UNIVERSIDADE DA CORUÑA

Escuela Politécnica Superior

Trabajo Fin de Grado

CURSO 2020/2021

DESARROLLO DE MÓDULOS DE VISIÓN POR COMPUTADOR EN PYTHON PARA LA DETECCIÓN DE OBJETOS EN UN ENTORNO DE PRUEBAS DE CONDUCCIÓN AUTÓNOMA

Grado en Ingeniería en Tecnologías Industriales

ALUMNO Miguel Cruz Irimia

TUTORES Francisco Javier Bellas Bouza

Alejandro Paz López

FECHA Julio 2021

1. Título y Resumen

Título

Desarrollo de módulos de visión por computador en Python para la detección de objetos en un entorno de pruebas de conducción autónoma.

Resumen

Este trabajo de fin de grado se enmarca en la línea de investigación de conducción autónoma que se lleva a cabo en el Grupo Integrado de Ingeniería (GII) de la UDC. En esta línea se utiliza el robot educativo Robobo para realizar estudios en aspectos de la conducción autónoma en un entorno de pruebas real y simulado. En este TFG se desarrollaron módulos de visión por computador en Python que resuelven problemas habituales en este entorno, como la detección de señales en tiempo real. Para ello, se identificaron las principales técnicas existentes, basadas en procesado de imagen tradicional y en redes de neuronas artificiales, para luego ser analizadas y comparadas en el entorno de pruebas con el objetivo de seleccionar la más eficiente y fiable. Para este trabajo experimental, se hizo uso de librerías específicas de visión y aprendizaje, como Keras y OpenCV, que fueron implementadas en el robot Robobo para su aplicación en el entorno real.

Título

Desenvolvemento de módulos de visión por computador en Python para a detección de obxectos nun entorno de probas de condución autónoma.

Resumo

Este proxecto de fin de grao forma parte da liña de investigación de condución autónoma realizada no Grupo de Enxeñaría Integrada (GII) da UDC. Nesta liña, o robot educativo Robobo úsase para realizar estudos en aspectos da condución autónoma nu contorno de probas real e simulado. Neste TFG desenvolvéronse módulos de visión por computador en Python para resolver problemas comúns neste contorno, como a detección de sinais en tempo real. Para iso, identificáronse as principais técnicas existentes, baseadas no procesamento de imaxes tradicionais e redes de neuronas artificiais, e logo foron analizadas e comparadas no entorno de probas para seleccionar a máis eficiente e fiable. Para este traballo experimental, o alumno fixo uso de bibliotecas específicas de visión e aprendizaxe, como Keras e OpenCV, que foron probadas no robot Robobo para a súa aplicación nun contorno real.

Title

Development of computer vision modules in Python for object detection in an autonomous driving testing environment.

Summary

This final degree project is part of the autonomous driving research line carried out in the Integrated Engineering Group (GII) of the UDC. In this line, the educational robot Robobo is used to carry out studies in different aspects of autonomous driving using a real and simulated environment. In this TFG, computer vision modules in Python were developed to solve common problems in this environment, such as detection real-time signal. For this, the main existing techniques were identified, based on traditional image processing or artificial neuron networks, which were analysed and compared in the test environment to select the most efficient and reliable ones. For this experimental work, the student made use of specific vision and learning libraries, such as Keras and OpenCV, which were tested on the Robobo robot for later application in the real environment.

UNIVERSIDADE DA CORUÑA

Escola Politécnica Superior

TRABAJO FIN DE GRADO

CURSO 2020/2021

DESARROLLO DE MÓDULOS DE VISIÓN POR COMPUTADOR EN PYTHON PARA LA DETECCIÓN DE OBJETOS EN UN ENTORNO DE PRUEBAS DE CONDUCCIÓN AUTÓNOMA

Grado en Ingeniería en Tecnologías Industriales

Documento

MEMORIA

Índice de contenido

Índice de figuras

Índice de tablas

2. Objetivos

Para una correcta compresión de este trabajo de fin de grado y de los temas que iremos tratando, empezaremos conociendo el objetivo principal y los sub-objetivos del mismo.

El objetivo principal de este proyecto es el desarrollo de módulos de visión por computador en lenguaje Python para la detección de objetos en un entorno de pruebas de conducción autónoma.

La consecución de este objetivo implica la realización de los siguientes subobjetivos:

1. Analizar el estado actual del entorno de pruebas de conducción autónoma que posee el Grupo Integrado de Ingeniería de la UDC, y establecer qué objetos se deberán detectar.

2. Analizar el estado del arte en reconocimiento de objetos mediante visión por computador, y seleccionar las técnicas a implementar en Python.

3. Conseguir un conjunto de imágenes reales (benchmark) adecuado para comparar las técnicas de detección de objetos.

4. Realizar pruebas de comparación de las técnicas en el conjunto de imágenes reales.

5. Conocer el robot móvil Robobo a nivel de características hardware, software y programación, así como el entorno de conducción autónoma que usaremos para la validación.

6. Desarrollar una metodología de toma de imágenes con el robot en el entorno.

7. Validar las técnicas de visión en las imágenes finales, primero en el computador, y finalmente en la operación en tiempo real del robot.

8. Caracterizar el algoritmo de detección de objetos en tiempo real en el robot y documentar las conclusiones del análisis técnico.

3. Estado del arte

Este Trabajo Fin de Grado (TFG) se enmarca en un proyecto de investigación del Grupo Integrado de Ingeniería (GII) de la Universidade da Coruña (UDC) con el que se pretende avanzar en la investigación de la conducción autónoma mediante la incorporación de algoritmos de reconocimiento de objetos, toma de decisiones, comportamientos colaborativos, etc. en una plataforma robótica móvil, conocida como Robobo.

Por tanto, en el marco de este TFG, nos centraremos en el área de la robótica educativa y de investigación, concretamente en los robots móviles inteligentes. En este ámbito, se desarrollarán librerías de visión por computador en Python, para la detección de objetos en tiempo real en un entorno de pruebas de conducción autónoma. En concreto, este TFG se centrará en la detección de señales de tráfico, una propiedad básica para crear comportamientos autónomos en este tipo de aplicaciones. En los siguientes sub-apartados nos centraremos en hacer una revisión de los principales campos afectados.

3.1 Robótica educativa

La creciente importancia que tiene la tecnología en el mundo hoy en día y su continuo desarrollo, hace que esta se convierta en parte integral del proceso de formación en la niñez y la juventud. Por esta razón es importante desarrollar propuestas en las que se ofrezca a niños y jóvenes la posibilidad de entrar en contacto con las nuevas tecnologías de manera formal y reglada [1].

La robótica educativa tiene su origen en los años 90, cuando comenzaron a crearse ciertos dispositivos de forma local con fines educativos para ser usados en talleres para alumnado de educación primaria. Sin embargo, no es hasta el año 2000 cuando la robótica educativa surge como herramienta de formación gracias al MIT (Massachusetts Institute of Technology), donde se estableció un convenio con la compañía LEGO para que se construyeran piezas que permitieran la integración de elementos de programación para que así los niños pudieran construir y programar dispositivos tecnológicos capaces de realizar ciertas acciones. De esta manera, se integraron las piezas de LEGO con el lenguaje de programación Logo, uno de los primeros lenguajes en el ámbito educativo. A continuación, haremos un recorrido de como se han ido adaptando distintos robots a esta disciplina.

Como era de esperar, una de las opciones más populares es el *Lego Mindstorm EV3* [2], versión moderna del comentado anteriormente, que se basa también en piezas de Lego (ver Figura 1). La pieza fundamental del set es el ladrillo inteligente EV3, cuya última versión cuenta con un procesador ARM9, un puerto USB para proporcionar WI-FI, lector de tarjetas Micro SD, altavoz, y una pantalla para mostrar información, aparte de numerosos puertos para la conexión de sensores y motores. A pesar de que para niveles superiores de educación es posible actualizar el firmware y programar el robot con Python, no soporta algoritmia compleja como reconocimiento de objetos o habla ya que no cuenta con la capacidad de cómputo necesaria, tampoco cuenta con sensores como la cámaras o láser.

Figura 1. Lego Mindstorm EV3

En la línea del anterior robot, encontramos otros como *Thymio* [3] (fundamentado en Aseba) y *Mbot* [4] (basado en Scratch 2.0 o en Arduino) que se han utilizado desde hace un tiempo tanto en la educación secundaria como en la universitaria (ver Figura 2 y 3). Estos comparten una falta de sensorización avanzada y una escasa o inexistente potencia de procesamiento que nos hará enfocarnos en robots más avanzados como *TurtleBot 3* o el robot *e-puck*.

Figura 2. Thymio

Figura 3. Mbot

Estos dos robots mencionados, sí cumplen las condiciones necesarias de un entorno de programación adaptado a distintos niveles educativos con una sensorización amplia y potente, además de controladores con potencia de cómputo suficiente. TurtleBot [5] es un kit robótico con software de código abierto y programable en ROS, además, cuenta con un sensor de distancia de 360 grados (LIDAR) y utiliza la tecnología SLAM (localización y mapeo simultáneos) para construir un mapa y conducir al robot por entornos de interior. Tiene ciertas limitaciones, como no poder usarse fuera de la enseñanza universitaria debido a la complejidad y conocimientos técnicos que requiere, y posee un pobre desempeño en tareas como reconocimiento de voz o interacción humano-robot al no tener sensores específicos para estas tareas, como se observa en la Figura 5. En el caso del e-puck (Figura 4), programable en distintos lenguajes como Python, C++ [6], encontramos que su procesador no es capaz de correr algoritmos potentes usados en la robótica del mundo real. Sus precios oscilan entre 500-600 €.

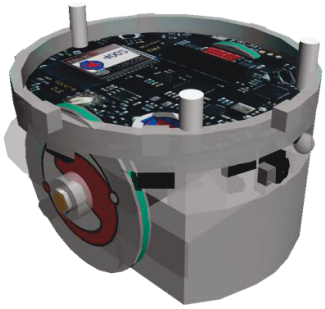

Figura 4. E-puck

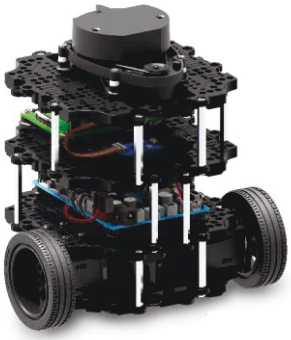

Figura 5. TurtleBot

A continuación, nos centraremos en dos robots de más alta gama que ya tienen capacidad para ejecutar algoritmos potentes, aunque con precios muy altos. El primero es el *Khepera IV* (ver Figura 6) un robot móvil especialmente compacto y sensorizado para cualquier actividad de laboratorio, aunque no adecuado para secundaria [7]. Este cuenta con herramientas para realizar pruebas de navegación, inteligencia artificial, sistemas multi-agentes o programación en tiempo real, y con un sistema operativo Linux completo, lo que facilita el desarrollo de aplicaciones con C/C++ o importar librerías.

Por otro lado, se encuentra el robot humanoide *NAO* [8]. Este modelo creado en 2008 mide 58 centímetros (ver Figura 7), es totalmente programable y es capaz de interactuar con todo tipo de público. Tiene aplicaciones útiles en educación (tanto para estudiantes como para profesores e investigadores), uso doméstico y empresas, ya que *NAO* es capaz de percibir el entorno a partir de sus múltiples sensores. Además de poseer suficiente poder de procesamiento para ejecutar los algoritmos más actuales, tiene un software llamado *Choreographe y* compatible con Windows, MAC y Linux, que permite programarlo sin tener conocimientos de un lenguaje de programación. También se puede programar en C++, Python, Java, Matlab y ROS para usuarios avanzados. Este modelo no es adecuado para estudios de conducción autónoma, y su elevado precio (alrededor de 6000€) hace que sea demasiado caro para ser usado en la mayoría de los centros educativos.

.

Figura 6.Khepera IV

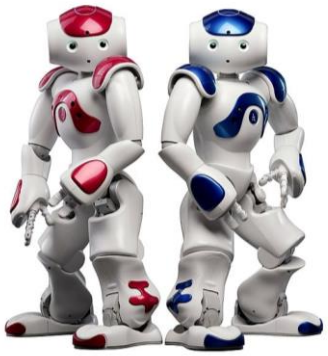

Figura 7. NAO

Una vez analizados los principales robots educativos, podemos decir que todos tienen ciertas limitaciones en las aulas que no permiten un acercamiento real de los alumnos a los usos y proyectos que se realizan en el mundo actual en la robótica autónoma. Entendiendo que nos encontramos en una época donde el procesamiento y análisis de datos a gran escala es básico, el uso de hardware para la ejecución de potentes algoritmos de aprendizaje automático, reconocimiento de habla o visión por computador cada vez están más presentes, por lo que existe la necesidad de la de la creación de un robot educativo con una serie de características que permitan al estudiantado aprender con robots capaces de manejar dichos algoritmos y con una sensorización adecuada. Para conseguir un robot con estas características, y además de un precio asequible, surge la idea de combinar una plataforma móvil con un smartphone, consiguiendo de esta manera una alta sensorización y el aprovechamiento de las capacidades y el poder de computacional ya existentes en un teléfono. Esto permite la existencia de varios niveles de programación, los cuales están organizados en bibliotecas que cubren diferentes áreas de las capacidades del robot, como: visión, sonido, control remoto, reconocimiento de voz, etc.

Con esta idea en mente, surge una nueva serie de robots basados en smartphone como *Romo*, *Fabe Explorer* y *WHEELPHONE* (Figuras 8, 9 y 10).

Figura 8. Romo

Figura 9. Fabe Explorer

Figura 10. WHEELPHONE

Romo [9] es una de las primeras plataformas móviles con smartphone, compatible con IOS y enfocada a la educación primaria, pero no apta para labores de nivel universitario o educación. Sus mayores inconvenientes fueron una ausencia total de sensores y que su aplicación dejó de actualizarse 2013, encontrándose actualmente desfasada y sin soporte oficial. En el caso de *Fable* [10] nos encontramos ante un sistema de construcción modular en el que los estudiantes pueden crear su propio robot en solo unos minutos. Este puede ser programado usando Blocky, una programación similar a Scratch visual y sencilla, además de la posibilidad de programación en Python para niveles más avanzados. Este robot puede ser adquirido en kit o por piezas en separado, pudiéndose ampliar con piezas de LEGO o impresión 3D para convertirlo en formas más interactivas como un robot más sociable. Sus principales inconvenientes son que no utiliza el procesador del Smartphone para analizar las imágenes y que su precio es elevado. En cuanto a WHEELPHONE [11], nos encontramos con un enfoque centrado en un uso universitario y de investigación, muy limitado para todas las etapas educativas.

Ante estas circunstancias surge una idea en el Grupo Integrado de Ingeniería de la UDC, de desarrollar una plataforma robótica móvil basada en Smartphone, pero que sea utilizable desde secundaria hasta la universidad para la enseñanza de la robótica inteligente. De esta forma nació, en el año 2016, el proyecto Robobo [12], en el cual se enmarca el presente TFG.

3.2 Robobo

En este subapartado se presenta el robot Robobo [13], un robot creado por MINT SL (spin-off de la UDC) y que está en continuo desarrollo, en el que varios alumnos han contribuido en su mejora a través de diferentes trabajos de fin de grado y de máster como este, y con el que realizaremos las pruebas en este proyecto (ver Figura 11).

Figura 11. Robobo

2.2.1 Hardware de Robobo

El Hardware del robot está compuesto por una base móvil y un Smartphone que se acoplará en ella, y se conecta por Bluetooth.

Toda la electrónica de la base Robobo [14] ha sido diseñada encima de una placa de circuito impreso (PCB) con la forma específica del robot, para acomodar encima todos los componentes electrónicos, sensores y actuadores, como veremos a continuación. Sus

principales componentes electrónicos son un micro-controlador basado en un *PIC32 microcontroller*. La base cuenta con dos ruedas, una unidad Pan-Tilt a la que se acopla el smartphone, cinco leds frontales y dos traseros, cinco sensores infrarrojos frontales y tres traseros, un conector micro USB tipo B y un botón de encendido. Dichos componentes aparecen indicados en la Figura 12.

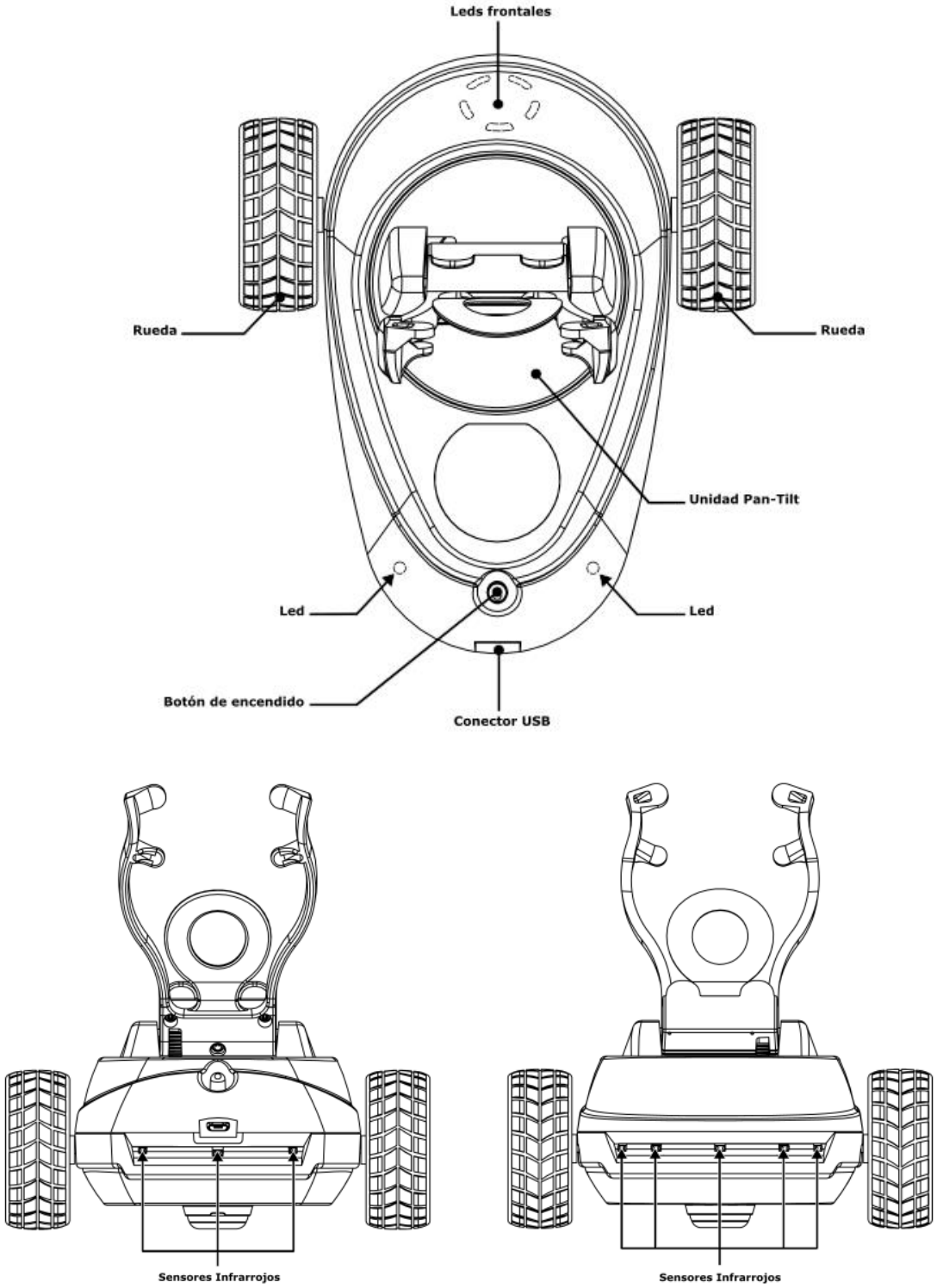

Figura 2. Base y sensores de Robobo

En cuanto al smartphone, a pesar de que los sensores, actuadores y procesadores dependen del modelo, podemos decir que aporta el siguiente hardware al robot:

- Dos cámaras de alta resolución.
- Una pantalla táctil LCD.
- Micrófono y altavoz.
- Giroscopio 3D, acelerómetro 3D y magnetómetro 3D.
- Sensores de luz, de proximidad y de temperatura.
- GPS.
- Conectividad WI-FI.
- Bluetooth.
- 3G/4G.

3.2.2 Software de Robobo

El framework de Robobo hace que sea programable desde un ordenador o Tablet, siempre que estén conectados a la misma red WI-FI, con Windows, MAC o Linux. El *framework* está organizado en tres jerarquías [15] dependiendo el nivel del programador: Scratch para principiantes, librerías en Python y JavaScript para usuarios intermedios, y librerías JAVA Android o ROS para usuarios avanzados. Todas las librerías necesarias están publicadas con licencias de código abierto. Así mismo, también cuenta con diferentes simuladores para permitir que los alumnos lo puedan programar sin depender totalmente del robot real. Estos también están orientados dependiendo de los diferentes niveles educativos y son: Unity, CoppeliaSim (ejemplo en la Figura 13) y Gazebo, como vemos en la Figura 14.

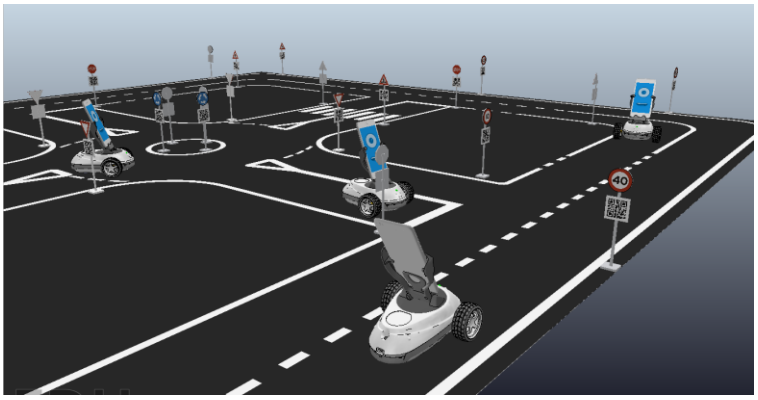

Figura 13. Simulación en CoppeliaSim

Figura 14. Niveles de programación Robobo

Podemos resumir las principales acciones que permite realizar:

- **Funciones de movimiento:** a través de métodos el robot se desplaza moviendo los motores alojados en la base, pudiendo rotar la orientación del Smartphone con los motores de la unidad PAN-TILT.

- **Funciones de detección:** son aquellas en las que el robot extrae datos de su entorno, como los IR, o todos los métodos basados en el uso de la cámara.

- **Funciones de interacción con el usuario:** estos métodos permiten aplicaciones de interacción usuario-robot, como expresión de emociones o reconocimiento de sonidos, la interacción táctil, síntesis de voz…

  Las librerías que se van a crear en este proyecto formarían parte de las funciones de detección, debido al desarrollo de algoritmos de visión artificial, que se podrán ejecutar en tiempo real en el propio procesador del Smartphone.

3.3 Conducción autónoma

Según un informe técnico reciente de la Carretera Nacional Autoridad de Seguridad Vial (NHTSA), el 94% de la carretera los accidentes son causados por errores humanos [16]. Sobre este trasfondo se están desarrollando sistemas de conducción automatizada con la promesa de prevenir y reducir accidentes, emisiones, transporte de personas con movilidad reducida y reducción de estrés relacionado con la conducción [17]. Se proyecta que los beneficios sociales anuales de los vehículos de conducción autónoma pueden ser alrededor de $800 billones para 2050 [18] gracias a la reducción de la congestión del tráfico, reducción de accidentes de tráfico, reducción del consumo energético y aumento de la productividad mediante la redistribución de la conducción.

Si empezamos por el origen de la idea del vehículo autónomo, también conocido como robótico, o informalmente como sin conductor o auto conducido, y su puesta en práctica, es mucho más vieja de lo que pueda parecer, en concreto, de principios del siglo XX [19]. Se considera que Francis Houdina, un ingeniero eléctrico de Nueva York en el daño 1925, fue el primero en poner en práctica el concepto de vehículo autónomo. Este era controlado a distancia por un control remoto que podía encender el vehículo, ponerlo en marcha, circular esquivando a otros vehículos y hacer sonar el claxon. Para ello creó su propia empresa y enseñó su primer prototipo al público en Manhattan. Este vehículo podría a ver seguido circulando si no fuera porque, finalmente, chocó con otro automóvil. Otro nombre muy destacado es el de Norman Bel Geddes, un diseñador industrial estadounidense que, con diseños de corte futurista, consiguió una gran fama. Destaca un trabajo que hizo para General Motors en Nueva York en 1939 de modelo futurista sobre un vehículo autónomo que funcionaba mediante electricidad controlado por radiocontrol y que se movía por un circuito eléctrico integrado en el pavimiento.

Estos hechos suponen el origen de una idea, pero distan mucho de la realidad actual que representa la conducción autónoma. El proyecto Eureka PROMETHEUS [20] se llevó a cabo en Europa entre 1987 y 1995 y fue uno de los primeros grandes estudios de conducción automatizada. El proyecto condujo al desarrollo de VITA II de Daimler-Benz, que tuvo éxito en conducir automáticamente por autopistas [21].

Figura 15. DARPA Grand Challenge

Más adelante, DARPA Grand Challenge, organizado por el Departamento de Defensa de EE. UU. en 2004, fue la primera gran competencia de conducción automatizada, y en ella ningún participante completó la ruta todoterreno de 150 millas. La mayor dificultad del desafío era la prohibición de cualquier intervención humana. Podemos ver algunos de los vehículos que participaron en la Figura 15. Más adelante, otro Gran Desafío DARPA similar tuvo lugar en 2005 donde cinco equipos lograron completar la ruta todoterreno [22], sin embargo, la conducción completamente automática en escenas urbanas se consideraba el mayor desafío del campo desde los primeros intentos.

Durante el Desafío Urbano de DARPA [23], en 2007, varios grupos de investigación de todo el mundo participaron con sus coches autónomos en un entorno de prueba inspirado en una ciudad. Seis equipos lograron completar el desafío y esto atrajo mucha atención y, aunque fue el evento más grande e importante hasta este momento, al entorno de prueba le faltaban ciertos aspectos para ser una verdadera escena de conducción urbana como peatones y ciclistas.

El aumento del interés público y del mercado potencial precipitó la aparición de vehículos autónomos con diferentes grados de automatización y captaron a los principales fabricantes de automóviles y compañías tecnológicas como Ford, Volkswagen, Tesla, Uber… entre los que podemos destacar Waymo [24], una empresa de Google desarrolladora de vehículos autónomos con servicio de taxi (ver Figura 16) y con millones de kilómetros recorridos sin conductor en carreteras públicas. Estos son los diferentes grados de automatización [25]:

- Nivel 0: Sin automatización en la conducción
- Nivel 1: Asistencia en la conducción
- Nivel 2: Automatización parcial
- Nivel 3: Automatización condicionada
- Nivel 4: Automatización elevada
- Nivel 5: Automatización completa

Figura 16. Coche autónomo Waymo

A pesar de todo, la conducción automatizada en los entornos urbanos aún no se ha logrado de forma total [26]. Accidentes causados por sistemas inmaduros [27] socavan la confianza en este sector y, además, cuesta vidas. El uso extendido de los vehículos

autónomas aún no es inminente [28], pero podemos hablar de su potencial impacto y de sus principales beneficios:

- Problemas solucionables: reducir emisiones contaminantes, congestiones en el tráfico y accidentes.

- Nuevas oportunidades: reajuste del tiempo de conducción, transporte de personas con movilidad reducida.

- Nuevas tendencias: considerar la movilidad como un servicio (MaaS), revolución logística.

Su evolución e implementación pueden reducir problemas causados por un comportamiento humano erróneo como la distracción, conducción bajo influencia de sustancias peligros o excesos de velocidad [29]. Además, considerando que el grupo de personas mayores (más de 60 años) está creciendo más rápido que los grupos más jóvenes [30], puede tener un gran impacto en la calidad de vida y productividad de una gran parte de la población. Por otro lado, los viajes compartidos tienen costes más bajos en comparación con la propiedad de vehículos con menos de 1000 km de kilometraje anual [31]. Se espera que la relación entre vehículos propios y compartidos sea de 50:50 en 2030 [32]. A continuación, trataremos principales áreas de estudios necesarias para crear vehículos de conducción autónoma.

i. Arquitectura del sistema:

A lo largo de los años se han establecido prácticas comunes en la arquitectura del sistema donde la mayoría de los vehículos autónomos dividen las tareas de conducción automatizada en subcategorías y emplean una matriz de sensores y algoritmos en varios módulos. Más recientemente, la conducción de extremo a extremo comenzó a surgir como una alternativa a enfoques modulares, volviéndose los modelos de aprendizaje profundo dominantes en muchas de estas tareas [33].

ii. Sensores y hardware:

Las unidades de hardware pueden categorizados en cinco: sensores exteroceptivos para la percepción, sensores propioceptivos para las tareas del control interno del estado del vehículo, matrices de comunicación, actuadores y unidades computacionales. Los sensores exteroceptivos se utilizan principalmente para percibir el entorno, que incluye objetos dinámicos y estáticos como zonas transitables, edificios, pasos de peatones, etc. Cámaras, LIDAR, sensores de radar y ultrasónicos son los más utilizados para esta tarea. Podemos ver una comparación detallada de estos sensores en la Tabla 1.

Modality	Affected by Illumination	Affected by weather	Color	Depth	Range	Accuracy	Size	Cost
Lidar	-	✓	-	✓	medium (< 200m)	high	large*	high*
Radar	-	-	-	✓	high	medium	small	medium
Ultrasonic	-	-	-	✓	short	low	small	low
Camera	✓	✓	✓	-	-	-	smallest	lowest
Stereo Camera	✓	✓	✓	✓	medium (< 100m)	low	medium	low
Flash Camera [77]	✓	✓	✓	✓	medium (< 100m)	low	medium	low
Event Camera [78]	limited	✓	-	-	-	-	smallest	low
Thermal Camera [79], [80]	-	✓	-	-	-	-	smallest	low

Tabla 1. Sensores exteroceptivos

iii. Localización y mapeo:

La localización es la tarea de encontrar la posición del yo en relación con un marco de referencia en un entorno [34], y es fundamental para cualquier robot móvil de conducción

autónoma. Los principales sensores utilizados para ello son: una fusión del Sistema de Posicionamiento Global y la Unidad de Medición Inercial (GPS-IMU) y el mapeo y localización simultáneos (SLAM), que consiste en una técnica usada por robots y vehículos autónomos para construir un mapa de un entorno desconocido en el que se encuentra, a la vez que estima su trayectoria al desplazarse dentro de este entorno [35].

iv. Percepción:

Percibir el entorno circundante y extraer información que puede ser fundamental para una navegación segura es un objetivo fundamental para la conducción autónoma. Entre la categoría de percepción destacamos por un lado la la detección de objetos basada en imágenes, la segmentación semántica (consiste en clasificar cada píxel de una imagen en una clase para detectar, carreteras, líneas de conducción ...) y la detección de objetos3D. Por otro lado, tienen mucha importancia el seguimiento de objetos y detección de carreteras y sus líneas. En la Figura 17 se representa una escena del seguimiento de las trayectorias de varios peatones en una intersección.

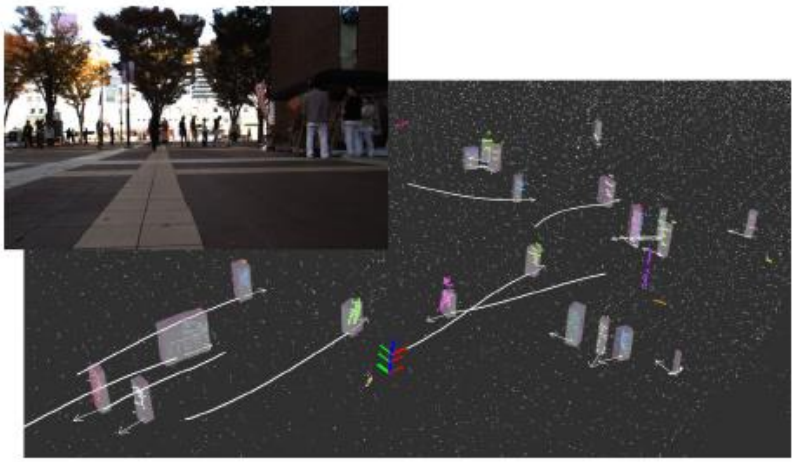

Figura 17. Seguimiento de objetos

v. Evaluación:

Un vehículo autónomo debe evaluar constantemente el riesgo general, el nivel de la situación y predecir las intenciones de los conductores humanos y peatones que hay a su alrededor. Una falta de evaluación aguda de un mecanismo puede provocar accidentes. Este campo se podría dividir en tres categorías: la evaluación de riesgo e incertidumbre, la evaluación del comportamiento de conducción humano (la intención del conductor humano circundante es más relevante para la predicción y la decisión a medio y largo plazo) y el reconocimiento del estilo de conducción. En la Figura 18 se muestra una evaluación del nivel de riesgo general en dos escenas de conducción diferentes.

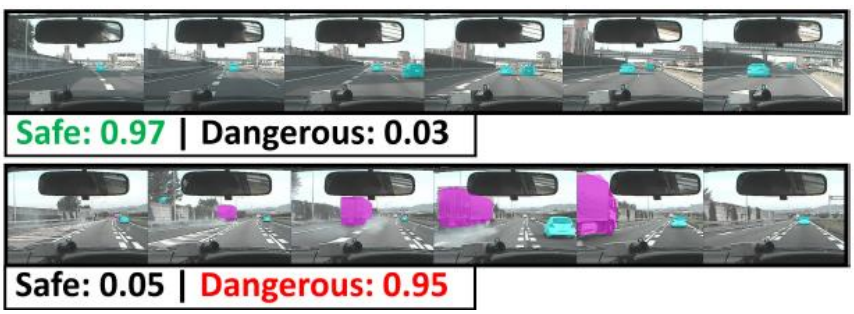

Figura 18. Evaluación de riesgo

vi. Planificación y toma de decisiones:

La planificación podemos separarla en dos tareas, la planificación global de ruta y la local. La global, es responsable de encontrar la ruta en la red de carreteras desde el origen hasta el destino final. Es un tema bien estudiado, de alto rendimiento y estándar en la industria durante más de una década, donde destaca el GPS.

La planificación local, consiste en la ejecución del plan global sin errores, es decir, debe encontrar trayectorias buscando evitar obstáculo y satisfacer ciertos criterios de optimización entre un punto de partida y un punto final.

vii. Interacción humano-máquina:

La comunicación y la interacción del vehículo con su conductor o pasajero dependerá del nivel de automatización.

Como hemos visto, las investigaciones en este campo todavía son muy importantes y por ello el GII se ha metido de lleno en él. Actualmente se está trabajando en un nuevo proyecto, The Robobo SmartCity, en el que se pretende estudiar e investigar distintos aspectos de la conducción autónoma, como comportamientos colaborativos entre diferentes robots autónomos.

3.4 Visión Artificial

La visión artificial [36] es una rama de la inteligencia artificial que tiene por objetivo modelar matemáticamente los procesos de percepción visual en los seres vivos y generar programas que permitan simular estas capacidades visuales por computadora. La visión artificial permite la asimilación de un mundo dinámico en tres dimensiones a partir de imágenes en dos dimensiones del mundo. Las imágenes pueden ser en blanco y negro o a color, pueden ser percibidas por una o varias cámaras, estáticas o móviles. Lo que la visión quiere deducir no solo se limita a la parte geométrica, sino que también busca determinar algunas propiedades de los materiales como pueden ser su color y textura. La visión artificial aplicada a la industrial abarca la informática, la óptica, la ingeniería mecánica y la automatización industrial.

En la Figura 19 se muestra el proceso que se sigue para extraer datos de una imagen, siguiendo una serie de pasos que incluye un pre-procesamiento de la imagen de entrada, extracción de características, reconocimiento, clasificación del tipo de objeto e interpretación de la escena.

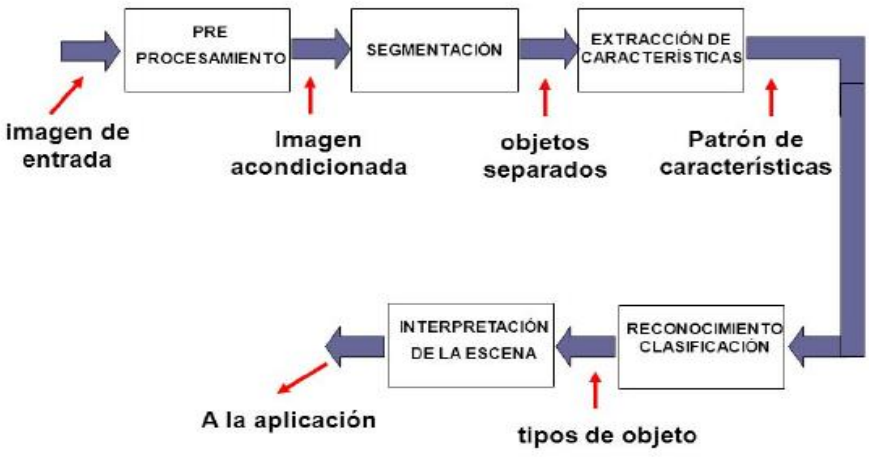

Figura 19. Etapas típicas de un sistema de visión artificial

3.4.1 OpenCV

OpenCV [37] es una de las librerías que utilizaremos, una biblioteca de código libre de visión artificial originalmente desarrollada por Intel. Desde que apareció su primera versión alfa en el mes de enero de 1999, se ha utilizado en una gran cantidad de aplicaciones, y hasta 2020 se la sigue mencionando como la biblioteca más popular de visión artificial [38]. Entre algunas aplicaciones de esta destacamos detección de movimiento, reconocimiento de objetos, reconstrucción 3D a partir de imágenes… Muchas de sus funcionalidades a la hora de representar imágenes son usadas para la elaboración de ilustraciones en este trabajo.

La biblioteca tiene más de 2500 algoritmos, al alcance de unas pocas líneas de código, y con la versatilidad de ser una librería multiplataforma disponible para Windows, Mac, Linux y Android. Distribuida bajo licencia BSD, lo que permite que pueda programarse con C, C++, Python, Java y Matlab y además permite su ejecución en diversas arquitecturas de hardware como smartphones, Raspberry Pi o PC. Otra de sus ventajas es la extensa documentación que posee, explicada y actualizada, con ejemplos de su uso e incluso tutoriales públicos para aquellas personas no iniciadas en su uso.

3.4.2 Reconocimiento de señales de tráfico

Las señales de tráfico se colocan a lo largo de las carreteras con la función de informar a los conductores sobre las condiciones de la carretera delantera, direcciones, restricciones o información de texto. Aunque las señales de tráfico tienen diferentes estructuras y apariencias en diferentes países, los tipos más esenciales de señales de tráfico son prohibitorios, de peligro, signos obligatorios y basados en texto. El reconocimiento de señales de tráfico [39] es una parte importante en los sistemas avanzados de asistencia al conductor (ADAS) y en sistemas de conducción automática. El primer paso clave del reconocimiento de señales es su detección (ver Figura 20), el cual es un problema desafiante debido a la existencia de diferentes tipos, la existencia de tamaños pequeños, escenas de conducción complejas y oclusiones.

Para la detección de señales, la cámara y el LIDAR son los dos sensores usados más populares. En los últimos años, ha habido una gran cantidad de algoritmos para la detección y localización de letreros basados en visión artificial y reconocimiento de patrones. Destacan los métodos basados en reconocimiento de colores, formas, formas y colores, métodos basados en aprendizaje automático y métodos basados en LIDAR.

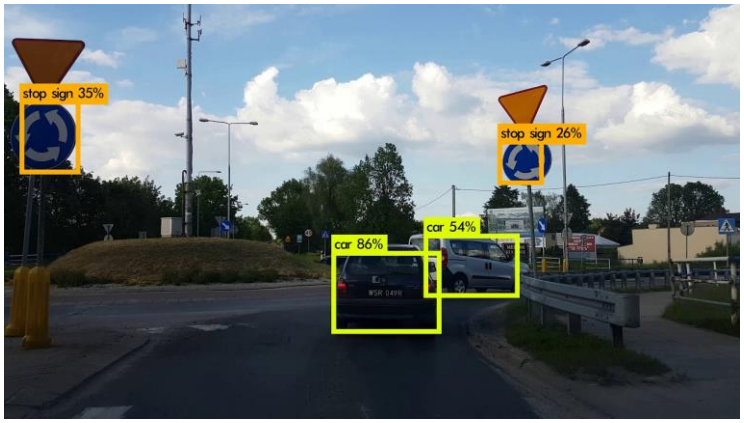

Figura 20. Detección de señales

A lo largo de los años, muchos métodos de detección fueron diseñados en base a la detección de colores especiales como azul, rojo y amarillo [40], utilizados para la reducción preliminar del espacio de búsqueda, seguido por algunos otros métodos de detección, como

de forma o borde, diseñados para detectar círculos, triángulos u octágonos. A través de los métodos de detección de formas y bordes también se podía extraer la posición exacta de una señal de tráfico. En la última década, a pesar de que existían métodos para la clasificación, la mayoría se encuentran desactualizados [41].

Recientemente, con el desarrollo del Machine Learning y metodologías de aprendizaje profundo, los métodos de detección basados en aprendizaje automático se han convertido gradualmente en los algoritmos más utilizados. Entre los métodos más destacados hoy en día para la detección de señales encontramos:

- Detección basada en AdaBoost.
- Detección basada en Support Vector Machines (SVM).
- Detección basada en Redes de Neuronas Artificiales.

Los métodos de detección basados en Machine Learning han logrado mucho éxito e importancia durante los últimos años, en este y en muchos más campos de la visión artificial, pero ¿qué es el Machine Learning?

3.5 Machine Learning

El Aprendizaje Automático o Machine Learning es una rama dentro de la Inteligencia Artificial, que comprende el aprendizaje a partir de la experiencia [42], basado en la construcción de programas computacionales que automáticamente mejoran su rendimiento en una tarea determinada en base a la experiencia. Esto permite usar los ordenadores y otros dispositivos con capacidad computacional para que aprendan a extraer los patrones y relaciones que hay en nuestros datos por sí solos. Esos patrones se pueden usar luego para predecir comportamientos y en la toma de decisiones [43]. En la práctica, la máquina corre internamente un algoritmo que, en base a la constante entrada de datos externos en su estructura, modifica ciertos parámetros internos con el objetivo de lograr predecir escenarios futuros, o tomar acciones de manera automática según ciertas condiciones. Dado que estas acciones se realizan de manera autónoma, sin intervención humana, se dice que el aprendizaje es automático.

El Machine Learning tiene una amplia gama de aplicaciones, incluyendo motores de búsqueda, diagnósticos médicos, detección de fraude en el uso de tarjetas de crédito, análisis del mercado de valores, clasificación de secuencias de ADN, reconocimiento del habla y del lenguaje escrito, juegos y robótica. Pero para poder abordar cada uno de estos temas es crucial en primer lugar distinguir los distintos tipos de problemas de aprendizaje automático con los que nos podemos encontrar. Para ello veremos los tres principales métodos de aprendizaje: supervisado, no supervisado y por refuerzo [44]. En la Figura 21 podemos ver un esquema donde se muestran estas formas de aprendizaje y los principales campos de uso que se asocian con cada uno de ellos.

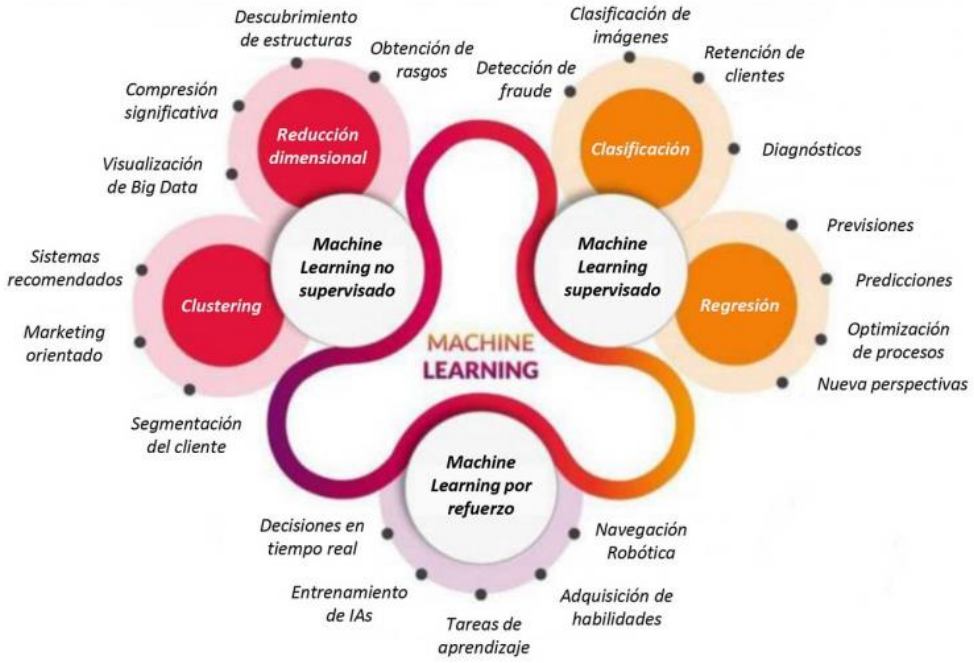

Figura 21. Formas de aprendizaje autónomo y campos de uso

- Aprendizaje supervisado: en los problemas de aprendizaje supervisado se enseña o entrena al algoritmo a partir de datos que ya vienen etiquetados con la respuesta correcta. Cuanto mayor es el conjunto de datos más puede aprender el algoritmo sobre el problema. Después de ser entrenado se le brindan nuevos datos sin las etiquetas de las respuestas correctas y el algoritmo de aprendizaje utiliza la experiencia adquirida durante la etapa de entrenamiento para predecir un resultado.

- Aprendizaje no supervisado: el algoritmo es entrenado usando un conjunto de datos que no tiene ninguna etiqueta; en este caso, nunca se le dice al algoritmo lo que representan los datos. La idea es que el algoritmo pueda encontrar por si solo patrones que ayuden a entender el conjunto de datos.

- Aprendizaje por refuerzo: el algoritmo aprende observando el mundo que le rodea. Su información de entrada es el feedback o retroalimentación que obtiene del mundo exterior como respuesta a sus acciones. Por lo tanto, el sistema aprende a base de ensayo-error.

A continuación, revisaremos algunos de los métodos de aprendizaje dentro de la inteligencia artificial más populares y cuyo uso puede ser adecuado en la tarea de aprendizaje y clasificación automática de señales de tráfico: Redes de Neuronas Artificiales, Deep Learning y Máquinas de Vectores Soporte (SVM).

3.5.1 Redes de Neuronas Artificiales

A pesar de que los conceptos de redes de neuronas artificiales o Deep Learning pueden parecer algo muy novedoso para muchas personas, su desarrollo empezó mucho tiempo atrás. En el año 1943, Walter Pitts y Warren McCulloh crean el primer modelo neuronal, donde crean una neurona que intenta modelar el comportamiento de una neurona natural [45]. Más adelante, entre la década del 50 y 60, Frank Rosenblatt logra el primer algoritmo que presentaba una red neuronal, a la que llama Perceptrón, utilizando las ideas introducidas por McCulloh. A pesar de que tuvo una buena recepción muy prometedora en los ámbitos

académicos, al final se probó que no podía ser entrenado para reconocer muchos otros tipos de patrones, lo que provocó un estancamiento en el avance de las redes neuronales durante algunos años. Entre 1970-1985 se conseguirá crear redes en que las salidas de una capa son utilizadas como entradas en la próxima (Redes Feedforward) y también se descubrirá el algoritmo de retro-propagación, que permitirá entrenar redes neuronales de múltiples capas de manera supervisada. Hasta 2006, se había alcanzado un punto en el que no se podían alcanzar ciertos niveles de aprendizaje por la falta de cómputo que, gracias al poder de las nuevas GPU y la aparición de nuevas ideas desaparece y es cuando se considera que se alcanza el Deep Learning, al lograr entrenar cientos de capas jerárquicas que conforman y potencian el aprendizaje profundo, dando una capacidad casi ilimitada a dichas redes [46]. Estas alcanzan su apogeo a partir de 2010 cuando, a consecuencia de la aparición de las redes sociales y el desarrollo del mundo web, se multiplica la generación de datos por individuo, lo que hace surgir una necesidad inmediata de herramientas y tecnologías que permitan aprovechar estos datos como son las redes neuronales de aprendizaje profundo y el Big Data.

El comportamiento de las redes neuronales pretende imitar al propio funcionamiento de las neuronas naturales. La neurona es una función matemática (ver Figura 22) que recibe unos datos de entrada (inputs) con los cuales realiza una serie de operaciones y produce unos datos de salida (outputs). Internamente, se realiza una suma ponderada con los datos de entrada y unos valores llamados pesos, los cuales son unos parámetros autoajustables que validarán la importancia que debe tener cada dato de entrada en la suma ponderada. A esta suma se le añade un parámetro “b” que suele llamarse parámetro de sesgo, que controla la predisposición de una neurona a dar su salida. La ecuación descrita anteriormente es $f(x1,x2)=b+1\ x1+w2x$.

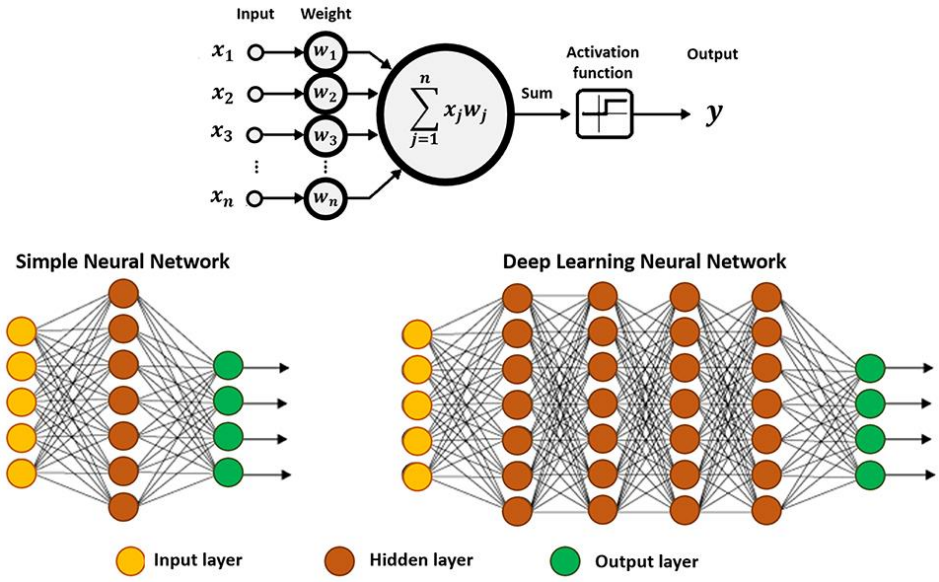

Figura 22. Estructura y matemática de una red neuronal artificial

Tras el sumatorio, el resultado atraviesa un bloque designado como *Threshold*. Esto es lo que se conoce como función de activación [47] y posee una gran importancia para el correcto funcionamiento de la neurona artificial. En las neuronas biológicas, si el estímulo supera un cierto valor umbral que viene predefinido, entonces la neurona se dispara, en caso negativo no. Las funciones de activación realizan esa misma tarea. Son funciones matemáticas como la tanh, la sigmoide o la ReLU que actúan a modo de filtro dejando solo pasar ciertos datos que se adecuen a su criterio e introduciendo una no linealidad a la salida. Las activaciones no lineales permiten que las redes neuronales se combinen entre sí de maneras mucho más complejas y modelen sistemas que de otra manera no podrían ser modelados.

La funcionalidad de una neurona artificial resulta muy reducida al menos que se combine con otras neuronas formando lo que se conoce como “red neuronal”. Las redes neuronales [48] se dividen en 3 partes (ver en Figura 22):

- Capa de entrada (Input Layer): está compuesto por las neuronas que asimilan los datos de entrada, como por ejemplo imagen o una tabla de datos.

- Capa oculta (Hidden Layer): es la red que realiza el procesamiento de información y hacen los cálculos intermedios. Cuantas más neuronas haya en esta capa, más complejos son los cálculos que se efectúan.

- Salida (Output Layer): es el último eslabón de la cadena y es la red que toma la decisión o realiza alguna conclusión aportando datos de salida. Esta capa es clave ya proporciona la información de salida de nuestra red neuronal. Su clave radica en el uso de las funciones de activación y de saber elegir correctamente las activaciones que más se adecuan al problema enfrentado.

Originalmente, las redes clásicas, también llamadas perceptrones multicapa clásicos, contenían una única capa oculta lo que limitaba enormemente su desempeño y potencial, cuando se descubre la ‘backpropagation’ o retro-propagación se hace realidad la posibilidad de utilizar más de una capa oculta. En la Figura 22 podemos ver la diferencia entre una red neuronal simple y una con varias capas. Con el paso del tiempo se ha ido buscando “más profundidad”, es decir, mayor número de capas ocultas y de neuronas. El algoritmo de retro-propagación que hizo esto posible consistía en recorrer una primera vez la red con los pesos inicializados de forma aleatoria y comparar el resultado de salida con el real para calcular de esta manera el error producido. A continuación, la red era recorrida de forma inversa (retro-propagación) ajustando los valores de los pesos mediante el uso de algoritmos de optimización como el descenso del gradiente, el cual permite converger hacia el valor mínimo de una función mediante un proceso iterativo. Este proceso se repetiría hasta que hubiera una convergencia y se produjera un margen de error aceptable entre los valores reales y predichos. Este proceso de propagación hacia delante y detrás ajustando los pesos hasta reducir el error del efecto esperado es lo que se conoce como entrenamiento. Veamos en la Figura 23 como podría ser la extracción de las características de una imagen a través de varias capas.

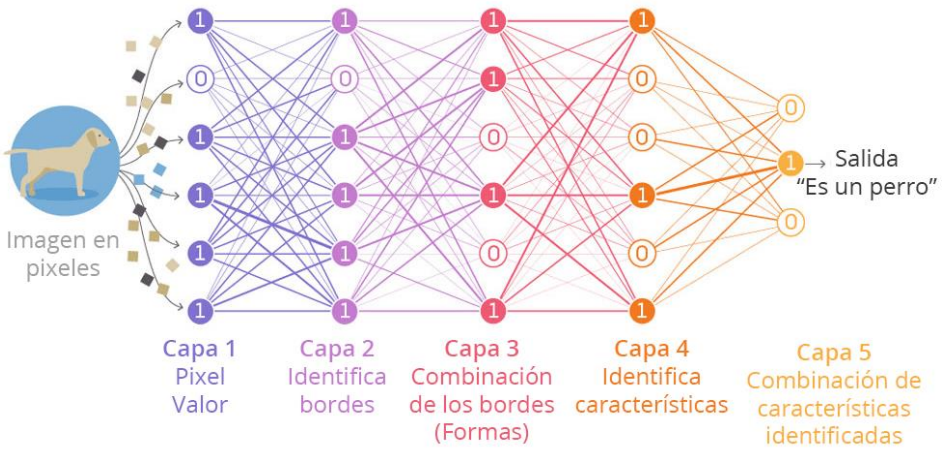

Figura 23. Extracción de características en capas ocultas

A continuación, estudiemos algunos ejemplos de los modelos y librerías que utilizaremos.

3.5.2 Perceptrón multicapa (MLP)

El perceptrón multicapa (MLP), es un algoritmo de aprendizaje supervisado [49] que aprende una función mediante el entrenamiento de un conjunto de datos, donde 'X' es el número de dimensiones para la entrada e 'Y' es el número de dimensiones para la salida. Dado un conjunto de características y un objetivo, puede aprender un aproximador de una función no lineal para clasificación o regresión. Es diferente de la regresión logística ya que entre la capa de entrada y la de salida, puede haber una o más capas no lineales, llamadas capas ocultas. La Figura 24 se muestra un MLP de una capa oculta con salida escalar.

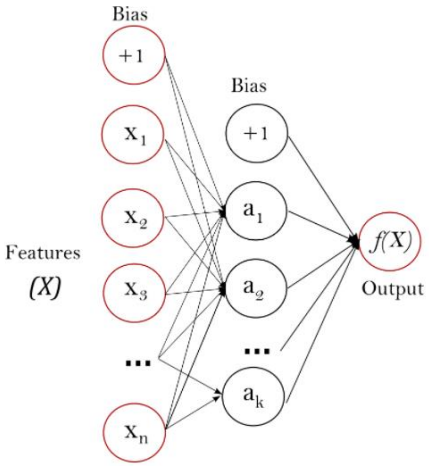

Figura 24. MLP

La MLP utiliza las capas ocultas y se entrena usando la retro-propagación. Esta red se entrena con dos vectores, una matriz "x" con un que incluye el número de ejemplos y características, es decir, contiene las muestras de entrenamiento representada en vectores de características y una matriz "y" del tamaño del número de ejemplos que contiene los valores objetivo (etiquetas de cada clase) para las muestras de entrenamiento. A continuación, veremos las ventajas y desventajas de este modelo:

↑ Capacidad para aprender modelos no lineales.

↑ Capacidad para aprender modelos en tiempo real (aprendizaje en línea).

↓ Los MLP con capas ocultas tienen una función de pérdida no convexa cuando existe más de un mínimo local. Por lo tanto, diferentes inicializaciones de peso aleatorio pueden llevar a una precisión de validación diferente.

↓ MLP requiere ajustar una serie de hiperparámetros, como la cantidad de neuronas, capas e iteraciones ocultas.

3.5.3 Máquinas de vectores de soporte (SVM)

Las máquinas de vectores de soporte (SVM), son también un algoritmo de aprendizaje supervisado utilizado para la clasificación, la regresión y la detección de valores atípicos [50]. Una máquina de vectores de soporte construye un hiperplano o un conjunto de hiperplanos en un espacio dimensional alto o infinito, de esta manera, se logra una separación eficaz por el hiperplano que tiene la mayor distancia a los puntos de datos de entrenamiento más cercanos de cualquier clase (el llamado margen funcional), ya que en general cuanto mayor

es el margen menor es el error de generalización del clasificador. La Figura 25 muestra la función de decisión para un problema linealmente separable, con tres muestras en los límites del margen, llamadas "vectores de soporte".

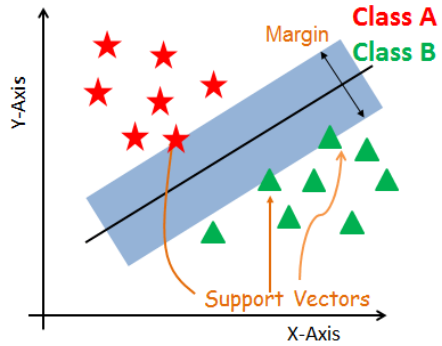

Figura 25. Máquina de vectores de soporte

En general, cuando el problema no se puede separar linealmente, los vectores de soporte son las muestras dentro de los límites del margen.

Como la mayoría de los problemas en el mundo real no son linealmente separables, se realiza una transformación no lineal de los datos de entrada “x” en otro espacio con funciones no lineales predeterminadas denominadas kernels (φ). Con las funciones de kernel se realiza una expansión de la dimensionalidad, tratando de buscar una relación lineal, véase un ejemplo en la Figura 26.

$$K(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x})^T \Phi(\mathbf{x}')$$

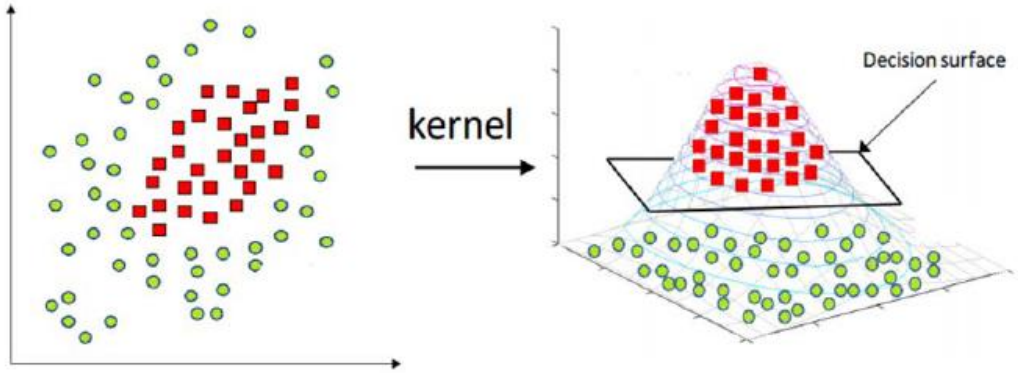

Figura 26. Ejemplo de clasificación no lineal

Cuando los problemas no son linealmente separables, se incluye un parámetro de regularización C que establece un compromiso entre el error de entrenamiento y la complejidad del modelo. Un C bajo será un modelo sencillo, con mayor error en el entrenamiento y suavidad en la frontera de decisión. Un C alto significa un modelo complejo, con poca suavidad en la frontera de decisión y riesgo de sobreajuste. Otro parámetro importante para definir será gamma, que define cuánta influencia tiene un solo ejemplo de entrenamiento. Cuanto mayor sea la gamma, más cercanos deben estar los otros ejemplos para que se vean afectados. La elección adecuada de C y gamma es fundamental para el rendimiento del SVM. Algunas de sus ventajas y desventajas son:

↑ Eficaz en espacios de gran dimensión, con dimensiones mayores que el número de muestras.

↑ Al utilizar un subconjunto de puntos de entrenamiento en la función de decisión (llamados vectores de soporte) son eficientes el uso de la memoria.

↑ Versátil: diferentes funciones de Kernel puedes ser especificadas en la función de decisión. Se proporcionan Kernels comunes, pero también es posible especificarlos personalizados.

↓ Si el número de características es mucho mayor que el número de muestras es crucial evitar sobreajustes al elegir las funciones del Kernel.

↓ Las SVM no proporcionan directamente estimaciones de probabilidad, estas se calculan mediante una costosa validación cruzada de cinco veces.

3.5.4 Redes convolucionales (CNN)

En el caso de la CNN nos encontramos ante un modelo de Deep Learning, unas redes multicapas que toman su inspiración del córtex visual de los animales. Consta con una arquitectura de varias capas que implementan la extracción de características para luego clasificar. Esto ayuda a ejecutar redes neuronales directamente sobre imágenes y las hace muy eficientes y fáciles en comparación con otros modelos. La principal diferencia entre las CNN y las redes neuronales ordinarias es que en ellas se asume que la entrada es una imagen [51].

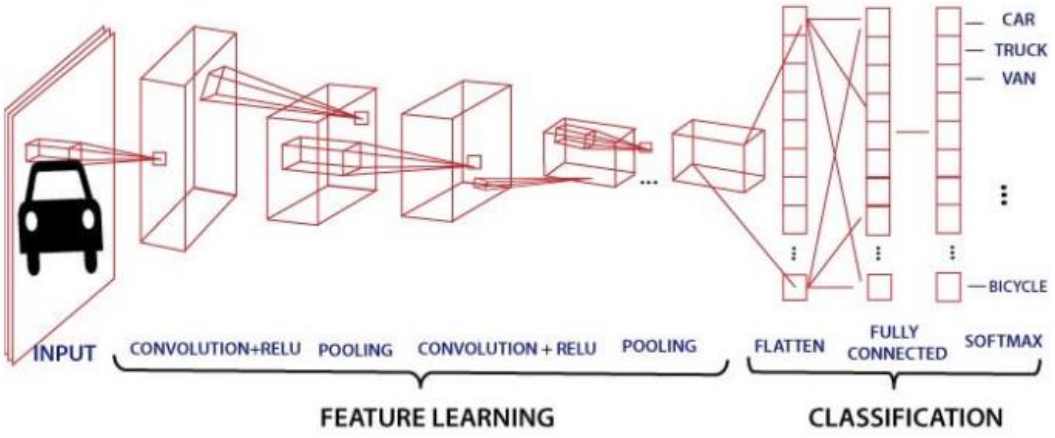

Figura 27. Modelo de CNN

Considerando que una imagen no es más es un conjunto de píxeles, la podemos aplanar y luego alimentar el modelo. Imaginemos el siguiente ejemplo de una imagen de entrada de tamaño 10 * 10 * 1, si el tamaño del Kernel es 3 * 3 * 1, la salida será una imagen de 8 * 8 * 1. Es decir, en CNN, en lugar de calcular cada píxel por separado, procesamos en el grupo de píxeles. Y este grupo está definido por el tamaño del Kernel (ver Figura 28).

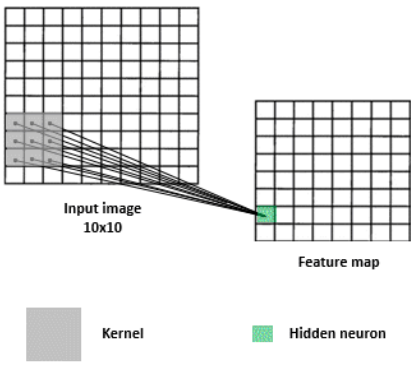

Figura 28. Función de Kernel

Podemos observar la arquitectura de una red convolucional simple en Figura 27, si la analizamos en detalle encontramos una parte basada en la extracción de características y otra en la parte de clasificación, siguiendo la siguiente estructura:

1. Input: esta capa incorporaría la imagen de entrada como una matriz de píxeles 3-D.

2. La capa convolucional hará el producto escalar entre la función de Kernel y una sub-matriz de la imagen de entrada del mismo tamaño que Kernel. Luego sumará todos los valores resultantes del producto escalar y este será el valor de un solo píxel de una imagen de salida. Este proceso se repite hasta cubrir toda la imagen de entrada y para todos los Kernel.

3. La siguiente capa RELU aplica una función de activación max (0, x) en todos los valores de píxeles de una imagen de salida.

4. La capa de agrupación (Pooling Layer) se utiliza para reducir la dimensión de un volumen de entrada, reduce el tamaño espacial de modo que disminuya la potencia computacional necesaria para procesar la imagen. Además, puede utilizar capas Dropout, que desactivarán aleatoriamente un porcentaje de las neuronas en cada capa oculta, acorde a una probabilidad de descarte, para reducir la complejidad del modelo.

   Hasta aquí, estas capas se encargan de la extracción de características. Las redes convolucionales también utilizan el descenso del gradiente y pueden añadirse tantas como se desee. A continuación, la siguiente capa procede a la clasificación:

5. La capa completamente conectada (FC-Layer) se utiliza para la clasificación de las características complejas extraídas de capas anteriores. Esta capa es la misma que la de las redes neuronales en las que cada neurona está conectada a todas las neuronas de la capa consecutiva. Para pasar una imagen de entrada a esta capa, necesitamos aplanar la imagen para que todos los valores de píxeles estén organizados en una columna. La salida final se calcula usando Softmax que da la probabilidad de cada clase para las características dadas.

3.5.5 Scikit-learn

Scikit-learn [52] es una librería muy popular en lenguaje Python para trabajar con Machine Learning e incluye la implementación de un gran número de algoritmos de aprendizaje. La podemos utilizar para clasificaciones, extracción de características, regresiones, agrupaciones, reducción de dimensiones, selección de modelos, o preprocesamiento. Posee una API que es consistente en todos los modelos y se integra muy bien con el resto de los

paquetes científicos que ofrece Python. Esta librería también nos facilita las tareas de evaluación, diagnóstico y validaciones cruzadas ya que nos proporciona varios métodos de fábrica para poder realizar estas tareas en forma muy simple.

3.5.6 TensorFlow

TensorFlow es una plataforma de código abierto para aprendizaje automático. Cuenta con un ecosistema integral y flexible de herramientas, bibliotecas y recursos de la comunidad que les permite a los investigadores innovar con el aprendizaje automático [53]. Esta biblioteca de código abierto para aprendizaje automático fue desarrollada por Google para satisfacer sus necesidades de sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y a los razonamientos usados por los humanos. TensorFlow fue originalmente desarrollado por el equipo de Google Brain para uso interno en Google antes de ser publicado bajo la licencia de código abierto Apache 2.0 el 9 de noviembre de 2015. Compila y entrena modelos de AA con facilidad mediante API intuitivas y de alto nivel como Keras, con ejecución inmediata, que permite una iteración de modelos inmediata y una depuración fácil. TensorFlow se construyó pensando en el código abierto y en la facilidad de ejecución y escalabilidad. Permite ser ejecutado en la nube, pero también en local. La idea es que cualquiera pueda ejecutarlo. Se puede ver a personas que lo ejecutan en una sola máquina, un único dispositivo, una sola CPU (o GPU) o en grandes clústeres. La disparidad es muy alta. Entre sus aplicaciones más conocidas destacan mejorar la fotografía de los smartphones, ayudar al diagnóstico médico y procesamiento de imágenes. Con esta librería utilizaremos las redes neuronales convolucionales o CNN.

4. Metodología

El presente TFG es un trabajo científico-teórico dentro de un marco de investigación. Como tal, es necesario establecer la metodología de trabajo utilizada para la realización de las pruebas, la obtención de datos y la redacción de conclusiones.

La metodología aplicada está acorde con los objetivos y sub-objetivos establecidos en el apartado correspondiente, de modo que se consiga su consecución:

1. **Analizar las necesidades actuales en el entorno de pruebas de conducción autónoma del GII para decidir qué objetos detectar:** Después de varias reuniones con los directores y otros miembros del GII, se decide que la librería ya existente de reconocimiento de objetos funciona adecuadamente y se estable como objeto de interés del proyecto el reconocimiento de señales de tráfico. Esto permitiría eliminar los actuales códigos QR o Aruco que se acoplan a las señales para su detección, y que quita realismo al entorno de pruebas.

2. **Escoger los métodos de identificación de señales tras analizar su estado del arte:** Una vez conocidos los objetos que queremos detectar, investigaremos y escogeremos las técnicas de detección de señales más apropiadas.

3. **Reunir un conjunto de imágenes de entrenamiento:** La creación de un conjunto de imágenes (benchmark) adecuado es indispensable a la hora de entrenar, comparar y seleccionar los distintos modelos escogidos para el reconocimiento de señales.

4. **Entrenar y comparar el funcionamiento de las técnicas escogidas con las imágenes del conjunto:** Se compararán las distintas técnicas empleadas a partir del conjunto de datos y se seleccionarán las mejores para realizar las pruebas en el entorno de pruebas del GII.

5. **Comprender el funcionamiento del robot Robobo y el entorno en el que se va a utilizar:** Debemos conocer las librerías de programación del robot que se usará, y las propiedades del entorno y las señales reales.

6. **Crear un programa para obtener recortes de señales del entorno utilizando el propio robot y su cámara:** La obtención de imágenes de señales a partir del recorte del área correspondiente a la señal en el conjunto de imágenes del entorno que ve el robot con la cámara, facilita la obtención de datos reales para probar la eficacia de las técnicas desarrolladas sobre el entorno real de conducción autónoma. Para ello es necesario desarrollar un programa que se ejecute en tiempo real en el robot y realice dicha tarea.

7. **Validar y comparar las técnicas de visión con las imágenes extraídas:** Primero se realizarán pruebas en el ordenador con las imágenes sacadas del entorno de conducción autónoma para después concluir las pruebas a través de una operación en tiempo real con el robot.

8. **Seleccionar el algoritmo de detección y extraer conclusiones:** Una vez escogido el algoritmo que utilizaremos en nuestra librería se probará y afinará en escenarios reales con el fin de extraer las conclusiones sobre su funcionamiento y reevaluar la decisión tomada en caso de que sea necesario.

5. Resultados

En este apartado ejecutaremos los pasos establecidos en el anterior apartado para el desarrollo de librerías de identificación y reconocimiento de un grupo de señales de tráfico preseleccionadas en una secuencia de video capturada en tiempo real. En pocas palabras, primero se explicará como es nuestro conjunto de datos de entrenamiento, después que modelos utilizaremos, compararemos y pre-seleccionaremos y, finalmente, conoceremos como y donde se realizará la toma de imágenes a través del robot para realizar las pruebas finales y seleccionar el modelo más sencillo y fiable.

Las librerías están escritas en Python, la detección se realiza utilizando OpenCV y la identificación se realiza usando distintas técnicas de Aprendizaje Máquina disponibles en las bibliotecas Scikit-Learn y Tensorflow.

5.1 Búsqueda y ajuste del conjunto de entrenamiento

Para entrenar y probar los diferentes modelos de identificación de señales, se utilizarán un conjunto de señales reales obtenidas a través de imágenes captadas por coches conduciendo a lo largo de diferentes países, recogidas de dos datasets diferentes: el conjunto alemán de reconocimiento de señales de tráfico alemán (GTSRB) [54] y el conjunto de datos belga (BelgiumTS Dataset) [55]. Se toma este punto de partida asumiendo que si el modelo de identificación es capaz de detectar esas señales reales también lo hará con las utilizadas en el entorno de pruebas, ya que las condiciones de visibilidad de las señales y la calidad de estas son más favorables.

El conjunto de datos seleccionado consta de 22 clases diferentes (divididas del 0-13 en rojas y del 14-21 en azules), seleccionadas entre las 42 y las 62 clases del conjunto alemán y belga, respectivamente, y un total de 3.385 imágenes de entrenamiento. Estas señales se han elegido del conjunto total teniendo en cuenta aquellas que serían más interesantes para utilizar en el entorno de pruebas de conducción autónoma. En la figura 29 podemos observar la frecuencia de aparición de cada clase en nuestro conjunto. Como podemos observar, existe un desbalance en el conjunto de datos, esto quiere decir que el número de observaciones no es el mismo para todas las clases usadas para la clasificación, lo que podría generar problemas en algunos tipos de clasificadores de Machine Learning con múltiples categorías.

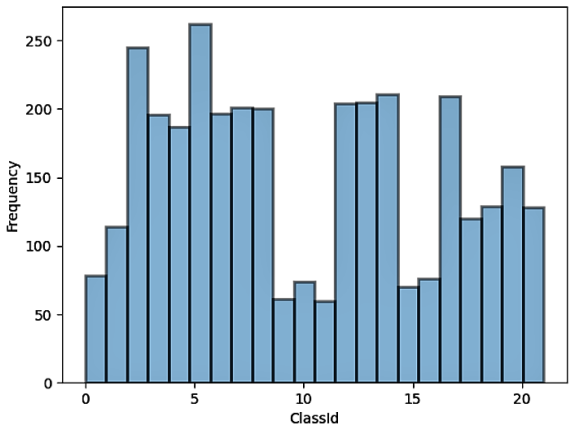

Figura 29. Frecuencia de aparición de cada clase

En nuestro conjunto aparecerán señales tomadas desde diferentes distancias y desde distintos ángulos, como veremos en la Figura 30 y 31. A continuación, veamos un listado de las señales escogidas y sus imágenes, ordenadas de la clase 0 a la 21.

- **Señales de peligro**: curva peligrosa hacia la izquierda (0), curva peligrosa hacia la derecha (1), paso de peatones (2), obras (3), semáforo (4), intersección (5).

- **Señales de prioridad**: ceda el paso (6), stop (7).

- **Señales de prohibición o restricción**: entrada prohibida (8), giro a la izquierda prohibido (9), giro a la derecha prohibido (10) y velocidades máximas de 20 (11), 30 (12) y 50 (13).

Figura 30. Señales de peligro, prioridad y prohibición

- **Señales de obligación:** sentido obligatorio recto (14), sentido obligatorio derecha (15), sentido obligatorio izquierda (16), sentido obligatorio curva derecha (17), sentido obligatorio curva izquierda (18), rotonda (19), zona de estacionamiento (20) y paso de peatones (21).

Figura 31. Señales de obligación

5.2 Búsqueda y prueba de los modelos para clasificación

El objetivo final de las librerías desarrolladas será la identificación en tiempo real de las señales capturadas con la cámara. Para ello, en este apartado se explicarán los modelos de aprendizaje automático que se han seleccionado, y el procesado realizado sobre el conjunto de datos previo para dar soporte a dicho aprendizaje.

En el campo del aprendizaje automático, el problema que se va a resolver es un problema de aprendizaje supervisado, en concreto de clasificación. Este consiste en tomar un vector de entradas X y decidir a cuál de las N clases posibles pertenecen. El punto más importante del problema de clasificación es que es discreto, cada ejemplo debe pertenecer únicamente a una clase, y el conjunto de clases cubre todo el espacio de salida posible.

En este TFG, se seleccionaron tres modelos de aprendizaje para resolver el problema, por su aplicabilidad a este tipo de casos: SVM, MLP y CNN. Los tres fueron explicados en el apartado de estado del arte.

5.2.1 Parametrización de los modelos

- **SVM**: debido a dificultades para conseguir resultados interesantes ajustando manualmente los parámetros C y gamma, se utiliza el parámetro gamma='scale', empleando una metodología similar a un programa de detección de señales en condiciones similares [56].

- **MLP**: en este caso, ya que las redes neuronales no tienen ningún criterio establecido para conocer que disposición de capas y neuronas es más adecuado, se han ido probando diferentes modelos de una, dos y tres capas con distintos números de neuronas. También, en cada configuración se ha ido variando el tamaño del batch (hiperparámetro que define el número de muestras para trabajar antes de actualizar los parámetros internos del modelo), de forma que en las tablas 3 y 4 se mostrará el mejor resultado de cada configuración de capas.

- **CNN**: en este tipo de redes hay muchos parámetros que se pueden modificar como el tamaño de los kernels y los filtros de cada capa, el número de capas, la capa dropout, número de batch, epochs … Para escoger estos modelos se buscaron problemas parecidos de detección de señales utilizando este tipo de redes de aprendizaje profundo [57] [58]. A partir de los dos modelos encontrados, se han ido variando los parámetros de batch y epochs (hiperparámetro que define el número de veces que el algoritmo de aprendizaje funcionará en todo el conjunto de datos de entrenamiento) durante la realización de los procesos de entrenamiento con nuestro conjunto de datos.

5.2.2 Pre-procesado de las imágenes

En el caso de las SVM y las MLP, para conseguir mejores resultados, se ha decido utilizar conjuntos de entrenamiento que tengan un menor número de clases, a través de la separación en dos modelos, uno para la detección de señales rojas y otro para las azules. Esto ya viene facilitado por el proceso de pre-procesmiento de la imagen, en la que para detectar el área de las señales se separa la detección en señales azules y rojas, ya que es una detección basada en buena parte en color. Para utilizar de forma eficaz este tipo de clasificadores automáticos con imágenes, es necesario realizar una fase previa de extracción de características utilizando algún algoritmo adecuado de visión por computador. El vector de características obtenido de esta forma será lo que se introduzca como entrada a dichos clasificadores, tanto durante la fase de aprendizaje como durante la fase de aplicación. En concreto, se utiliza el método Histograma de Gradientes Orientados HOG [59], que resulta muy adecuado para extraer la forma predominante de imágenes de este tipo (señales de tráfico). Así, el procesado de imagen, en esta fase, será el mostrado en la Figura 32, donde primero se lee la imagen, se transforma a escala de gris, se reduce el tamaño, se extrae su vector de características con HOG y, finalmente, se alimenta al clasificador automático.

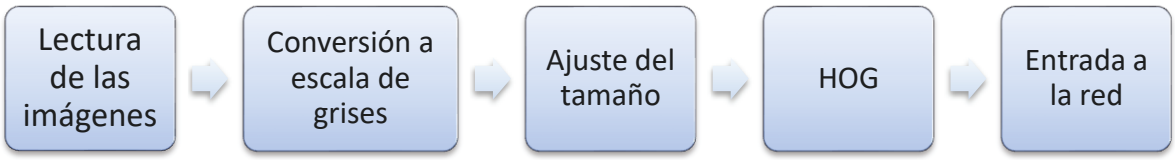

Figura 32. Pre-procesado en MLP Y SVM

En las redes convolucionales (CNN), el proceso será diferente, ya que la extracción de características la realiza la propia red CNN en sus primeras capas y en sus últimas realiza la clasificación a partir del resultado de las capas estas, una vez ha aprendido a generalizar cada clase en base a un conjunto reducido de características. Después de la lectura de la imagen, se ajustará su tamaño y se creará una matriz a partir de la imagen, utilizando Numpy [60], que será el input de la red (ver en la Figura 33).

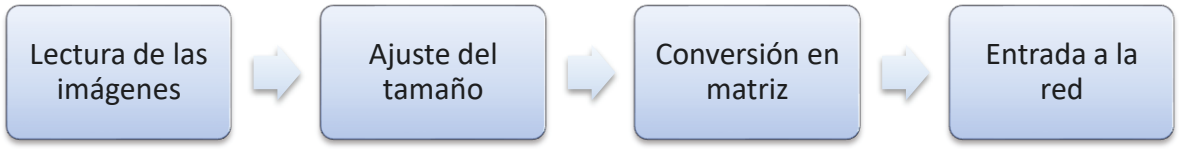

Figura 33. Pre-procesado en CNN

5.2.3 Comparación de modelos

Una vez escogidos los distintos modelos que vamos a utilizar, es el momento de compararlos y elegir cuales son los más apropiados. Para ello, es necesario establecer un método de comparación. En este caso se utilizará el método score [61], que mide la precisión del modelo, entendida como el porcentaje de valores clasificados correctamente con respecto al total de elementos de un proceso de ejecución y test basado en validación cruzada. Para realizar la validación cruzada, se utilizará la librería Kfold [62], que divide el conjunto de datos en K subconjuntos distintos de aproximadamente el mismo tamaño, usando una K=5 en nuestras pruebas. Además, se utiliza ShuffleSplit [63] para asegurar un submuestreo aleatorio.

Una vez ejecutadas las pruebas, se dividirán los resultados en tres tablas ordenadas de mejor a peor score. Recordemos que esto se debe a que habrá tres tipos de red, una para el conjunto completo de las imágenes y una para cada color. Una vez vistos los resultados, se analizará el score, la desviación y el tiempo de ejecución de cada modelo para sacar las conclusiones sobre el rendimiento. En caso de tener resultados muy parecidos se analizarían las matrices de confusión, con el fin de analizar si el modelo comete muchos fallos en un tipo de clase, lo que la haría ser menos interesante.

5.2.3.1 Resultados de reconocimiento de señales rojas

MLP/SVM ROJO			
	SCORE	**DESVIACIÓN**	**TIEMPO (s)**
MLP(24)	0,981	±0,0015	16,7
MLP(18,24)	0,974	±0,0103	19,7
MLP(18)	0,971	±0,0165	15,14
MLP(18,18)	0,958	±0,0347	20,27
MLP(9)	0,955	± 0,0111	14,46
SVM	0,938	± 0,0114	22,41
MLP(12,12,12)	0,922	±0,0102	16,54
MLP(6)	0,903	±0,0487	15,19
MLP(9,9)	0,901	±0,0463	25,51
MLP(9,18)	0,896	±0,0524	29,45
MLP(18,18,18)	0,874	±0,0468	23,77
MLP(8,6)	0,861	±0,066	20,04
MLP(8,12,24)	0,771	±0,0902	19,43
MLP(9,9,9)	0,757	±0,02001	22,99
MLP(6,6)	0,757	±0,04892	26,19

Tabla 2. Evaluación de modelos SVM Y MLP para señales rojas

Como vemos en la Tabla 2, la mejor red es la formada por una capa oculta con veinticuatro neuronas y como su desviación, score y tiempo de ejecución son mejores que la MLP(24), será la que guardaremos para realizar futuras pruebas.

5.2.3.2 Resultados de reconocimiento de señales azules

MLP/SVM AZUL			
	SCORE	**DESVIACIÓN**	**TIEMPO (s)**
MLP (9)	0,997	±0,0056	16,09
MLP(18)	0,996	±0,0054	17,16
MLP(18,18)	0,996	±0,0123	19,72
MLP(18,24)	0,995	±0,0020	18,52
MLP(24)	0,995	±0,0012	17,49
MLP(18,18,18)	0,986	±0,0168	19,86
SVM	0,985	±0,0175	21,88
MLP(9,18)	0,985	±0,0125	13,91
MLP(6)	0,984	±0,0128	11,66
MLP(12,12,12)	0,981	±0,0179	16,07
MLP (9,9)	0,965	±0,0378	17,33
MLP(6,6)	0,961	±0,0479	18,1
MLP(8,6)	0,956	±0,0465	18,93
MLP(9,9,9)	0,913	±0,0545	23,66
MLP(8,12,24)	0,886	±0,0768	19,84

Tabla 3. Evaluación de modelos SVM Y MLP para señales rojas

En la Tabla 3 podemos ver que el mejor score lo tiene una red de una capa oculta y nueve neuronas. Además, por ser un modelo más sencillo y con menor tiempo de ejecución, esta red, la MLP(9), será la que guardemos para la comparación final.

5.2.3.3 Resultados de las CNN

En el caso de las CNN nos hemos basado en dos modelos diferentes a los que nos hemos referido anteriormente [58] [59] y que se diferencian en complejidad y diversos parámetros como el número de capas ocultas o filtros de kernel. Está fuera del alcance de este trabajo realizar pruebas exhaustivas con los diferentes parámetros de este tipo de red, por lo que se han seleccionado los valores estándar proporcionados por los autores. Podemos observar los parámetros establecidos en la Figura 34 y 35, donde se ve que el modelo 1 es más sencillo que el 2 ya que cuenta con 8 tipos de capas y no cuenta con capas Dropout. Por otro lado, el modelo 2 está formado por 12 tipos de capas. Realizamos tres pruebas diferentes en cada modelo, variando únicamente los hiperparámetros batch y epochs (ver Tabla 4). Por ello llamaremos a las pruebas del primer modelo 1.1, 1.2 y 1.3 y a las del segundo 2.1, 2.2 y 2.3. Los resultados se muestran en la Tabla 4, donde se observa cómo el mejor score es el del modelo 2.1, aunque con ligera diferencia respecto al siguiente modelo (1.2). Debemos recordar que estas redes son entrenadas sin separar las señales rojas de las azules, por lo que, aunque su score final es inferior al de la mejor pareja MLP, guardaremos el modelo 2.1 por su mejor score y menor tiempo de ejecución para probarlo más adelante.

CNN					
	EPOCHS	**BATCH**	**SCORE**	**DESVIACIÓN**	**TIEMPO(s)**
CCN(2.1)	20	50	0,9876	±0,26	92,45
CCN(1.2)	15	50	0,9846	±0,15	100.53
CCN(1.1)	20	50	0,9835	±0,15	38,16
CCN(1.3)	20	100	0,9802	±0,69	37.29
CCN(2.2)	15	50	0,9775	±1,17	77,81
CCN(2.3)	20	100	0,9645	±1,32	100.73

Tabla 4. Evaluación de la CNN

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=X_train.shape[1:]))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(22, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Figura 34. CNN 1

```
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=X_train.shape[1:]))
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(22, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Figura 35. CNN 2

5.2.3.4 Guardado de los modelos

Tanto en tiempo de ejecución como en mejor score, la pareja MLP es la que ha proporcionado mejores resultados. Como queremos realizar las pruebas finales con más modelos, guardaremos por un lado la mejor red convolucional y por otro lado la mejor red MLP roja y la mejor MLP azul. Las SVM también se probarán, simplemente por analizar también el resultado de este modelo, ya que recibió scores más bajos que muchas otras redes. En el caso de la SVM y la MLP se guardarán los modelos en ficheros a través de la librería joblib de Python [64] y en el de la CNN utilizando la función 'model.save 'de TensorFlow [65]. Esto se utilizará para que, más adelante, durante la ejecución del detector de señales en tiempo real se puedan cargar, y así ejecutar la identificación de señales.

5.3 Validación en Robobo

Hasta ahora, solo hemos probado los modelos con conjuntos de datos libremente accesibles con imágenes provenientes de vías urbanas e interurbanas reales. No obstante, el objetivo de este TFG es lograr una librería que funcione adecuadamente en el entorno Robobo SmartCity.

Figura 36. Robobo Smart City

La Robobo SmartCity (ver Figura 36) es un modelo a escala de un barrio de una ciudad representado por una maqueta de 3,5 m x 4 m que se crea ante la idea de utilizar la conducción autónoma y las ciudades inteligentes como entornos para la enseñanza de la robótica. Este trazado urbano está formado por una carretera exterior de doble sentido, que rodea la parte interior de la ciudad, varios edificios, pasos de cebra, indicaciones en la carretera y lo que nos interesa, señales de tráfico (ver Figura 37).

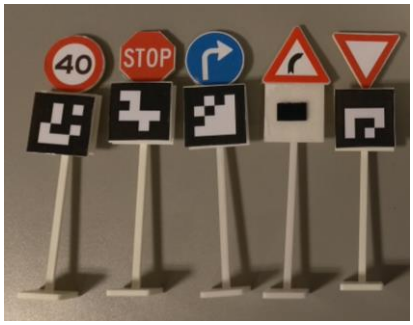

Figura 37. Ejemplos de señales de la Robobo Smart City

En el siguiente apartado se explicará como se realizará la detección de las señales en la Robobo SmartCity y, a continuación, se probarán los diferentes modelos a través de un programa de funcionamiento en tiempo real con Robobo.

5.3.1 Captura y clasificación de imágenes en el entorno robótico

Para tomar las nuevas imágenes de prueba, se hace circular al robot en una calle con varias señales y se crea un programa de recorte que vaya guardando las distintas señales capturadas a través de la cámara del robot, para el posterior análisis de los modelos. En las Figuras 38 y 39 podemos ver como se realiza este proceso. A continuación, explicaremos como funciona el programa creado para el recorte de señales.

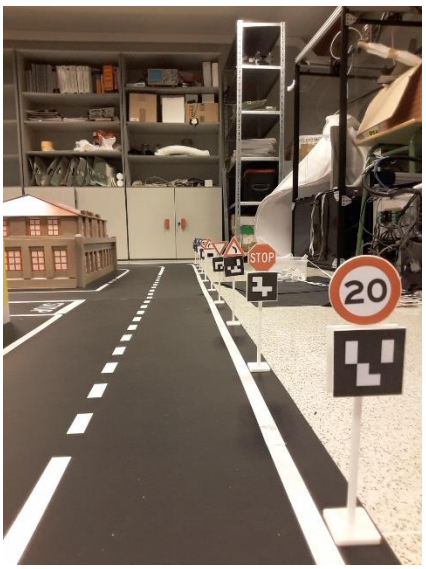

Figura 39. Vista desde la cámara de Robobo

Figura 38. Vista exterior del Robobo

Para poder separar las señales rojas de las azules, el programa de recorte se hará utilizando técnicas de visión artificial, siguiendo los pasos que se describen a continuación.

5.3.1.1 Segmentación de los colores deseados

Antes de segmentar los colores, realizamos una ecualización de histograma en la imagen para que se pueda realizar un ajuste de contraste (ver Figura 40). Esto generalmente ayuda a eliminar áreas demasiado brillantes u oscuras y también ayuda a limpiar la imagen de entrada.

La imagen ecualizada se convierte a formato HSV ya que facilita la segmentación por color. A continuación, definimos los límites más bajos y altos de las máscaras que utilizaremos para extraer los colores en los valores del formato HSV, en ellas se aumentarán los colores azules o rojos y hará el resto de la imagen negra (ver en la Figura 41). En el caso de las imágenes rojas se necesitará crear dos máscaras diferentes en lugar de una debido a que en el formato HSV este color se encuentra en dos esquinas diferentes del canal de color.

Figura 40. Imagen con ajuste de contraste

Una vez separados los colores tendremos todos los rojos o azules de la imagen, esto incluye partes de la imagen que no son señales y que debemos separar. Para ello, se crea un espacio gris personalizado de los colores que más nos interesan, separando los 3 canales de la imagen y mejorando el que nos interese (Figura 41). Esto favorece la eliminación de colores que no queremos y aumenta los deseados para una mejor detección.

5.3.1.2 Búsqueda de la región de interés

Para encontrar las regiones de interés utilizamos el detector de puntos clave MSER [66], este método de detección de áreas de color es muy utilizado en reconocimiento de señales, a la hora de encontrar una forma continua y una relación dimensional.

Esto nos sirve para desechar las regiones que no cumplen ciertos requisitos de área. Las regiones conservadas, serán limpiadas, filtradas y resaltadas utilizando operaciones morfológicas de visión por computador (ver Figura 41).

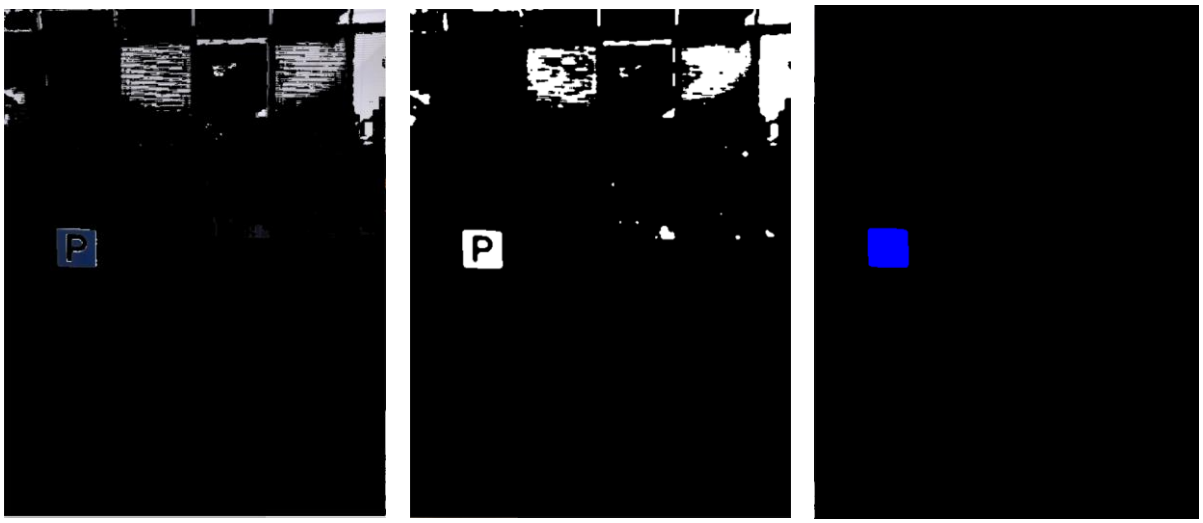

Figura 41. Máscara azul, espacio gris y regiones conservada

5.3.1.3 Búsqueda del contorno

La imagen con las regiones restantes se procesa con una función para detectar contornos y estos se clasifican en función del área que ocupan en la imagen. De esta forma, podremos calcular el espacio que ocupa el contorno en la imagen. Como ya sabemos el tamaño que ocupan las señales en la imagen, aunque este dependerá de la distancia a la que queramos que sea captado**,** podemos establecer unos requisitos de radio para comprobar si el contorno se ajusta a lo que buscamos. De esta forma, cualquier región que no cumpla el radio requerido será desechada.

5.3.1.4 Identificación de la señal

En este punto se parte de las áreas de la imagen que han sido recortadas automáticamente por ser candidatas para contener señales de tráfico. A la hora de identificar una señal de tráfico, utilizaremos los modelos entrenados, que tomarán una imagen como entrada y devolverán la probabilidad de pertenencia a cada clase. Como ya se ha comentado, en el caso de la red convolucional, se utilizará como entrada la imagen original escalada según el tamaño de la red, sin embargo, no será así en el caso de los otros clasificadores, donde utilizaremos el histograma de gradientes orientados (HOG) como método previo de extracción de características (ejemplo en la Figura 42).

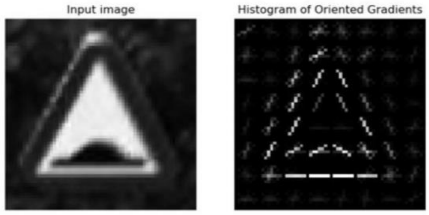

Figura 42. Extracción de características HOG

5.3.2 Prueba de los modelos seleccionados con imágenes del entorno

Una vez recortadas las señales de la imagen completa, se procede a comparar las mejores redes MLP, la mejor red convolucional y las mejores SVM. De esta forma, escogemos los mejores modelos de cada método de aprendizaje después de haberlos entrenado y probado con señales de conducción de carreteras reales para realizar unas nuevas pruebas con las señales del entorno robótico. Como las imágenes de prueba son nuevas, nos interesa volver a comparar varios modelos diferentes. Para ello, se utilizan 40 imágenes de 10 clases diferentes y se consiguen los scores mostrados en la Tabla 6.

	SCORE
MLP AZUL (9)	**0,9375**
MLP ROJA(24)	**0,875**
CNN 2.1	0,825
SVM AZUL	0,8333
SVM ROJA	0,7916

Tabla 5. Resultados de los modelos con señales del entorno robótico

Como podemos comprobar, los scores son inferiores a los mostrados en las Tablas 2, 3 y 4. Esto puede ocurrir por varias causas como la diferencia en el recorte de la imagen, la diferencia de las cámaras utilizadas, las condiciones de luz o el modelado de las señales del entorno robótico, entre otras.

A la vista de estos resultados, se tomó la decisión de re-entrenar los modelos añadiendo imágenes sacadas de la Robobo SmartCity al conjunto original, para analizar si esto mejora los resultados de test. Aunque esto se podía haber realizado al principio, nuestro conjunto de datos real contenía señales de tráfico que aún no han sido modeladas para el entorno robótico y que también podrían utilizarse en entornos de simulación, donde se pueden añadir texturas reales. De esta forma, al añadir estas nuevas imágenes, el conjunto de datos se hace más realista para nuestro entorno de conducción, en cuanto a que se añaden imágenes de señales impresas en 3D y a escala esperando que esto pueda mejorar el rendimiento de los modelos. Por tanto, se han añadido 7 imágenes de la Robobo SmartCity a cada una de las clases del conjunto original, y se han repetido los entrenamientos de los modelos seleccionados anteriormente, con los mismos parámetros de aprendizaje. En la tabla 7 observamos los nuevos resultados.

	SCORE
MLP AZUL (9)	**1**
MLP ROJA (24)	**0,9583**
SVM AZUL	0,9375
CNN 2.1	0,9
SVM ROJA	0,875

Tabla 6. Resultados de los modelos reentrenados con señales del entorno robótico

Como se puede observar en la tabla, ahora los resultados son mucho mejores, y todos los modelos se acercan al nivel que proporcionaban cuando se probaron solo con imágenes reales. Por familias de modelos, podemos decir que los mejores resultados combinados

siempre se obtuvieron con las redes MLP roja y la azul, que serán las que escogeremos finalmente para su implantación en el robot real.

5.3.3 Resultados de las pruebas con el robot en movimiento

Todo lo que hemos realizado hasta aquí se ha hecho para cumplir el objetivo final de validar el funcionamiento de una solución desarrollada como parte de la operativa del robot real en un entorno de una ciudad inteligente. Por lo tanto, necesitamos comprobar que los modelos seleccionados, que serán las dos MLP, detecten las diferentes señales de la Robobo Smart City y con qué precisión lo hacen en diferentes condiciones.

Para ello, se desarrolla un programa de control en tiempo real del vídeo de la cámara del robot para que, mientras el Robobo circula y toma imágenes a través de la cámara, se introduzcan los recortes que se vayan realizando como entrada de la red y nos devuelva la probabilidad de pertenencia a cada clase. Este programa debe establecer la conexión con el robot y su cámara, poner en movimiento el robot, utilizar el programa de recortes de señales explicado anteriormente, cargar el modelo a utilizar y alimentarlo con dichos recortes de señales para obtener las distintas probabilidades de pertenencia a las distintas clases. En dicho programa se realizan varios ajustes para establecer la distancia a la que queremos que se detecte la señal y la probabilidad mínima para que se considere la detección como válida. Una vez visto su funcionamiento, este programa sería el que se añadiría a la librería de detección de señales en tiempo real.

Llegados a este punto, queremos conocer si el programa funciona correctamente y como lo hace a distintas velocidades y frames por segundo, para considerar si sería válido para implementarse en el Robobo para su circulación en la Smart City. Es objeto de interés saber si el robot es capaz de clasificar las señales a distintas velocidades, que serán las establecidas como velocidades máximas en los diferentes tramos de la SmartCity y como influye el número de frames por segundo en cada caso. En la Tabla 7 podemos observar los resultados obtenidos, en la Figura 43 algunos ejemplos de estas pruebas y, a continuación, los detalles de las pruebas:

-Se utiliza la familia de modelos MLP.
-Existe un recorrido con 10 señales a detectar.
-Se modifican los parámetros de velocidad y frames por segundo.
-Se considera que cada señal correctamente detectada será un acierto y cada señal mal identificada un falso acierto.
-Se utilizan condiciones de buena visibilidad de las señales.

MLP				
	Velocidad	**Frames/s**	**Nº de aciertos**	**Nº de falsos aciertos**
Prueba 1	20	10	10	2
Prueba 2	20	5	10	1
Prueba 3	20	1	9	1
Prueba 4	30	10	10	2
Prueba 5	30	5	10	1
Prueba 6	30	1	8	0
Prueba 7	50	10	10	1
Prueba 8	50	5	9	1
Prueba 9	50	1	5	0

Tabla 7. Resultados en tiempo real

Como podemos ver, los resultados obtenidos cumplen con lo esperado. Cuanto más frames por segundo procese el Robobo a lo largo de la prueba, más se asegura la detección de todas las señales y también aumenta el número de falsos aciertos. Esto se debe a que el robot está ejecutando el algoritmo muchas veces y aumenta la probabilidad de error, lo que podría volverse a nuestro favor con la implementación de un software que solo considere una detección como válida después de que se produzca varias veces, como sugeriremos más adelante como trabajo futuro. Cuando procesamos menos frames por segundo, el algoritmo no recibe imágenes suficientes para detectar cada señal, lo que hace que disminuya el número de detecciones a medida que aumenta la velocidad del robot.

Para poder visualizar algunos casos de acierto y error en las pruebas realizadas, la imagen obtenida por streaming desde el Smartphone, se pasa por la librería y muestra la señal identificada en la pantalla del ordenador, a la izquierda de esta, extrayendo una imagen del conjunto de entrenamiento de la clase predicha, como se muestra en la Figura 43.

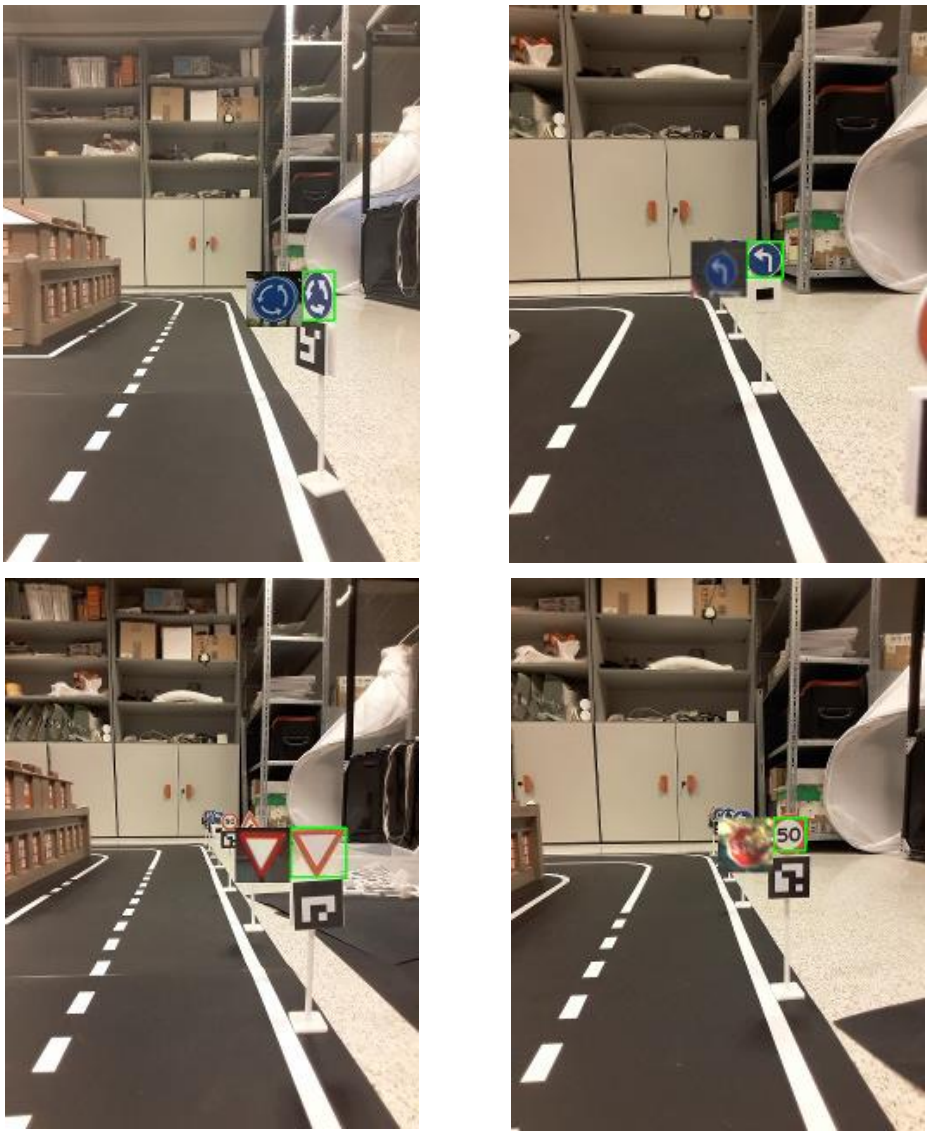

Figura 43. Ejemplos de acierto y error en detección en tiempo real

En cuanto a los resultados obtenidos, hemos conseguido desarrollar una librería que cumple el objetivo principal de detectar todas las señales probadas en el entorno robótico (en distintas posiciones y orientaciones). Su principal desventaja es la incorrecta clasificación de

alguna señal en determinadas ocasiones, lo que podría generar un incorrecto comportamiento del robot que perjudicaría su uso en un entorno de conducción autónoma sin ningún tipo de intervención humana. Este resultado se podría utilizar de forma sencilla para que otros desarrollen programas de control con el Robobo, como el comentado anteriormente o alguna otra idea como bajar la velocidad del robot o aumentar los frames si la detección de la señal varía, aprovechando directamente una nueva capacidad del mismo para identificar señales en tiempo real.

6. Discusión

El presente TFG se centra en el desarrollo de una librería de detección de señales de tráfico en tiempo real en un entorno de pruebas de robótica autónoma. Para ello, se realizan una serie de experimentos sobre diferentes algoritmos de visión por computador. Una vez desarrollada, el robot será capaz de identificar y reconocer señales de tráfico mientras funciona como un vehículo autónomo, y se utilizará en el GII para investigación en el ámbito de la conducción autónoma.

Se tuvo que realizar una serie de investigaciones sobre cada algoritmo, ya que era objeto de interés conseguir un modelo de detección que fuese lo suficiente preciso y rápido para que pudiese realmente ser utilizado en tareas de investigación. Todos los datos y conclusiones recogidos durante las pruebas tienen el objetivo de ayudar a continuar distintos estudios en esta área, además de crear una documentación técnica fiable sobre su funcionamiento en un entorno real.

Durante la realización de este trabajo he conseguido aprender a utilizar Python y varias librerías avanzadas como TensorFlow, Keras y OpenCV. Además, conseguí iniciarme en la comprensión de muchos conceptos y técnicas que antes eran ambiguos para mí, como el Machine Learning, los distintos tipos de aprendizaje, el funcionamiento de una red neuronal, que es el Deep Learning y cómo se utiliza en visión artificial, etc. También manejé por primera vez un robot, siendo capaz de programarlo para realizar diversas tareas, lo que supuso para mi un gran acercamiento a este mundo, ya que en realidad considero que no hay mejor manera para entender algo que probándolo uno mismo.

En función de los resultados finales obtenidos se han pensado distintas maneras para seguir mejorando la librería creada. En el marco de este TFG, se escoge la red que se añadirá a la librería de detección después de una serie de pruebas con distintas imágenes. Esta red es capaz de detectar señales y proporciona unos resultados que aún pueden ser mejorados. Como trabajo futuro se podrían realizar una serie de tareas para afinar mejor el funcionamiento de esta librería. A continuación, se comentan distintas posibilidades de trabajo futuro.

Como comentamos anteriormente, en nuestra librería se procesa el vídeo de la cámara del robot en tiempo real, lo que supone porcesar varios frames de video por segundo, lo que hace que el robot en muy poco tiempo pueda ver la misma señal varias veces. Esto, que podría llevar a un aumento de errores, podría aprovecharse en nuestro favor creando un programa que, para decidir si la detección es correcta, tenga en consideración las evidencias de varias detecciones consecutivas. De esta forma, después de predecir dos, tres, o las veces que se considere necesaria la misma clase en función de varios frames, el programa nos retornaría la clase final asignada a la señal con un mayor grado de confianza.

Otra forma de intentar mejorar la librería sería reajustando el conjunto de datos de entrenamiento de los modelos con más imágenes, particularmente creando un conjunto de datos más balanceado.

Por otro lado, en el caso de que en el futuro se volviese necesario aumentar el conjunto de datos o de clases, sería interesante volver a trabajar con las CNN, debido a su alta eficiencia en el procesamiento de imágenes, sobre todo cuando el volumen de datos es mayor.

Estas recomendaciones se basan en mi experiencia obtenida a la hora de realizar y analizar las todas las pruebas realizadas y comentarlas con mis tutores.

En cuanto a Robobo, estoy muy agradecido de poder haberme acercado tanto a utilizar personalmente un robot. Considero que esta librería formará parte de muchas otras con las que se realizarán estudios de investigación autónoma muy interesantes, incluso podría llegar el día en que esta librería no fuese necesaria gracias a la aplicación de herramientas de localización y estudio de comportamientos colaborativos en una ciudad inteligente, por lo que estoy muy agradecido de haber trabajado y aprendido en este proyecto.

7. Conclusiones

En este capítulo se discutirán las conclusiones globales extraídas en relación a los objetivos que se plantearon al principio del proyecto.

Recordemos que el objetivo principal de este proyecto es el desarrollo de módulos de visión por computador en lenguaje Python para la detección de objetos en un entorno de pruebas de conducción autónoma. Para ello, se había dividido en los siguientes objetivos secundarios:

1. Analizar el estado actual del entorno de pruebas de conducción autónoma que posee el Grupo Integrado de Ingeniería de la UDC, y establecer qué objetos se deberán detectar.

2. Analizar el estado del arte en reconocimiento de objetos mediante visión por computador, y seleccionar las técnicas a implementar en Python.

3. Conseguir un conjunto de imágenes reales (benchmark) adecuado para comparar las técnicas de detección de objetos.

4. Realizar pruebas de comparación de las técnicas en el conjunto de imágenes reales.

5. Conocer el robot móvil Robobo a nivel de características hardware, software y programación, así como el entorno de conducción autónoma que usaremos para la validación.

6. Desarrollar una metodología de toma de imágenes con el robot en el entorno.

7. Validar las técnicas de visión en las imágenes finales, primero en el computador, y finalmente en la operación en tiempo real del robot.

8. Caracterizar el algoritmo de detección de objetos en tiempo real en el robot y documentar las conclusiones del análisis técnico.

Los primeros objetivos (1 y 2) se realizaron, en colaboración con mis tutores, analizando cuales eran las necesidades actuales del GII y cuales eran los algoritmos más actualizados y recomendados para la detección de señales. Después, definimos una metodología adecuada de pruebas para el análisis de las distintas técnicas de detección para, a continuación, aplicarlas en el entorno real.

El tercer subobjetivo presentó mayor dificultad. Para conseguir reunir el conjunto de imágenes fue necesario realizar una búsqueda, establecer cuales serían las señales más adecuadas para el entorno robótico y a continuación intentar reunirlas todas. Algunas señales, como ciertas de velocidad, no se consiguieron, y en el caso de muchas otras hizo falta utilizar dos conjuntos de datos diferentes para conseguir un número adecuado y razonablemente balanceado de datos.

En cuanto a las pruebas de comparación de los distintos modelos, previamente escogidos, se me facilitaron cuadernos de Jupyter de asignaturas de Aprendizaje Máquina y Visión Artificial de la EPS en los que se explicaba los pasos clave para entrenar los distintos modelos, como compararlos entre ellos y como, finalmente, aplicarlos. En el caso de las redes CNN tuve que investigar por mi cuenta como realizar la validación cruzada y como guardarlos, ya que utiliza librerías diferentes a los otros modelos. Se establecieron una serie de parámetros de comparación entre todos los modelos y seleccioné los mejores de cada modelo para realizar las pruebas finales.

Sobre el quinto subobjetivo, el de conocer el robot Robobo y su funcionamiento tampoco fue complicado, ya que existe información de calidad acerca del funcionamiento de su

hardware y de su software y de como utilizarlo. Enlazando con el siguiente subobjetivo, se utilizaron los manuales existentes de Robobo para aprender a utilizarlo, activar su cámara y sus funciones de movimiento. Una vez aprendido a utilizar la cámara, aún tenía que realizar un programa para el recorte de las señales de cada color. Para ello, se encontraron trabajos que se enfrentaban a problemas similares y se conseguí automatizar exitosamente el recorte de las imágenes inspirándome en las técnicas utilizadas en dichos trabajos. Para ello, fue necesario ir ajustando manualmente los distintos filtros de pre-procesado de imagen y comprobar que las distintas máscaras de colores, detección de contornos y resto de técnicas de visión por computador utilizadas funcionaban correctamente para que el recorte se realizase adecuadamente. Una vez conseguido, desarrollé un programa de control del robot para la captura en tiempo real de diferentes imágenes presentes en un entorno real de ciudad inteligente para a continuación utilizarlas como conjunto de datos de test para comparar los modelos ya entrenados.

En cuanto al séptimo objetivo, para realizar la validación de los distintos modelos con las imágenes finales, se introdujeron los recortes de las señales de la ciudad de Robobo Smart City como conjuntos de prueba y se analizó cual funcionaba mejor. Para la comparación, se utilizó el método del score. Una vez obtenidos los resultados, decidimos probar a reentrenar los modelos con dichas señales, las del entorno robótico, para analizar si esto hacía mejorar el rendimiento los modelos. A la vista de que los resultados mejoraban, finalmente escogí los modelos reentrenados que mejores resultados habían proporcionado repetidamente, las MLP, para comprobar su funcionamiento en la ciudad.

Finalmente llegamos al último apartado, en el que había que validar el algoritmo elegido estudiando su comportamiento en tiempo real con el robot en movimiento. Este sin duda fue el apartado más difícil, ya que se tuvo que desarrollar un programa que realizase los recortes de las señales a través de la cámara del Robobo, mientras este se encontraba en movimiento, y se introdujesen en las redes MLP como entrada para que devolviesen la probabilidad de pertenencia a cada clase. Una vez llegados a este punto, lo más importante era que la librería diseñada realmente consiguiera detectar las señales y devolvernos las predicciones de cada clase en tiempo real. En este programa, hubo que modificar ciertos parámetros como la distancia a la que queríamos que detectase la imagen o la probabilidad mínima necesaria para considerar una predicción como válida. Una vez conseguido, se estudiaron los resultados obtenidos a distintas velocidad y distintos frames por segundo para conocer su funcionamiento bajo distintas condiciones. Ya con los resultados delante pude elaborar una serie de conclusiones para reflexionar sobre su futura implementación como librería del robot.

En conclusión, después de haber seleccionado el algoritmo que mejor funcionaba y haber conseguido que detectase las señales a través de la cámara de Robobo en tiempo real, y habiendo documentado todo el proceso y las pruebas realizadas para conseguirlo, se puede considerar que se han cumplido con los objetivos presentados en el presente Trabajo de Fin de Grado.

8. Referencias

[1] Bravo Sánchez, Flor Ángela, Forero Guzmán, Alejandro, “La robótica como un recurso para facilitar el aprendizaje y desarrollo de competencias generales”, Universidad de Salamanca,2012. [Online] Recuperado de:

https://revistas.usal.es/index.php/eks/article/view/9002/9247

[2] Edacom, “¿Qué es mindstorms ev3 de lego® education?” [Online]. Accesible en:

https://blog.edacom.mx/que-es-lego-mindstorms-education-ev3

[3] Thymio Wiki, “Thymio II” [Online]. Accesible en:

http://wiki.thymio.org/en:thymio.

[4] Makeblocks, “Robot Mbot.” [Online]. Accesible en:

https://www.programoergosum.com/cursos-online/robotica-educativa/249-robotica-educativa-con-mbot/que-es-mbot

[5] Robotis e-Manual, “Turtlebot3” [Online]. Accesible en:

http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

[6] E-puck, “e-puck education robot” [Online]. Accesible en:

http://www.e-puck.org/

[7] KTEAM, “Khepera-IV” [Online]. Accesible en:

https://www.k-team.com/khepera-iv

[8] Alive Robots, “NAO, los robots del futuro ya son una realidad” [Online]. Accesible en:

https://aliverobots.com/nao/

[9] Kickstarter, “Romo, The smartphone robot for everyone” [Online]. Accesible en:

https://www.kickstarter.com/projects/peterseid/romo-the-smartphone-robot-for-everyone?lang=es

[10] Shape Robotics, "This is Fable Connect" [Online]. Accesible en:

https://www.shaperobotics.com/connect/

[11] “WHELLPHONE.” [Online]. Accesible en:

http://www.wellphone.com

[12] F. Bellas et al., “Robobo: The Next Generation of Educational Robot,” Ferrol, A Coruña, Spain, 2015.

[13] ROBOBO, “TheRoboboproject.” [Online]. Accesible en:

https://therobobоproject.com/robobo-smartphone-robot/

[14] MINT, “Presentación de la base Robobo” [Online]. Accesible en:

https://education.therobobоproject.com/presentacion-de-robobo/presentacion-de-la-base-robobo

[15] MINT, “Robobo programming wiki” [Online]. Accesible en:

https://github.com/mintforpeople/robobo-programming/wiki

[16] S. Singh, “Critical reasons for crashes investigated in the national motor vehicle crash causation survey,"Washington, DC, USA, Tech. Rep. DOT HS 812 115, 2015.

[17] T. J. Crayton and B. M. Meier, ``Autonomous vehicles: Developing a public health research agenda to frame the future of transportation policy,'' J. Transp. Health, vol. 6, pp. 245252, Sep. 2017.

[18] W. D. Montgomery, R. Mudge, E. L. Groshen, S. Helper, J. P. MacDufe, and C. Carson, “America's workforce and the self-driving future: Realizing productivity gains and spurring economic growth,'' Securing America's Future Energy, Washington, DC, USA, Tech. Rep., 2018.

[19] Hipertextual, “El origen del vehículo autónomo” [Online]. Accesible en:

https://hipertextual.com/2020/08/origen-historia-vehiculo-autonomo

[20] Jan Stuart, “Programme For a European Trafc System With Highest Eficiency and Unprecedented Safety” [Online]. Accesible en:

https://researchportal.port.ac.uk/portal/en/projects/european-eureka-project--programme-for-a-european-traffic-of-highest-efficiency-and-unprecedented-safety-prometheus(7258e5d4-51ec-4114-84b1-f107e2dd9cb5).html

[21] B. Ulmer, “VITA II-active collision avoidance in real trafc,'' in Proc. Intell. Vehicles Symp., 1994, pp. 16.

[22] M. Buehler, K. Iagnemma, and S. Singh, Eds., “The 2005 DARPA Grand Challenge: The Great Robot Race, vol. 36.” Berlin, Germany: Springer, 2007.

[23] M. Buehler, K. Iagnemma, and S. Singh, Eds., “The DARPA Urban Challenge: Autonomous Vehicles in City Trafc,” vol. 56. Berlin, Germany: Springer, 2009.

[24] Waymo LLC, “Technology,” [Online]. Accesible en:

https://waymo.com/tech/

[25] S. International, “Levels of driving automation.” Patente J3016, 2016.

[26] S. Hecker, D. Dai, and L. Van Gool, ``End-to-end learning of driving models with surround-view cameras and route planners,'' in Proc. Eur. Conf. Comput. Vis. (ECCV), 2018, pp. 435453.

[27] D. Lavrinc. “This is How Bad Self-Driving Cars Suck in Rain”. [Online]. Accesible en:

https://jalopnik.com/this-is-how-badself-driving-cars-suck-in-the-rain-1666268433

[28] Ekim Yurtsever, Jacob Lambert, Alexander Carballo And Kazuya Takeda“ A Survey of Autonomous Driving: Common Practices and Emerging Technologies”

[29] W. D. Montgomery, R. Mudge, E. L. Groshen, S. Helper, J. P. MacDufe, and C. Carson, “America's workforce and the self-driving future: Realizing productivity gains and spurring economic growth,'' Securing America's Future Energy, Washington, DC, USA, Tech. Rep., 2018.

[30] DESA, UN, “World population prospects the 2017 revision key ndings and advance tables,'' in Proc. World Popul. Prospect., 2017, pp. 146.

[31] Deloitte, “Deloitte Global Automotive Consumer Study Advanced Vehicle Technologies and Multimodal Transportation” Global Focus Countries, 2019 [Online]. Accesible en:

https://www2.deloitte.com/content/dam/Deloitte/us/Documents/manufacturing/us-global-automotive-consumer-study-2019.pdf

[32] Federatione Internationale de l'Automobile (FiA) Region 1, “The Automotive Digital Transformation and the Economic Impacts of Existing Data Access Models.” [Online]. Accesible en:

https://www.aregion1.com/wp-content/uploads/2019/03/The-Automotive-Digital-Transformation_Full-study.pdf

[33] R. McAllister, Y. Gal, A. Kendall, M. van der Wilk, A. Shah, R. Cipolla, and A. Weller, “Concrete problems for autonomous vehicle safety: Advantages of Bayesian deep learning,'' in Proc. 26th Int. Joint Conf. Artif. Intell., Aug. 2017, pp. 19.

[34] S. Kuutti, S. Fallah, K. Katsaros, M. Dianati, F. Mccullough, and A. Mouzakitis, “A survey of the state-of-the-art localization techniques and their potentials for autonomous vehicle applications,'' IEEE Internet Things J., vol. 5, no. 2, pp. 829846, Apr. 2018.

[35] Wikipedia, “Localización y modelado simultáneos” [Online]. Accesible en:

https://es.wikipedia.org/wiki/Localizaci%C3%B3n_y_modelado_simult%C3%A1neos

[36] Rodríguez Torres Ricardo Miguel, “Estado Del Arte Sobre Vision Artificial” [Online]. Accesible en:

https://es.scribd.com/document/379793433/Estado-Del-Arte-Sobre-Vision-Artificial1

[37] OpenCV, “About OpenCV.” [Online]. Accesible en:

https://opencv.org/about

[38] “Lista de bibliotecas populares de visión artificial en 2019” [Online]. Accesible en:

https://svitla.com/blog/overview-of-modern-computer-vision-tools

[39] Chunsheng Liu 1, Shuang Li1, Faliang Chang1, And Yinhai Wang 2, “Machine Vision Based Traffic Sign Detection Methods: Review, Analyses and Perspectives”

[40] H. Gómez-Moreno, S. Maldonado-Bascón, P. Gil-Jiménez, and S. Lafuente-Arroyo, “Goal evaluation of segmentation algorithms for trafc sign recognition,'' IEEE Trans. Intell. Transp. Syst., vol. 11, no. 4, pp. 917930, Dec. 2010.

[41] M.-Y. Fu and Y.-S. Huang, “A survey of trafc sign recognition,'' in Proc. ICWAPR, Qingdao, China, Jul. 2010, pp. 119124.

[42] Ana Umaquinga, Luis Suárez, Omar Oña, “Machine learning: Importance, advances, techniques and applications” Pasto, 19-20 de abril del 2018. (Universidad Técnica del Norte / Ibarra-Ecuador). [Online]. Accesible en:

https://www.researchgate.net/publication/325860155_Machine_learning_Importance_advances_techniques_and_applications

[43] Research Gate, “Introducción al Machine Learning con BigML” [Online]. Accesible en:

https://www.researchgate.net/publication/338517309_Introduccion_al_Machine_Learning_con_BigML

[44] Telefónica, “Tipos de aprendizaje en Machine Learning: supervisado y no supervisado” [Online]. Accesible en:

https://empresas.blogthinkbig.com/que-algoritmo-elegir-en-ml-aprendizaje/

[45] Fran Ramírez “Historia de la IA” 20 julio, 2018 [Online]. Accesible en:

https://empresas.blogthinkbig.com/historia-de-la-ia-frank-rosenblatt-y-e/

[46] Aprende Machine Learning, “Breve Historia de las Redes Neuronales Artificiales” [Online]. Accesible en:

https://www.aprendemachinelearning.com/breve-historia-de-las-redes-neuronales-artificiales/

[47] Smart Panel, “Undestanding the activations neural network” [Online]. Accesible en:

https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

[48] “Conceptos Básicos Sobre Redes Neuronales” [Online]. Accesible en:

http://grupo.us.es/gtocoma/pid/pid10/RedesNeuronales.htm

[49] Scikit-Learn,”Neural network models” [Online]. Accesible en:

https://scikit-learn.org/stable/modules/neural_networks_supervised.html

[50] Scikit-Learn, “Support Vector Machines” [Online]. Accesible en:

https://scikit-learn.org/stable/modules/svm.html

[51] Sanket Doshi, “Convolutional Neural Network: Learn And Apply” [Online]. Accesible en:

https://medium.com/@sdoshi579/convolutional-neural-network-learn-and-apply-3dac9acfe2b6

[52] Wikipedia, “Scikit-learn” [Online]. Accesible en:

https://es.wikipedia.org/wiki/Scikit-learn

[53] TensorFlow, “Introducción a TensorFlow” [Online]. Accesible en:

https://www.tensorflow.org/learn

[54] Institut für Neuroinformatik, “The German Traffic Sign Recognition Benchmark” [Online]. Accesible en:

https://benchmark.ini.rub.de/gtsrb_news.html

[55] “BelgiumTS Dataset” [Online]. Accesible en:

https://btsd.ethz.ch/shareddata/

[56] Sanchit Gupta, “Traffic Sign Detection & Recognition” [Online]. Accesible en:

https://medium.com/lifeandtech/traffic-sign-detection-recognition-f6e741543619

[57] Piyush Malhotra, “Traffic sign recognition using deep neural networks” [Online]. Accesible en:

https://towardsdatascience.com/traffic-sign-recognition-using-deep-neural-networks-6abdb51d8b70

[58] Sanket Doshi, “Traffic Sign Detection using Convolutional Neural Network” [Online]. Accesible en:

https://towardsdatascience.com/traffic-sign-detection-using-convolutional-neural-network-660fb32fe90e

[59] Scikit-Image, “Histogram of Oriented Gradients” [Online]. Accesible en:

https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_hog.html

[60] Numpy, “What is NumPy?” Accesible en:
https://numpy.org/doc/stable/user/whatisnumpy.html

[61] Scikit-Learn, “Score” [Online]. Accesible en:
https://runebook.dev/es/docs/scikit_learn/modules/model_evaluation

[62] Scikit-Learn, “K-fold” [Online]. Accesible en:
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html

[63] Scikit-Learn, “SuffleSplit” [Online]. Accesible en:
https://scikitlearn.org/stable/modules/generated/sklearn.model_selection.ShuffleSplit.html

[64] Scikit-Learn, “Model persistence” [Online]. Accesible en:
https://pypi.org/project/joblib/

[65] TensorFlow, “Guardar y cargar modelos” [Online]. Accesible en:
https://www.tensorflow.org/tutorials/keras/save_and_load

[66] Scikit-image, “Blob Detection” [Online]. Accesible en:
https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_blob.html

9. Anexo I: Documentación

Para registrar y utilizar la librería creada se incluye en la documentación de Robobo, en un espacio de Github del GII reservado para proyectos de visión. La documentación incluye lo necesario para implementar el reconocimiento de señales, es decir, los modelos entrenados, el programa a utilizar para la detección en tiempo real y una serie de explicaciones de lo que se va haciendo. Se encuentran disponibles en la carpeta 'traffic_sign_recognition' en el siguiente enlace:

https://github.com/GII/robobo_vision/tree/master/traffic_sign_recognition