):  
2     def play(self, car):  
3         if(car.back_wheels._speed > 0):  
4             logging.debug('Yendo RECTO...')  
5             car.keep_following = False
```

```
6         car.front_wheels.turn(car.STRAIGHT_ANGLE)
7 # En 10 segundos recorre 1 metro con una velocidad
8 # de 20
9 t = 10
10 v = 20
11 actual_vel = car.back_wheels._speed
12 tiempo = (t * v) / actual_vel
13 time.sleep(tiempo)
14 car.keep_following = True
```

Código 4.4: Ejemplo del código del comportamiento del coche al detectar la señal de ir recto.

- **Señales de izquierda y derecha.** Para estas señales se realizaron una animaciones de luces naranjas en la tira LED, para simular los intermitentes. De nuevo, consultando la velocidad actual del robot, se calculó el tiempo necesario para recorrer un metro, pero esta vez girando las ruedas a 45º o a 135º, dependiendo de si se detecta la señal de izquierda o derecha respectivamente.
- **Semáforo rojo.** Se ajustó el atributo “*keep_following*” a *False* y se paró el coche poniendo a 0 la velocidad. De esta forma, hasta que no se detecte la señal de semáforo verde, el coche no continúa la conducción.
- **Semáforo verde.** Se consulta la velocidad actual del robot y, si es 0, se pone a 20. En todos los casos, se ajusta el atributo “*keep_following*” a *True*.

CAPÍTULO 5

Implantación y pruebas

5.1 Puesta en marcha en distintos circuitos

Fueron muchas las veces que se puso en funcionamiento el robot y se probó el programa de conducción hasta llegar a la solución final. Era un trabajo de prueba y mejora, cada vez se iba afinando más la tarea de detección de señales, tanto la manera de inferir el modelo, como el comportamiento del robot.

Cuando se obtuvo una solución funcional, se realizaron una serie de pruebas en diferentes circuitos, en distintos suelos y alterando la iluminación. Primeramente se establecieron los circuitos, ya que, como bien se ha mencionado anteriormente, se trata de un entorno controlado. Esto significa, que para poder simular que el coche va a la izquierda, por ejemplo, se tiene que construir un carril con una bifurcación y la señal de ir a la izquierda. Así pues, primero se probaron las señales que no cambiaban el rumbo de la trayectoria en un circuito cerrado y con un único carril. Despues, se probaron las señales de ir recto, a la izquierda y a la derecha en otro circuito adaptado a dichas acciones.



Figura 5.1: Los dos tipos de circuitos utilizados para poner a prueba la solución desarrollada.
Fuente: Elaboración propia.

Finalmente, se probaron esos mismos circuitos o parecidos en otros suelos y cambiando la iluminación.

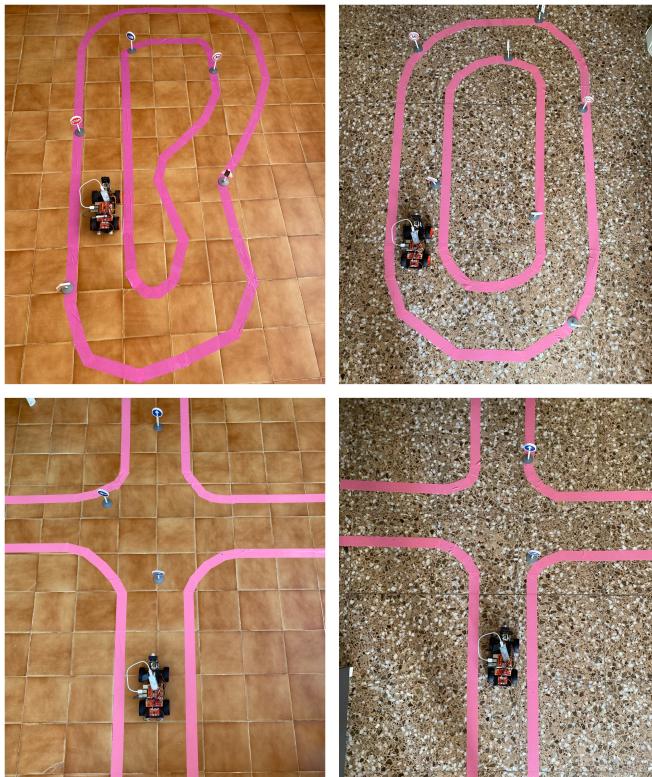


Figura 5.2: Ejemplos de algunos circuitos usados en las pruebas con distintos suelos e iluminaciones.

Fuente: Elaboración propia.

5.2 Análisis de los resultados obtenidos

Como resultado, en términos generales, el coche robótico conseguía reconocer todas las señales de tráfico estipuladas y utilizadas en el entrenamiento del modelo. También, era capaz de reaccionar sin ningún problema y de manera automática al detectarlas.

En las **pruebas realizadas en circuitos cerrados**, el coche no se salía en ningún momento del carril. Esto quiere decir, que la inferencia del nuevo modelo, finalmente no ralentizaba la tarea de seguimiento del carril. Por otro lado, en las señales utilizadas en estas pruebas, el modelo las detectaba correctamente el 90% de los casos, bajando un poco la precisión cuando la luz o las condiciones eran más adversas. Por ejemplo, cuando la cámara estaba orientada en contra de la luz. Pero, en general, se puede decir que el modelo consigue detectar correctamente las señales en la mayoría de las situaciones, y se puede considerar un modelo robusto, teniendo en cuenta el entorno controlado a baja escala, donde la seguridad vial no es un objetivo a alcanzar.

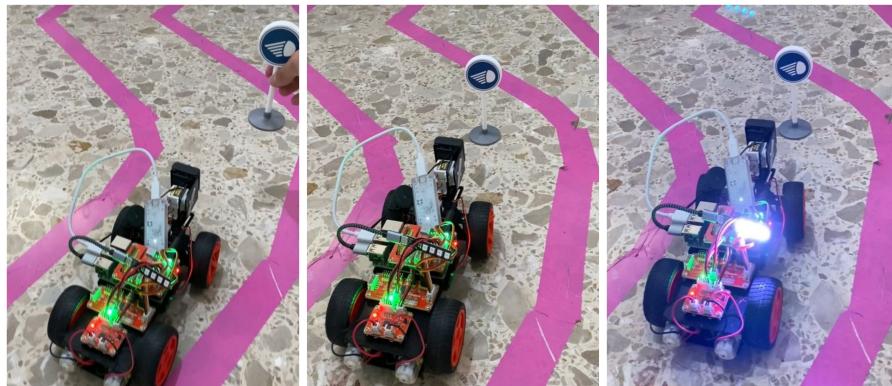


Figura 5.3: El coche autónomo detecta y reacciona ante la señal de encendido de luces.

Fuente: Elaboración propia.

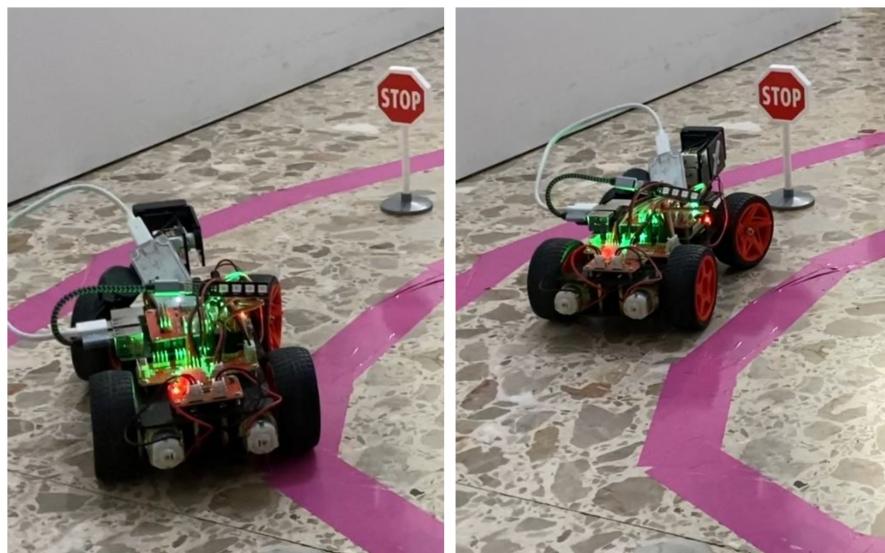


Figura 5.4: El coche autónomo se detiene al detectar la señal de *STOP*.

Fuente: Elaboración propia.

En cuanto a las **pruebas con carriles bifurcados**, las tres señales utilizadas (recto, izquierda y derecha), también se conseguía detectarlas en la mayoría de las ocasiones, siendo menor la precisión obtenida en condiciones desfavorables de luz. Por lo que respecta a las acciones a realizar para cada señal, estas eran más complicadas de simular, y se tuvo que corregir la forma del carril en alguna ocasión para que el coche pudiese completarlo hasta el final. Esto quiere decir que, por ejemplo, para que el coche realizase correctamente el giro programado a la izquierda, se tuvo que corregir el ángulo de cierre de la curva o la intensidad de curvatura; para que, al finalizar la maniobra, el vehículo pudiese seguir detectando y siguiendo correctamente el carril.

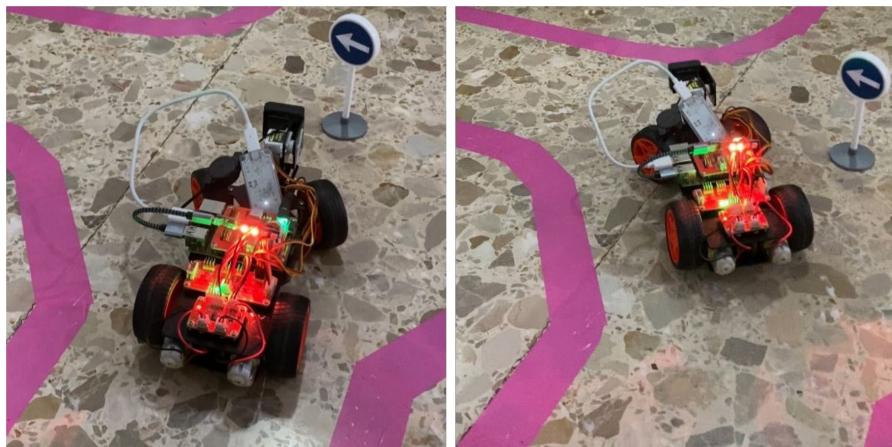


Figura 5.5: El coche autónomo detecta y reacciona ante la señal de sentido obligatorio izquierdo.
Fuente: Elaboración propia.

CAPÍTULO 6

Conclusiones

Durante la realización del presente trabajo, se ha dotado de un sistema de reconocimientos de señales de tráfico a un pequeño coche robótico, permitiendo interpretar de forma autónoma un conjunto de señales establecido en un entorno controlado. En este capítulo final, se realiza un repaso a los diferentes objetivos y subobjetivos marcados al inicio del proyecto, se extraen algunas conclusiones y conocimientos obtenidos y se exponen algunas propuestas de mejora para futuras continuaciones del trabajo.

Empezando por los subobjetivos estipulados al principio:

- 1. Implementar un programa para la recolecta de *frames* y posterior creación de una base de datos de las señales de tráfico a entrenar.**

Este objetivo se ha cumplido con el desarrollo y adaptación del programa principal de conducción del que se disponía, haciendo uso de la librería *picar* de SunFounder para controlar el robot. La nueva versión del programa Python permite controlar totalmente el robot mediante teclado, pudiendo parar debidamente el pequeño coche y capturar imágenes manualmente. También, mediante el uso de diferentes *scripts* de procesado de imágenes se ha conseguido aumentar el conjunto de datos. Para finalizar, gracias a la herramienta web Make Sense, se ha conseguido anotar correctamente los *frames* de entrenamiento y, por tanto, terminar de construir el *dataset*. Una mejora para futuras ampliaciones sería la cámara, un sensor de más calidad mejoraría notablemente el resultado final.

- 2. Entrenar un modelo de inteligencia artificial usando aprendizaje profundo para la detección de señales de tráfico.**

Se consiguió entrenar este modelo de detección de señales, gracias al uso de la API de detección de objetos de TensorFlow y con el modelo preentrenado “ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8”. También, se pudo alcanzar este objetivo haciendo uso del entorno Google Colab y la GPU T4 que proporciona. Para futuras mejoras de este modelo, se podrían utilizar los pesos de esta última versión y reentrenar

el modelo con más imágenes, intentando obtener, de esta forma, una mejor precisión. También, se podría aumentar el número de señales a detectar, o que detecte peatones e incluso señales del pavimento.

3. Desarrollar un programa principal para la conducción autónoma del robot.

Este objetivo ha sido cumplido con el desarrollo del programa Python “autonomus_driver_2024.py”. Se ha conseguido que ambos modelos puedan coexistir teniendo en cuenta la limitación de capacidad de cómputo. Como mejora se puede plantear la adquisición de otra TPU Coral Edge o la compra de otra TPU más potente y compatible con la Raspberry, que pueda inferir los dos modelos sin problemas.

4. Ejecutar la solución desarrollada y realizar pruebas.

Gracias al logro de todos los anteriores subobjetivos, se pudo poner en funcionamiento el sistema de detección de señales y se realizaron diferentes pruebas de la solución en distintos circuitos, suelos y variando iluminaciones.

Dado que se han alcanzado estos cuatro subobjetivos que, dividían y estructuraban el trabajo a realizar para llevar a cabo el desarrollo de la solución, se puede decir que se ha logrado el objetivo principal de este proyecto: **conseguir que el coche robótico pueda reconocer señales de tráfico y actuar en consecuencia, todo esto en un escenario a pequeña escala que se asemeje a la realidad.**

Además, durante el desarrollo de la solución final, se han adquirido una serie de enseñanzas útiles para la realización de futuros proyectos similares:

- Trabajar con equipos robustos y de mayor calidad, no solo aumenta la calidad final del proyecto, sino que también proporciona comodidad y eficiencia durante el desarrollo.
- La mejor forma de mejorar el resultado de un modelo de detección de objetos es aumentando el conjunto de datos de entrenamiento, variando las posiciones, los ángulos, la iluminación y anotando de manera precisa y uniforme las *bounding boxes* de los correspondientes objetos a detectar.
- Es posible entrenar un modelo de detección de señales con un *dataset* reducido (2000 imágenes) y usando un modelo de detección de objetos preentrenado.
- El modelo de detección de objetos es más pesado que el de clasificación del ángulo de giro, por tanto, para poder ejecutar ambos en tiempo real es necesario más capacidad de cómputo.

Por otro lado, existen muchas líneas abiertas de mejora para este proyecto. Partiendo de un coche con una autonomía parcial, que es capaz de seguir carriles y detectar señales

de tráfico, se puede mejorar este robot, obteniendo un mayor grado de autonomía. Algunas de las mejoras propuestas son:

- **La mejora de hardware**, mejorando la cámara, la TPU o incluso la Raspberry, ya que a veces el lector de la tarjeta micro SD daba algún problema
- **Mejorar el modelo de seguimiento de carriles**, para que pueda conducir usando otros colores de carriles o ajustando la velocidad en las curvas.
- **Mejorar el modelo de detección de señales**, ampliando el número de señales a detectar o identificando peatones u otros objetos.
- **Implementar un sistema de aparcamiento autónomo**, dado que, la librería permite al coche ir marcha atrás. Quizás usando una segunda cámara como cámara trasera o usando únicamente el sensor principal.

En conclusión, se dan por alcanzados los objetivos de este proyecto, habiendo conseguido que el vehículo robótico pueda detectar un conjunto de señales y las interprete correctamente, todo ello mientras conduce de manera autónoma sin salirse de un carril delimitado por dos cintas. Además, se ha contribuido humildemente a esta área de la informática cada vez más creciente, dejando público el trabajo, para que otros particulares puedan experimentar y mejorar en el ámbito de la detección de objetos. Es muy importante fomentar, intercambiar conocimientos y motivar a la comunidad de científicos e ingenieros para que la tecnología siga avanzando.

Bibliografía

- [1] de la Torre, A. (2024, febrero). La DGT ya anticipa la llegada de coches completamente autónomos: Un nuevo reglamento está en marcha para 2024. *Xataka*. Obtenido de <https://www.xataka.com/movilidad/dgt-anticipa-llegada-coches-completamente-autonomos-nuevo-reglamento-esta-marcha-para-2024>.
- [2] Moncalvillo González, A. (2024). Seguimiento de carril por visión y conducción autónoma con Aprendizaje por Imitación. *Obtenido de https://burjcdigital.urjc.es/handle/10115/38557*
- [3] Terrones Rodríguez, A. L. (2021). Una aproximación general al desarrollo de los coches autónomos. *CTS: Revista iberoamericana de ciencia, tecnología y sociedad*, 16(47), 153-175. *Obtenido de https://dialnet.unirioja.es/servlet/articulo?codigo=8022371*
- [4] Alabyad, N., Hany, Z., Mostafa, A., Eldaby, R., Tagen, I. A., & Mehanna, A. (2024, Marzo). From Vision to Precision: The Dynamic Transformation of Object Detection in Autonomous Systems. *2024 6th International Conference on Computing and Informatics (ICCI)* (pp. 332-344). IEEE.
- [5] Agencia de Proyectos de Investigación Avanzados de Defensa. (2024). Wikipedia. *Obtenido de https://es.wikipedia.org/wiki/Agencia_de_Proyectos_de_Investigaci%C3%B3n_Avanzados_de_Defensa*
- [6] Anitua Galdón, M. G. (2019). Las estrategias de las empresas automovilísticas con el coche autónomo y los nuevos jugadores. *Obtenido de https://repositorio.comillas.edu/xmlui/bitstream/handle/11531/27730/TFG-%20Anitua%20GaldoIn%2c%20MariIa%20Gabriela.pdf?sequence=1&isAllowed=y*
- [7] Trovão, J. P. (2024). Advancing Automotive Technologies [Automotive Electronics]. *IEEE Vehicular Technology Magazine*, 19(1), 106-C3. *Obtenido de https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10474561*

- [8] Silva Lozada, B. R. (2024). Diseño e implementación de un algoritmo para la segmentación del carril en carretera que permita el manejo autónomo de un robot móvil usando visión por computador. *Obtenido de* <https://repository.unab.edu.co/bitstream/handle/20.500.12749/26048/Tesis.pdf?sequence=4&isAllowed=y>
- [9] García, E. (2012). Visión artificial. *FUOC Fundación para la Universitat Oberta de Catalunya. Obtenido de* <https://ftp.isdi.co.cu/Biblioteca/BIBLIOTECA%20UNIVERSITARIA%20DEL%20ISDI/COLECCION%20DE%20LIBROS%20ELECTRONICOS/LE-1069/LE-1069.pdf>
- [10] Tello, Jesús. (2006). La visión artificial y las operaciones morfológicas en imágenes binarias. *Obtenido de* https://www.researchgate.net/profile/Jesus-Tello/publication/236656570_La_vision_artificial_y_las_operaciones_morfologicas_en_imagenes_binarias/links/00b7d518b7d18bf0c6000000/La-vision-artificial-y-las-operaciones-morfologicas-en-imagenes-binarias.pdf
- [11] García, I. & Caranqui, V. (2015, enero – diciembre). La visión artificial y los campos de aplicación. *Tierra Infinita*(1), 98-108. *Obtenido de* <https://revistasdigitales.upec.edu.ec/index.php/tierrainfinita/article/view/76/2992>
- [12] Vila, J., Hernández, M., Feliciano, S., & Hernández, J. L. (s.f.). Detección de objetos basados en HOG/SVM en una imagen. *Obtenido de* [https://www.innovaingenieria.uagro.mx/innova/index.php/innova/article/download/93/44/#:~:text=HOG%20\(Histograma%20De%20Gradientes%20orientados\)&text=descriptor%20de%20caracter%C3%ADsticas%20es%20generalizar,lo%20vea%20bajo%20diferentes%20condiciones](https://www.innovaingenieria.uagro.mx/innova/index.php/innova/article/download/93/44/#:~:text=HOG%20(Histograma%20De%20Gradientes%20orientados)&text=descriptor%20de%20caracter%C3%ADsticas%20es%20generalizar,lo%20vea%20bajo%20diferentes%20condiciones)
- [13] Toya Sherdek, C. (2016, febrero 26). LBP y ULBP – Local Binary Patterns y Uniform Local Binary Patterns. César Troya Sherdek. *Obtenido de* <https://cesartroyasherdek.wordpress.com/2016/02/26/deteccion-de-objetos-vi/>
- [14] Boujema, K. S., Akallouch, M., Berrada, I., Fardousse, K., & Bouhouste, A. (2021). ATTICA: a dataset for arabic text-based traffic panels detection. *IEEE Access*, 9, 93937-93947. *Obtenido de* <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9466101>
- [15] Latina, M.A., Van Russel R. Dela Cruz, J., & Delos Santos, F.D. (2022, diciembre). Empty Glass Bottle Defect Detection Based on Deep Learning with CNN Using

- SSD MobileNetV2 Model. 2022 *IEEE 14th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment, and Management (HNICEM)*, 1-6.
- [16] Building occupancy management solution using the TensorFlow Object Detection API. (2021). *GreenWaves Technologies. Obtenido de* <https://greenwaves-technologies.com/wp-content/uploads/2021/05/Occupancy-mngnt-with-TensorFlow-Object-Detection-API.pdf>



ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenibles	Alto	Medio	Bajo	No Procede
ODS 1. Fin de la pobreza.				
ODS 2. Hambre cero.				
ODS 3. Salud y bienestar.				
ODS 4. Educación de calidad.				
ODS 5. Igualdad de género.				
ODS 6. Agua limpia y saneamiento.				
ODS 7. Energía asequible y no contaminante.				
ODS 8. Trabajo decente y crecimiento económico.				
ODS 9. Industria, innovación e infraestructuras.				
ODS 10. Reducción de las desigualdades.				
ODS 11. Ciudades y comunidades sostenibles.				
ODS 12. Producción y consumo responsables.				
ODS 13. Acción por el clima.				
ODS 14. Vida submarina.				
ODS 15. Vida de ecosistemas terrestres.				
ODS 16. Paz, justicia e instituciones sólidas.				
ODS 17. Alianzas para lograr objetivos.				



Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

Este Trabajo Fin de Grado se relaciona con gran parte de los Objetivos de Desarrollo Sostenible. El desarrollo de un sistema de visión artificial que permite detectar señales de tráfico, es un proyecto que contribuye a la tecnología de conducción autónoma y, por tanto, a la expansión de vehículos autónomos. Este tipo de vehículos aportan a la sociedad una gran cantidad de beneficios y avances alineados con los Objetivos de Desarrollo Sostenible planteados por la Organización de las Naciones Unidas. A continuación, se explican detalladamente las diferentes relaciones con los ODS del presente trabajo.

Primeramente, la automatización de los vehículos de empresas distribuidoras supondría una reducción del costo de transporte por carretera. Esto, desencadenaría en una mejora en la vida de aquellas personas que necesitan ser administradas con productos de primera necesidad. Dado que, el abaratamiento del transporte provocaría también la bajada de precio de alimentos y productos básicos como los medicamentos, se puede decir que este proyecto se alinea con los ODS **fin de la pobreza** (1), **hambre 0** (2) y **reducción de las desigualdades** (10).

Por otro lado, como bien se ha mencionado en el capítulo 1 de introducción, los coches autónomos reducen notablemente la tasa de mortalidad por accidentes de tráfico, ya que, la mayoría de estos accidentes se producen por fallos humanos. Por tanto, los sistemas de conducción autónoma contribuyen notablemente al objetivo **salud y bien estar** (3).

En lo que respecta al objetivo de **igualdad de género** (5), un transporte público totalmente autónomo, como puede ser la flota de taxis de Waymo, ayudaría positivamente a la reducción del número de casos de mujeres que sufren acoso de cualquier índole al subirse al coche de un desconocido. Los coches autónomos, además de no requerir de un conductor, también ofrecen una vigilancia, haciendo más seguro el trayecto de vuelta a casa.

El objetivo que más relación tiene con este proyecto es el de **industria, innovación e infraestructura** (9), ya que se trabaja con tecnologías punteras, como la inteligencia artificial y la robótica, en constante investigación y que, a menudo, desencadenan en innovaciones. Además, se incita a otros desarrolladores a contribuir en la investigación y crecimiento de la tecnología. Por otro lado, la conducción autónoma también tiene una fuerte relación con la industria y la infraestructura, debido a que, sin el desarrollo y adaptación de ambas, no se puede globalizar esta tecnología.

Por último, la correcta implantación y expansión de estos vehículos autónomos, en concreto del transporte público, como las redes de taxis sin conductor o autobuses



autónomos, podría contribuir a la bajada de precio del transporte y desencadenar en una menor adquisición de vehículos personales, por lo que también existe una relación con los ODS **energía asequible y no contaminante** (7), **ciudades y comunidades sostenibles** (11) y **producción y consumo responsables** (12).