UNIVERSIDAD EUROPEA DE MADRID

ESCUELA DE ARQUITECTURA, INGENIERÍA Y DISEÑO

ÁREA INGENIERÍA INDUSTRIAL

MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

Estudio y desarrollo de conductores autónomos en miniatura

Alumno: D. Luis Javier Abuin Martínez

Director: D. Víctor Manuel Padrón Nápoles

SEPTIEMBRE 2022

TÍTULO: Estudio y desarrollo de conductores autónomos en miniatura

AUTOR: Luis Javier Abuin Martínez

DIRECTOR DEL PROYECTO: Víctor Manuel Padrón Nápoles

FECHA: 23 de septiembre de 2022

RESUMEN

El objetivo de este Proyecto Fin de Máster es llevar a cabo un estudio en cuanto al hardware y del software que se requieren para desarrollar conductores autónomos en miniatura.
Se procederá a realizar un análisis previo sobre la plataforma hardware utilizada comentando el porqué de su elección, así como también un análisis de los algoritmos de inteligencia artificial que se utilizan para dar autonomía a vehículos y robots autónomos.

Se mostrará un "roadmap" de conducción autónoma para enseñar los distintos niveles de autonomía que existen y en cual encaja este proyecto. Se desarrollará también el hardware del sistema que se ha utilizado y como se ha construido el conductor autónomo, además de la explicación del funcionamiento de los principales algoritmos implementados.

Por último, se realizarán pruebas en un entorno virtual si es posible, pero sobre todo en campo, para recoger datos e información de los test y obtener conclusiones acerca del funcionamiento del hardware escogido en combinación con el software o algoritmo implementado.

ABSTRACT

The aim of this Master's Thesis is to carry out a study of the hardware and software required to develop miniature autonomous drivers. A preliminary analysis of the hardware platform used will be carried out, commenting on the reasons for its choice, as well as an analysis of the artificial intelligence algorithms used to provide autonomy to autonomous vehicles and robots.

A roadmap of autonomous driving will be shown to demonstrate the different levels of autonomy that exist and where this project fits in. The hardware of the system that has been used and how the autonomous driver has been built will also be developed, as well as an explanation of the functioning of the main algorithms implemented.

Finally, tests will be carried out in a virtual environment, if possible, but above all in the field, to collect data and information from the tests and to obtain conclusions about the functioning of the chosen hardware in combination with the implemented software or algorithm.

Índice

Índice de Figuras

Índice de Tablas

Capítulo 1. INTRODUCCIÓN

Siempre me han motivado las tecnologías y poder hacer todo de una manera autónoma, optima y eficiente. Soy un entusiasta de la robótica, en su día hice el máster de robótica y automática en la UPM, y de los coches en general por eso creo que el mejor TFM que podría haber escogido es este sin duda, ya que combina dos de mis pasiones.

Otro de los factores que me ha motivado a hacerlo es la posibilidad de colaborar e investigar en algo tan importante, que, aunque ya tenga bastante camino avanzado todavía quedan cosas por aportar, solucionar y mejorar.

Por último, hay que destacar la dificultad no solo del proyecto, ya que es algo relativamente novedoso y por tanto es difícil encontrar soluciones adecuadas, sino también la complejidad de hacerlo mientras se está trabajando, mientras realizas otros proyectos profesionales a los que tienes que dedicar tiempo y tienen prioridad, ya que trabajo para una empresa y de vez en cuando estoy viajando por este motivo.

1.1 Planteamiento del problema

Con el desarrollo en la segunda mitad del siglo XX en nuevas tecnologías, como la robótica, la aparición de algoritmos de inteligencia artificial (IA) y Machine Learning, los avances en electrónica con nuevos semiconductores de mayor velocidad y capacidad, nuevos componentes hardware y sobre todo la reducción de costes que estos han tenido a lo largo de los años, se ha hecho posible la integración de estas tecnologías en nuestra vida creando nuevos dispositivos inteligentes como los smartphones, o introduciéndolos en algunos ya conocidos como los vehículos.

Esta introducción de las nuevas tecnologías en los vehículos ha aportado diversas mejoras en el último siglo y además ha abierto la posibilidad de que al igual que ocurre con los robots que operan de forma autónoma, los vehículos que conducimos los seres humanos acaben siendo autónomos, con la diferencia de que un robot en una industria, lo hace en un entorno muy controlado y un vehículo lo haría en un entorno mucho más caótico e impredecible, que el que tienen los robots actualmente en la industria, debido a la cantidad de factores y variables que entran en juego.

Este es el problema al que nos enfrentamos, cómo desarrollar la capacidad de que un vehículo que conducimos en el día a día, acabe siendo conducido por un sistema computacional que se comporta como un conductor autónomo, de la misma forma que nosotros lo hacemos o incluso mejor, a ser posible, es decir respetando las normas de tráfico, sin causar accidentes ya sean contra objetos, o peor aún, poniendo en peligro seres vivos, bajo cualquier situación meteorológica y llegando al destino indicado de forma rápida, segura y eficaz.

1.2 Objetivos del proyecto

El objetivo de este proyecto es contribuir de una pequeña forma a la solución de este problema seleccionando un hardware específico de entre las compañías que dedican parte de sus recursos

al desarrollo de componentes concretos para optimizar la computación de los algoritmos de inteligencia artificial.

Además de seleccionar un hardware específico y dedicado a la IA y al Machine Learning, se deberá seleccionar una plataforma robótica que proporcione la estabilidad, autonomía energética y movimiento necesarios para imitar lo máximo posible la conducción de un vehículo.

La combinación de ambos nos proporciona tanto movimiento como la capacidad de albergar y procesar algoritmos de inteligencia artificial que modifiquen el comportamiento del conductor autónomo en función de lo que capten los sensores.

A la hora de realizar el proyecto se ha dividido en hitos que se pueden desglosar de esta manera:

- Estudio preliminar de las diferentes plataformas hardware
- Compra de material hardware necesario
- Montaje del hardware escogido
- Instalación del sistema operativo
- Puesta a punto de la placa y de los sensores
- Configuración de la comunicación inalámbrica
- Pruebas de algoritmos básicos y conceptos de inteligencia artificial
- Estudio de las diferentes plataformas software de inteligencia artificial
- Recogida de datos e información de campo
- Comparativa de algoritmos propios para la conducción autónoma
- Propuesta de trabajos futuros

1.3 Documentación del proyecto

La documentación del proyecto está formada por las siguientes partes:

- Memoria
- Anexo I. Código de los algoritmos implementados
- Anexo II. Presupuesto

Siendo la estructura de la memoria la siguiente:

- **Estado del arte:**
  Se hará una introducción sobre la inteligencia artificial, el Machine Learning, además del estado en el que se encuentran los desarrollos hardware y software específicos para poder realizar el proyecto.

- **Roadmap de conducción autónoma:**
  En este punto se comentará la trayectoria de conducción autónoma que existe, describiendo los diferentes niveles que hay y donde encaja este proyecto.

- **Hardware del sistema**
  En este apartado se describen los componentes del robot, el montaje y los sensores correspondientes.

- **Desarrollo software del sistema**
  En este capítulo se van a explicar los algoritmos utilizados, como funcionan y porque se han usado.

- **Pruebas en campo**
  Se describirán las pruebas de campo hechas y se mostrará la diferente información recogida por los algoritmos.

- **Conclusiones y trabajos futuros**
  Al final se obtendrán conclusiones basadas en las pruebas y se propondrán trabajos futuros que se pueden complementar a este proyecto para seguir desarrollando la idea.

- **Bibliografía**
  Contiene los enlaces y referencias a las que el proyecto hace alusión.

Finalmente, en el anexo I se analizará y explicará detalladamente el código utilizado y en el anexo II se expondrán los costes que ha tenido este proyecto.

Capítulo 2. ESTADO DEL ARTE

2.1 Introducción

Este capítulo tiene por objeto introducir al lector en los algoritmos de inteligencia artificial, los entornos de prueba, las plataformas robóticas y el hardware dedicado a la IA, de manera que más adelante le resulten familiares algunos conceptos y comprenda el estado actual de estas tecnologías y desarrollos.

2.2 Hardware

En este punto se describirá la plataforma hardware utilizada que albergará tanto el sistema operativo como los algoritmos que moverán el robot y usarán los sensores para comprender el entorno que le rodea y actuar en consecuencia. Además, se describirá también la plataforma robótica que se va a utilizar y que llevará la placa hardware.

Hay que mencionar que existen múltiples tarjetas electrónicas que pueden servir para controlar un robot o ejecutar algoritmos, como pueden ser las siguientes [1]:

- Arduino
- Raspberry PI
- Asus Tinker Board
- LePotato
- La Frite
- Rock64 Media Board
- Rock Pi
- Banana Pi
- Odroid
- UDOO
- PocketBeagle
- Orange Pi plus
- Onion Omega 2 plus
- Nvidia Jetson Nano
- BeagleBone Black
- SparkFun

Pero la mayoría de ellas solo actúan como controlador, no como un ordenador, no tienen un sistema operativo al uso, una de las excepciones es la Raspberry Pi, pero no está especialmente diseñada para el desarrollo de la inteligencia artificial. Hay que tener en cuenta también que la

mayoría no están diseñadas especialmente para probar algoritmos de inteligencia artificial y es probable que se ralenticen, disminuyendo su eficacia en las operaciones.

Aparte de lo mencionado en el anterior párrafo, hay que tener en cuenta que se debe tener un buen soporte por parte de la compañía, una gran comunidad donde haya mucha gente que se dedique a lo mismo y se comparta el conocimiento, tanto de los proyectos o de las soluciones implementadas, como de los errores que se han ido encontrando por el camino, para poder resolverlos o corregirlos cuanto antes y no volver a repetirlos.

Otro punto a tener en cuenta para la elección de la placa es que la compañía nos ofrezca un kit de desarrollo con el que podemos iniciarnos en la programación y seguir la evolución de los algoritmos desde cero. Si además se tiene una plataforma robótica donde se pueda anclar fácilmente es otro punto a favor ya que los algoritmos de movimiento y los anclajes estarán especialmente adaptados a ella.

En base a lo mencionado en este punto se ha decidido escoger el kit de desarrollo que ofrece NVIDIA llamado “Jetson Nano 2GB Developer Kit”. La compañía ofrecía más placas y de mayor potencia como la “Jetson Xavier” o “Jetson AGX”, pero para comenzar a desarrollar, esta es la mejor opción, pues ofrece un sistema operativo en el que se pueden ejecutar algoritmos de inteligencia artificial sin problema, el coste no es tan alto como las otras opciones y además se puede anclar a una plataforma robótica llamada “Jetbot”. En conclusión, tiene todo lo necesario para poder ejecutar el proyecto con garantías.

2.2.1 Jetson Nano 2GB Developer Kit

El kit de desarrollo de NVIDIA Jetson Nano 2GB es finalmente el que hemos seleccionado para realizar el proyecto. El contenido de este kit incluye un módulo Jetson sin especificación de producción (P3448-0003) que va unido a una placa portadora de referencia (P3542-0000). Existen dos revisiones de este kit, uno que incluye el adaptador inalámbrico 802.11ac y cable de carga (945-13541-0000-000) y otro que no incluye ninguna de las dos cosas anteriores (945-13541-0000-000), el cuál es el que finalmente se ha escogido dado que la otra opción no se comercializaba actualmente.

Este kit es compatible con el pack SDK NVIDIA JetPack, que está basado en un sistema operativo Linux y contiene todos los controladores, bibliotecas, API’s de inteligencia artificial y visión por computador, suporta tecnologías en la nube y de orquestación, y además contiene herramientas para desarrolladores, documentación y códigos de prueba. Es necesario una tarjeta micro SD para cargar este sistema operativo.

La placa Jetson Nano está compuesta por los siguientes componentes:

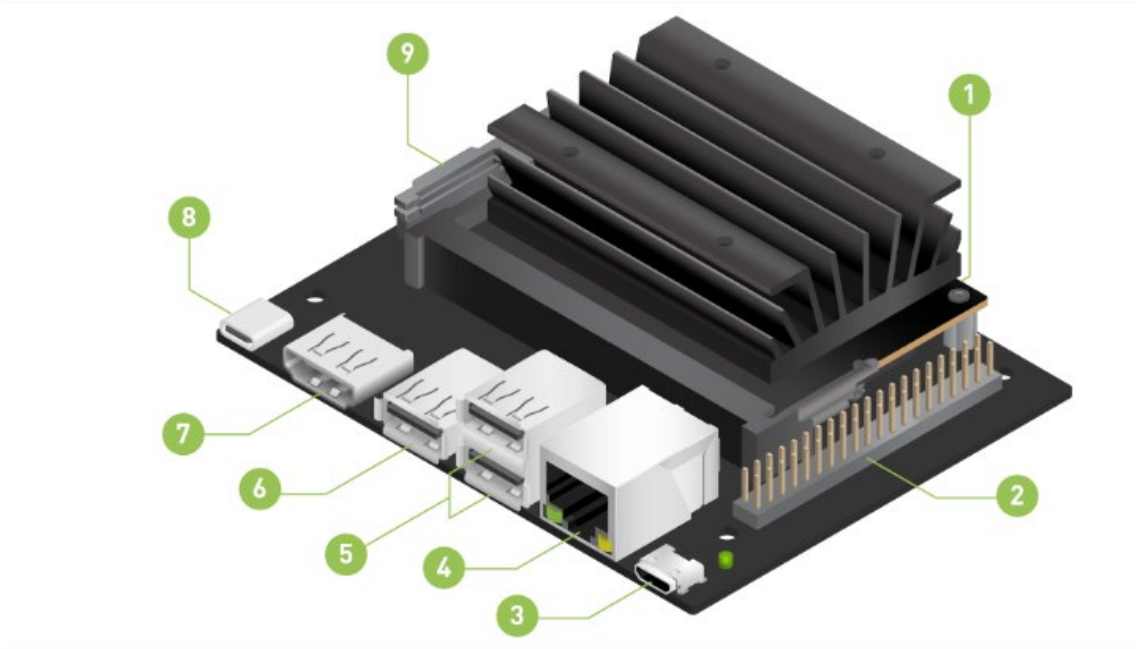


Figura 1 - Descripción de componentes de la placa Jetson Nano [2]

1. Ranura para tarjetas micro SD para el almacenamiento principal
2. Cabezal de expansión de 40 pines
3. Puerto micro-USB para el modo dispositivo
4. Puerto Gigabit Ethernet
5. Puertos USB 2.0 (x2)
6. Puerto USB 3.0 (x1)
7. Puerto de salida HDMI
8. USB-C para la entrada de alimentación de 5V
9. Conector de cámara MIPI CSI-2

Vistas del kit de desarrollo:

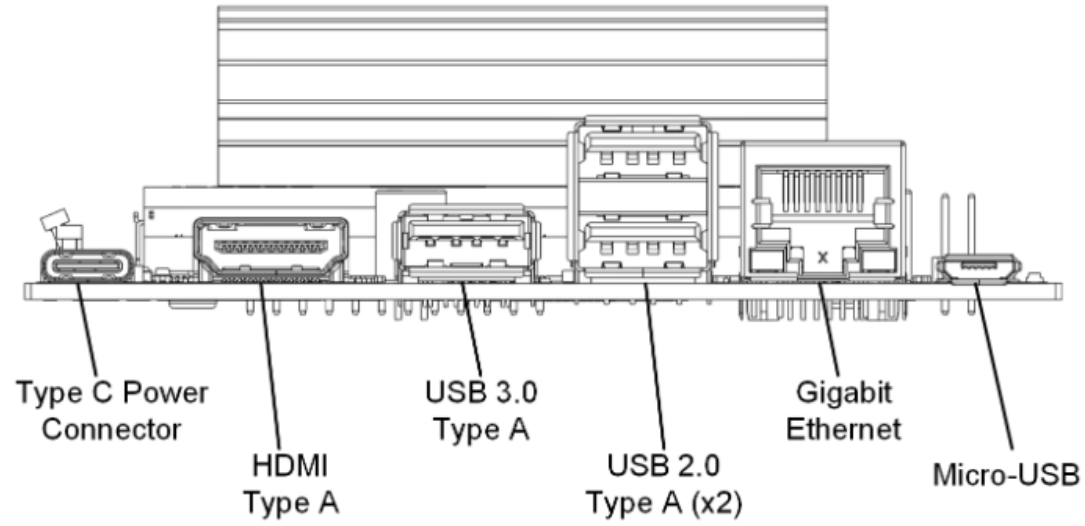


Figura 2 – Vista delantera de la Jetson Nano [3]

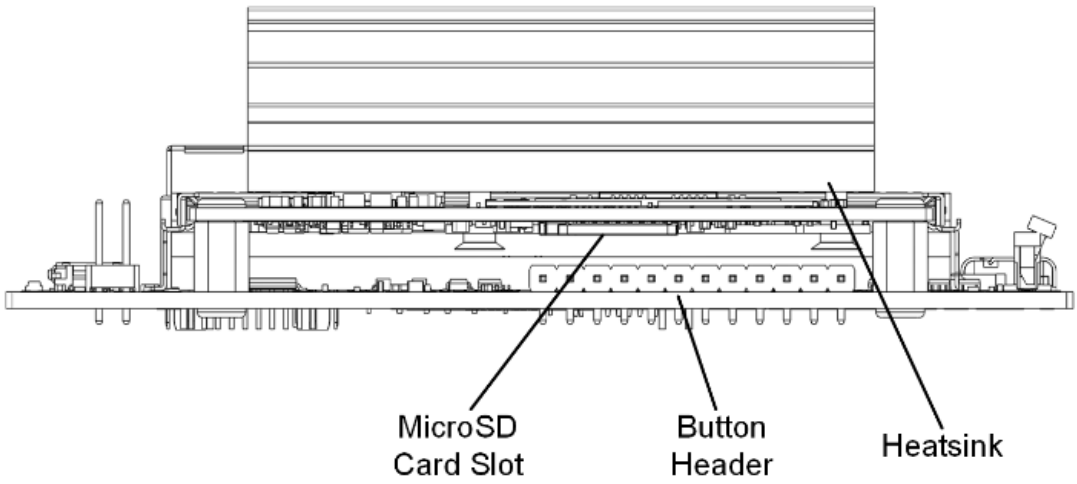


Figura 3 - Vista trasera de la Jetson Nano [3]

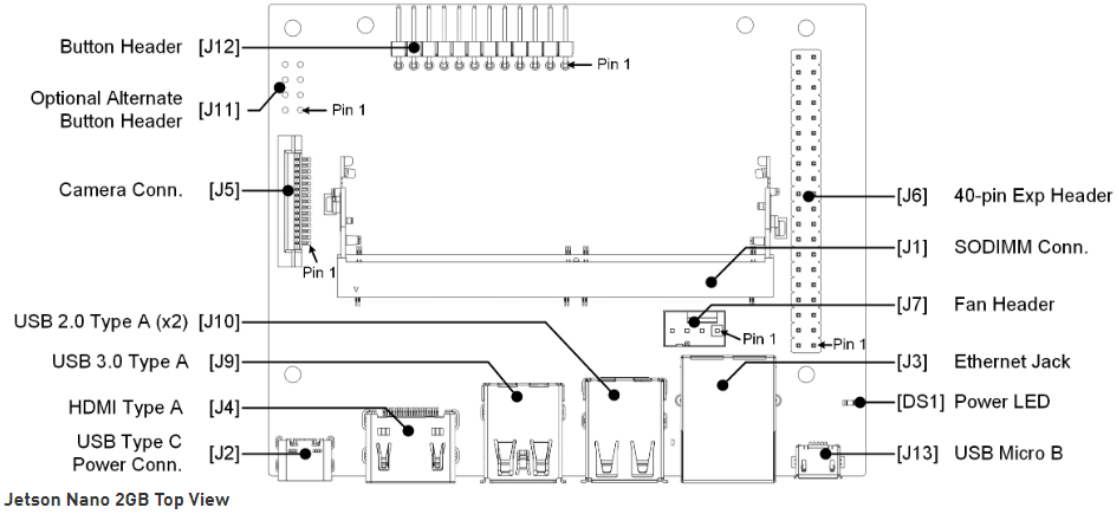


Figura 4 - Vista desde arriba de la Jetson Nano [3]

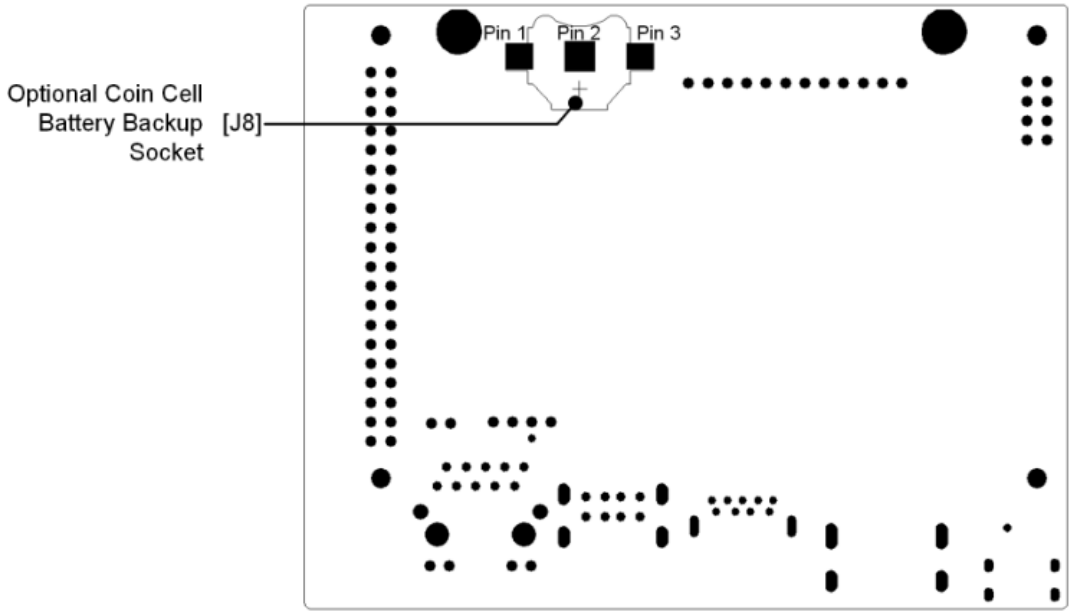


Figura 5 - Vista desde abajo de la Jetson Nano [3]

2.2.2 Jetbot

En cuanto a la plataforma robótica existen diversas marcas que proporcionan los materiales y componentes necesarios para construir por uno mismo un vehículo en miniatura capaz de realizar un desplazamiento entre dos puntos. Además, también pueden incorporar diversos sensores, baterías para tener cierta autonomía y una placa controladora de los motores y sensores. Algunas de estas marcas que proporcionan un robot en el mercado son:

- OSOYOO
- ELEGOO
- Freenove
- KEYESTUDIO
- WAVESHARE
- SPARKFUN
- LEGO

Una vez que hemos elegido la placa controladora y con la que vamos a implementar nuestros algoritmos de Machine Learning debemos seleccionar una plataforma robótica.

Al final se ha seleccionado las plataformas robóticas de la marca Waveshare, ya que las que tienen en el mercado están especialmente diseñadas para portar una placa Jetson nano, una placa de expansión con baterías y la cámara. Entre las plataformas que ofrece se ha decidido seleccionar el jetbot, ya que sobre todo es más económico y es la opción perfecta para iniciarse.

Figura 6 – Jetbot [4]

Hay que destacar que Waveshare es un proveedor de Nvidia, diseñadora de Jetson nano, y tiene una comunidad de usuarios bastante amplia, lo que permite recibir soporte en caso de

necesidad o también por ejemplo ahorrarnos la compra del chasis pudiéndolo imprimir en 3D con los diseños que se comparten en la comunidad y que además podrían permitir llevar más sensores adaptados al diseño del robot.

Figura 7 - Jetbot DIY [5]

Otro aliciente es que este robot tiene ya un diseño digital que puede llegar a ser implementado en un entorno virtual de pruebas como se verá en el apartado 2.3.2 de este capítulo.

2.3 Software

En esta sección del capítulo dos, se va a explicar que es la inteligencia artificial, el Machine Learning y el Deep Learning.

2.3.1 Inteligencia artificial

Básicamente la inteligencia artificial se refiere a los sistemas o máquinas que son capaces de imitar la inteligencia humana para llevar a cabo tareas a través de algoritmos computacionales, y que además pueden mejorar de forma iterativa a partir de la información que van recopilando del entorno y de sus propias acciones. Algunos ejemplos pueden ser [6]:

- "Chatbots" que comprenden más rápido los problemas de los clientes y ofrecen soluciones más eficientes [6].
- Asistentes inteligentes que analizan grandes conjuntos de datos para mejorar la programación [6].

- Motores de recomendación que proporcionan sugerencias dependiendo de los hábitos de consumo de los usuarios [6].

La diferencia de la inteligencia artificial con un ordenador corriente es que no responde de manera lineal y constante, como sería el apagado o encendido del computador presionando un botón, es decir obedeciendo ordenes, sino que es capaz de interpretar datos y situaciones, respondiendo de una forma diferente en cada caso y aprender de cada uno de ellos [6].

2.3.1.1 Machine Learning

El Machine Learning una disciplina del campo de la inteligencia artificial, la cual a través de algoritmos da la capacidad a los computadores de identificar patrones en una cantidad masiva de datos y en base a esto realizar un análisis predictivo para realizar tareas de forma autónoma sin la necesidad de que hayan tenido que ser programados, es decir aprendiendo.

Existen varios tipos de entrenamiento para que una inteligencia artificial aprenda. Además de tener que aprender a través de cálculos, esta también aprende a través de la información y los datos que recopila y que proveen las personas que la están capacitando, y también con la interacción con los usuarios. Los tipos de entrenamiento son los siguientes:

1. **Supervisado**

El aprendizaje supervisado necesita conjuntos de datos etiquetados, es decir, le decimos al modelo qué es lo que queremos que aprenda.

Por poner un ejemplo, supongamos que tenemos una heladería y durante los últimos años hemos estado registrando diariamente datos climatológicos, temperatura, mes, día de la semana, etc., y también hemos hecho lo propio con el número de helados vendidos cada día. En este caso, seguramente nos interesaría entrenar un modelo que, a partir de los datos climatológicos, temperatura, etc. (características del modelo) de un día concreto, nos diga cuántos helados se van a vender (la etiqueta a predecir) [7].

Dependiendo del tipo de etiqueta, dentro del aprendizaje supervisado existen dos tipos de modelos:

- Los modelos de clasificación, que producen como salida una etiqueta discreta, es decir, una etiqueta dentro de un conjunto finito de etiquetas posibles. A su vez, los modelos de clasificación pueden ser binarios si tenemos que predecir entre dos clases o etiquetas (enfermedad o no enfermedad, clasificación de correos electrónicos como “spam” o no

"spam") o multiclase, cuando se tiene que clasificar más de dos clases (clasificación de imágenes de animales, análisis de sentimientos, etc.) [7].

- Los modelos de regresión producen como salida un valor real, como el ejemplo que comentábamos de los helados. [7]

2. Sin supervisión

Por su parte, el aprendizaje no supervisado trabaja con datos que no han sido etiquetados. No tenemos una etiqueta que predecir. Estos algoritmos se usan principalmente en tareas donde es necesario analizar los datos para extraer nuevo conocimiento o agrupar entidades por afinidad [7].

Este tipo de aprendizaje también tiene aplicaciones para reducir dimensionalidad o simplificar conjuntos de datos. En el caso de agrupar datos por afinidad, el algoritmo debe definir una métrica de similitud o distancia que le sirva para comparar los datos entre sí. Como ejemplo de aprendizaje no supervisado tenemos los algoritmos de agrupamiento o clustering, que podrían aplicarse para encontrar clientes con características similares a los que ofrecer determinados productos o destinar una campaña de marketing, descubrimiento de tópicos o detección de anomalías, entre otros [7].

Por otro lado, en ocasiones, algunos conjuntos de datos como los relacionados con información genómica tienen grandes cantidades de características y por varias razones, como, por ejemplo, reducir el tiempo de entrenamiento de los algoritmos, mejorar el rendimiento del modelo o facilitar la representación visual de los datos, necesitamos reducir la dimensionalidad o número de columnas del conjunto de datos. Los algoritmos de reducción de dimensionalidad utilizan técnicas matemáticas y estadísticas para convertir el conjunto de datos original en uno nuevo con menos dimensiones a cambio de perder un poco de información. Ejemplos de algoritmos de reducción de dimensionalidad son PCA, t-SNE o ICA. [7]

3. Semi-supervisado

En ocasiones, es muy complicado disponer de un conjunto de datos completamente etiquetado. Imaginemos que somos los dueños de una empresa de fabricación de productos lácteos y queremos estudiar la imagen de marca de nuestra empresa a través de los comentarios que los usuarios han publicado en redes sociales. La idea es crear un modelo que clasifique cada comentario como positivo, negativo o neutro para, después, hacer el estudio. Lo primero que hacemos es bucear por las redes sociales y recolectar dieciséis mil mensajes donde se menciona a nuestra empresa. El problema ahora es que no tenemos etiqueta en los datos, es decir, no sabemos cuál es el sentimiento de cada comentario. Aquí entra en juego el aprendizaje semi-supervisado. Este tipo de aprendizaje tiene un poco de los dos anteriores. Usando este enfoque, se comienza etiquetando manualmente algunos de los comentarios. Una vez tenemos una pequeña porción de comentarios etiquetados, entrenamos uno o varios algoritmos de aprendizaje supervisado sobre esa pequeña parte de datos etiquetados y

utilizamos los modelos resultantes del entrenamiento para etiquetar el resto de los comentarios. Finalmente, entrenamos un algoritmo de aprendizaje supervisado utilizando como etiquetas las etiquetadas manualmente más las generadas por los modelos anteriores. [7]

4. **Por refuerzo**

Por último, el aprendizaje por refuerzo es un método de aprendizaje automático que se basa en recompensar los comportamientos deseados y penalizar los no deseados. Aplicando este método, un agente es capaz de percibir e interpretar el entorno, ejecutar acciones y aprender a través de prueba y error. Es un aprendizaje que fija objetivos a largo plazo para obtener una recompensa general máxima y lograr una solución óptima. El juego es uno de los campos más utilizados para poner a prueba el aprendizaje por refuerzo. AlphaGo o Pacman son algunos juegos donde se aplica esta técnica. En estos casos, el agente recibe información sobre las reglas del juego y aprende a jugar por sí mismo. Al principio, evidentemente, se comporta de manera aleatoria, pero con el tiempo empieza a aprender movimientos más sofisticados. Este tipo de aprendizaje se aplica también en otras áreas como la robótica, la optimización de recursos o sistemas de control [7].

2.3.1.2 Deep Learning

El Deep Learning o aprendizaje profundo, es un tipo de Machine Learning, el cual hace pasar una gran cantidad de datos por numerosas capas de procesamiento algorítmico para conseguir que un ordenador aprenda por su propia cuenta y acabe realizando tareas similares a las que realiza el ser humano, como el reconocimiento del habla o procesamiento del lenguaje natural y el reconocimiento de imágenes por visión por computador [8].

Los algoritmos de Deep Learning se basan en redes neuronales artificiales estructuradas en capas y conectadas entre sí para transmitir información, input layer (entrada), hidden layer (ocultas) y output layer (salida). Los datos entran por la primera capa, en la que hay varias neuronas artificiales, las cuales se van activando o no dependiendo de los datos que van pasando a través de ellas y produciendo unos datos de salida [8].

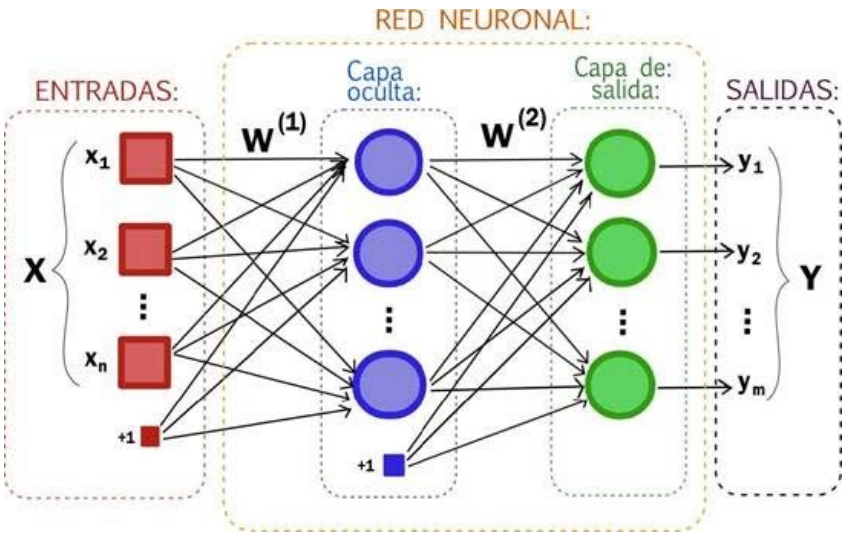


Figura 8 - Red neuronal [9]

Como cada neurona está conectada a otra a través de unos enlaces, el valor de la salida de la neurona anterior es multiplicado por un valor llamado, peso. Estos pesos en los enlaces lo que hacen es incrementar o inhibir el estado de activación de las neuronas adyacentes. De la misma manera a la salida de la neurona existe una función, llamada función de activación, que modifica el valor resultante o impone un límite que no debe sobrepasarse antes de propagar la información a otra neurona [10].

Estos sistemas aprenden y se van formando así mismos actualizando el valor de los pesos de las neuronas tratando de minimizar la función de pérdida (loss) para dar un resultado correcto.

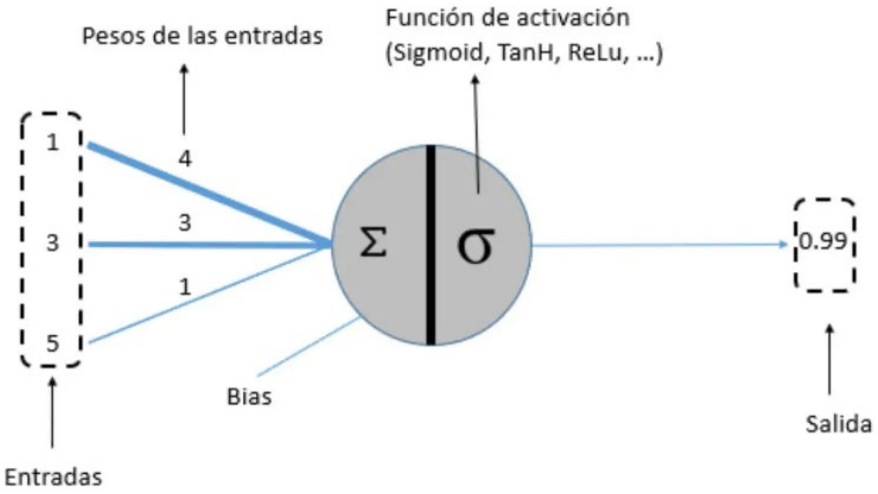


Figura 9 - Esquema de una red neuronal [10]

A continuación, se describen un par de ejemplos de Deep Learning:

- **Procesamiento del lenguaje natural**

Se trata de una rama de la inteligencia artificial que trata de que las computadoras puedan llegar a entender, interpretar y manipular el lenguaje humano

- **Visión por computador**

Es una disciplina del Machine Learning que incluye métodos capaces de adquirir, procesar, analizar y comprender las imágenes del mundo real con el fin de que estás sean tratadas por un computador y pueda realizar un análisis en base a la información numérica o simbólica recibida.

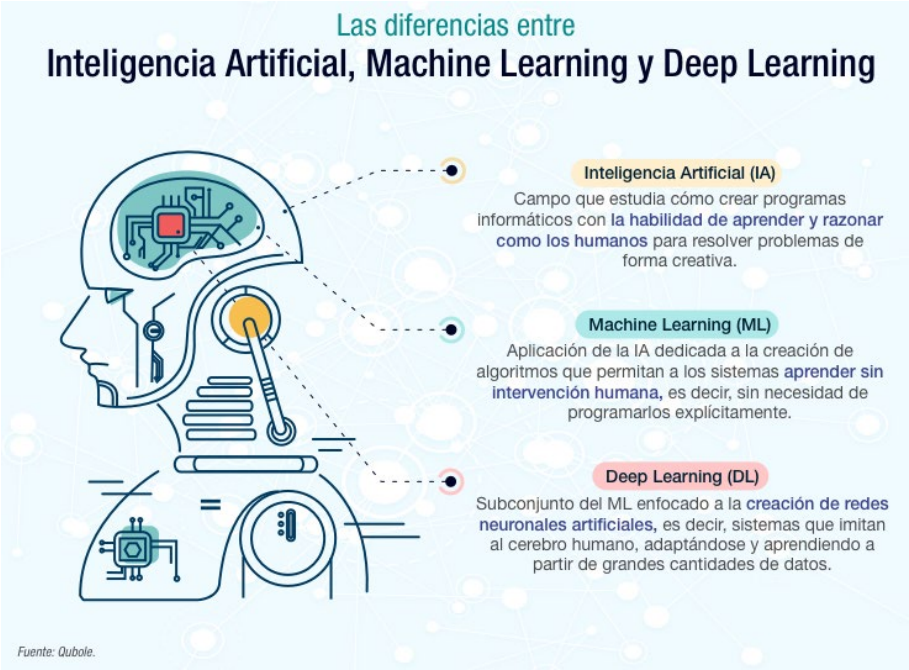


Figura 10 - Diferencias entre IA, ML y DL [8]

2.3.2 Entornos de prueba

En este apartado se describirán algunos entornos de prueban que se han usado en este proyecto y otros que pueden ser muy útiles en el desarrollo en concreto de vehículos autónomos o de la robótica en general.

2.3.2.1 JupyterLab

JupyterLab es una aplicación web con una arquitectura cliente-servidor que permite editar y ejecutar cuadernos u otros tipos de archivo utilizando un sencillo navegador web. La aplicación se puede ejecutar en cualquier equipo tanto en red local como con conexión a internet y es compatible con los principales navegadores web.

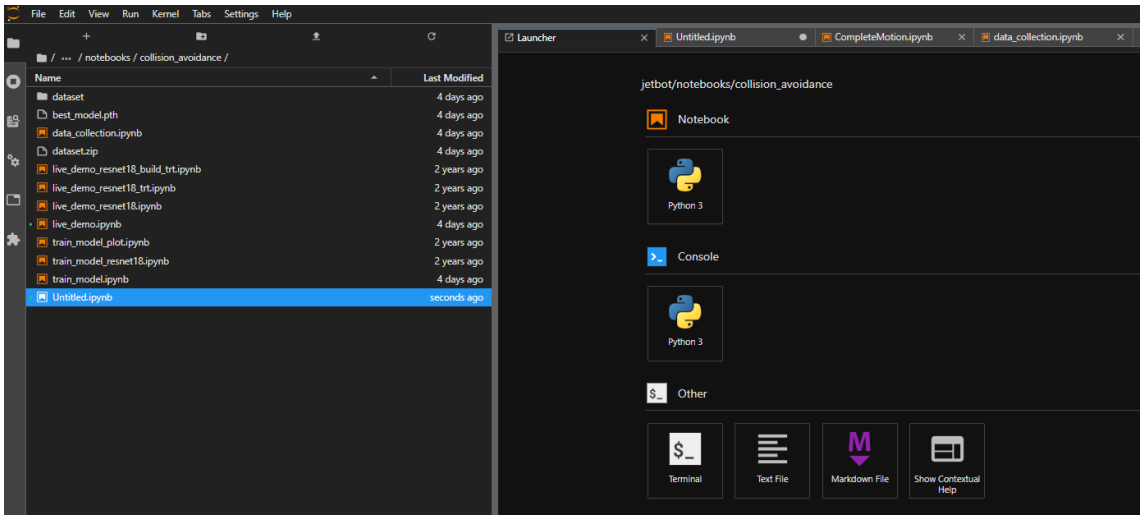


Figura 11 - JupyterLab

Un cuaderno en JupyterLab está formado por una serie de celdas que pueden tener código fuente, textos, ecuaciones, gráficos, controles interactivos y otros elementos, y que tiene la extensión ".ipynb". Una celda que contiene un código se puede ejecutar y lo que se obtiene aparece a continuación de esa celda, como una parte del cuaderno. Esto facilita al usuario el cambio de datos y el análisis en tiempo real, además permite la colaboración entre usuarios para programar [11].

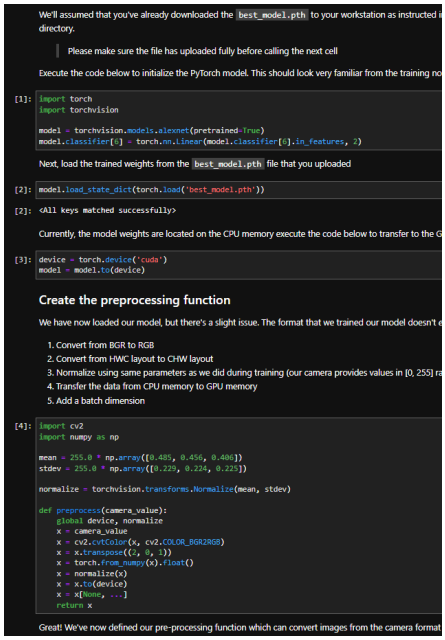


Figura 12 - Ejemplo de un cuaderno con celdas

Una vez que se ha instalado y se inicia el servidor web, ya se puede entrar desde esta dirección http://localhost:8888. Dentro, el entorno contiene un menú en la parte superior desde el que podemos abrir archivos, ejecutarlos, editarlos y parar los núcleos para reestablecer sesión. La parte central se divide en dos, la parte izquierda muestra el árbol de carpetas y en la parte derecha es donde se muestran los diferentes cuadernos y celdas para ser editadas.

2.3.2.2 ROS

ROS (Robot Operating System) es un framework para el desarrollo de software para robots y que tiene la funcionalidad de un sistema operativo de código abierto en un clúster heterogéneo que está mantenido por la OSRF (Open Source Robotics Foundation).

Este sistema operativo proporciona los mismos servicios que se espera de un sistema operativo, incluyendo la abstracción de hardware, el control de dispositivos de bajo nivel, implementación de funciones de uso común, transferencia de mensajes entre procesos y gestión de paquetes. Además, proporciona herramientas y bibliotecas para obtener, construir, escribir y ejecutar código en varios ordenadores.

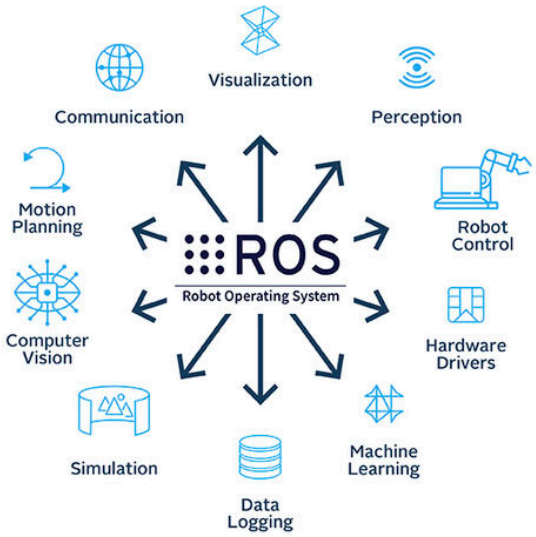


Figura 13 - ROS, Robot Operating System [13]

El objetivo principal de este sistema operativo es apoyar la reutilización de código en la investigación y desarrollo de la robótica. Esto lo hace gracias a que es un marco de trabajo distribuido de procesos o también llamados nodos, con los que pueden agruparse paquetes y pilas de ejecución y soporta un sistema de repositorio de códigos. Además, puede integrarse con otros softwares de desarrollo para robots como son OpenRAVE y Player, es fácil de probar y testear con la herramienta rostest, es escalable para grandes sistemas de ejecución y procesos, e integra librerías como OpenCV para algoritmos de visión por computador [14].

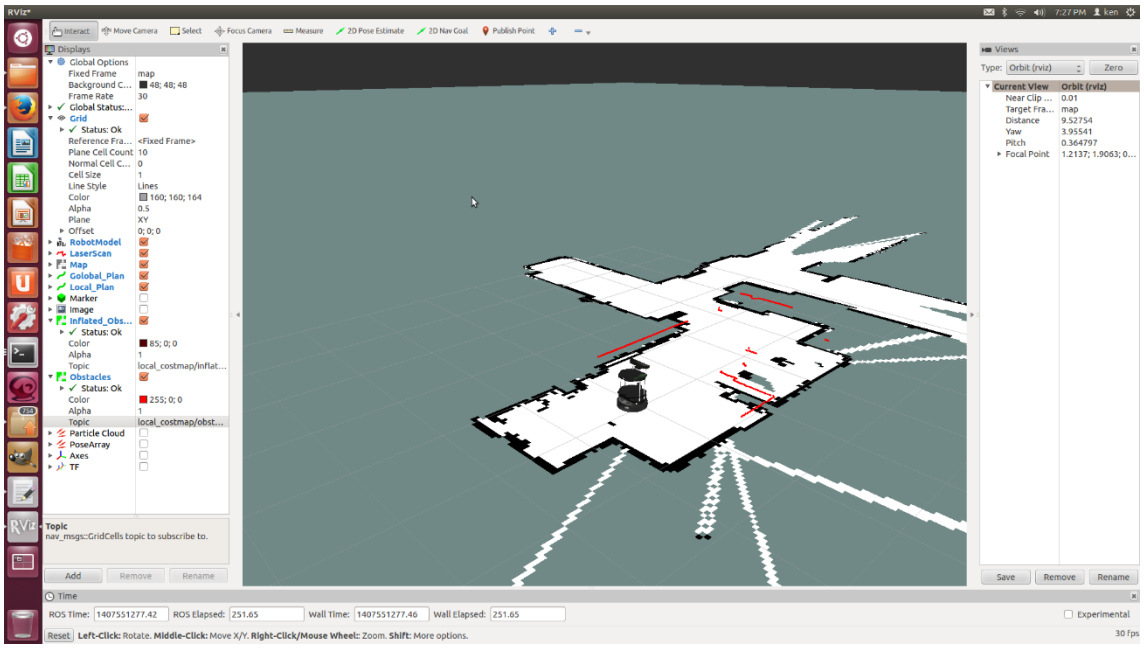


Figura 14 - Entorno de visualización Rviz [15]

Uno de los paquetes más importantes que integra ROS es la herramienta Rviz, que se trata de un software de visualización 2D y 3D, el cual mediante diferentes complementos o plugins es posible visualizar el modelo del robot, las malos y los marcos de referencia para facilitar el diseño y la programación de estos. También permite visualizar y registrar la información de los sensores como pueden ser cámaras, láseres, LIDAR o ultrasonidos, pudiendo crear mapas de la zona que está recorriendo el robot como muestran las figuras siguientes.

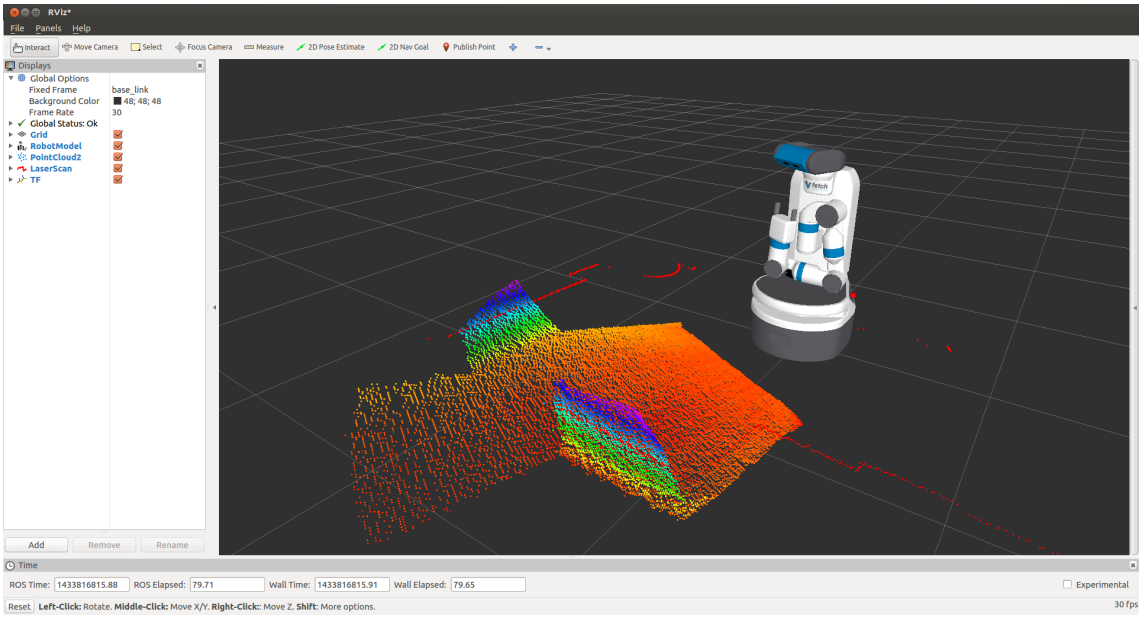


Figura 15 - Visualización de sensores en Rviz [16]

2.3.2.3 Gazebo

El software Gazebo se trata de un simulador de robótica 3D de código abierto, el cual simula es capaz de simular el mundo real ya que incorpora el motor físico “ODE”, además de la librería de renderizado OpenGL y código para simulación de sensores y actuadores [17].

La diferencia con Rviz es que este primero es un visualizar, es decir muestra lo que el robot cree que está ocurriendo en su entorno, mientras que Gazebo lo que muestra es lo que realmente está ocurriendo ya que es un simulador físico real.

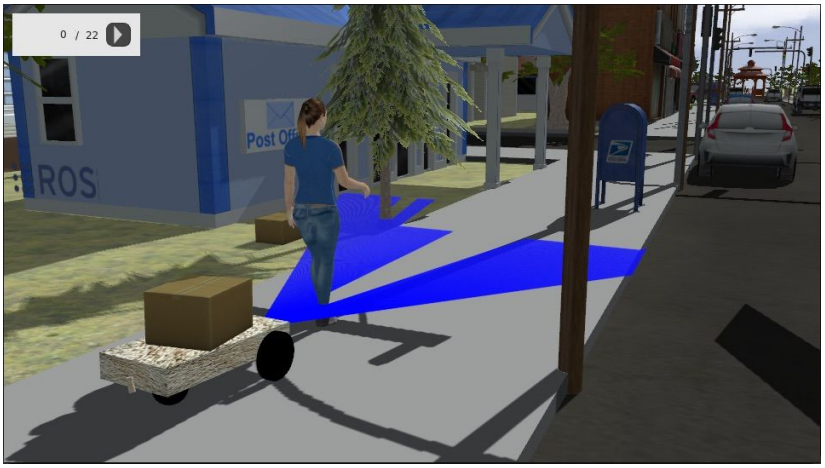


Figura 16 - Software de simulación real Gazebo [18]

Este software puede combinarse con ROS y con Rviz y al final ofrecer un entorno de simulación real, que será útil sobre todo si no podemos tener uno para poder probar el software robótico desarrollado o si se quieren adelantar las pruebas en entorno real se puede simular previamente para reportar fallos que puedan ser catastróficos.

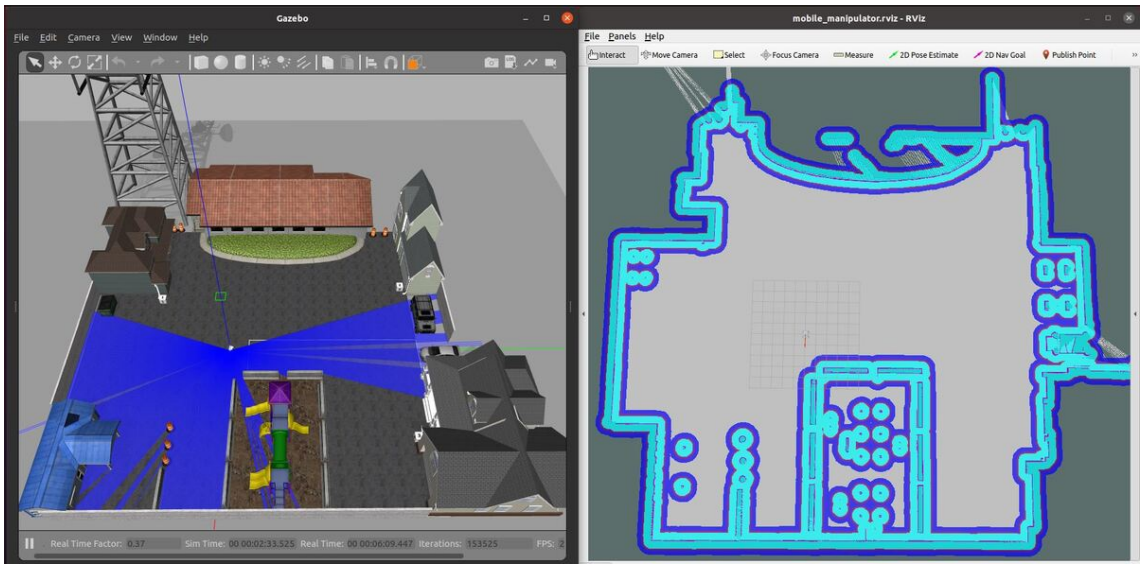


Figura 17 – Combinación de entornos Rviz y Gazebo [19]

Capítulo 3. ROADMAP DE CONDUCCIÓN AUTÓNOMA

3.1 Introducción

En este capítulo se van a abordar los conceptos básicos de la conducción autónoma, los niveles que tiene, como se ha llegado a este punto, cual es el futuro y donde encajaría este proyecto.

3.2 Visión general

En este punto se va a mostrar una visión general del estado actual de las tecnologías de conducción autónoma.

La conducción automatizada se considera una de las tecnologías clave y de los principales avances tecnológicos que influyen y dan forma a nuestra movilidad y calidad de vida futuras. Los principales impulsores para lograr mayores niveles de conducción automatizada son:

- La seguridad: Reducir los accidentes causados por errores humanos.
- Eficiencia y objetivos medioambientales: Aumentar la eficiencia del sistema de transporte y reducir tiempo en el tráfico congestionado. Un tráfico más fluido ayudará a disminuir el consumo de energía y las emisiones de los vehículos.
- Confort: Permitir la libertad del usuario para realizar otras actividades cuando los sistemas automatizados estén activos.
- Inclusión social: Garantizar la movilidad de todos, incluidos los usuarios de edad avanzada y discapacitados.
- Accesibilidad: Facilitar el acceso al centro de las ciudades.

La conducción automatizada debe considerarse, por tanto, un aspecto clave para la política europea de transportes de transporte, capaz de apoyar varios objetivos y retos sociales, como la seguridad vial, la descarbonización, las ciudades inteligentes, la inclusión social, etc. En términos tecnológicos, el avance hacia una conducción altamente automatizada se considera un proceso evolutivo para garantizar que todas las partes implicadas involucrados puedan desarrollarse y evolucionar con el ritmo adecuado. [20]

Todo este proceso comenzó hace años con el desarrollo de los sistemas de ayuda a la conducción como son el ABS (Anti-Lock Braking System o Sistema Antibloqueo de Frenos), el ESP (Elektronisches Stabilitätsprogramm o Programa Electrónico de Estabilidad) y los sistemas avanzados de asistencia al conductor, ADAS (Advance Driving Assitance System) como pueden ser el detector de ángulo muerto, la alerta de tráfico cruzado o el control de crucero adaptativo.

Sin embargo, los gobiernos y las principales empresas fabricantes se enfrentan a retos y lagunas existentes, tanto en cuanto a tecnología como a legislación, que tendrán que abordar durante estos años para dar paso a la implementación del vehículo autónomo.

Se espera que, para la década de 2050, los vehículos sean totalmente autónomos, compartidos y además electrificados. Esa es la visión general y el camino que se espera hacer realidad algún día.

3.3 Conceptos básicos

Esta sección se introducen los términos básicos de los niveles de automatización, los dominios de diseño operativo y los casos de uso.

Actualmente existen definidos 6 niveles de automatización de la conducción, y son los siguientes:

- Nivel 0: Sin automatización de la conducción.
- Nivel 1: Asistencia al conductor.
- Nivel 2: Automatización parcial de la conducción
- Nivel 3: Automatización condicional de la conducción.
- Nivel 4: Alta automatización de la conducción.
- Nivel 5: Automatización total de la conducción.

SAE level	Name	Narrative Definition	Execution of Steering and Acceleration/ Deceleration	*Monitoring* of Driving Environment	Fallback Performance of *Dynamic Driving Task*	System Capability *(Driving Modes)*
***Human driver* monitors the driving environment**						
0	**No Automation**	the full-time performance by the *human driver* of all aspects of the *dynamic driving task*, even when enhanced by warning or intervention systems	Human driver	Human driver	Human driver	n/a
1	**Driver Assistance**	the *driving mode*-specific execution by a driver assistance system of either steering or acceleration/deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	Human driver and system	Human driver	Human driver	Some driving modes
2	**Partial Automation**	the *driving mode*-specific execution by one or more driver assistance systems of both steering and acceleration/ deceleration using information about the driving environment and with the expectation that the *human driver* perform all remaining aspects of the *dynamic driving task*	**System**	Human driver	Human driver	Some driving modes
***Automated driving system* ("system") monitors the driving environment**						
3	**Conditional Automation**	the *driving mode*-specific performance by an *automated driving system* of all aspects of the dynamic driving task with the expectation that the *human driver* will respond appropriately to a *request to intervene*	System	**System**	Human driver	Some driving modes
4	**High Automation**	the *driving mode*-specific performance by an automated driving system of all aspects of the *dynamic driving task*, even if a *human driver* does not respond appropriately to a *request to intervene*	System	System	**System**	Some driving modes
5	**Full Automation**	the full-time performance by an *automated driving system* of all aspects of the *dynamic driving task* under all roadway and environmental conditions that can be managed by a *human driver*	System	System	System	**All driving modes**

Figura 18 - Niveles de conducción autónoma [20]

Los dominios de diseño operativo (ODD, operative design domain) se refieren a los límites funcionales que tiene un sistema para un nivel de automatización específico. Es decir, este

concepto es el que acotaría los distintos niveles de conducción autónoma, estableciendo límites para saber cuándo se pasa al siguiente nivel de automatización [21].

Existe otro concepto a tener en cuenta que son las tareas dinámicas de conducción (DTT, Dynamic Driving Task), que se refieren a las funciones necesarias para controlar un vehículo en una carretera con tráfico, excluyendo eso sí la parte estratégica, es decir, la programación de viajes o destinos [21]. Estas funciones serían las siguientes:

1. El control del movimiento lateral del vehículo mediante la dirección (operativa);
2. Control del movimiento longitudinal del vehículo mediante la aceleración y la desaceleración (operacional);
3. Control del entorno de la conducción mediante la detección, el reconocimiento, la clasificación y la preparación de respuestas a objetos y eventos
4. Preparación de la respuesta (operativa y táctica)
5. Ejecución de la respuesta a objetos y eventos (operativa y táctica);
6. Planificación de maniobras (táctica);
7. Mejora de la visibilidad mediante iluminación, señalización y gestos, etc. (táctica).

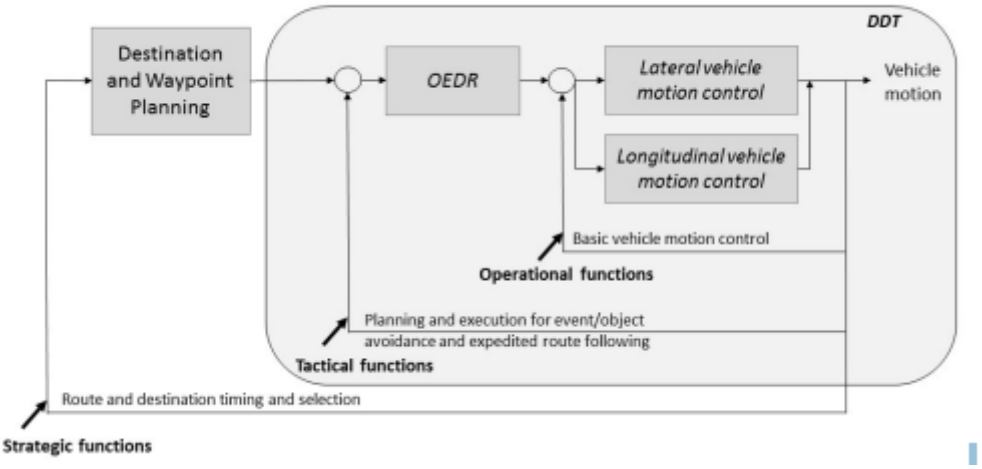


Figura 19 - Vista esquemática de las tareas de conducción [21]

Los puntos 3 y 4 hacen referencia a subtareas de detección y respuesta a objetos y eventos, también llamado OEDR (Object and Event Detection and Response), siendo la parte más compleja y los puntos principales donde encajaría este proyecto [21].

En esta tabla se representa mejor los niveles de conducción autónoma junto con los conceptos que hemos visto anteriormente.

Level	Name	Narrative definition	DDT		DDT fallback	ODD
			Sustained lateral and longitudinal vehicle motion control	OEDR		
Driver performs part or all of the *DDT*						
0	No Driving Automation	The performance by the *driver* of the entire *DDT*, even when enhanced by *active safety systems*.	*Driver*	*Driver*	*Driver*	n/a
1	Driver Assistance	The *sustained* and *ODD*-specific execution by a *driving automation system* of either the *lateral* or the *longitudinal vehicle motion control* subtask of the DDT (but not both simultaneously) with the expectation that the *driver* performs the remainder of the *DDT*.	*Driver* and *System*	*Driver*	*Driver*	Limited
2	Partial Driving Automation	The *sustained* and *ODD*-specific execution by a *driving automation system* of both the *lateral* and *longitudinal vehicle motion control* subtasks of the *DDT* with the expectation that the *driver* completes the *OEDR* subtask and *supervises* the *driving automation system*.	**System**	*Driver*	*Driver*	Limited
ADS ("*System*") performs the entire *DDT* (while engaged)			*System*	**System**	*Fallback-ready user (becomes the driver during fallback)*	Limited
3	Conditional Driving Automation	The *sustained* and *ODD*-specific performance by an *ADS* of the entire DDT with the expectation that the *DDT fallback-ready user* is *receptive* to *ADS*-issued *requests to intervene*, as well as to *DDT performance-relevant system failures* in other *vehicle* systems, and will respond appropriately.				
4	High Driving Automation	The *sustained* and *ODD*-specific performance by an *ADS* of the entire *DDT* and *DDT fallback* without any expectation that a *user* will respond to a *request to intervene*.	*System*	*System*	**System**	Limited
5	Full Driving Automation	The *sustained* and unconditional (i.e., not *ODD*-specific) performance by an *ADS* of the entire *DDT* and *DDT fallback* without any expectation that a *user* will respond to a *request to intervene*.	*System*	*System*	*System*	**Unlimited**

Tabla 1 - Niveles de conducción autónoma de acuerdo al estandar J3016-2018 [21]

A continuación, tenemos un ejemplo de cómo actuaría el sistema para sistema de nivel 4 para hacerse una idea general del funcionamiento de un sistema de conducción autónoma:

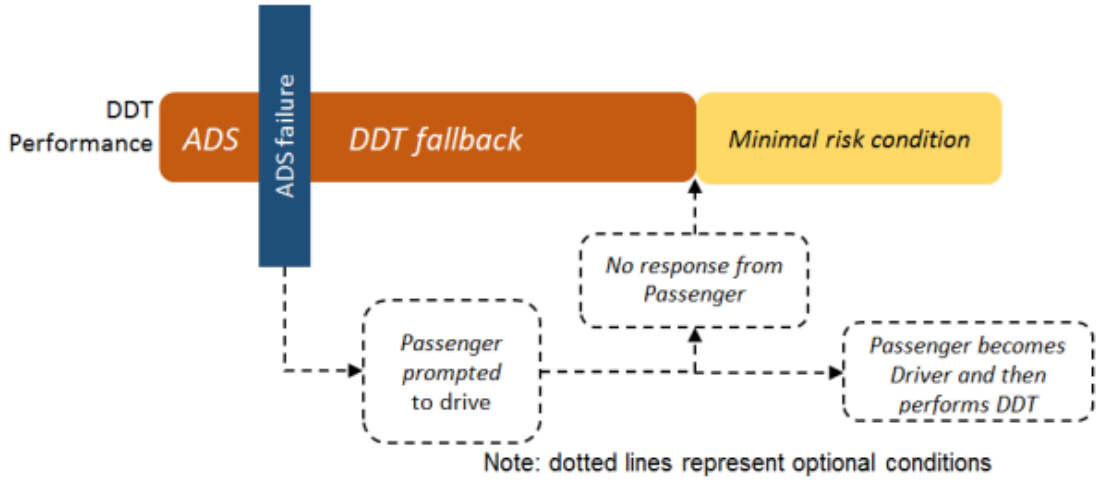


Figura 20 - Caso de uso del funcionamiento del sistema para un nivel 4 [21]

En este ejemplo, el vehículo está en el modo conducción autónoma con nivel 4. En el caso de que el sistema falle, pueden ocurrir dos cosas, que la realimentación del sistema decida por si mismo poner el vehículo y a sus ocupantes en unas condiciones de mínimo riesgo, llegando incluso a pararse, o que el conductor pueda tomar el control mediante una notificación previa y poner el vehículo a salvo el mismo [21].

3.4 Niveles de conducción automatizada

En este punto se presentan más detalladamente los diferentes niveles de conducción en relación con los conceptos que se han visto en el apartado anterior.

3.4.1 Nivel 0

El nivel mínimo de automatización de la conducción corresponde al nivel 0, ya que la tarea de conducir recae completamente sobre la persona que conduce el vehículo, inclusive cuando este se refuerza con sistemas de seguridad activa. Por tanto, en este nivel no existe conducción autónoma.

Level of *Driving Automation*	Role of *User*	Role of *Driving Automation System*
DRIVER* PERFORMS THE *DYNAMIC DRIVING TASK (DDT)		
Level 0 - *No Driving Automation*	*Driver* (at all times): • Performs the entire *DDT*	*Driving Automation System* (if any): • Does not perform any part of the *DDT* on a *sustained* basis (although other *vehicle* systems may provide warnings or support, such as momentary emergency intervention)

Figura 21 - Nivel 0 de conducción autónoma [21]

3.4.2 Nivel 1

El nivel de conducción autónoma 1 corresponde con la capacidad de dar asistencia al conductor por parte del vehículo, realizando este el resto de la conducción. Es decir, el sistema, el vehículo, debe de realizar una subtarea de control, como puede ser el movimiento lateral o longitudinal, pero no ambas simultáneamente. Y se esper que el conductor realice el resto de la tarea de conducción, supervisando además la realización por parte del sistema de la subtarea, ósea el movimiento lateral o longitudinal.

Level of *Driving Automation*	Role of *User*	Role of *Driving Automation System*
Level 1 - *Driver Assistance*	*Driver* (at all times): • Performs the remainder of the *DDT* not performed by the *driving automation system* • *Supervises* the *driving automation system* and intervenes as necessary to maintain safe *operation* of the *vehicle* • Determines whether/when engagement or disengagement of the *driving automation system* is appropriate • Immediately performs the entire *DDT* whenever required or desired	*Driving Automation System* (while engaged): • Performs part of the DDT by executing either the *longitudinal* ***or*** the *lateral vehicle motion control* subtask • Disengages immediately upon *driver* request

Figura 22 - Nivel 1 de conducción autónoma [21]

Un ejemplo de este caso sería el control de crucero adaptativo, ya que controla la aceleración y el frenado, o el freno automático de emergencia en caso de una posible colisión inminente.

3.4.3 Nivel 2

El nivel de conducción autónoma 2 se caracteriza por un sistema de conducción autónomo que realiza las subtareas de control de la conducción del movimiento lateral y longitudinal, pero a diferencia del nivel 1, este sistema las puede hacer de forma simultánea. Esto quiere decir que puede controlar tanto la aceleración y frenado, movimiento longitudinal, como la dirección, movimiento lateral, gracias a que tiene una mayor conciencia del entorno. En este nivel, el conductor debe controlar la conducción además de supervisar las tareas automáticas.

Level of *Driving Automation*	Role of *User*	Role of *Driving Automation System*
Level 2 - *Partial Driving Automation*	*Driver* (at all times): • Performs the remainder of the *DDT* not performed by the *driving automation system* • *Supervises* the *driving automation system* and intervenes as necessary to maintain safe *operation* of the *vehicle* • Determines whether/when engagement and disengagement of the *driving automation system* is appropriate • Immediately performs the entire *DDT* whenever required or desired	*Driving Automation System* (while engaged): • Performs part of the *DDT* by executing both the *lateral* ***and*** the *longitudinal vehicle motion control* subtasks • Disengages immediately upon *driver* request

Figura 23- Nivel de conducción autónoma 2 [21]

En este nivel se pueden retirar las manos y la vista temporalmente, un ejemplo seria la asistencia en carretera, en atascos o al aparcar, lo que incluiría un sistema de mantenimiento de carril trabajando juntamente con un programador de velocidad.

3.4.4 Nivel 3

El nivel de conducción autónoma 3 se refiere a que un sistema autónomo de conducción (ADS) pueda llevar a cabo de forma sostenida y específica, dependiendo de la velocidad o condiciones meteorológicas) las tareas de conducción (DTT) de manera que el usuario o conductor este informado, y pueda monitorizar el sistema, aunque no sea necesario y estar preparado para una solicitud de intervención, por ejemplo, cuando se produzca un fallo del sistema o una rotura de un componente clave para mantener la conducción. En el caso de que se produzca un fallo y el usuario tome el control, este puede llevar el vehículo hacía condiciones de riesgo mínimo o seguir manejando el vehículo una vez desconectado el sistema.

Level 3 – *Conditional Driving Automation*	*Driver* (while the ADS is not engaged): • Verifies operational readiness of the *ADS-equipped vehicle* • Determines when engagement of *ADS* is appropriate • Becomes the *DDT fallback-ready user* when the *ADS* is engaged *DDT fallback-ready user* (while the *ADS* is engaged): • Is *receptive* to a *request to intervene* and responds by performing *DDT fallback* in a timely manner • Is *receptive* to *DDT performance-relevant system failures* in *vehicle* systems and, upon occurrence, performs *DDT fallback* in a timely manner • Determines whether and how to achieve a *minimal risk condition* • Becomes the *driver* upon requesting disengagement of the *ADS*	*ADS* (while not engaged): • Permits engagement only within its *ODD* *ADS* (while engaged): • Performs the entire *DDT* • Determines whether *ODD* limits are about to be exceeded and, if so, issues a timely *request to intervene* to the *DDT fallback-ready user* • Determines whether there is a *DDT performance-relevant system failure* of the *ADS* and, if so, issues a timely *request to intervene* to the *DDT fallback-ready user* • Disengages an appropriate time after issuing a *request to intervene* • Disengages immediately upon *driver* request

Figura 24 - Nivel de conducción autónoma 3 [21]

Un ejemplo de este nivel sería al estar en un atasco, el sistema puede tener el control y el conductor desconectar completamente de la conducción, ya que es el sistema el que lleva la dirección, aceleración y frenado. Cuando el vehículo atraviesa el atasco, el sistema lanza una solicitud para que el usuario reanude el control. El vehículo también debe monitorizar el estado del conductor para poder detenerse de manera segura si el conductor no puede tomar el control.

3.4.5 Nivel 4

En este nivel de conducción autónoma el sistema es capaz de monitorizar el entorno de conducción además de manejar todas las funciones de conducción sin esperar que el usuario responda a una solicitud de intervención. Es decir que el propio sistema llevará el vehículo hacia unas condiciones de riesgo mínimo en caso de fallo y sin esperar ninguna intervención por parte del usuario, que en este nivel se le considera pasajero en las situaciones en las que el sistema (ADS) toma el control.

Level 4 - High Driving Automation	*Driver/dispatcher* (while the *ADS* is not engaged):
	• Verifies operational readiness of the *ADS-equipped vehicle*
	• Determines whether to engage the *ADS*
	• Becomes a *passenger* when the *ADS* is engaged only if physically present in the *vehicle*
	
	Passenger/dispatcher (while the *ADS* is engaged):
	• Need not perform the *DDT* or *DDT fallback*
	• Need not determine whether and how to achieve a *minimal risk condition*
		ADS (while not engaged):
		• Permits engagement only within its *ODD*
		
		ADS (while engaged):
		• Performs the entire *DDT*
		• May issue a timely *request to intervene*
		• Performs *DDT fallback* and transitions automatically to a *minimal risk condition* when:
		  • A *DDT performance-relevant system failure* occurs or
		  • A *user* does not respond to a *request to intervene* or

Figura 25 - Nivel de conducción autónoma 4 [21]

Como ejemplo, se considera un nivel 4 a la tarea de aparcar, pero sin ninguna supervisión del usuario. Conducir por autopista sin supervisión también se consideraría de este nivel, pero la conducción antes de llegar a la autopista y al salir la realiza el usuario. Si este no respondiera cuando llegase el momento, el sistema pondría el vehículo en riesgo mínimo, Esa también es una diferencia con el nivel 3, ya que en el nivel anterior la tarea de poner el vehículo en condiciones de riesgo mínimo la realiza el conductor.

3.4.6 Nivel 5

El nivel de conducción 5 quiere decir que es totalmente autónomo, es decir que realiza la tarea de la conducción de manera sostenida e incondicional o no especifica, independientemente de las restricciones meteorológicas, horarias, de velocidad o geográficas, y no espera que un usuario responda a una solicitud de intervención. Sin embargo, puede haber condiciones que tampoco son manejables por un conductor y en las que el sistema tampoco sería capaz de completar el viaje, por ejemplo, carreteras inundadas o, tormentas de nieve. En estos casos el sistema ejecutara la conducción hasta llegar a unas condiciones de riesgo mínimo, y esperará hasta que las condiciones vuelvan a ser favorables para reanudar la conducción. En este nivel como en el anterior, el usuario no necesita supervisar el sistema ni ser receptivo a solicitudes de intervención.

Level 5 - Full Driving Automation	*Driver/dispatcher* (while the *ADS* is not engaged):
	• Verifies operational readiness of the *ADS*-equipped vehicle
	• Determines whether to engage the *ADS*
	• Becomes a *passenger* when the *ADS* is engaged only if physically present in the *vehicle*
	
	Passenger/dispatcher (while the *ADS* is engaged):
	• Need not perform the *DDT* or *DDT fallback*
	• Need not determine whether and how to achieve a *minimal risk condition*
	• May perform the *DDT fallback* following a *request to intervene*
	• May request that the ADS disengage and may achieve a *minimal risk condition* after it is disengaged
	• May become the *driver* after a requested disengagement
		ADS (while not engaged):
		• Permits *engagement* of the *ADS* under all driver-manageable on-road conditions
		
		ADS (while engaged):
		• Performs the entire *DDT*
		• Performs *DDT fallback* and transitions automatically to a *minimal risk condition* when:
		  ◦ A *DDT performance-relevant system failure* occurs or
		  ◦ A *user* does not respond to a *request to intervene* or
		  ◦ A *user* requests that it achieve a *minimal risk condition*
		• Disengages, if appropriate, only after:
		  ◦ It achieves a *minimal risk condition* or
		  ◦ A *driver* is performing the *DDT*
		• May delay a *user*-requested disengagement

Figura 26 - Nivel de conducción autónoma 5 [21]

Como ejemplo en este caso, el vehículo de nivel 5 sería capaz de realizar un viaje programado con un destino independientemente de si está en una autopista, de los puntos de partida y llegada o de las condiciones de la carretera, tráfico o meteorológicas.

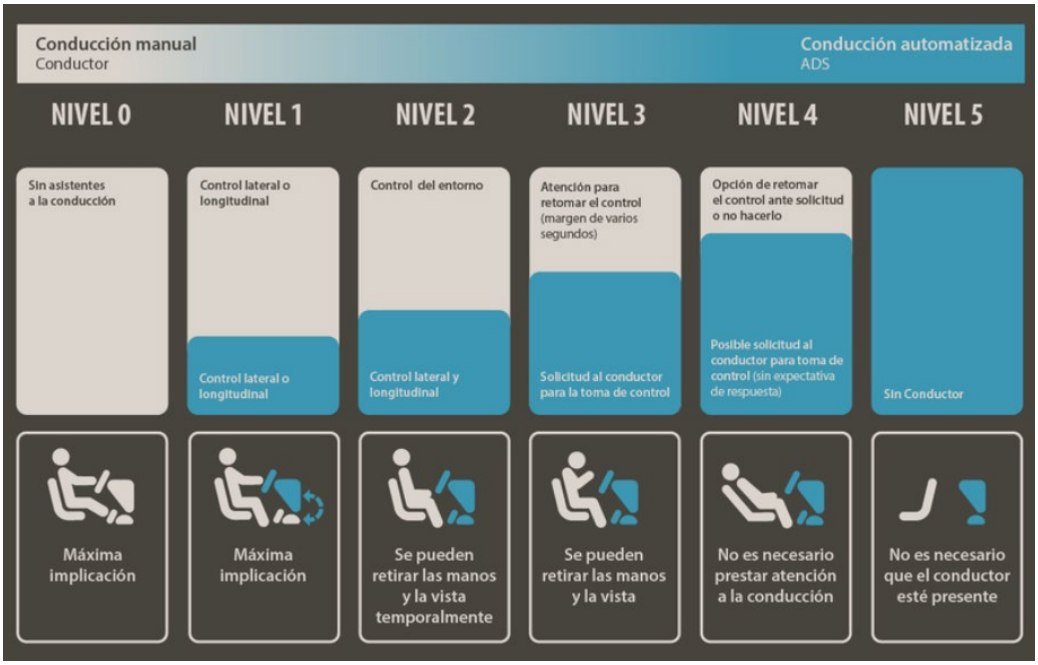


Figura 27 - Representación de los niveles de conducción autónoma [22]

3.5 Futuro de la conducción autónoma

Los sistemas de advertencia, asistencia y sistemas avanzados de asistencia al conductor (ADAS) ya se encuentran disponibles en el marcado para incrementar la automatización tanto en los vehículos de pasajeros como en los de carga.

El camino que va a continuar la automatización de la conducción va a ser progresivo, si bien ya tenemos sistemas de nivel 2, en los que el conductor tiene que estar pendiente pero el sistema puede realizar la conducción de forma autónoma para unas condiciones específicas. Un ejemplo de vehículo autónomo de nivel 2 sería el Tesla Model S, pero también se está empezando a desplegar los sistemas de nivel 3, como puede ser el Mercedes Benz Clase S o el Honda Legend EX.

Estos avances quieren decir que el desarrollo propuesto para conseguir e implementar en los vehículos de pasajeros y de carga la conducción autónoma se está cumpliendo como se puede apreciar en la imagen posterior.

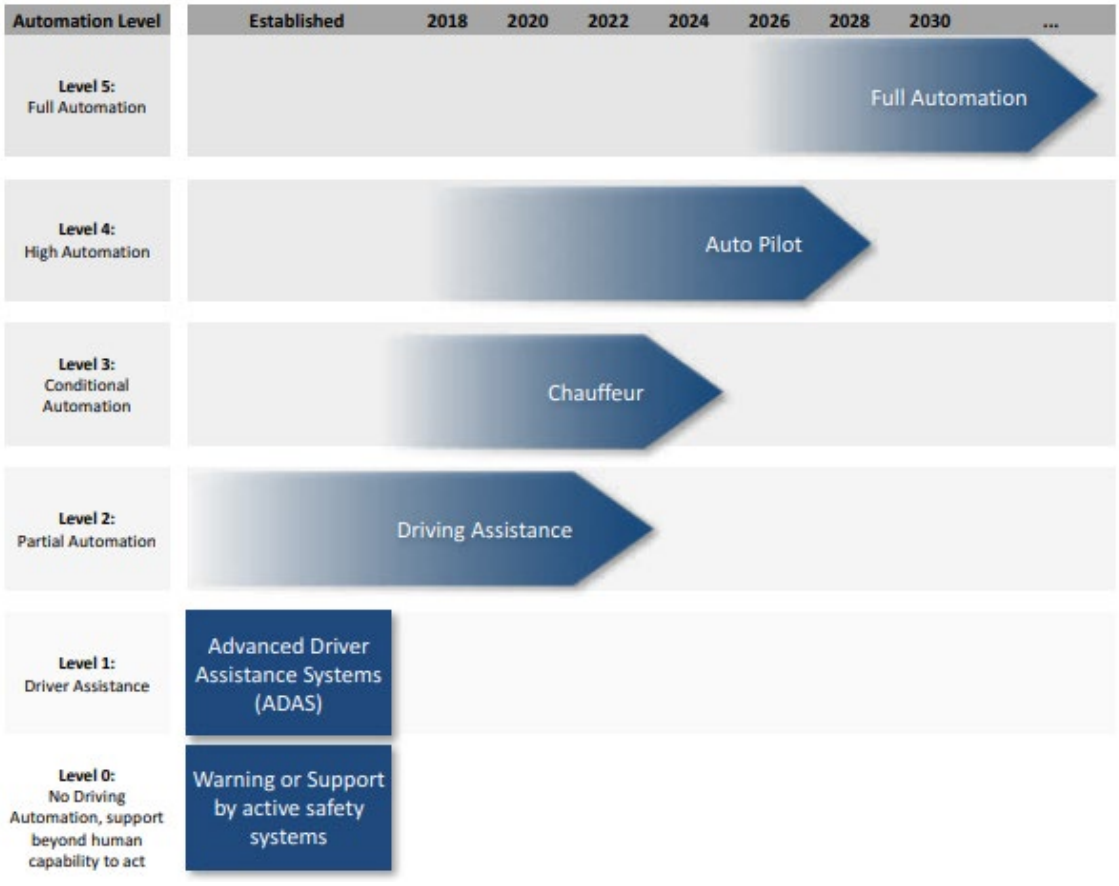


Figura 28 - Camino hacia el desarrollo de la conducción autónoma [20]

La ruta que debe seguir el desarrollo del coche autónomo de pasajeros es compartida también por los vehículos de carga, así como de los autocares. Como se ha comentado antes nos encontramos en el nivel 3 de desarrollo y ya empezamos a tener sistemas capaces de ser autónomos en situaciones de tráfico saturado tanto urbano como en autopista, o conducción regular en autopistas, que prácticamente sería un desarrollo del siguiente nivel de automatización.

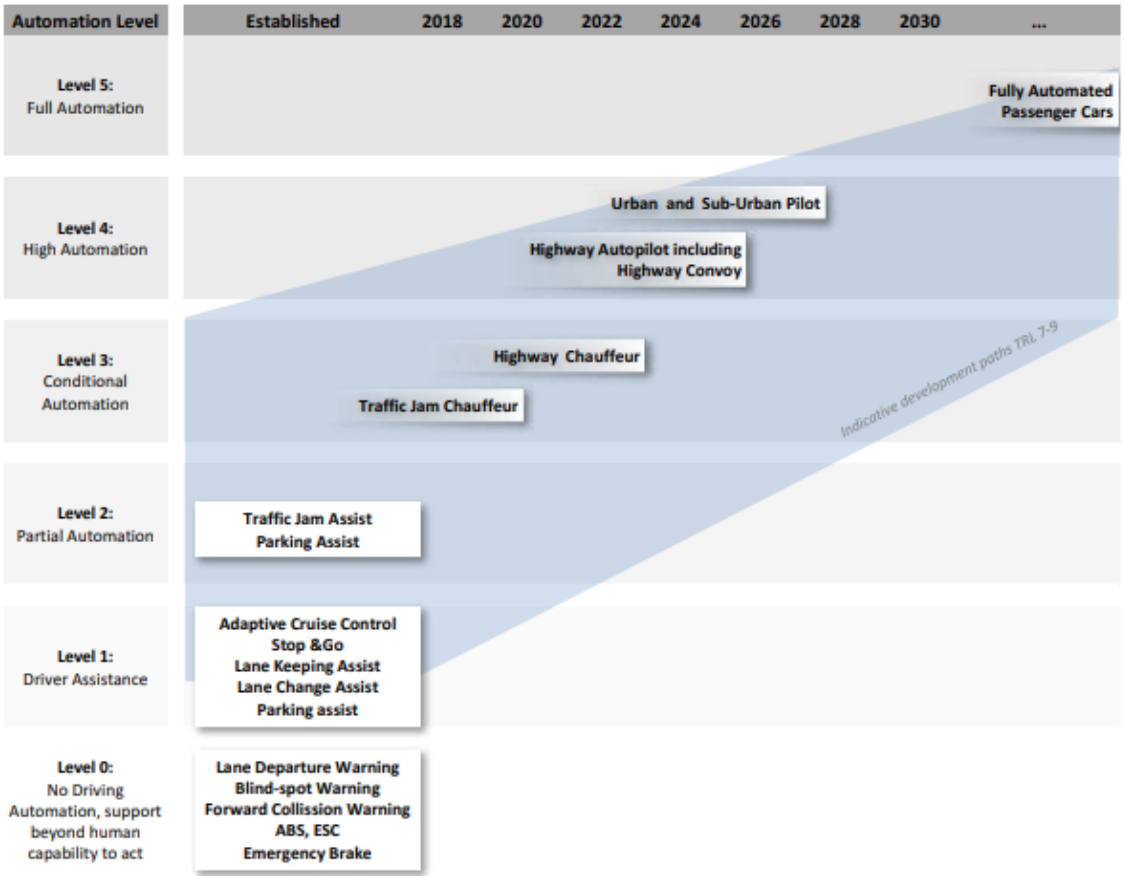


Figura 29 - Camino hacia el desarrollo de vehículos autónomos de pasajeros [20]

En cuanto a la investigación para poder continuar con los desarrollos y alcanzar los diferentes niveles, la UE lleva financiando la investigación y la innovación en el ámbito de la conducción autónoma más de una década, comenzando con el 6º Programa Marco hasta el Horizonte 2020 con un presupuesto de unos 300 millones de euros.

El objetivo de estos programas es centrarse en proyectos piloto de demostración a gran escala para probar sistemas de conducción altamente automatizados para turismos, transporte de mercancías o servicios de movilidad en zonas urbanas.

Otras prioridades en estos programas es la aceptación por parte de los usuarios, el diseño de una interfaz hombre-máquina segura, adaptación de la infraestructura vial en apoyo a la automatización y los procedimientos de prueba y validación de las funciones de la conducción automatizada.

La Comisión Europea también ofrece un apoyo financiero a las partes interesadas en el desarrollo de una red de corredores transfronterizos paneuropeos con la tecnología 5G para pruebas a gran escala y un despliegue temprano de la conectividad que ayudaría a la conducción tanto conectada como automatizada, ya que los vehículos podrían comunicarse entre sí o con el sistema de carreteras, advirtiendo de situaciones de peligro o la comunicación de los movimientos y parámetros entre vehículos para adaptar la conducción.

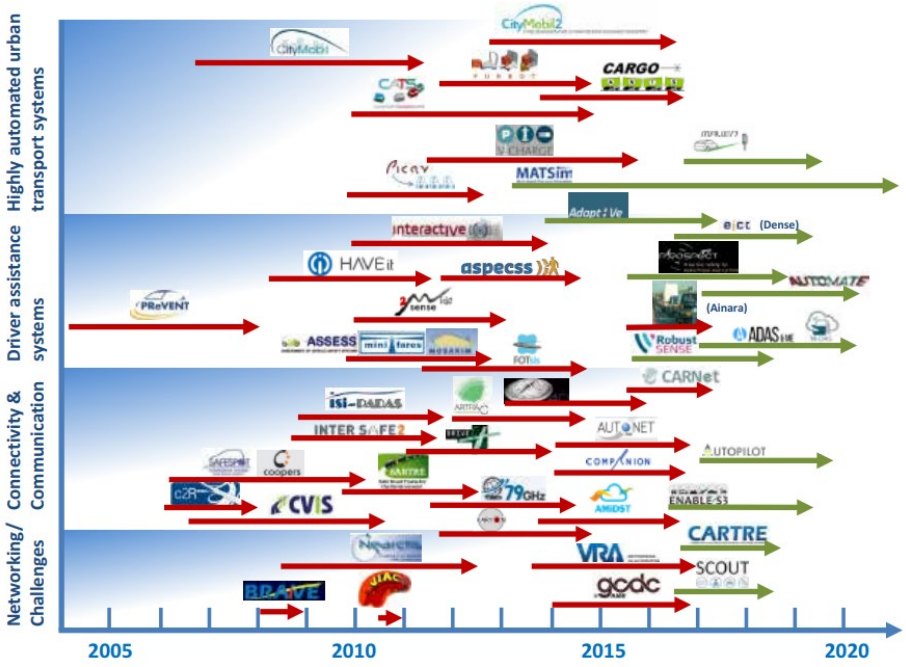


Figura 30 - Vista de los proyectos en marcha para desarrollar el vehículo autónomo [20]

Los proyectos CARTRE y SCOUT por ejemplo pretenden acelerar el desarrollo creando una base de conocimiento para las actividades de investigación, identificar GAPS y prioridades mediante un intercambio de datos. También se pretende establecer una hoja de ruta exhaustiva y estructurada que establezca interdependencias técnicas y no técnicas para acelerar los procesos, sobre todo en los niveles 4 y 5 SAE. Estos dos proyectos han detectado ámbitos técnicos y no técnicos como [23]:

- Tecnología a bordo de vehículos
- Infraestructura física y digital
- Conectividad de servicios compartidos
- Factores humanos
- Concienciación de los usuarios
- Aceptación social
- Ética
- La validación de la seguridad
- Aptitud para la circulación
- Inteligencia artificial
- Big Data
- Evaluación socioeconómica
- Sostenibilidad

Algunos de los proyectos que se han realizado o se realizan en España son los siguientes:

- **Un millón de kilómetros:** Este proyecto consiste, como su nombre indica, en la realización de un millón de kilómetros con vehículos autónomos en vías e infraestructuras reales para recabar datos y experiencias de usuario para favorecer la implantación de los nuevos servicios asociados a la conducción autónoma y la movilidad [24].

- **BRAVE**: es un proyecto sobre conducción autónoma, en concreto sobre la automatización condicionada (Nivel de automatización 3), financiado por el programa Horizonte 2020 de la Unión Europea y lanzado en 2017 por un consorcio internacional. La base de este proyecto es considerar que la introducción de vehículos automatizados en las vías públicas sólo podrá tener éxito cuando el usuario esté en el centro de todos los aspectos técnicos, es decir, cuando la tecnología vaya de la mano de aspectos como la aceptación del usuario, la seguridad vial, las consideraciones sociales, económicas, legales y éticas [25].

- **R3CAV:** incluye el desarrollo de una nueva arquitectura adaptable -tanto en hardware como software- destinada a la creación de futuros vehículos autónomos y conectados, capaces de operar con diferentes niveles de autonomía. Tiene dos casos de uso principales: el primero de ellos en el desarrollo de un prototipo de vehículo autónomo en entorno controlado industrial, que se llevará a cabo en la factoría de Renault en Palencia; y el segundo de ellos en Alcobendas donde se probarán tanto sistemas de conectividad en vehículos Renault como en vehículos altamente automatizados de transporte de personas [26].
- **EZ10:** Se trata de un proyecto en el que participan ASLA, CRTM, UAM y la DGT y que pone en marcha el primer autobús autónomo y eléctrico que circula por las instalaciones de la UAM, recorriendo 3.7 km en cada viaje [27].

Además de los proyectos hay que tener en cuenta que al principio no se recogía en la ley la existencia del vehículo autónomo, pero ahora los países comienzan a introducirlos en su regulación, como es el caso de España que ha reformado su ley de tráfico para incluir la denominación de coche autónomo, pero también Alemania ha introducido la regulación para el coche autónomo de nivel 4 con un proyecto de ley.

Esto hace indicar que poco a poco el vehículo autónomo es aceptado y se va introduciendo en nuestra sociedad siguiendo la hoja de ruta desarrollada. Todavía queda mucho por hacer, pero en esta década seguramente lleguemos a tener un coche autónomo de nivel 3 y 4.

3.6 Donde encaja este proyecto

Este proyecto de estudio e investigación de vehículos autónomos en miniatura tiene el propósito de ayudar en la investigación y desarrollo de los vehículos autónomos poniendo a prueba dos de las partes involucradas, que son el software y el hardware.

Primero para el hardware, ya que se han diseño tarjetas electrónicas específicas para poder desarrollar nuevos algoritmos o poner a prueba otros ya creados, como puede ser la Jetson nano, de una forma económica y abierta a todo el mundo, para que haya una gran cantidad de usuarios que puedan dar "*feedback*" de los proyectos, algoritmos y eficiencia del hardware. Y una vez probado en este tipo de hardware llevarlo a otro más profesional de manera que este último no se vea comprometido por los nuevos desarrollos, ya que es mucho menos económico y no saldría rentable si ocurriese algún fallo irreversible.

Segundo en cuanto al software, ya que la gran cantidad de usuarios puede poner a prueba diferentes algoritmos o crear otros nuevos que ayuden al desarrollo de la conducción autónoma y que más adelante puedan llegar a ser probados en proyectos de mayor envergadura.

Además, este tipo de proyectos pueden acercar más a la sociedad la inteligencia artificial y dejar de verla poco a poco no como una amenaza si no como una ayuda en nuestro día a día como puede ser la conducción.

Es en estos puntos donde encajaría mejor este proyecto, al final es un pequeño paso en un desarrollo de grandes proporciones.

Capítulo 4. HARDWARE DEL SISTEMA

4.1 Introducción

En esta sección se va a ver una descripción de todos los componentes que conforman el robot, así como una guía para su montaje y otra guía para la creación de una pista de pruebas portátil.

4.2 Descripción de los componentes del robot

A continuación, se van a describir brevemente los componentes del kit de desarrollo jetbot AI 2GB.

Figura 31 - Componentes del kit de desarrollo jetbot [28]

1. Jetson Nano 2GB
2. Tarjeta de memoria microSD 64 GB
3. Chasis principal
4. Estructura de la cámara
5. Base acrílica aislante para la cámara
6. Placa de expansión controladora
7. Cámara CSI
8. Antena WiFi (Ha sido sustituido por un USB WiFi, No incluido en el kit)

9. Motores de 6V y 200 rpm
10. Ruedas
11. Rodamientos
12. Baterías de 18650 3,7V y 9700 mAh, no incluidas en el kit
13. Fuente de alimentación de 5V y 2,5A
14. Joystick PS4/PC
15. Destornilladores
16. Cable agrupado de 6 pines
17. Tornillería
18. Llave
19. Ventilador
20. Lector de tarjetas microSD

4.3 Montaje del robot

Esta sección del capítulo contiene una guía de montaje paso a paso del jetbot.

4.3.1 Montaje de los motores

Los motores deben de ir atornillados al chasis del robot con 4 tornillos del tipo M3*25 , dos para cada uno, introduciéndolos por el exterior y ajustando con tuercas del mismo tamaño por el lado interior del chasis, al otro lado del motor. Los motores deben ir orientados como muestra la imagen siguiente.

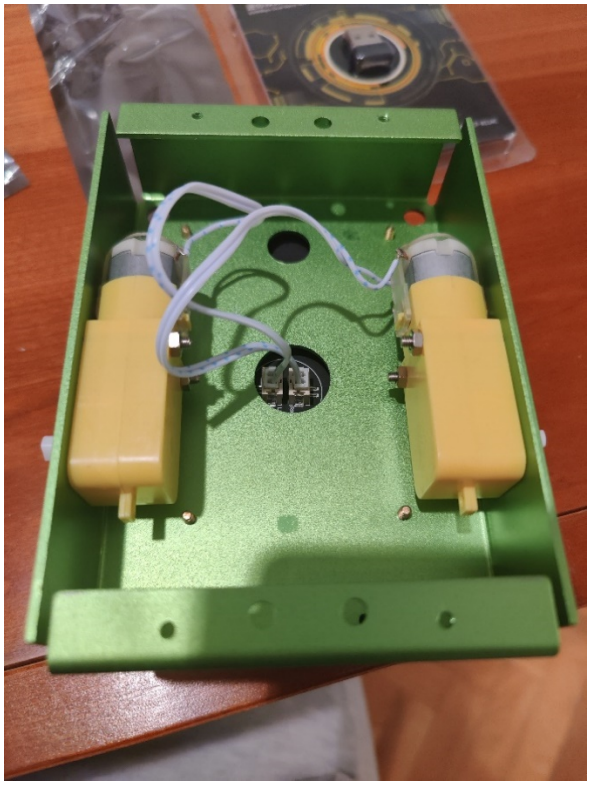


Figura 32 - Montaje de los motores

4.3.2 Colocación del chasis de la cámara

Para colocar el chasis de la cámara y la cámara, se debe juntar primero este mismo, con la placa de aislante acrílico y después la cámara, como se muestra en la siguiente figura. Cuando todo este junto procedemos a atornillar todo con los 4 tornillos y tuercas de nylon de medida M2*6.

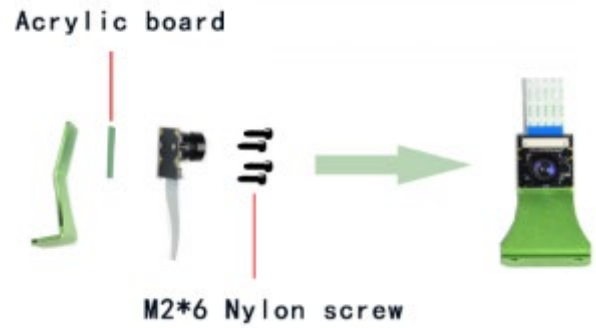


Figura 33 - Colocación de la cámara [28]

Una vez que tenemos esto, se debe atornillar esta estructura al chasis mediante tornillos de medida M3*6. Quedando el conjunto de la siguiente forma.

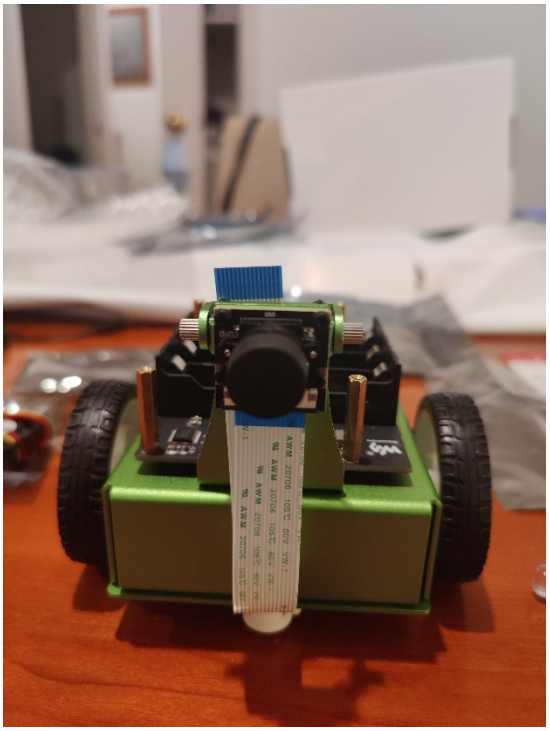


Figura 34 - Montaje de la cámara

4.3.3 Montaje de la placa de expansión

Antes de montar la placa de expansión sobre el chasis se debe atornillar unas columnas de separación por su parte superior, para que queden salientes y posteriormente se pueda colocar la placa Jetson nano. Para esto se colocan las 4 columnas de métrica M2.5*25 y se atornillan a la placa mediante 4 tornillos, uno para cada columna, de métrica M2.5*5.

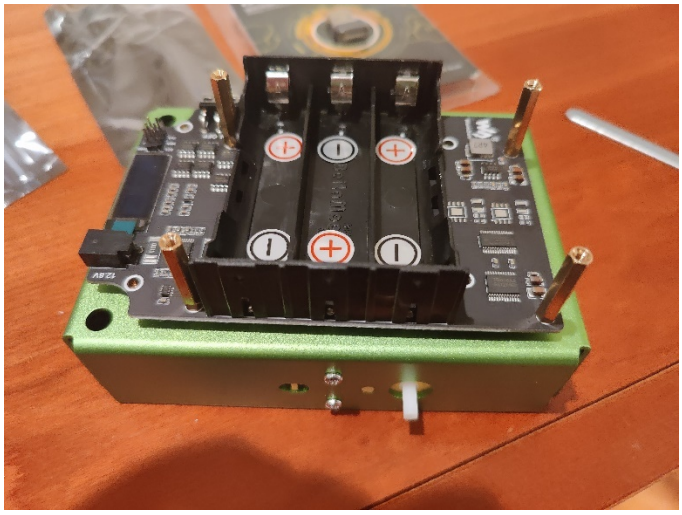


Figura 35 - Montaje de la placa controladora de expansión

La placa de expansión debe ir montada sobre la parte superior del chasis, quedando encajado el conector de los motores en el orificio central. Para atornillarla se deben colocar 4 tornillos por el interior de métrica M3*6, luego una pequeña columna de separación de métrica M3*1. Se coloca la placa y se atornilla a la columna de separación, otros 4 tornillos de métrica M3*6.

Figura 36 - Montaje de la placa controladora de expansión

4.3.4 Conexión de los motores a la placa de expansión

Una vez colocada la placa de expansión, podemos dar la vuelta al chasis y conectar los motores con la misma orientación que aparece en la imagen. Este paso es muy importante porque después procederemos a cerrar el conjunto y si nos equivocamos con la posición al conectarlos el robot funcionará de manera inversa y deberemos de volver a abrir el chasis para corregir la conexión.

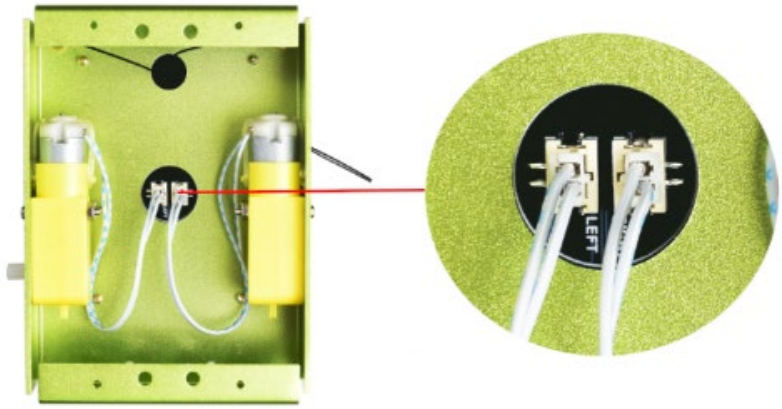


Figura 37 - Conexión de los motores a la placa de expansión [28]

4.3.5 Montaje de los rodamientos

Para los rodamientos deben introducirse primero en la carcasa de plástico y después atornillarlos a la placa inferior del chasis, pero dejando que tengan un poco de juego para que puedan rodar adecuadamente.

Figura 38 - Montaje de los rodamientos

Una vez colocados, procedemos a colocar la parte inferior del chasis sobre la parte superior cubriendo los motores, encajándola y atornillándola con tornillería del tipo M3*6, quedando el conjunto como aparece en la imagen posterior.

Figura 39 - Montaje del chasis inferior

4.3.6 Colocación de las ruedas

Este paso es el más sencillo, tan solo debemos colocar cada una de las ruedas en la parte sobresaliente del chasis y rotativa de los motores. Simplemente van encajadas, sin atornillar.

Figura 40 - Montaje de las ruedas

4.3.7 Colocación de las baterías.

Se deben usar 3 baterías del tipo 18650, con una capacidad mayor a 7500 mAh. Esto es importante pues, con que una tenga una capacidad inferior ni el robot ni la placa Jetson nano se encenderán. Para colocarlas se debe seguir la indicación de los símbolos, y hacer coincidir los de la placa de expansión junto con los de las baterías de manera que, positivo debe ir con positivo y negativo con negativo.

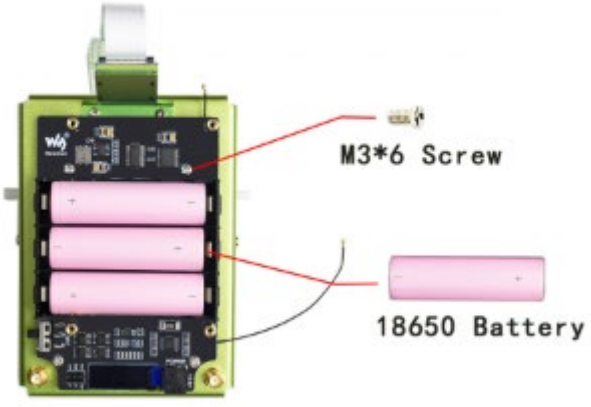


Figura 41 - Colocación de las baterías [28]

4.3.8 Montaje de la placa Jetson nano

Después de montar las baterías el siguiente paso es colocar la placa Jetson nano, la cual se sitúa encima de la placa de expansión, pero va sobre las columnas separadoras atornillada con 4 tornillos de métrica M2.5*5.

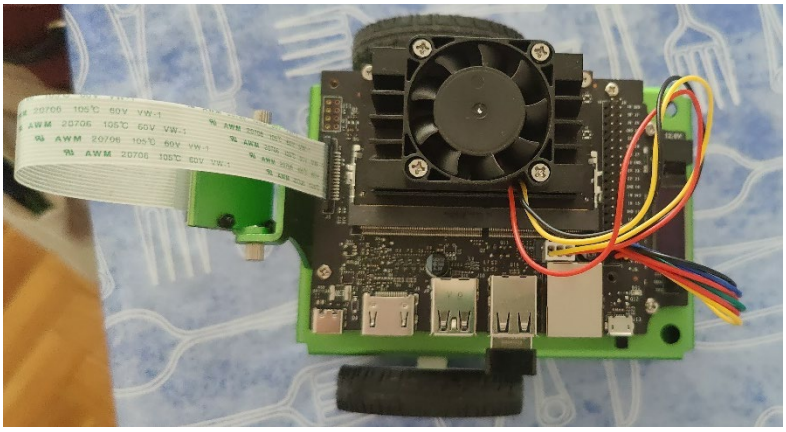


Figura 42 - Montaje de la placa jetson nano

Una vez colocada, conectaremos el cable de la cámara con la parte azul hacia fuera.

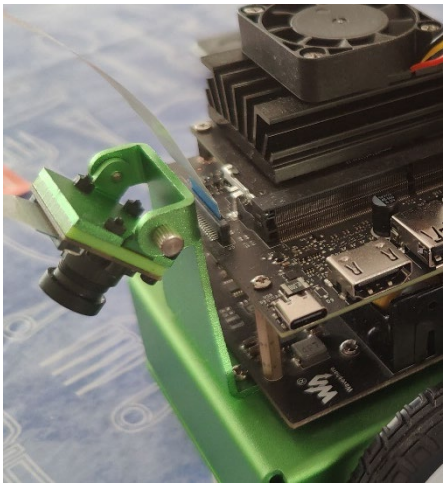


Figura 43 - Conexión de la cámara con la placa jetson nano

4.3.9 Montaje del ventilador y conexión entre placas

Por último, se procederá a montar el ventilador sobre el disipador de la placa Jetson nano. Para ello se atornillará con 4 tornillos de métrica M2*6 y se conectará al conector de 4 pines, dejando uno sin conectar como muestra la figura siguiente.

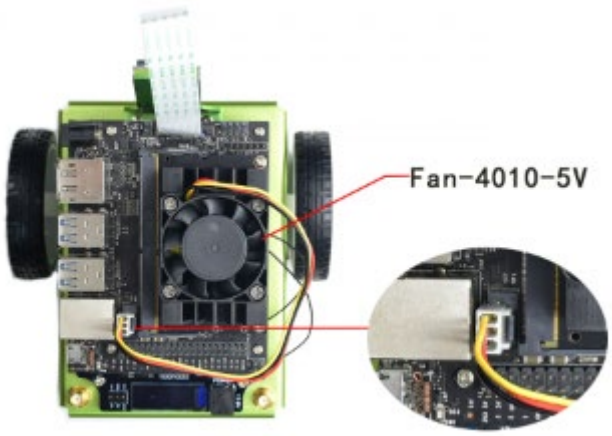


Figura 44 - Conexión y montaje del ventilador [28]

También hay que conectar la placa de expansión con la Jetson nano mediante 6 cables unidos por dos conectores hembra. Estos cables se conectarán desde el conector de la placa de expansión hasta los primeros pines GPIOs de la Jetson nano de la siguiente forma.

Figura 45 - Conexión entre placas [28]

Por último, se conecta el WiFi USB, se introduce la tarjeta microSD por la parte trasera de la jetson nano y el montaje del robot estaría completado.

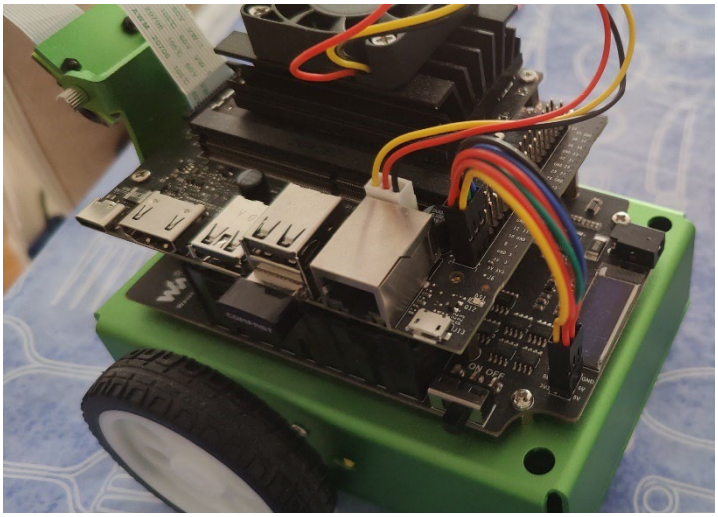


Figura 46 - Otra perspectiva de las conexiones entre placas y el ventilador

4.4 Creación y montaje de la pista de pruebas

Para poder probar los diferentes algoritmos y el movimiento del robot en un mismo entorno real de pruebas, se ha decidido hacer un circuito o una pista de pruebas que pueda ser plegable y ligera para poder llevarla a cualquier lugar y poder ponerlo a prueba en cualquier momento.

Este diseño será modular y cada pieza corresponderá a una parte del circuito, pudiendo ser sustituida sin problemas en caso de desgaste.

Para ello se han comprado 9 cartulinas negras de tamaño A3 y una cinta de pintor blanca, puede parecer un poco rudimentario, pero es útil para lo que se necesita, es económico, flexible, permite un diseño modular y además ligero para poder llevarlo a cualquier lado.

Se van a disponer las cartulinas formando un rectángulo de 3x3. Para unir unas con otras y permitir esa flexibilidad y que sea plegable, se va a pegar la cinta de pintor por la parte de abajo, uniendo una tras otra siguiendo este esquema.

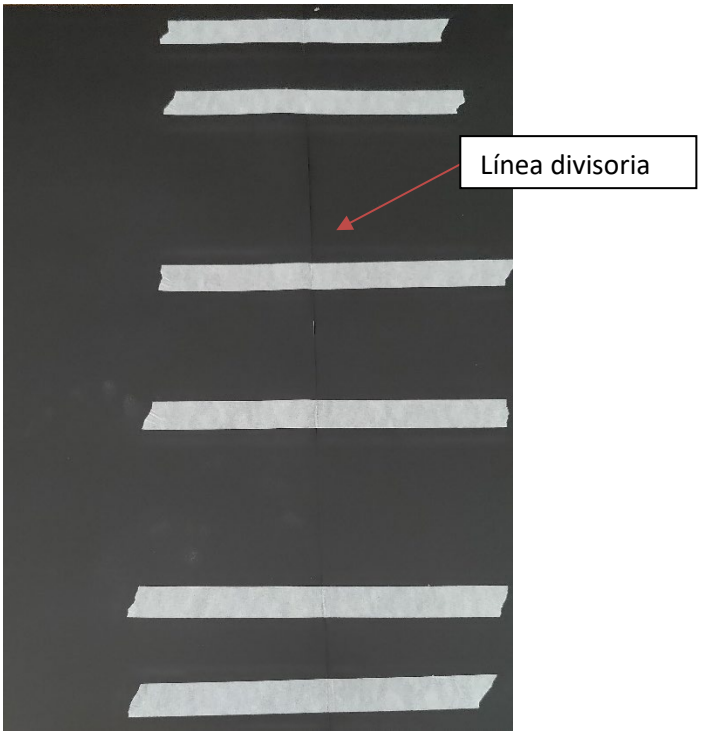


Figura 47 - Unión de las piezas del circuito de pruebas

Una vez que están todas unidas se le da la vuelta a la pista de pruebas y se empieza a crear el circuito por el que circulará el robot. En este caso se han dispuesto 3 calles comunicadas y con un cruce central. Cada calle mide aproximadamente 25cm de ancho para que le robot tenga espacio suficiente y tiempo de reacción para corregir su posición dado su tamaño. La pista de pruebas ha quedado de la siguiente forma, como muestra la figura.

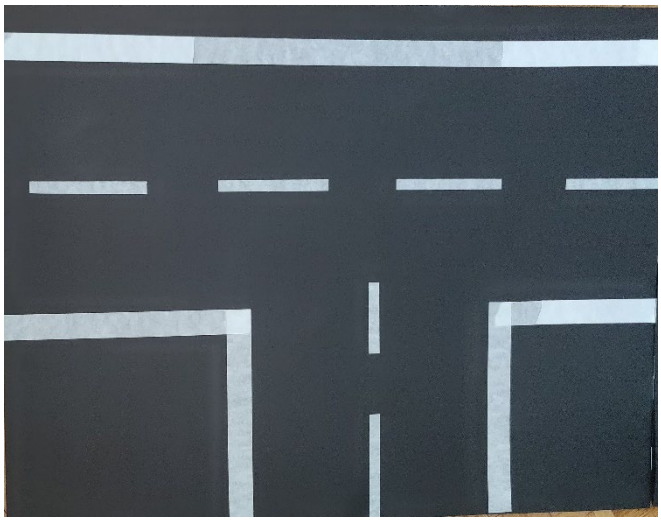


Figura 48 - Sección de la pista de pruebas

Capítulo 5. DESARROLLO SOFTWARE DEL SISTEMA

5.1 Introducción

En este capítulo se van a explicar los algoritmos de inteligencia artificial que se han utilizado para que el robot aprenda en base a una colección de imágenes y pueda seguir un camino, así como evitar obstáculos.

5.2 Algoritmos de IA implementados

5.2.1 ResNet18

Se trata de una red neuronal convolucional, CNN (Convolutional Neural Network) que está basada en bloques residuales y fue presentada por Microsoft. Esta red consiguió tener menos complejidad que otras redes convolucionales, como la VGGNet, y bajar la tasa de error hasta un 3,57% gracias a los bloques residuales y a las 152 capas [30].

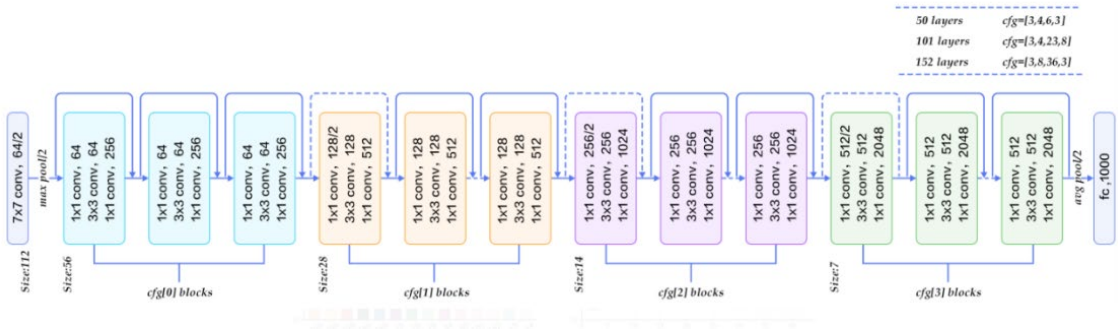


Figura 49 - Arquitectura de la red ResNet [29]

El concepto principal de la creación de ResNet se basa en que una red neuronal a partir de cierta profundidad, en contra de lo que pueda parecer, no siempre va a mejorar, y empiezan a aparecer el sobre entrenamiento, el desvanecimiento del gradiente (vanishing gradient) y problemas con la dimensionalidad. Es decir que la red deja de aprender y su precisión se estanca. Sin embargo, las redes convolucionales que son profundas porque tienen más capas, aprenden mejor, porque son capaces de saltarse algunas de esas capas y así evitar los problemas anteriormente mencionados propios de redes profundas que utilizan todas las capas [30].

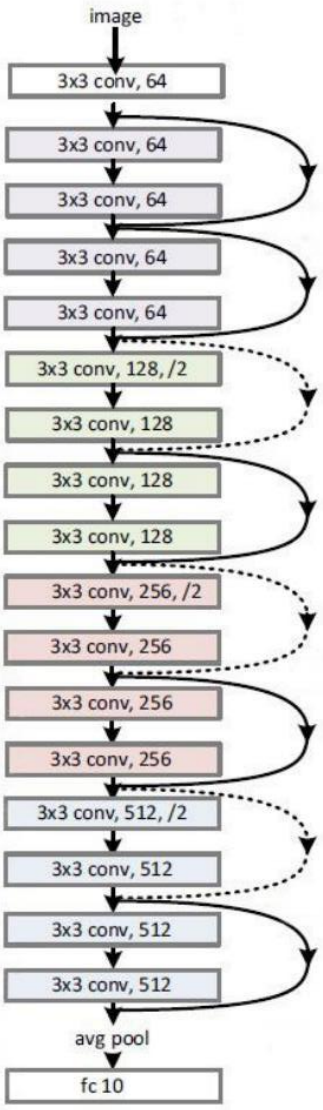


Figura 50 - Arquitectura ResNet18 [31]

Aquí es donde subyace la idea de esta red, ya que lo que hace es saltarse capas introduciendo una conexión residual (una capa identidad). La siguiente imagen muestra el salto de capas a partir de la identidad.

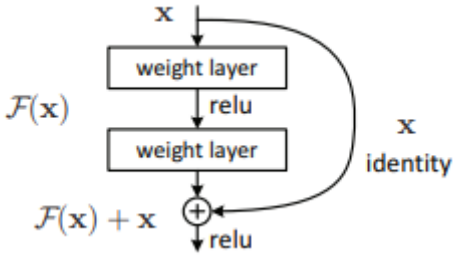


Figura 51 - Esquema de un bloque residual [30]

El funcionamiento es el siguiente, la entrada x va ir pasando por todas las capas conv-ReLu-conv-ReLu. Esta operación da como resultado un F(x). La salida de este bloque es F(x) + x que sería la H(x). Una red convolucional tradicional se debe conseguir que F(x) y H(x) tengan el mismo valor pero para este tipo de red lo que se calcula es que valor hay que añadir a F(x) a su entrada x. Es decir añadir una pequeña alteración a la entrada para conseguir una representación ligeramente alterada. La diferencia con las redes convolucionales tradicionales es que no añaden esa relación con su entrada original. Los autores dicen que esto hace que la optimización del mapa residual sea más sencillo y con el algoritmo de backpropagation, el gradiente no se desvanece por las operaciones de suma que se van realizando a través de la red [30, 32].

Un punto interesante de la red es que con 152 capas se obtiene una precisión muy alta, y que después de las dos primeras el tamaño del volumen se ha reducido de 224x224 a 56x56. Aumentar el número de capas puede dar un resultado mejor pero también puede crear overfitting.[30, 32]

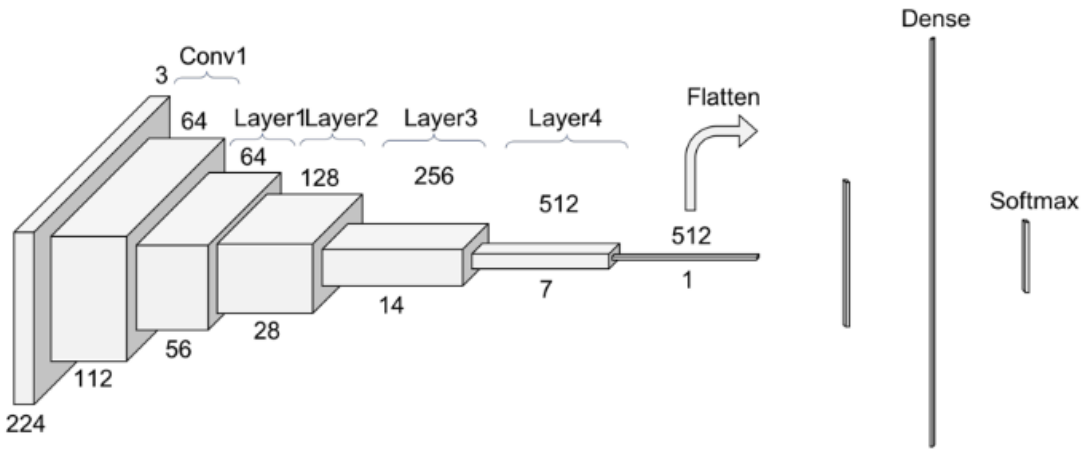


Figura 52 - Otra vista de las capas de la red ResNet [32]

5.2.1.1 1ª convolución

El primer bloque antes de entrar en las capas comunes de la red consiste en una convolución, más una normalización por lotes, más una operación para agrupar los máximos posibles.

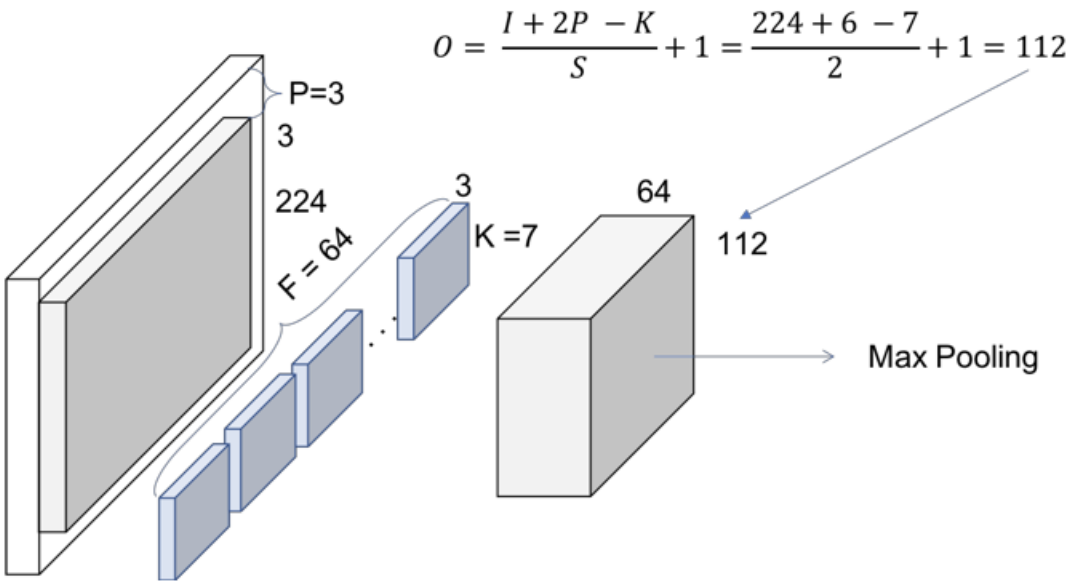


Figura 53 - Convolución de una red ResNet [32]

La red neuronal ResNet18 normalmente usa un kernel de tamaño 7 y un mapa de características de 64, esto hace que tengamos dividida la imagen en secciones de 64x7 y se rellene con ceros el resto. Al final el tamaño de salida que se obtiene apilando las secciones es de 112x112x64 [32].

5.2.1.2 Bloques

Cada capa de una ResNet está compuesta por varios bloques, esto es debido a que cuando una red de estas características profundiza, lo hace normalmente *aumentando el número de operaciones dentro de cada bloque*, pero sin alterar el número de capas. Con lo que a operación se refiere, quiere decir que se realiza una convolución, una normalización por lotes y una activación de la red a una entrada. La siguiente figura sería un ejemplo de las operaciones que realiza un bloque [32].

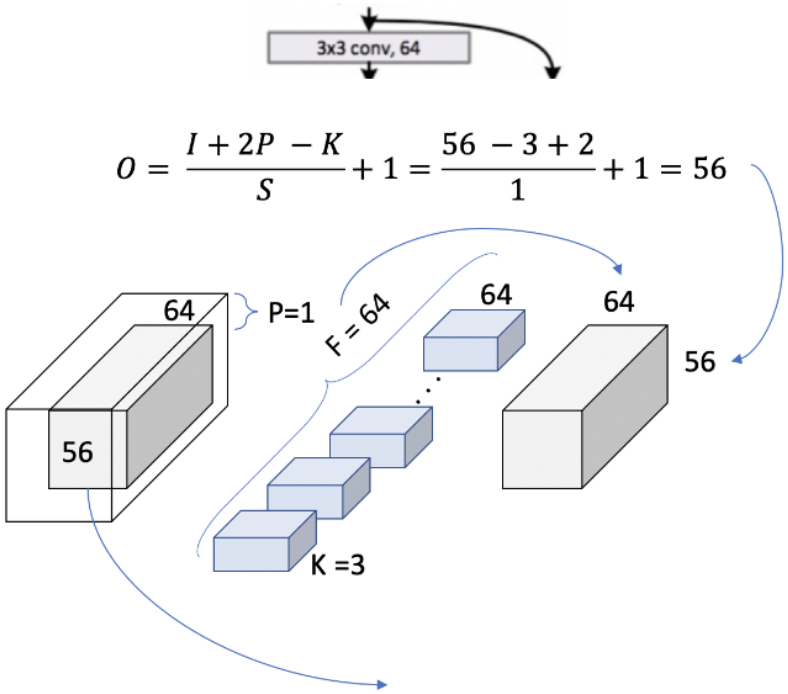


Figura 54 - Representación de un bloque básico [32]

Se puede comprobar que el volumen no cambia dentro de un bloque, al final se obtiene una salida de 56x56. La siguiente figura muestra cómo se extendería esta misma operación para cubrir un bloque entero de 3x3 [32].

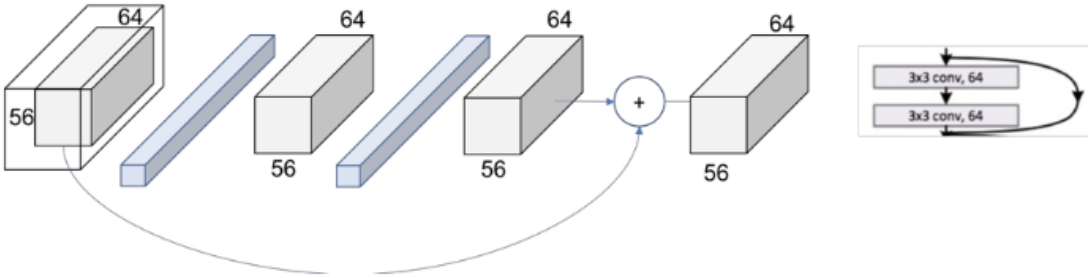


Figura 55 - Operaciones de un bloque completo 3x3 [32]

La figura siguiente mostraría lo mismo pero extendido para una capa de la red neuronal de 3x3.

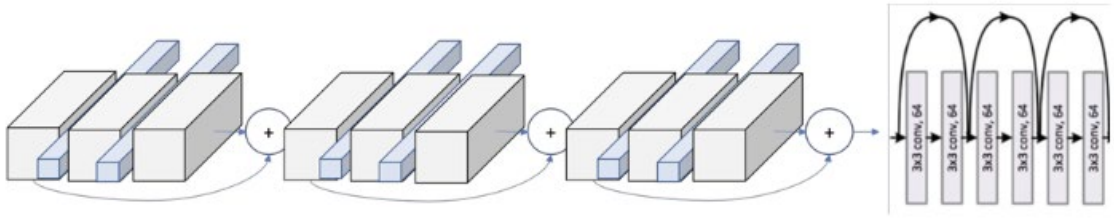


Figura 56 - Esquema de capas y bloques de operación de la red neuronal [32]

5.2.1.3 Patrones en las capas

Se puede ver como patrón en las capas el cambio de dimensionalidad y la primera operación de cada capa que se encarga de reducir la dimensión por lo que también habría que dimensionar el volumen que pasa por la conexión de salto. Por tanto, existiría una diferencia entre el volumen en el paso normal y el paso con salto así que hay que realizar convoluciones 1x1, en los dos, de tal manera que quede como la siguiente imagen [32].

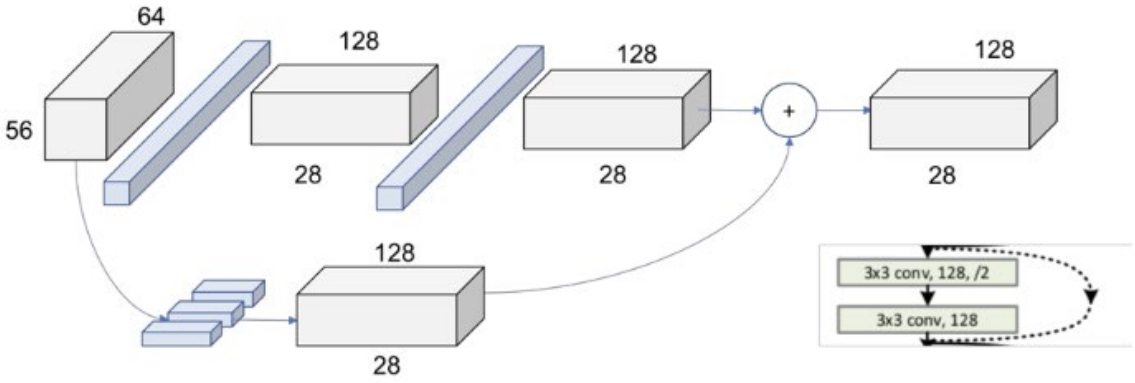


Figura 57 - Convolución 1x1 en salto [32]

Con esto los dos volúmenes de cada salida tienen el mismo tamaño y pueden ser sumados, el comportamiento es el mismo a lo largo del conjunto de capas como muestra la siguiente imagen [32].

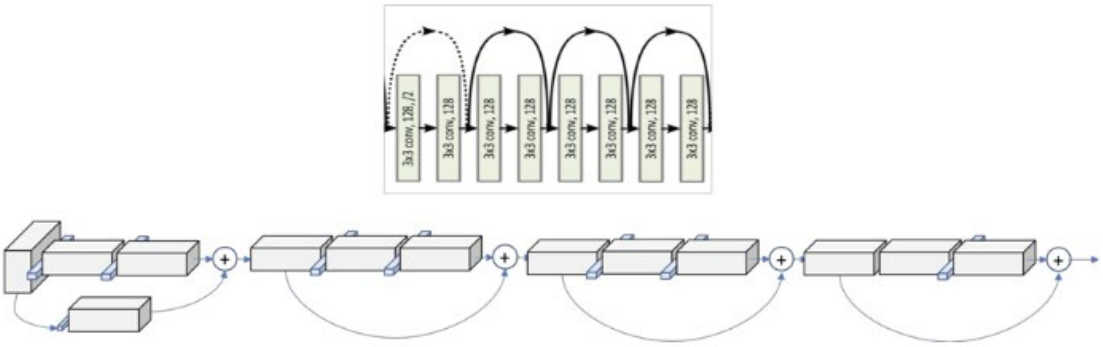


Figura 58 - Imagen global del conjunto de capas [32]

5.2.1.4 Caso Jetbot

Para nuestro caso el input será una imagen que hemos obtenido anteriormente de la pista, se procederá a realizar una transformación de la imagen para que sea del tamaño de 224x224, se normalizará y se realizará también unos cambios en el brillo, contraste y saturación, para obtener más inputs. Nuestra salida serán las coordenadas X, Y que previamente hemos ido guardando de cada imagen. Con esto se comenzará a entrenar a la red neuronal que tendrá una capa final "*full connect*" con 512 de dimensión del mapa de características de entrada y una dimensión de característica de salida.

Una vez entrenada, obtenemos un modelo el cual usaremos para cuando el robot obtenga de nuevo imágenes, estás sean preprocesadas para poder hacer cálculos en tiempo real y después ejecutadas por el modelo para saber cuáles son las coordenadas a las cuales el robot debe ir en base a la imagen nueva obtenida.

5.2.2 ResNet34

El funcionamiento de la red neuronal convolucional residual ResNet34 es similar a la red que se ha visto en el apartado anterior con la diferencia de que esta red contiene más capas, en concreto 34, a diferencia de la anterior que tenía 18. También se diferencia en el procesamiento ya que la ResNet 18 puede llegar a 1.8 billones de operaciones de coma flotante o FLOPS.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figura 59 - Comparativa entre las diferentes redes residuales ResNet [32]

Esto no garantiza que el resultado vaya a ser mejor como se ha explicado anteriormente y como muestran los resultados en la sección de pruebas de campo.

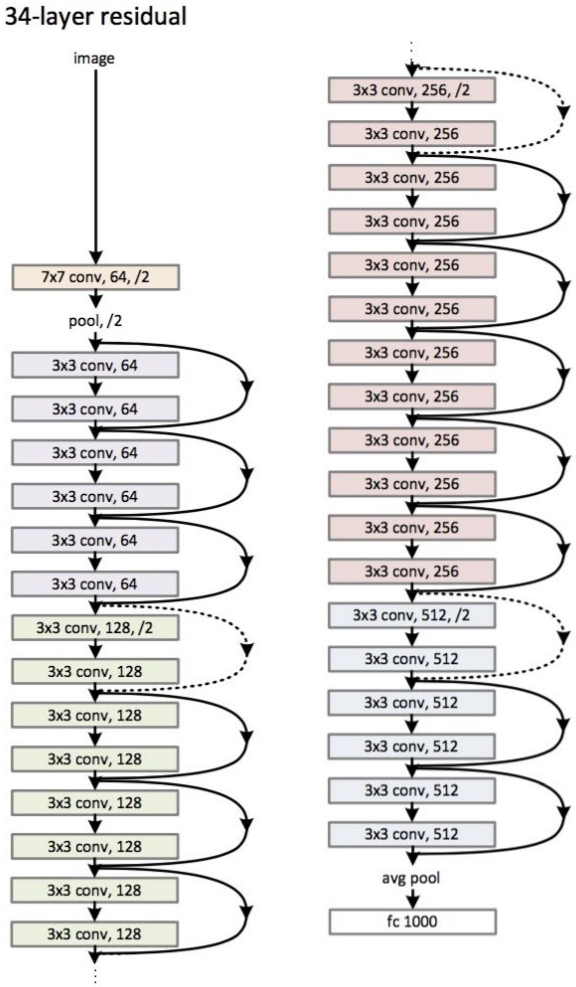


Figura 60 - Arquitectura ResNet34 [32]

5.2.3 Alexnet

Alexnet es una red neuronal convolucional, al igual que la que se ha visto en el punto 5.2.1.y fue la primera red neuronal que utilizó una GPU para acelerar el proceso.

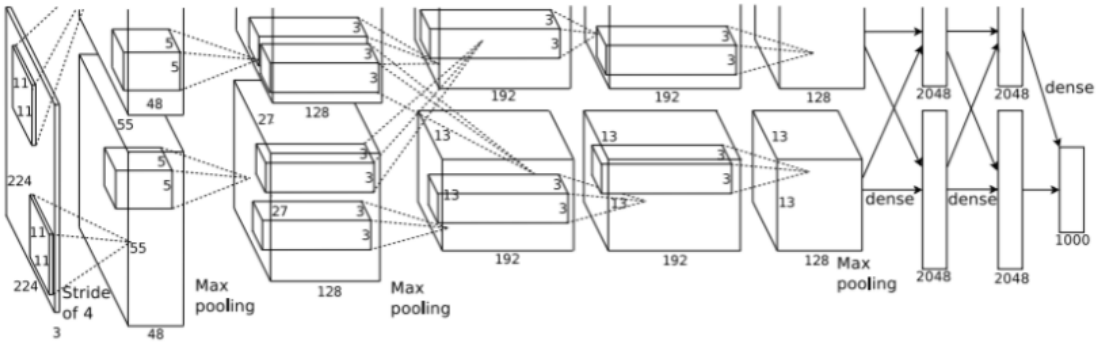


Figura 61 - Arquitectura de la red neuronal Alexnet [33]

La arquitectura de Alexnet comprende 8 capas con sus correspondientes pesos constituidas por lo siguiente:

- 5 capas convolucionales
  - 3 capas con max-pooling
  - 2 capas de normalización
- 2 capas completamente conectadas
- 1 capa softmax

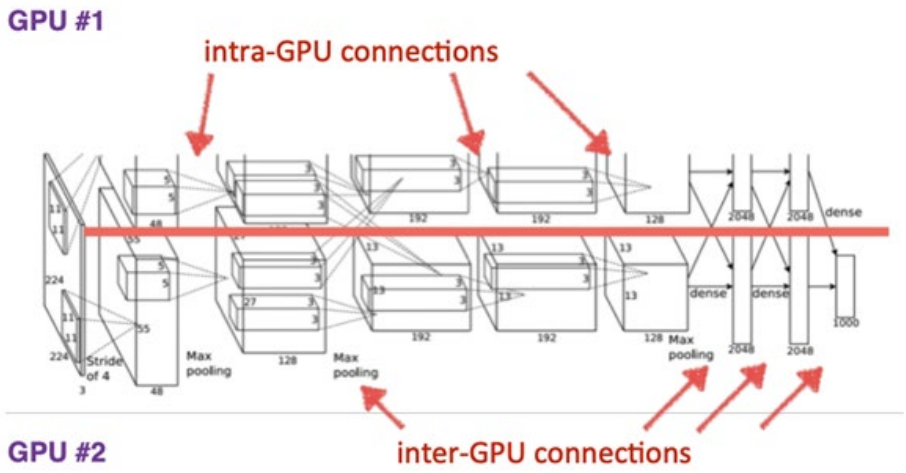


Figura 62 - Arquitectura de la red neuronal Alexnet (2)

Algunas de las características de esta red son:

- Cada capa convolucional está constituida por filtros convolucionales y por una función de activación no lineal ReLu.
- Las capas de agrupación o “pooling” realmente utilizan una maximización solapada que agrupa las reducciones de dimensionalidad.
- El tamaño de la entrada es fijo debido a la presencia al final de capas totalmente conectadas, y esta entrada tiene un volumen de 224x224x3, pero al rellenar algunos huecos queda con el siguiente volumen 227x227x3.
- Tiene 60 millones de parámetros

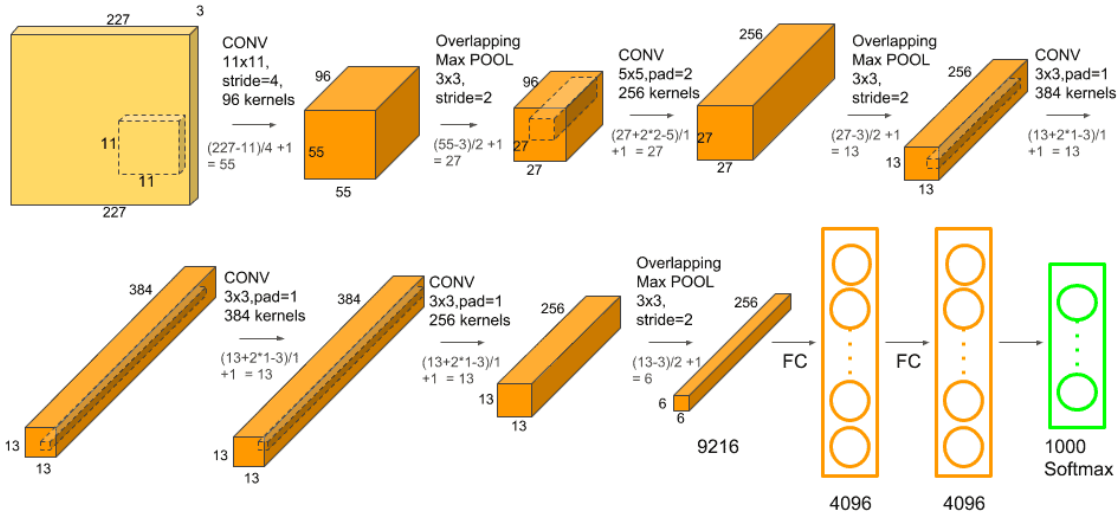


Figura 63 - Capas de la red neuronal Alexnet [34]

La técnica de agrupación o “max-pooling” consiste permite reducir la dimensionalidad y hacer suposiciones sobre las características contenidas en cada una de las regiones en las que se divide la imagen.

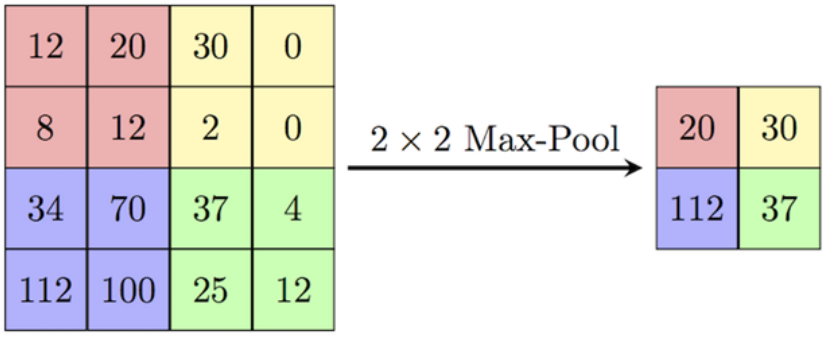


Figura 64 - Reducción de dimensionalidad o “max pooling” [35]

La parte de solapamiento lo que hace es superponer las ventanas, como vemos en la Figura 63, adyacentes sobre las que se calcula esa reducción de dimensionalidad entre sí [35].

Otra de las técnicas usadas en esta red neuronal es la no linealidad ReLu que sirve para entrar mucho más rápido una red CNN que otras que utilizan funciones de activación por saturación como Tanh o Sigmodi [35].

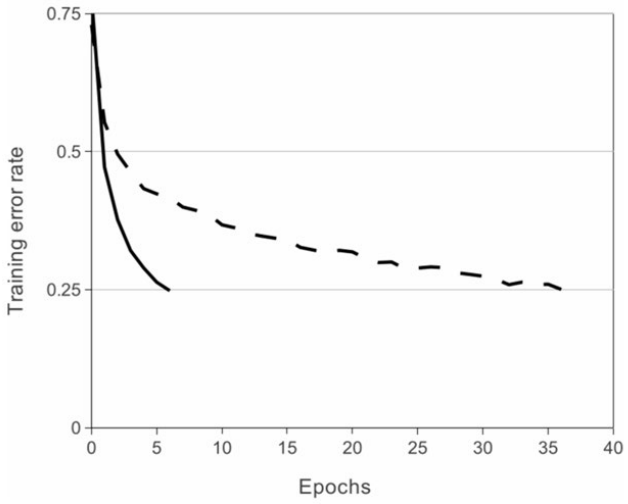


Figura 65 - CNN vs Tanh [33]

La gráfica superior muestra esta mejora en cuanto a la rapidez de entrenamiento en comparación con una Tanh.

Esta red neuronal al tener 60 millones de parámetros tiene problemas con el sobreentrenamiento. Para reducir esto se recurren a dos métodos:

Aumento de datos: se aumenta el conjunto de datos con las imágenes recogidas, creando espejo de esas imágenes, recortándolas de forma aleatoria, cambiando la luminosidad o la saturación de color. De esta forma disponemos de un conjunto mayor de imágenes para entrenar la red neuronal y evitar el "overfitting" [35].

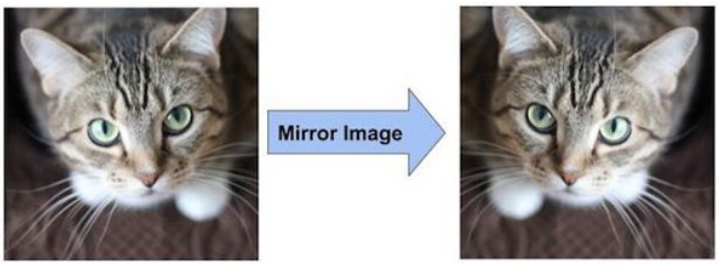


Figura 66 - Imagen espejo [35]

Abandono o "dropout": este método consiste en el abandono de neuronas que tienen una probabilidad de 50%. Cuando una neurona es descartada o abandonada, esta deja de contribuir

tanto en la propagación hacía delante como en la regresión. Cada entrada a la red neuronal pasa por una arquitectura diferente, como se muestra en la próxima figura, esto hace que se dé como resultado que los parámetros de peso de cada neurona aprendidos sean más robustos y eviten ser sobreentrenadas [35].

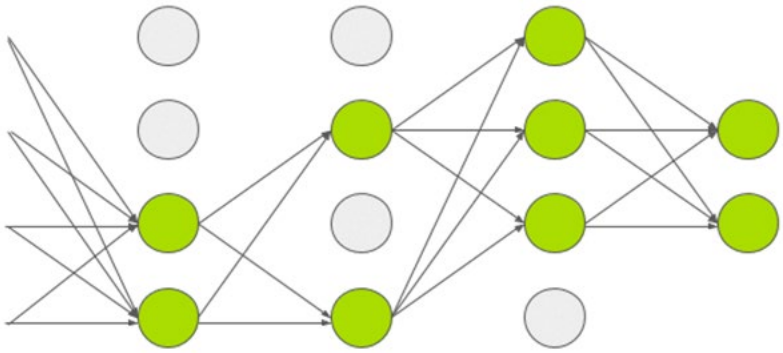


Figura 67 - "Dropout" [35]

Capítulo 6. RESULTADOS DE ENTRENAMIENTO

En este capítulo se muestran los resultados que se han obtenido al entrenar los distintos modelos de redes neuronales para las dos acciones que va a realizar el robot que son, el seguimiento de carretera y evitar la colisión con obstáculos.

Antes de entrar en materia con los resultados se van a describir algunos términos que se utilizarán más adelante.

Época: Se trata de un ciclo a través del conjunto de datos que se utilizan para entrenar una red neuronal. Está compuesto de lotes de datos e iteraciones.

Batch: Se refiere al lote o conjunto de datos. Es el número de muestras con las que se entrena la red neuronal antes de cambiar al ciclo siguiente y volver a ajustar los parámetros internos del modelo.

Subentrenamiento: El Subentrenamiento o “underfitting” se produce cuando un modelo se entrena con pocos datos y la red neuronal no es capaz de aprender adecuadamente por falta de muestras y en consecuencia no generaliza el conocimiento.

Sobreentrenamiento: El sobreentrenamiento u “overfitting” se produce cuando un modelo se entrena con muchos datos, y la red neuronal comienza a aprender del ruido o entradas de datos imprecisas. Al final, el modelo no categoriza bien esos datos debido al exceso de detalles y ruido.

Training: Se refiere a la parte en la que la red neuronal realiza el aprendizaje a través del conjunto de datos y durante unas épocas determinadas para obtener el modelo con los parámetros internos ajustados.

Test: El test se refiere a un lote o conjunto de datos que se utiliza para conocer la precisión del modelo entrenado. Este lote no puede formar parte del conjunto de datos de entrenamiento.

Loss: Se trata de la función de pérdida o “loss”, la cual evalúa la desviación entre las predicciones realizadas por la red neuronal y los valores reales de las observaciones utilizadas durante el aprendizaje. Cuanto menor sea este resultado más eficiente es la red neuronal, para conseguir esto lo que se hace es ajustar los parámetros internos del modelo.

Error = Prediccion_real - Prediccion_realizada

Descenso del gradiente: El descenso del gradiente es el proceso de minimizar la función de error ajustando poco a poco los parámetros de aprendizaje o pesos.

6.1 Seguimiento de carretera

Para que el robot llegue a seguir una ruta marcada por una carretera se han realizado 6 pruebas con diferentes algoritmos, parámetros, y conjuntos de datos. Estas pruebas fueron necesarias para crear un modelo adecuado, pero también para responder a las siguientes cuestiones:

- ¿Hay alguna diferencia entre entrenar en la propia Jetson nano o en un portátil?
- ¿Hay diferencias en el entrenamiento si cambiamos el número de épocas?
- ¿Afecta poner el modo espejo a los entrenamientos de la red?
- ¿Existe diferencia entre una red ResNet 18 y una red ResNet34?
- ¿Hay diferencias en el entrenamiento con un mayor conjunto de datos?
- ¿Mejora el resultado usar una red neuronal preentrenada?

6.1.1 Prueba 1 – Entrenamiento en CPU y en GPU (cuda)

En la primera prueba se han realizado dos entrenamientos, uno en la propia placa Jetson nano con GPU y otro en un portátil. Los dos entrenamientos han usado los siguientes parámetros:

Dispositivo	Red Neuronal	Conjunto de datos	Epoch	Espejo	Preentrenamiento
GPU	ResNEt18	1000	70	No	Sí
CPU	ResNEt18	1000	70	No	Sí

Tabla 2 - Parámetros configurados para la prueba 1

6.1.1.1 Resultados

Con la GPU:

Epoch	Training	Test
1	0,151585	0,07247
2	0,06989	0,104914
3	0,067393	0,104326
4	0,063299	0,078047
5	0,066668	0,081172
6	0,067656	0,195719
7	0,063348	0,06358
8	0,060589	0,071445
9	0,061358	0,077722
10	0,061984	0,062778
11	0,061318	0,066096
12	0,062948	0,060804
13	0,0593	0,058684

14	0,059647	0,059658
15	0,061707	0,068918
16	0,062425	0,071479
17	0,063628	0,055777
18	0,060559	0,059179
19	0,056218	0,065029
20	0,059071	0,064302
21	0,066449	0,054255
22	0,060595	0,056999
23	0,059456	0,084257
24	0,060121	0,061235
25	0,061788	0,060639
26	0,058468	0,064616
27	0,059313	0,064582
28	0,057822	0,056415
29	0,056626	0,064263
30	0,061418	0,057573
31	0,05925	0,07833
32	0,059084	0,059975
33	0,056962	0,072146
34	0,057411	0,068247
35	0,057957	0,064058
36	0,055654	0,086564
37	0,057772	0,065196
38	0,056474	0,07382
39	0,058641	0,07574
40	0,056057	0,069698
41	0,055082	0,062985
42	0,055973	0,063418
43	0,057687	0,097831
44	0,057338	0,552218
45	0,063666	0,184236
46	0,061752	0,057914
47	0,062053	0,169097
48	0,063335	0,080282
49	0,065202	0,067925
50	0,064711	0,085596
51	0,060953	0,066571
52	0,059287	0,078263
53	0,060033	0,059684
54	0,060431	0,06097
55	0,058826	0,0566
56	0,058321	0,059294
57	0,058168	0,087273
58	0,058817	0,131715
59	0,059683	0,084656

60	0,058161	0,062275
61	0,057821	0,102471
62	0,06001	0,059972
63	0,05987	0,058768
64	0,061143	0,059943
65	0,058684	0,061556
66	0,057315	0,068681
67	0,05673	0,059937
68	0,055667	0,05712
69	0,054519	0,058631
70	0,054593	0,057927

Tabla 3 - Resultado prueba 1 CPU

Con la CPU:

Epoch	Training	Test
1	0,162972	0,075903
2	0,068902	0,091611
3	0,067033	0,067062
4	0,066675	0,070201
5	0,061701	0,06814
6	0,061857	0,067087
7	0,062696	0,062729
8	0,060529	0,066659
9	0,060086	0,061842
10	0,064902	0,072303
11	0,061025	0,060687
12	0,060207	0,060668
13	0,059312	0,065296
14	0,06184	0,090944
15	0,058747	0,060523
16	0,06378	0,059285
17	0,058605	0,059035
18	0,059909	0,066658
19	0,059577	0,065819
20	0,060076	0,062438
21	0,057111	0,054617
22	0,061624	0,059402
23	0,060397	0,067045
24	0,057294	0,06655
25	0,057632	0,065277
26	0,061223	0,069972
27	0,056549	0,06416
28	0,057717	0,068168
29	0,060756	0,059456

30	0,066399	0,065396
31	0,058229	0,08235
32	0,06011	0,060439
33	0,05737	0,081527
34	0,055006	0,059078
35	0,055799	0,075619
36	0,057212	0,268725
37	0,056732	0,066984
38	0,061926	0,062072
39	0,057877	0,05953
40	0,056664	0,061049
41	0,056093	0,067132
42	0,057985	0,067716
43	0,060664	0,069577
44	0,060726	0,123034
45	0,057715	0,060912
46	0,056406	0,059602
47	0,057545	0,055924
48	0,058039	0,071122
49	0,060134	0,07584
50	0,054938	0,062323
51	0,054903	0,067925
52	0,054686	0,072406
53	0,060447	0,084703
54	0,058779	0,055844
55	0,057251	0,064887
56	0,055405	0,06595
57	0,058459	0,070334
58	0,058759	0,07199
59	0,058776	0,08877
60	0,055132	0,065712
61	0,056491	0,060435
62	0,057004	0,060708
63	0,056397	0,065814
64	0,056942	0,068029
65	0,057477	0,060932
66	0,05914	0,060586
67	0,054457	0,070266
68	0,055515	0,065344
69	0,056766	0,062326
70	0,054895	0,062364

Tabla 4 - Resultado prueba 1 CPU

6.1.1.2 Gráficas

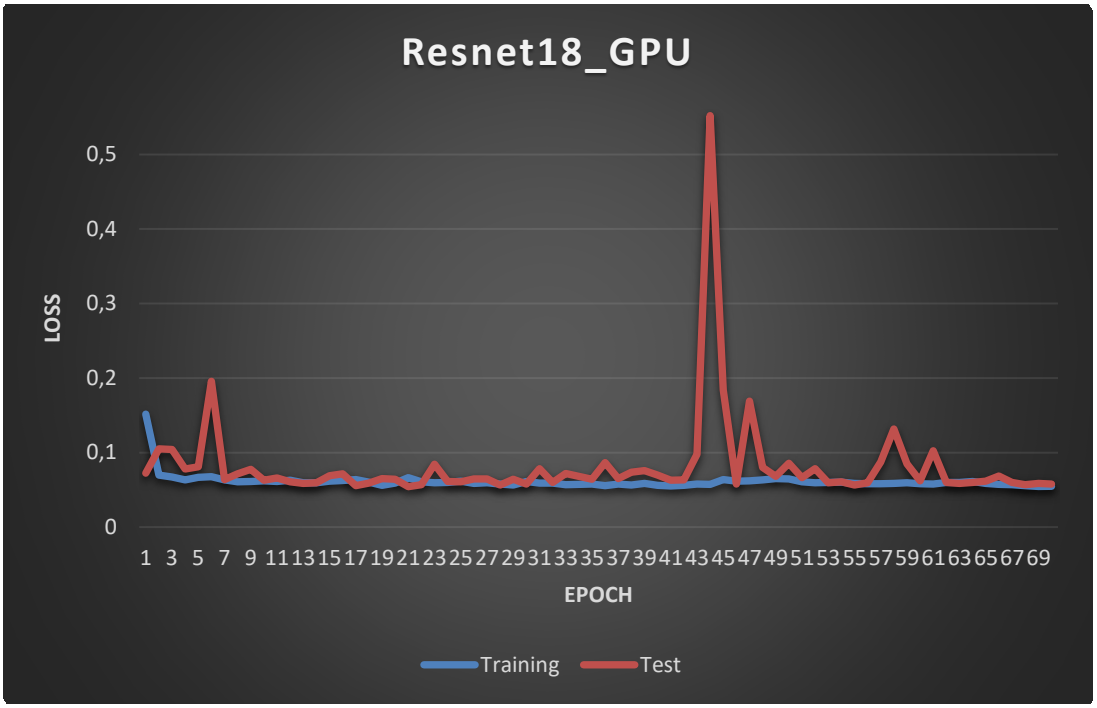


Figura 68 - Gráfica prueba 1 GPU- Training vs Test

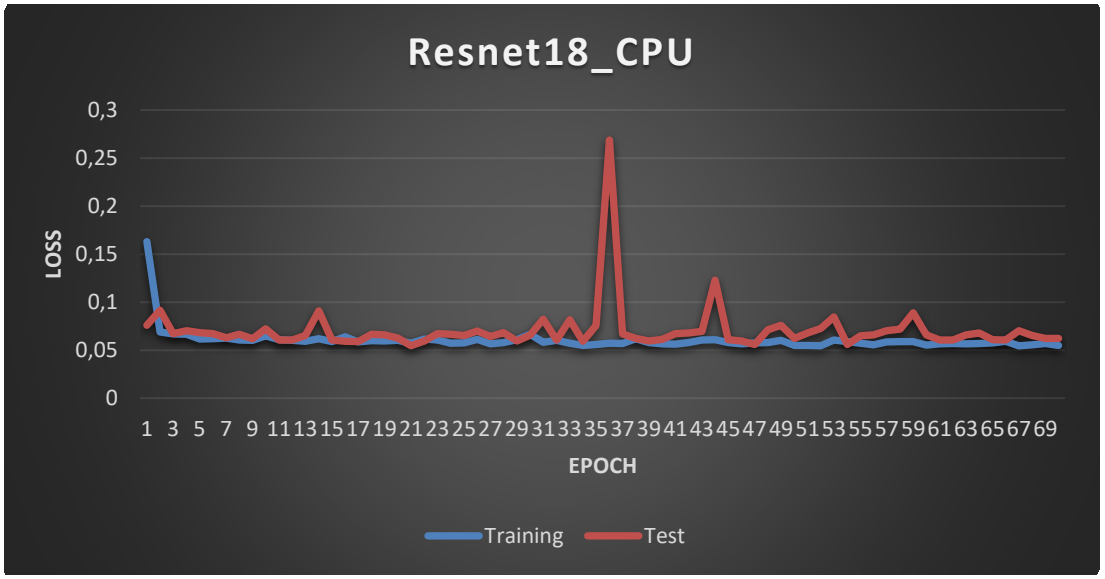


Figura 69 – Gráfica prueba 1º CPU - Training vs Test

6.1.2 Prueba 2 – Modo espejo

En esta segunda prueba se ha activado el parámetro de espejo, el cual aumenta el conjunto de datos con imágenes invertidas. Los parámetros con los que se han configurado las redes neuronales son los siguientes:

Dispositivo	Red Neuronal	Conjunto de datos	Epoch	Espejo	Preentrenamiento
GPU	ResNet18	2100	50	Sí	Sí
GPU	ResNet18	2100	50	No	Sí

Tabla 5 - Parámetros configurados para la prueba 2

6.1.2.1 Resultados

Sin espejo:

Epoch	Training	Test
1	0,248029	0,049809
2	0,029239	0,031246
3	0,03118	0,040139
4	0,024817	0,030735
5	0,029311	0,033801
6	0,019051	0,024947
7	0,02071	0,021829
8	0,018255	0,021678
9	0,02098	0,021556
10	0,017938	0,038349
11	0,019411	0,016755
12	0,017079	0,017902
13	0,013037	0,022217
14	0,019781	0,014092
15	0,014771	0,023205
16	0,014178	0,018057
17	0,015006	0,017071
18	0,012395	0,020992
19	0,012376	0,028241
20	0,012794	0,01683
21	0,011522	0,018453
22	0,009961	0,016816
23	0,010321	0,013394
24	0,010722	0,019075
25	0,012406	0,032233
26	0,014899	0,015226
27	0,012704	0,020798
28	0,010891	0,017179
29	0,008577	0,024381

30	0,010278	0,025289
31	0,009666	0,013522
32	0,00821	0,019879
33	0,008028	0,01639
34	0,007186	0,018574
35	0,007795	0,020603
36	0,010551	0,018049
37	0,011129	0,01752
38	0,007905	0,017374
39	0,008132	0,027055
40	0,007421	0,013892
41	0,008117	0,025053
42	0,008619	0,017777
43	0,006769	0,015164
44	0,006076	0,015958
45	0,006449	0,018935
46	0,006415	0,016642
47	0,007348	0,018736
48	0,007054	0,021301
49	0,008018	0,019467
50	0,007342	0,02148

Tabla 6 - Resultados prueba 2 sin espejo

Con espejo:

Epoch	Training	Test
1	0,284279	0,040494
2	0,038837	0,033642
3	0,030268	0,022545
4	0,029627	0,017214
5	0,028965	0,021502
6	0,024969	0,017036
7	0,025629	0,025969
8	0,024978	0,026138
9	0,022481	0,025165
10	0,020202	0,030149
11	0,020838	0,019113
12	0,019326	0,02217
13	0,019321	0,017431
14	0,016298	0,017944
15	0,016489	0,01917
16	0,018032	0,024825
17	0,01659	0,023444

18	0,012908	0,01629
19	0,014487	0,016781
20	0,013676	0,031183
21	0,013609	0,023245
22	0,011508	0,015957
23	0,010771	0,021627
24	0,012704	0,021787
25	0,011324	0,018438
26	0,010976	0,015225
27	0,01173	0,021378
28	0,011768	0,017022
29	0,012032	0,034285
30	0,016091	0,021799
31	0,008712	0,019074
32	0,008493	0,017071
33	0,008208	0,013835
34	0,007178	0,019483
35	0,007841	0,023226
36	0,008313	0,017022
37	0,009215	0,015774
38	0,008875	0,017762
39	0,007456	0,017942
40	0,007301	0,021622
41	0,006976	0,017802
42	0,006819	0,017636
43	0,006044	0,020398
44	0,006125	0,017497
45	0,006535	0,018416
46	0,008084	0,018673
47	0,00719	0,021113
48	0,006346	0,019415
49	0,007521	0,018522
50	0,007518	0,019769

Tabla 7 - Resultados prueba 2 con espejo

6.1.2.2 Gráficas

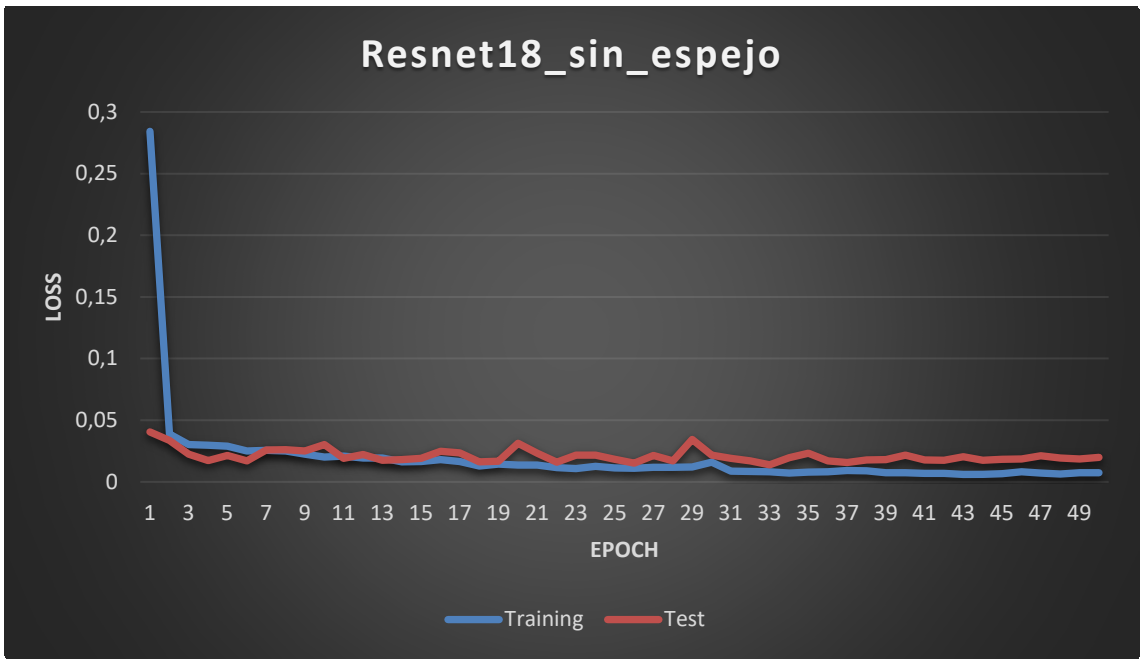


Figura 70 - Gráfica prueba 2 sin espejo - Training vs Test

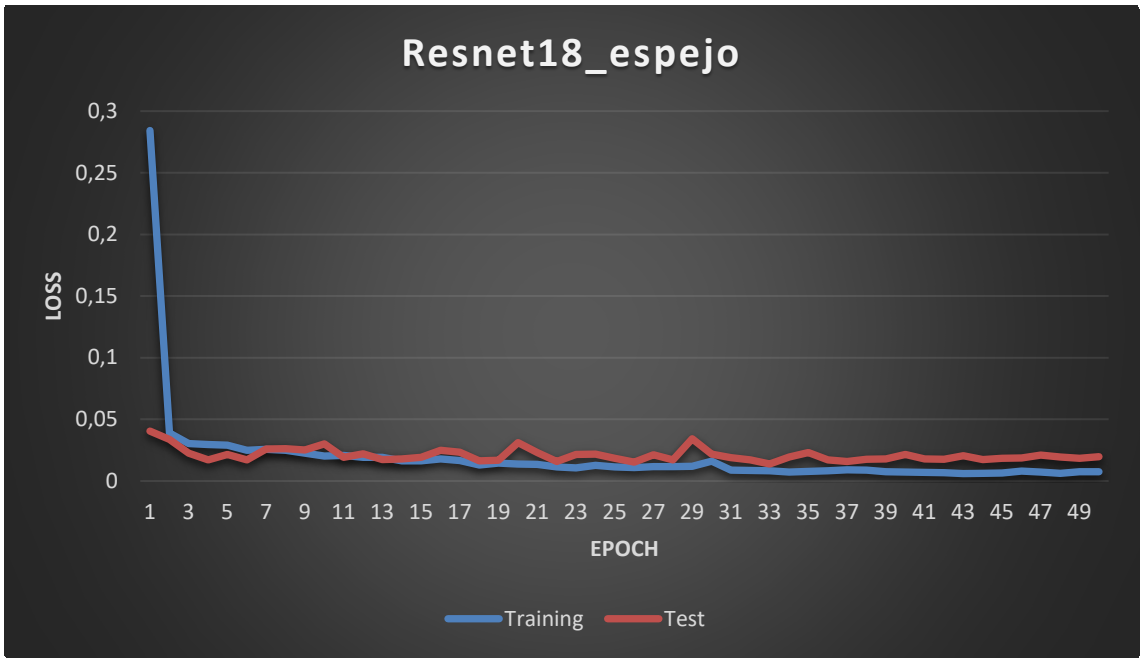


Figura 71 - Gráfica prueba 2 con espejo - Training vs test

6.1.3 Prueba 3 – Diferencia de épocas

En la prueba 3 se configuraron dos redes neuronales distintas para que el número de épocas del entrenamiento fuese distinto y saber si se estaba sobreentrenando o por el contrario todavía se podía entrenar más la red neuronal. Los parámetros de las redes neuronales que se han utilizado son los siguientes:

Dispositivo	Red Neuronal	Conjunto de datos	Epoch	Espejo	Preentrenamiento
GPU	ResNet18	2100	50	No	Sí
GPU	ResNet18	2100	70	No	Sí

Tabla 8 - Parámetros configurados para la prueba 3

6.1.3.1 Resultados

Con 50 épocas:

Epoch	Training	Test
1	0,186685	0,028807
2	0,037706	0,021449
3	0,026824	0,028508
4	0,026579	0,015137
5	0,022153	0,036017
6	0,025608	0,02721
7	0,020287	0,017594
8	0,018693	0,015748
9	0,017273	0,025249
10	0,016175	0,016788
11	0,014639	0,019156
12	0,015808	0,016188
13	0,015016	0,016437
14	0,012856	0,01814
15	0,013982	0,02145
16	0,013963	0,012832
17	0,011489	0,017864
18	0,012324	0,018492
19	0,01012	0,016622
20	0,010645	0,016489
21	0,009796	0,016687
22	0,010478	0,016365
23	0,009554	0,013662
24	0,013481	0,01414
25	0,010906	0,01844
26	0,011169	0,012117
27	0,012159	0,014013

28	0,009303	0,016372
29	0,009797	0,014278
30	0,009785	0,013456
31	0,008001	0,012177
32	0,007126	0,015117
33	0,008095	0,014899
34	0,00803	0,015425
35	0,006387	0,014656
36	0,006911	0,015671
37	0,009581	0,015165
38	0,006767	0,014744
39	0,007359	0,016649
40	0,006399	0,016443
41	0,005715	0,019288
42	0,005462	0,014729
43	0,005941	0,013035
44	0,006866	0,021983
45	0,008655	0,016417
46	0,007846	0,018666
47	0,00722	0,023587
48	0,005944	0,022698
49	0,00668	0,014624
50	0,006065	0,015834

Tabla 9 - Resultados prueba 3 con 50 épocas

Con 70 épocas:

Epoch	Training	Test
1	0,094734	0,03655
2	0,03514	0,024643
3	0,029457	0,020157
4	0,027485	0,028917
5	0,023555	0,033313
6	0,02194	0,030689
7	0,020668	0,021907
8	0,021004	0,020696
9	0,0194	0,013047
10	0,01786	0,030208
11	0,016945	0,038716
12	0,015721	0,028185
13	0,015377	0,021301
14	0,021916	0,099451
15	0,017987	0,025079
16	0,016292	0,036355
17	0,014639	0,057905

18	0,014845	0,022377
19	0,012907	0,014333
20	0,012688	0,030101
21	0,011439	0,026828
22	0,012405	0,020095
23	0,012945	0,019917
24	0,011735	0,016649
25	0,010459	0,015046
26	0,009956	0,018292
27	0,012009	0,04447
28	0,019615	0,016738
29	0,016106	0,017898
30	0,011758	0,017659
31	0,009481	0,018385
32	0,009183	0,016992
33	0,008357	0,022631
34	0,00818	0,027916
35	0,007493	0,01837
36	0,0061	0,023106
37	0,007946	0,017653
38	0,007181	0,013991
39	0,006991	0,020327
40	0,005888	0,016784
41	0,005512	0,026599
42	0,006067	0,023411
43	0,00549	0,017416
44	0,009025	0,022128
45	0,006809	0,016863
46	0,004808	0,019356
47	0,004925	0,025057
48	0,004391	0,017986
49	0,004289	0,022074
50	0,004104	0,021824
51	0,004371	0,015648
52	0,005322	0,016532
53	0,004293	0,01851
54	0,003708	0,01531
55	0,003569	0,019402
56	0,005716	0,021929
57	0,006087	0,018263
58	0,004008	0,018048
59	0,00327	0,015523
60	0,002833	0,017058
61	0,003163	0,019641
62	0,003349	0,015902
63	0,003003	0,01536

64	0,004206	0,018324
65	0,003828	0,015797
66	0,003537	0,016683
67	0,002884	0,015486
68	0,002758	0,015369
69	0,002663	0,016926
70	0,003171	0,01483

Tabla 10 - Resultados prueba 3 con 70 épocas

6.1.3.2 Gráficas

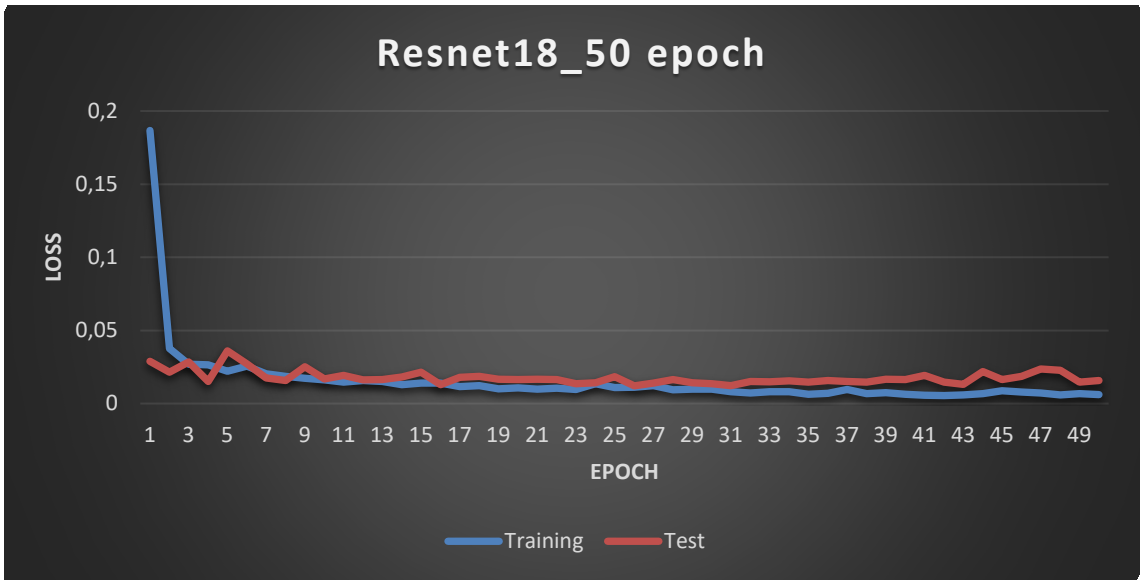


Figura 72 - Gráfica prueba 3 con 50 épocas - Training vs Test

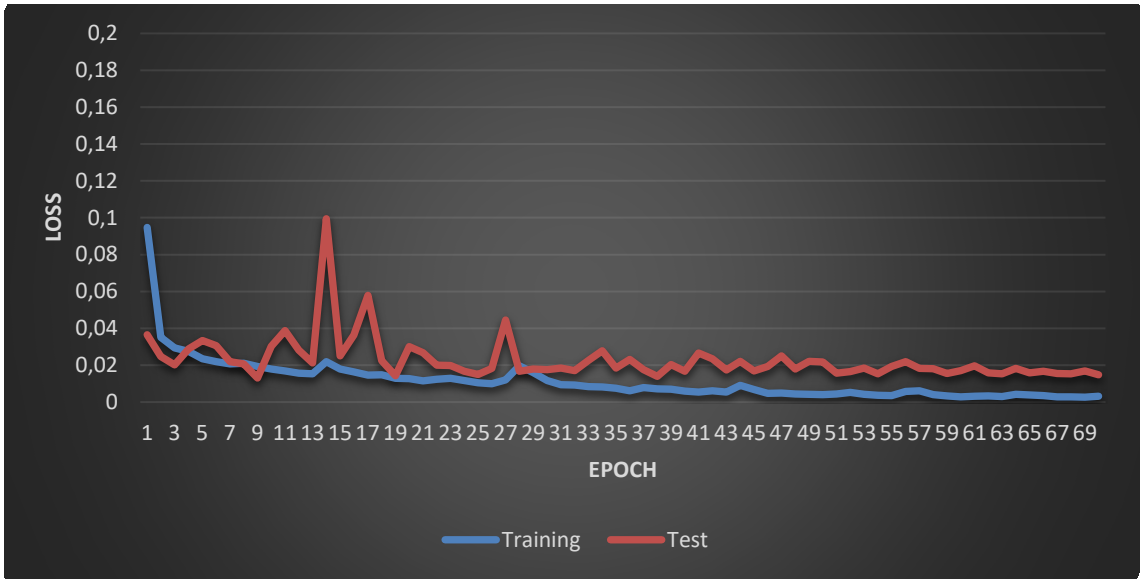


Figura 73 - Gráfica prueba 3 con 70 épocas - Training vs Test

6.1.4 Prueba 4 – Comparativa ResNet18 vs ResNet 34

En la prueba 4 se han entrenado dos tipos distintos de redes, una red resNet18 y otra red ResNet34. Los parámetros configurados para esta prueba son los siguientes:

Dispositivo	Red Neuronal	Conjunto de datos	Epoch	Espejo	Preentrenamiento
GPU	ResNet18	2100	50	No	Sí
GPU	ResNet34	2100	50	No	Sí

Tabla 11 - Parámetros de configuración de la prueba 4

6.1.4.1 Resultados

Con ResNet18:

Epoch	Training	Test
1	0,186685	0,028807
2	0,037706	0,021449
3	0,026824	0,028508
4	0,026579	0,015137
5	0,022153	0,036017

6	0,025608	0,02721
7	0,020287	0,017594
8	0,018693	0,015748
9	0,017273	0,025249
10	0,016175	0,016788
11	0,014639	0,019156
12	0,015808	0,016188
13	0,015016	0,016437
14	0,012856	0,01814
15	0,013982	0,02145
16	0,013963	0,012832
17	0,011489	0,017864
18	0,012324	0,018492
19	0,01012	0,016622
20	0,010645	0,016489
21	0,009796	0,016687
22	0,010478	0,016365
23	0,009554	0,013662
24	0,013481	0,01414
25	0,010906	0,01844
26	0,011169	0,012117
27	0,012159	0,014013
28	0,009303	0,016372
29	0,009797	0,014278
30	0,009785	0,013456
31	0,008001	0,012177
32	0,007126	0,015117
33	0,008095	0,014899
34	0,00803	0,015425
35	0,006387	0,014656
36	0,006911	0,015671
37	0,009581	0,015165
38	0,006767	0,014744
39	0,007359	0,016649
40	0,006399	0,016443
41	0,005715	0,019288
42	0,005462	0,014729
43	0,005941	0,013035
44	0,006866	0,021983
45	0,008655	0,016417
46	0,007846	0,018666
47	0,00722	0,023587
48	0,005944	0,022698
49	0,00668	0,014624
50	0,006065	0,015834

Tabla 12 - Resultados prueba 4 con ResNet18

Con ResNet34:

Epoch	Training	Test
1	0,13589	0,024041
2	0,035414	0,023235
3	0,025717	0,016802
4	0,030111	0,031826
5	0,02304	0,024797
6	0,019943	0,022918
7	0,019988	0,018503
8	0,019131	0,024073
9	0,020441	0,012073
10	0,018025	0,018735
11	0,017244	0,059923
12	0,015611	0,128107
13	0,014166	0,015338
14	0,013251	0,017503
15	0,013253	0,018333
16	0,013043	0,332699
17	0,014825	0,051176
18	0,01249	0,014913
19	0,014643	0,116213
20	0,021455	0,019855
21	0,013576	0,012589
22	0,013319	0,018869
23	0,011089	0,017559
24	0,009944	0,026369
25	0,011471	0,022005
26	0,009304	0,034953
27	0,009019	0,019735
28	0,008469	0,019096
29	0,009483	0,012134
30	0,00871	0,019607
31	0,009579	0,065183
32	0,011132	0,014986
33	0,00832	0,011296
34	0,009915	0,017926
35	0,01259	0,025303
36	0,008509	0,015678
37	0,006339	0,01304
38	0,00661	0,016081
39	0,009216	0,013367
40	0,009093	0,024456
41	0,013384	0,232278

42	0,007439	0,018261
43	0,006349	0,011251
44	0,006094	0,017346
45	0,005456	0,013255
46	0,005647	0,019102
47	0,005458	0,012902
48	0,004793	0,024721
49	0,006071	0,014983
50	0,005955	0,012419

Tabla 13 - Resultados prueba 4 con ResNet34

6.1.4.2 Gráficas

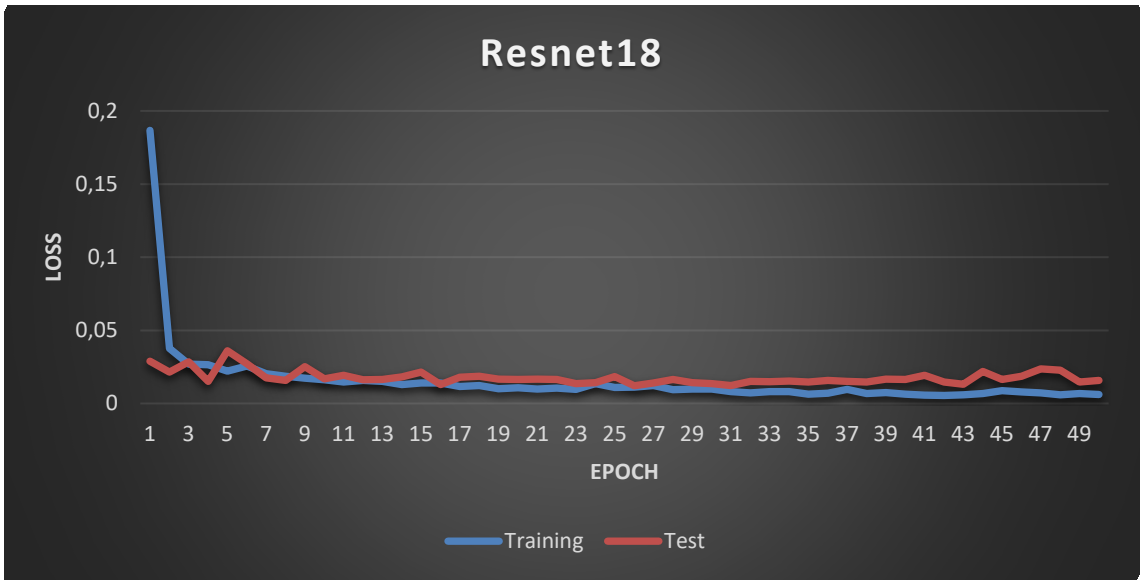


Figura 74 - Gráfica prueba 4 ResNet18 - Training vs Test

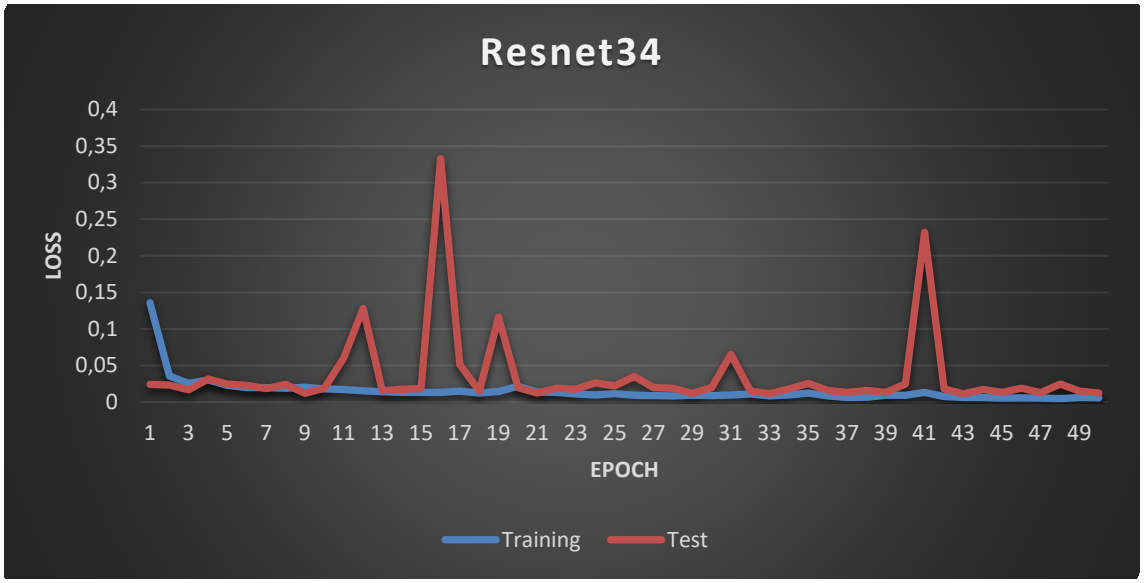


Figura 75 - Gráfica prueba 4 ResNet34 - Training vs Test

6.1.5 Prueba 5 – Diferentes conjuntos de datos

En esta prueba se ha utilizado un conjunto de datos con 1000 imágenes y otro conjunto con 2100 imágenes para entrenar un mismo tipo de red neuronal. Los parámetros configurados para esta prueba son los siguientes:

Dispositivo	Red Neuronal	Conjunto de datos	Epoch	Espejo	Preentrenamiento
GPU	ResNet18	1000	70	No	Sí
GPU	ResNet18	2100	70	No	Sí

Tabla 14 - Parámetros de configuración de la prueba 5

6.1.5.1 Resultados

Con 1000 imágenes:

Epoch	Training	Test
1	0,143226	0,063315
2	0,071053	0,067715
3	0,067549	0,07683
4	0,060915	0,096425
5	0,06296	0,083175
6	0,065062	0,060344
7	0,060275	0,065386
8	0,064713	0,057407
9	0,058005	0,062755

10	0,063316	0,074849
11	0,059332	0,060395
12	0,063633	0,070982
13	0,057433	0,090096
14	0,061504	0,078044
15	0,0648	0,066689
16	0,056819	0,061899
17	0,058249	0,060173
18	0,05918	0,069427
19	0,059517	0,075577
20	0,057882	0,068535
21	0,060432	0,060355
22	0,059342	0,066154
23	0,059145	0,063251
24	0,058732	0,068749
25	0,05901	0,085108
26	0,059892	0,062004
27	0,058492	0,062703
28	0,057521	0,076824
29	0,058564	0,069781
30	0,05867	0,068457
31	0,057556	0,069305
32	0,058941	0,06571
33	0,055558	0,062373
34	0,058099	0,072614
35	0,056619	0,064101
36	0,058991	0,071195
37	0,059897	0,064684
38	0,055822	0,075063
39	0,059376	0,079571
40	0,056354	0,063108
41	0,059995	0,058559
42	0,057393	0,059125
43	0,05581	0,067465
44	0,05701	0,064265
45	0,057345	0,066443
46	0,054957	0,06795
47	0,057773	0,09107
48	0,06042	0,070603
49	0,058227	0,064988
50	0,057948	0,071399
51	0,059753	0,069342
52	0,060255	0,059937
53	0,056497	0,060019
54	0,058912	0,067366
55	0,055281	0,06411

56	0,059939	0,074217
57	0,059467	0,064694
58	0,055495	0,06197
59	0,058225	0,071528
60	0,055319	0,065812
61	0,056601	0,061305
62	0,055352	0,061913
63	0,057204	0,070998
64	0,056151	0,060685
65	0,055673	0,061231
66	0,054456	0,065047
67	0,057073	0,06413
68	0,057766	0,072655
69	0,056963	0,066431
70	0,055668	0,065731

Tabla 15 - Resultados prueba 5 con 1000 imágenes

Con 2100 imágenes:

Epoch	Training	Test
1	0,099363	0,054069
2	0,049616	0,045601
3	0,050843	0,049286
4	0,047807	0,046851
5	0,048815	0,04596
6	0,047837	0,04732
7	0,046525	0,052954
8	0,048273	0,046639
9	0,048443	0,049062
10	0,045374	0,055263
11	0,048474	0,056493
12	0,045964	0,048133
13	0,045642	0,046729
14	0,045801	0,060658
15	0,045091	0,058672
16	0,043621	0,043369
17	0,048127	0,049114
18	0,048365	0,067289
19	0,046277	0,052454
20	0,044621	0,047568
21	0,045204	0,050086
22	0,044651	0,051893
23	0,046677	0,049875
24	0,044501	0,049263
25	0,044752	0,050308

26	0,044747	0,046207
27	0,044473	0,047401
28	0,045561	0,06233
29	0,047905	0,077457
30	0,044439	0,058493
31	0,043189	0,046444
32	0,044178	0,060625
33	0,045842	0,05006
34	0,042744	0,050125
35	0,043679	0,050028
36	0,041941	0,049156
37	0,042284	0,04727
38	0,043788	0,049404
39	0,043618	0,052507
40	0,041528	0,049379
41	0,042043	0,050507
42	0,042069	0,048269
43	0,040889	0,047447
44	0,041592	0,047974
45	0,041638	0,054213
46	0,0432	0,058452
47	0,041021	0,05469
48	0,04324	0,071548
49	0,041977	0,056316
50	0,041882	0,049694
51	0,040511	0,04714
52	0,040102	0,051035
53	0,038774	0,053135
54	0,039617	0,052884
55	0,039163	0,062055
56	0,039358	0,102624
57	0,039538	0,048829
58	0,038682	0,065265
59	0,038449	0,050003
60	0,039811	0,052256
61	0,040052	0,050409
62	0,039792	0,051188
63	0,038947	0,053264
64	0,038268	0,050806
65	0,038639	0,090571
66	0,03731	0,053829
67	0,038906	0,060539
68	0,037906	0,070937
69	0,036533	0,050825
70	0,037602	0,058744

Tabla 16 - Resultados prueba 5 con 2100 imágenes

6.1.5.2 Gráficas

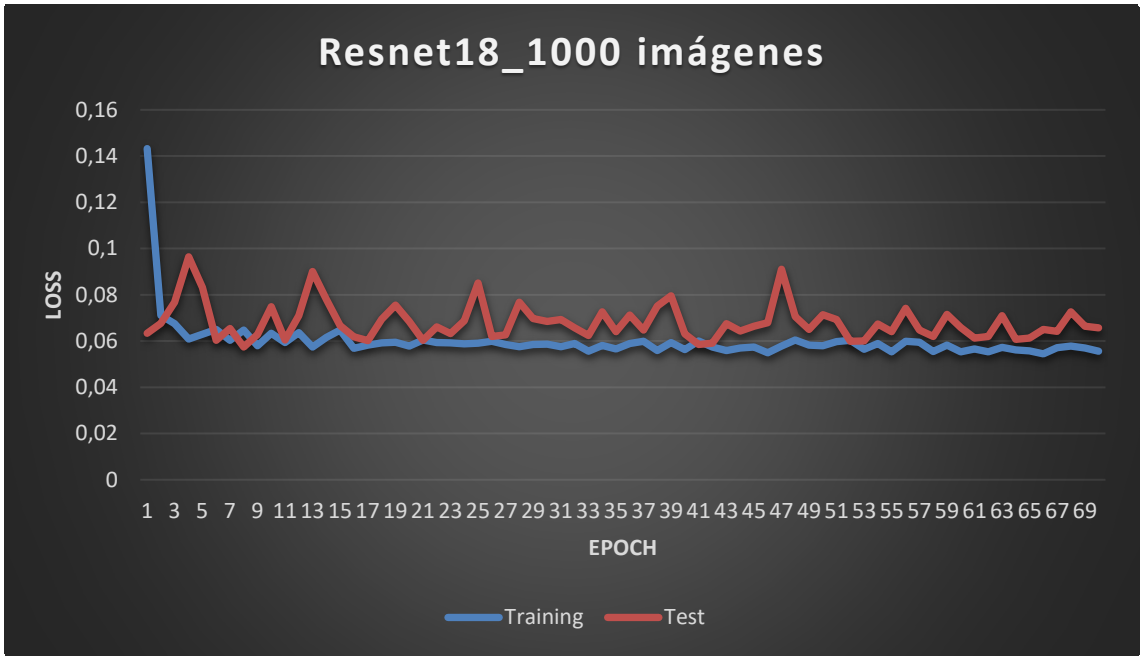


Figura 76 - Gráfica prueba 5 con 1000 imágenes - Training vs Test

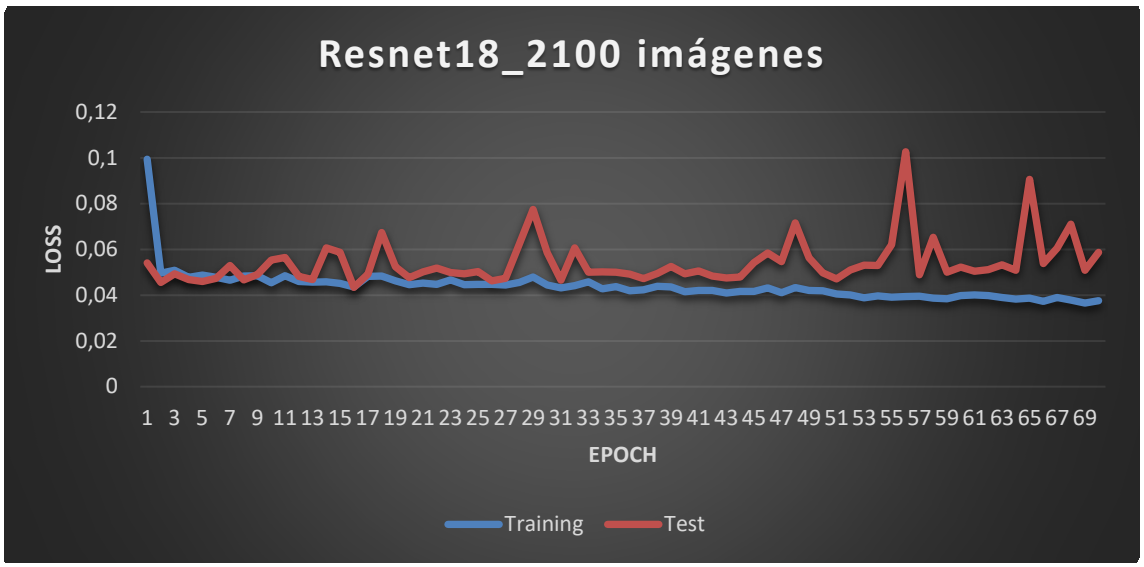


Figura 77 - Gráfica prueba 5 con 2100 imágenes - Training vs Test

6.1.6 Prueba 6 – Preentrenamiento

La prueba 6 consistió en utilizar redes neuronales que ya han sido preentrenadas, es decir que ya han sido configuradas con unos pesos determinados. Para esta prueba se han configurado las redes neuronales con los siguientes parámetros:

Dispositivo	Red Neuronal	Conjunto de datos	Epoch	Espejo	Preentrenamiento
GPU	ResNet34	1000	70	No	Sí
GPU	ResNet34	2100	70	No	No

Tabla 17 - Parámetros de configuración de la prueba 6

6.1.6.1 Resultados

Con preentrenamiento:

Epoch	Training	Test
1	0,156989	0,077117
2	0,074635	0,086268
3	0,071623	0,081031
4	0,071196	0,064523
5	0,067674	0,068856
6	0,063485	0,067258
7	0,061995	0,066049
8	0,060939	0,073225
9	0,061589	0,070741
10	0,059416	0,078962
11	0,061424	0,065207
12	0,059777	0,065901
13	0,061462	0,07077
14	0,061331	0,066766
15	0,059896	0,067701
16	0,062029	0,072274
17	0,058465	0,06693
18	0,05923	0,070331
19	0,05819	0,081069
20	0,057923	0,069151
21	0,059801	0,063387
22	0,061394	0,069882
23	0,06165	0,061887

24	0,05753	0,072525
25	0,059658	0,073604
26	0,057351	0,07157
27	0,05614	0,066381
28	0,057687	0,078617
29	0,057833	0,065306
30	0,056868	0,070638
31	0,057345	0,075363
32	0,05896	0,073075
33	0,057907	0,070156
34	0,057515	0,08153
35	0,056889	0,067008
36	0,056844	0,077277
37	0,056708	0,075538
38	0,055338	0,07346
39	0,056769	0,153505
40	0,063106	0,071727
41	0,059463	0,064188
42	0,059227	0,068551
43	0,056349	0,129644
44	0,0558	0,072851
45	0,054	0,0721
46	0,05422	0,067765
47	0,054316	0,077451
48	0,05585	0,075603
49	0,055769	0,073445
50	0,057185	0,070639
51	0,05514	0,071756
52	0,053283	0,077872
53	0,052576	0,076764
54	0,052496	0,074924
55	0,053733	0,071245
56	0,054029	0,074453
57	0,052335	0,078647
58	0,05179	0,072843
59	0,051357	0,073457
60	0,051596	0,080411
61	0,052007	0,077294
62	0,050517	0,078756
63	0,050182	0,078326
64	0,05151	0,079864
65	0,049687	0,072493
66	0,050047	0,080156
67	0,049931	0,078182
68	0,04945	0,078089
69	0,049881	0,084049

Tabla 18 - Resultado prueba 6 con preentrenamiento

Sin preentrenamiento:

Epoch	Training	Test
1	0,161307	0,078526
2	0,079183	0,077168
3	0,076803	0,111134
4	0,070481	0,068508
5	0,065948	0,062522
6	0,066252	0,102821
7	0,067102	0,076125
8	0,066037	0,071508
9	0,064002	0,064021
10	0,063839	0,071952
11	0,065417	0,102009
12	0,063038	0,069139
13	0,063987	0,065897
14	0,063862	0,067788
15	0,060988	0,07194
16	0,059334	0,070586
17	0,060189	0,075351
18	0,056976	0,060024
19	0,057192	0,074011
20	0,058413	0,076952
21	0,06017	0,065777
22	0,059828	0,083386
23	0,057723	0,075278
24	0,059502	0,0635
25	0,058428	0,509851
26	0,058721	0,071141
27	0,057611	0,088451
28	0,056717	0,076101
29	0,05828	0,066878
30	0,056059	0,067062
31	0,058582	0,068782
32	0,057031	0,067747
33	0,055861	0,068077
34	0,05525	0,076179
35	0,05534	0,065228
36	0,056215	0,182467
37	0,057991	0,07767
38	0,060381	0,076521

39	0,061385	0,067867
40	0,06146	0,072737
41	0,055968	0,079242
42	0,055431	0,071252
43	0,055291	0,071158
44	0,053917	0,070677
45	0,05359	0,070329
46	0,053064	0,067077
47	0,052803	0,068965
48	0,05357	0,069274
49	0,053344	0,070839
50	0,052918	0,08281
51	0,052059	0,076842
52	0,053189	0,074109
53	0,052924	0,08989
54	0,052313	0,093905
55	0,053093	0,071183
56	0,05193	0,070788
57	0,052839	0,075138
58	0,051332	0,06947
59	0,049877	0,088748
60	0,050798	0,07502
61	0,051695	0,075104
62	0,049335	0,079973
63	0,04994	0,079308
64	0,048929	0,08029
65	0,048917	0,082722
66	0,048948	0,079361
67	0,048502	0,080552
68	0,049771	0,084944
69	0,049551	0,092545
70	0,048176	0,082446

Tabla 19 - Resultados prueba 6 sin preentrenamiento

6.1.6.2 Gráficas

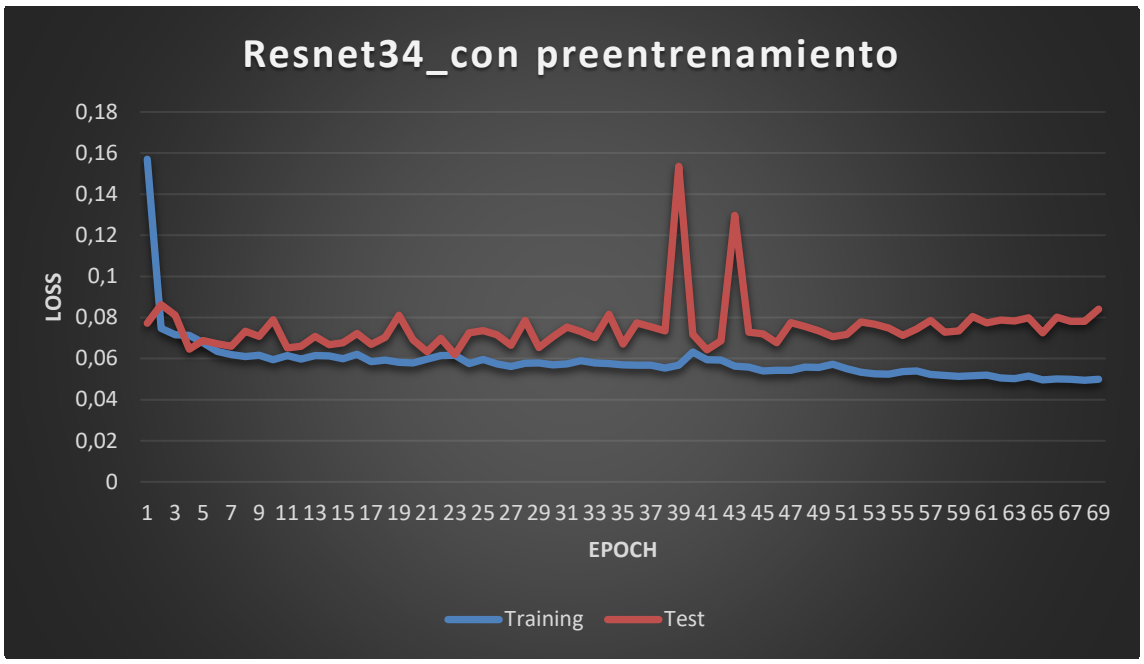


Figura 78 - Gráfica prueba 6 con preentrenamiento - Training vs Test

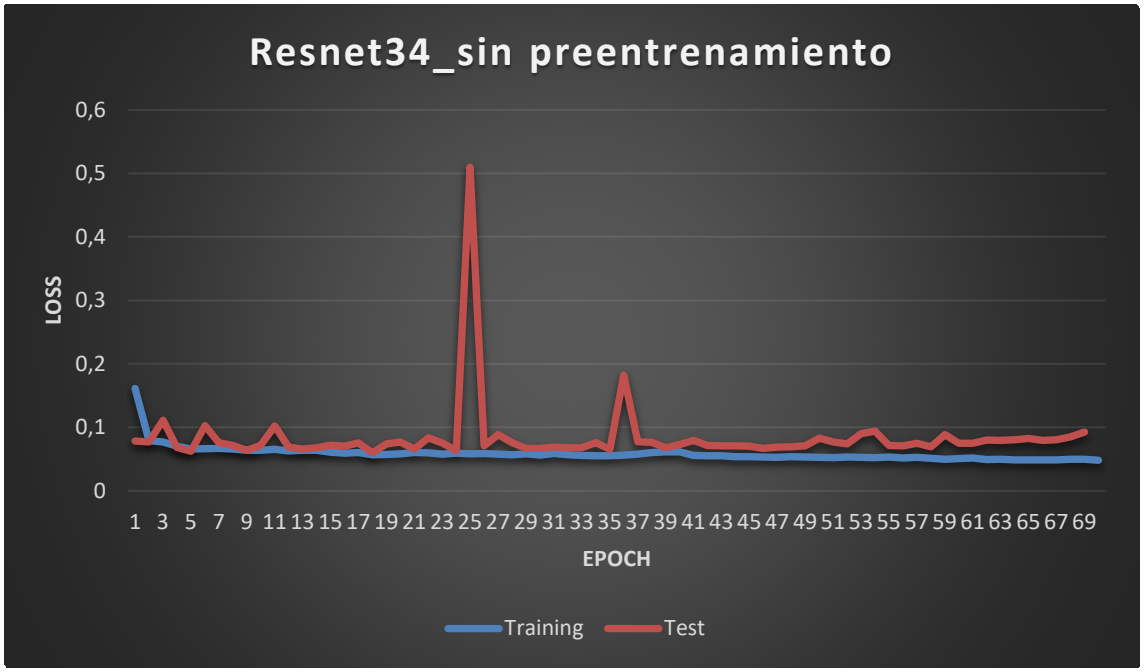


Figura 79 - Gráfica prueba 6 sin preentrenamiento - Training vs Test

6.2 Evitar la colisión con obstáculos

Para que el robot llegue a evitar los obstáculos que se encuentra durante su trayectoria se han realizado 3 pruebas con diferentes algoritmos, parámetros, y conjuntos de datos. Estas pruebas fueron necesarias para crear un modelo adecuado, pero también para responder a las siguientes cuestiones:

- ¿Hay diferencias en el entrenamiento si cambiamos el número de épocas?
- ¿Existe diferencia entre una red ResNet 18 y una red Alexnet?
- ¿Hay diferencias en el entrenamiento con un mayor conjunto de datos?
- ¿Mejora el resultado usar una red neuronal preentrenada?

6.2.1 Prueba 1 – Diferencia de épocas

En esta prueba se configuró el mismo tipo de red neuronal (Alexnet) pero con diferentes épocas, para saber si se estaba llegando al sobreentrenamiento o todavía se podía entrenar más la red neuronal. Se volvió a probar esta diferencia de épocas porque se utilizó una red neuronal diferente a las utilizadas en el seguimiento por carretera. Los parámetros configurados son los siguientes:

Dispositivo	Red Neuronal	Conjunto de datos	Epoch	Preentrenamiento
GPU	Alexnet	285	30	No
GPU	Alexnet	285	50	No

Tabla 20 - Parámetros de configuración prueba 1 evitar obstáculos

6.2.1.1 Resultados

Con 30 épocas:

Epoch	Accuracy
1	0,46
2	0,60
3	0,46
4	0,46
5	0,48
6	0,46
7	0,46
8	0,52
9	0,46
10	0,54
11	0,46
12	0,44

13	0,62
14	0,44
15	0,56
16	0,50
17	0,64
18	0,64
19	0,64
20	0,66
21	0,66
22	0,58
23	0,72
24	0,72
25	0,70
26	0,78
27	0,76
28	0,64
29	0,86
30	0,84

Tabla 21 - Resultados prueba 1 evitar colisiones 30 épocas

Con 50 épocas:

Epoch	Accuracy
1	0,38
2	0,38
3	0,38
4	0,38
5	0,38
6	0,38
7	0,38
8	0,38
9	0,38
10	0,54
11	0,40
12	0,34
13	0,58
14	0,52
15	0,56
16	0,62
17	0,62

18	0,56
19	0,48
20	0,44
21	0,52
22	0,74
23	0,78
24	0,64
25	0,78
26	0,66
27	0,84
28	0,80
29	0,78
30	0,86
31	0,80
32	0,88
33	0,78
34	0,96
35	0,88
36	0,88
37	0,92
38	0,90
39	0,88
40	0,94
41	0,94
42	0,96
43	0,72
44	0,84
45	0,94
46	0,94
47	1,00
48	0,98
49	0,94
50	0,98

Tabla 22 - Resultados prueba 1 evitar colisiones 50 épocas

6.2.1.2 Gráficas

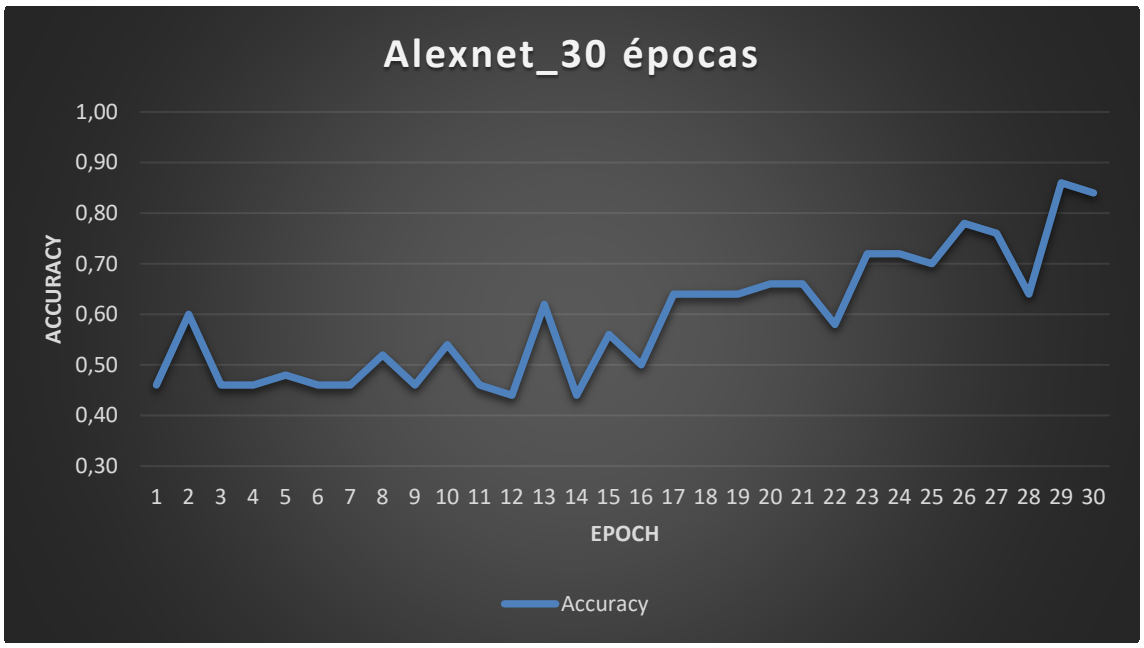


Figura 80 - Gráficas prueba 1 Alexnet 30 épocas

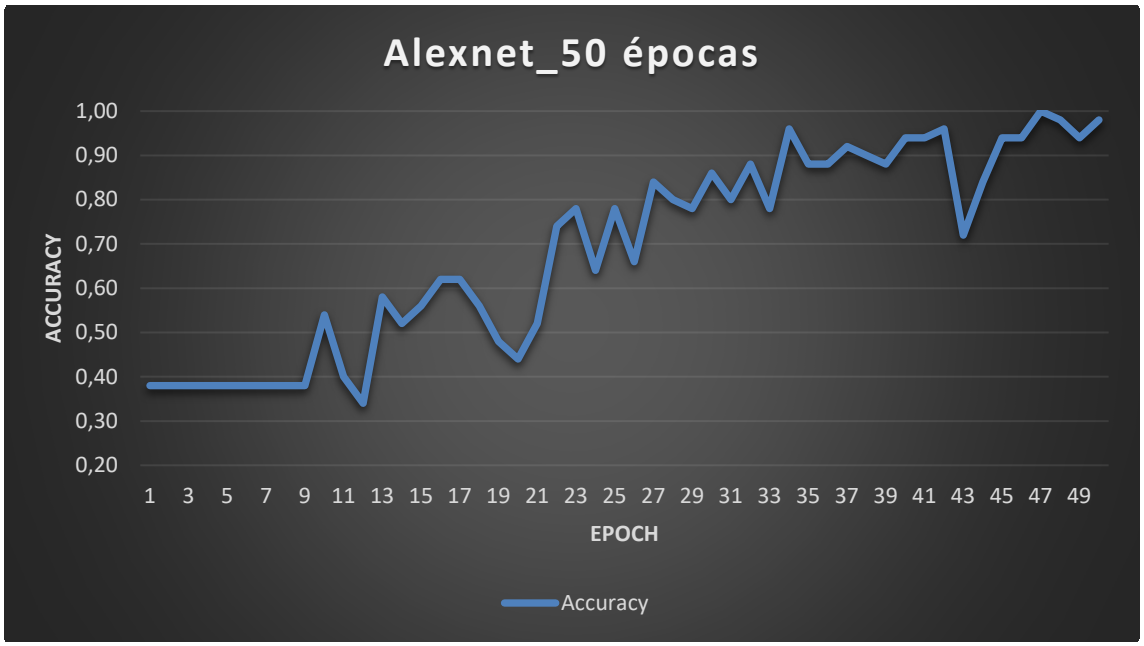


Figura 81 - Gráfica prueba 1 Alexnet 50 épocas

6.2.2 Prueba 2 – Aumento del conjunto de datos

En esta prueba se ha utilizado un conjunto de datos con 285 imágenes de obstáculos y 285 imágenes de paso libre, y otro conjunto con 450 imágenes para cada opción, para entrenar un mismo tipo de red neuronal. Los parámetros configurados para esta prueba son los siguientes:

Dispositivo	Red Neuronal	Conjunto de datos	Epoch	Preentrenamiento
GPU	Alexnet	285	30	No
GPU	Alexnet	450	30	No

Tabla 23 - Parámetros de configuración prueba 2 evitar obstáculos

6.2.2.1 Resultados

Con 285 imágenes:

Epoch	Accuracy
1	0,46
2	0,60
3	0,46
4	0,46
5	0,48
6	0,46
7	0,46
8	0,52
9	0,46
10	0,54
11	0,46
12	0,44
13	0,62
14	0,44
15	0,56
16	0,50
17	0,64
18	0,64
19	0,64
20	0,66
21	0,66
22	0,58
23	0,72
24	0,72
25	0,70
26	0,78

27	0,76
28	0,64
29	0,86
30	0,84

Tabla 24 - Resultados prueba 2 evitar colisiones 285 imágenes

Con 450 imágenes:

Epoch	Accuracy
1	0,96
2	0,98
3	0,98
4	0,98
5	0,96
6	0,98
7	0,98
8	0,96
9	0,98
10	0,98
11	0,98
12	0,98
13	0,98
14	0,98
15	0,98
16	0,98
17	0,98
18	0,98
19	0,98
20	0,98
21	0,98
22	0,98
23	0,98
24	0,96
25	0,98
26	0,98
27	0,98
28	0,98
29	0,98
30	0,98

Tabla 25 - Resultados prueba 2 con 450 imágenes

6.2.2.2 Gráficas

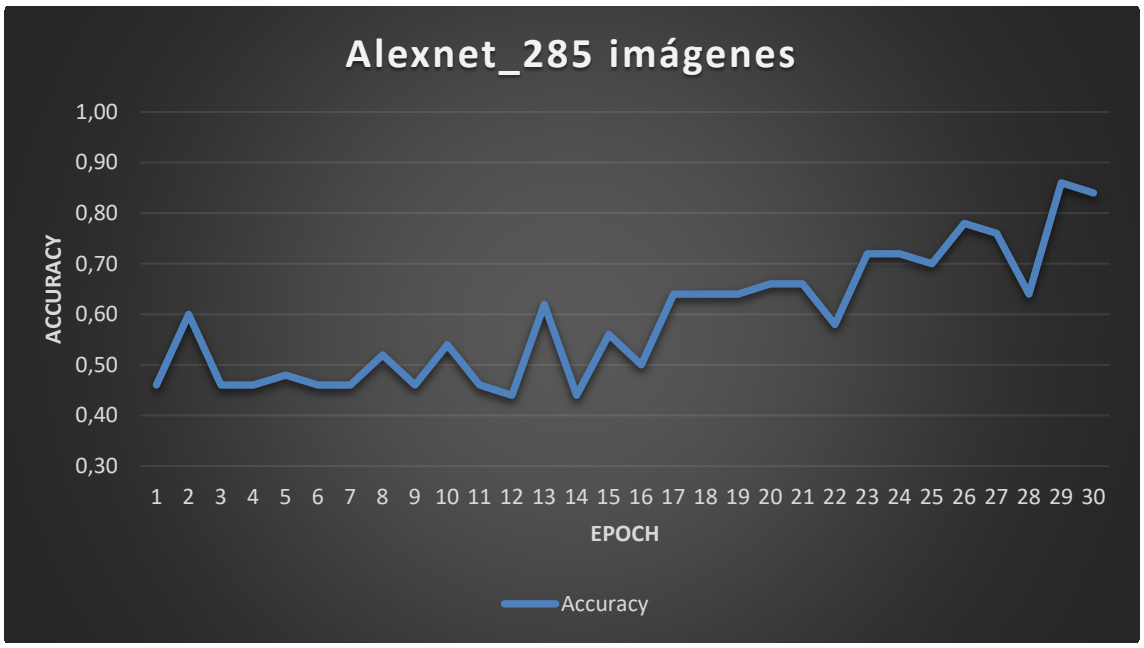


Figura 82 - Gráfica prueba 2 con 285 imágenes

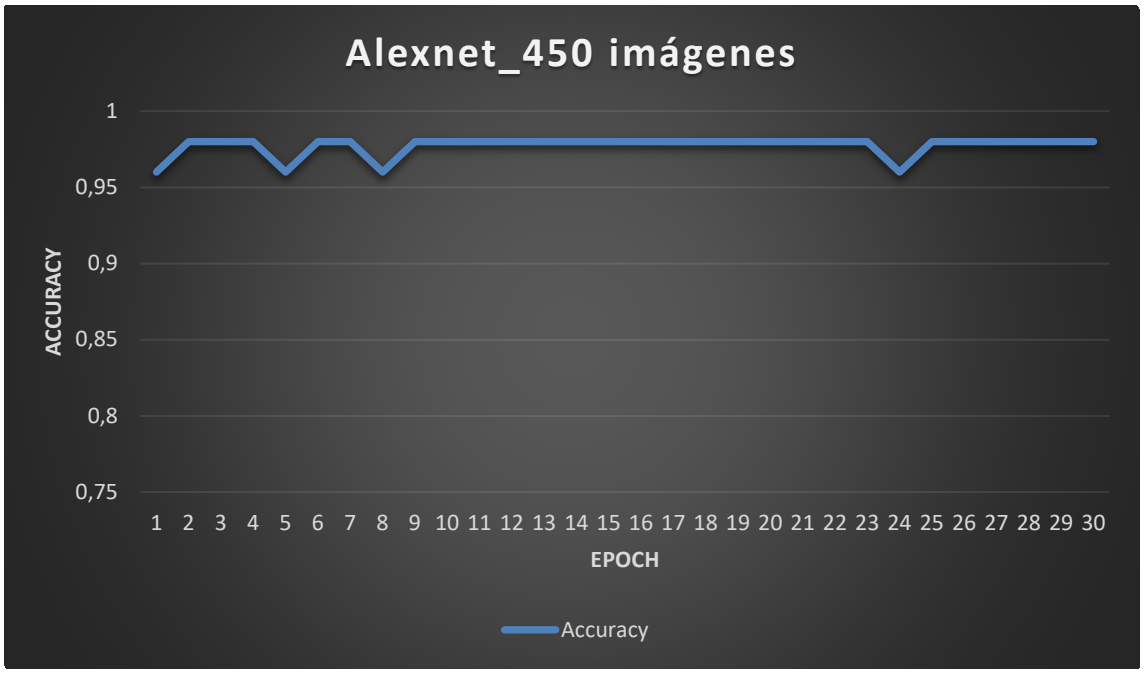


Figura 83 - Gráfica prueba 2 con 450 imágenes

6.2.3 Prueba 3 – Red neuronal preentrenada

La prueba 3 consistió en utilizar dos redes neuronales. Una que ha sido ya configurada con unos pesos determinados y otra que se entrena desde cero. Para esta prueba se han configurado las redes neuronales con los siguientes parámetros:

Dispositivo	Red Neuronal	Conjunto de datos	Epoch	Preentrenamiento
GPU	Alexnet	285	50	Sí
GPU	Alexnet	285	50	No

Tabla 26 - Parámetros de configuración prueba 3 evitar obstáculos

6.2.3.1 Resultados

Sin preentrenamiento:

Epoch	Accuracy
1	0,38
2	0,38
3	0,38
4	0,38
5	0,38
6	0,38
7	0,38
8	0,38
9	0,38
10	0,54
11	0,40
12	0,34
13	0,58
14	0,52
15	0,56
16	0,62
17	0,62
18	0,56
19	0,48
20	0,44
21	0,52
22	0,74
23	0,78
24	0,64
25	0,78
26	0,66
27	0,84
28	0,80

29	0,78
30	0,86
31	0,80
32	0,88
33	0,78
34	0,96
35	0,88
36	0,88
37	0,92
38	0,90
39	0,88
40	0,94
41	0,94
42	0,96
43	0,72
44	0,84
45	0,94
46	0,94
47	1,00
48	0,98
49	0,94
50	0,98

Tabla 27 - Resultados prueba 3 sin preentrenamiento

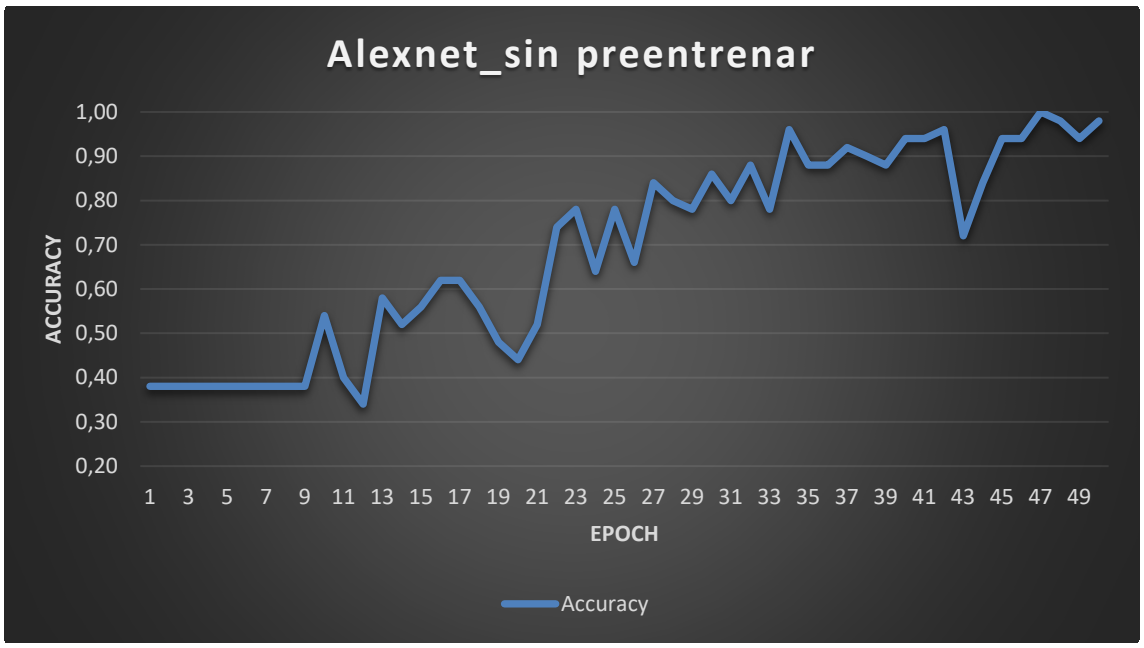
Con preentrenamiento:

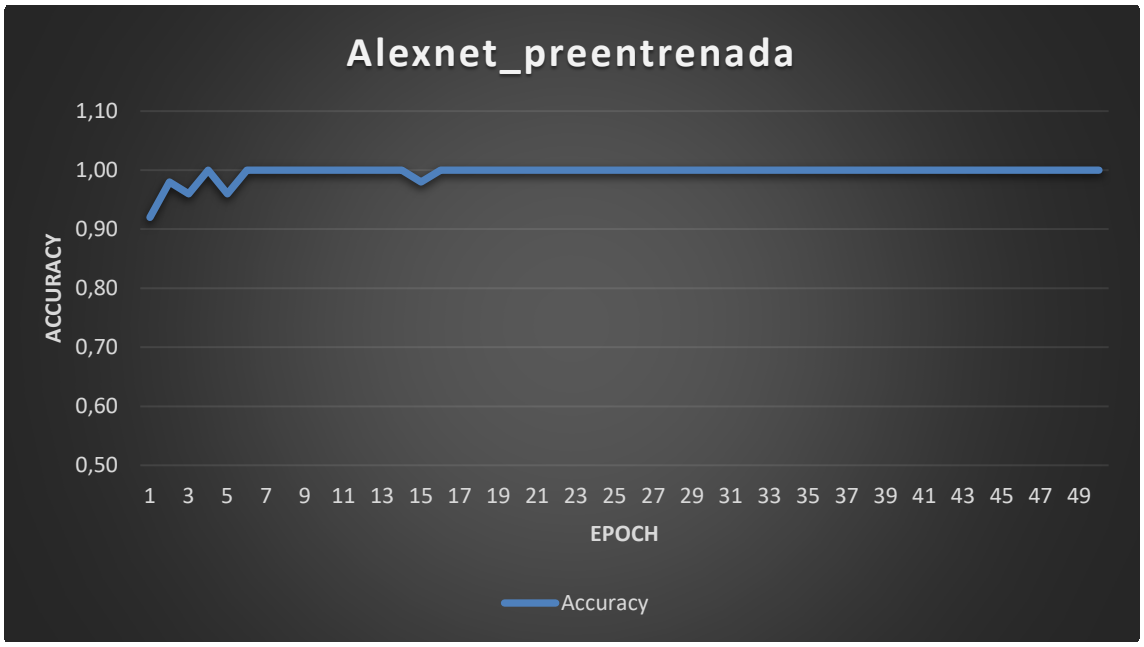
Epoch	Accuracy
1	0,92
2	0,98
3	0,96
4	1,00
5	0,96
6	1,00
7	1,00
8	1,00
9	1,00
10	1,00
11	1,00
12	1,00
13	1,00
14	1,00
15	0,98
16	1,00
17	1,00
18	1,00

19	1,00
20	1,00
21	1,00
22	1,00
23	1,00
24	1,00
25	1,00
26	1,00
27	1,00
28	1,00
29	1,00
30	1,00
31	1,00
32	1,00
33	1,00
34	1,00
35	1,00
36	1,00
37	1,00
38	1,00
39	1,00
40	1,00
41	1,00
42	1,00
43	1,00
44	1,00
45	1,00
46	1,00
47	1,00
48	1,00
49	1,00
50	1,00

Tabla 28 - Resultado prueba 3 con preentrenamiento

6.2.3.2 Gráficas





6.2.4 Prueba 4 – Comparativa ResNet18 vs Alexnet

En la prueba 4 se han entrenado dos tipos distintos de redes, una red resNet18 y otra red Alexnet. Los parámetros configurados para esta prueba son los siguientes:

Dispositivo	Red Neuronal	Conjunto de datos	Epoch	Preentrenamiento
GPU	Alexnet	450	30	No
GPU	ResNet34	450	30	No

Tabla 29 - Parámetros de configuración prueba 4 evitar obstáculos

6.2.4.1 Resultados

Con Alexnet:

Epoch	Accuracy
1	0,96
2	0,98
3	0,98
4	0,98
5	0,96
6	0,98
7	0,98
8	0,96
9	0,98
10	0,98
11	0,98
12	0,98
13	0,98
14	0,98
15	0,98
16	0,98
17	0,98
18	0,98
19	0,98
20	0,98
21	0,98
22	0,98
23	0,98
24	0,96
25	0,98
26	0,98
27	0,98
28	0,98
29	0,98
30	0,98

Tabla 30 - Resultados prueba 4 con Alexnet

Con ResNet18:

Epoch	Accuracy
1	0,86
2	0,92
3	0,90
4	0,92
5	0,94
6	0,92
7	0,92
8	1,00
9	0,94
10	0,94
11	0,96
12	0,88
13	0,96
14	0,96
15	0,96
16	0,94
17	0,96
18	0,98
19	0,96
20	0,94
21	0,94
22	0,92
23	0,98
24	0,98
25	0,98
26	0,98
27	0,94
28	0,96
29	0,94
30	0,94

Tabla 31 - Resultados prueba 4 con ResNet18

6.2.4.2 Gráficas

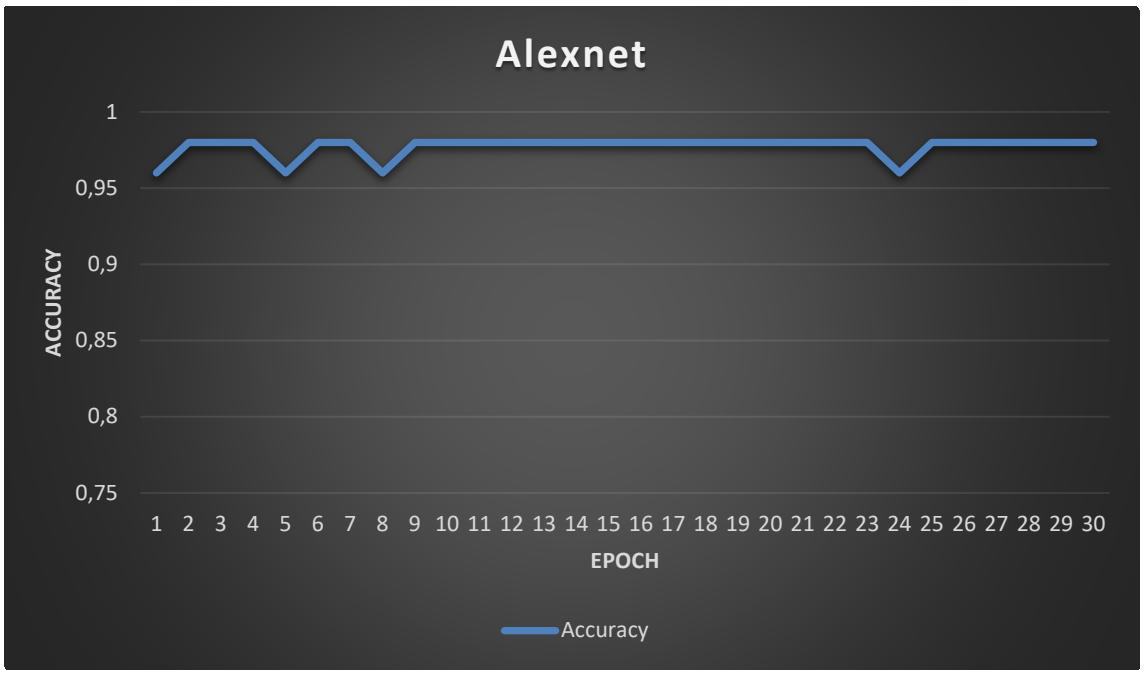


Figura 84 - Gráfica prueba 4 Alexnet

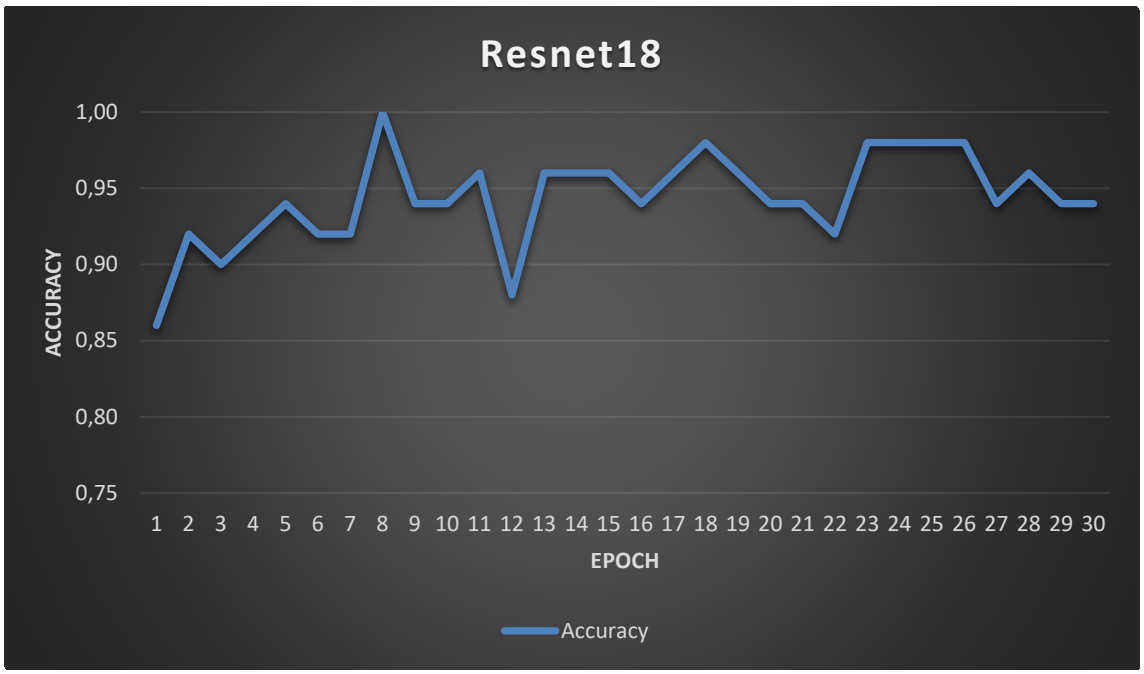


Figura 85 - Gráfica prueba 4 ResNEt18

6.3 Conclusiones de los entrenamientos realizados

Las conclusiones que se pueden sacar de los entrenamientos realizados para el seguimiento de una carretera son las siguientes:

- Al realizar la primera prueba no hubo una diferencia destable entre entrenar en la propia Jetson Nano o entrenar en un portátil estándar, ya que la tabla de resultados y la gráfica no muestran esa diferencia. La diferencia podría estar en cuanto al tiempo de procesamiento, pero los dos entrenamientos tardaron aproximadamente 3 horas y media.
- La activación del modo espejo para tener un conjunto de imágenes mayor no parece que afecte al entrenamiento, dado que comparando las gráficas con espejo y sin espejo son muy parecidas. Sin embargo, al realizar las pruebas de campo con este modo activado, el robot acabo teniendo muchas más salidas de pista que sin activarlo.
- El aumento de épocas indica que todavía se puede llegar a entrenar más el modelo y que con 70 épocas alcanzamos un error menor. Esto no quiere decir que si seguimos aumentando se va a obtener un modelo mejor ya que hay que tener en cuenta el sobreentrenamiento. Por tanto, realizar un entrenamiento con 70 épocas es adecuado.
- Comparando cuál de las dos redes puede llegar a entrenar un modelo mejor de red neuronal, obtenemos que la red neuronal ResNet34 no es del todo adecuada ya que alguna de las imágenes de test no es detectada correctamente, al final tiene más errores que la red neuronal ResNet18.
- Con la prueba 5 queda claro que con un mayor conjunto de datos se obtiene un menor número de errores. Por tanto, es necesario tener un buen conjunto de imágenes con las que realizar el entrenamiento de la red neuronal.
- La última prueba indica que se realiza un mejor entrenamiento si lo hacemos con una red neuronal que ya ha sido preentrenada, es decir que ya tiene ajustados unos pesos determinados.

Por tanto, para el seguimiento de un camino, la configuración adecuada de entrenamiento de la red neuronal será la siguiente:

Las conclusiones que se obtienen para el caso de la detección de obstáculos son iguales a las anteriores en lo que a conjunto de datos, redes preentrenadas y número de épocas se refiere. Sin embargo, las gráficas y los datos indican en este caso que una red neuronal del tipo Alexnet obtiene un resultado de entrenamiento mejor que una del tipo ResNet18.

Capítulo 7. PRUEBAS DE CAMPO

Para las pruebas en campo se ha tenido en cuenta los resultados obtenidos en los entrenamientos de las redes neuronales del capítulo 6. Estas pruebas en campo se han realizado con los modelos de redes neuronales definidos en el punto 6.3 ya que tienen los mejores parámetros para realizar tanto el seguimiento a través de una carretera como para evitar obstáculos en la trayectoria.

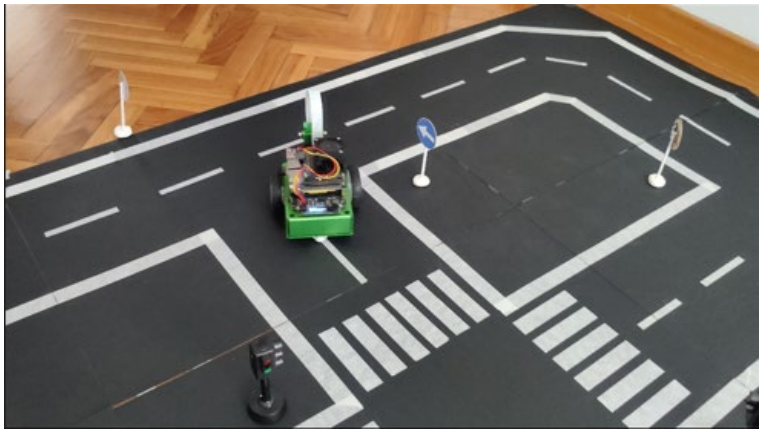


Figura 86 - Robot realizando una prueba de campo 1

Los resultados de estas pruebas determinan lo eficaces que son ambos modelos de redes neuronales a la hora de cumplir su cometido, además estas pruebas definen también los parámetros de movimiento del robot que son los siguientes:

- Velocidad
- Giro
- Respuesta (kd)
- Sesgo de dirección

Para ajustar estos parámetros se han realizado horas de ensayos basados en prueba y error, y ajustando todos los valores al mismo tiempo ya que dependen unos de otros.

Figura 87 - Robot realizando una prueba de campo 2

Una vez realizadas las pruebas de campo sobre la pista se ha llegado a la siguiente conclusión respecto a los valores de movimiento del robot.

- El valor de velocidad debe estar entre 0.09 – 0.11, si se pone un valor por debajo de 0.09 el robot se quedará parado bastantes veces ya que la estructura del robot y las ruedas no son del todo adecuadas para la superficie de la pista de pruebas. Tampoco se debe de poner un valor mayor de 0.11 ya que la pista de pruebas no es muy grande y al robot no le da tiempo a efectuar los giros y se saldría de la pista.

- Los valores adecuados para realizar los giros deben estar entre 0.02 – 0.04, siendo el valor 0.03 el valor perfecto para realizar los giros de una forma suave y rápida para el tamaño del circuito y de las curvas que el robot tiene que tomar.

- El valor de rapidez de respuesta (kd) se ha ajustado a 0.05. A pesar de que hasta 0.15 daba buena respuesta en la mayor parte del circuito, en los cruces el movimiento llegaba a ser muy brusco corrigiendo la dirección rápidamente para llegar al punto siguiente, por ese motivo se acabó reduciendo hasta 0.05 ya que la respuesta era adecuada en toda la pista de pruebas.

- En cuanto al valor de sesgo de dirección se ha mantenido en 0, ya que, al desplazar la barra de selección tanto a valores positivos como negativos, el robot acababa por salirse de los límites del circuito continuamente.

En resumen, los valores son los que aparecen en la imagen siguiente:

```
speed_gain_slider = ipywidgets.FloatSlider(min=0.0, max=1.0, step=0.01, value=0.10, description='velocidad')
steering_gain_slider = ipywidgets.FloatSlider(min=0.0, max=1.0, step=0.01, value=0.03, description='giro')
steering_dgain_slider = ipywidgets.FloatSlider(min=0.0, max=0.5, step=0.001, value=0.05, description='respuesta kd')
steering_bias_slider = ipywidgets.FloatSlider(min=-0.3, max=0.3, step=0.01, value=0.0, description='sesgo de dirección')
```

Figura 88 - Parámetros de movimiento del robot

Estos valores son también aplicables para evitar los obstáculos que el robot se encuentre en la trayectoria que se utiliza el mismo modelo para el movimiento del robot.

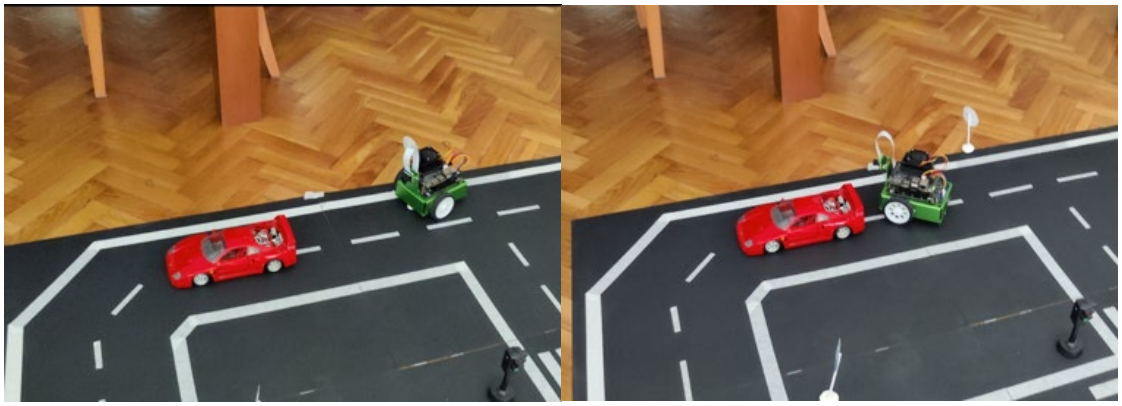


Figura 89 - Robot avanzando ante un obstáculo y parando cuando está apunto de colisionar

Capítulo 8. CONCLUSIONES Y FUTURAS LÍNEAS DE TRABAJO

En este capítulo se detallarán las conclusiones de este proyecto, así como las líneas de trabajo futuras para poder ampliarlo y seguir investigando sobre la inteligencia artificial y la conducción autónoma en robots y otros vehículos.

8.1 Conclusiones

Como conclusiones cabe destacar la importancia de tener un buen conjunto de datos o imágenes para poder entrenar a la red neuronal, ya no solo por la cantidad sino también por la variabilidad de este, ya que el robot en este caso no siempre va a funcionar en las mismas condiciones de luminosidad, y además el fondo de las imágenes obtenidas sobre de la pista de pruebas también puede variar, alterando el funcionamiento del robot a lo largo de la trayectoria.

Destacar también la opción de que algunas redes neuronales propuestas pudieran optimizarse, como la red ResNet18, ya que este influye positivamente en el reconocimiento de la pista de pruebas y de los obstáculos que pudiera tener el robot alrededor.

La estructura y algunos de los componentes del robot eran mejorables, sobre todo las ruedas ya que en este caso no funcionaban de manera adecuada sobre la superficie planteada de la pista de pruebas y llegaban a girar, pero sin tener la suficiente fricción con la superficie para poder mover el robot. Esto aparte de las pérdidas de tiempo que pudiera ocasionar que no es lo más relevante, sí que podría afectar a la trayectoria del robot ya que la rueda podría llegar a tener la fricción suficiente en cualquier momento y de forma repentina, cambiando el siguiente punto de la trayectoria rápidamente y no pudiendo a veces corregir a tiempo.

Otro punto es la dificultad del proyecto en sí, ya que aparte de tener que entender el funcionamiento de una red neuronal y sus procesos matemáticos, la complejidad es mayor a la hora de programarla porque el robot dispone de una versión, la placa Jetson Nano tiene otra versión, el lenguaje de programación utilizado, Python, tiene otra versión y las librerías que se utilizan para inteligencia artificial o visión por computador y librerías matemáticas tienen otra versión. El problema es que muchas veces estas versiones no son compatibles entre sí, provocando muchos errores de ejecución y no pudiendo avanzar en nuevos métodos de entrenamiento o de reconocimiento de imágenes.

Por último, destacar que el proyecto a pesar de la dificultad y costes y tiempo de entrega del kit de desarrollo escogido, me ha parecido muy interesante realizarlo ya que he aprendido en qué estado está actualmente la conducción autónoma y los términos que se utilizan para

hablar de ella. También volver a recordar lo que ya sabía de la inteligencia artificial y métodos de entrenamiento, pero sobre todo poder poner en práctica esos conocimientos y otros nuevos que he aprendido en algo que funciona y no solo es teórico o virtual si no que es real. Animo al lector de este proyecto, si le apasiona la inteligencia artificial, a continuar con las líneas futuras planteadas en el siguiente punto de este capítulo.

8.2 Futuras líneas de trabajo

En cuanto a las líneas de trabajo futuras sobre este proyecto pueden señalarse las siguientes:

- Se puede probar a añadir otro tipo de sensores, como sensores de ultrasonidos, para detectar obstáculos que se encuentren en medio de la trayectoria y ayudar a la detección de estos por medio de la cámara ya que a veces puede llegar a fallar este método, y con la ayuda de otro sensor, tendríamos una probabilidad más alta de detectarlo y evitar la colisión. Se puede sustituir la cámara CSI por otra que tenga infrarrojos y así tener visión en condiciones de baja luminosidad o añadir un LIDAR con el que se podría obtener la distancia que existe hasta un objeto midiendo el tiempo entre la emisión y la recepción de haz láser pulsado. Esto podría ser útil también para saber si tenemos suficiente distancia de frenado con el objeto que se encuentra en la trayectoria.
- Además del entrenamiento propuesto por Machine Learning se puede probar a enseñar al robot a seguir un camino mediante el entrenamiento por refuerzo. Se entrena al robot a que reconozca algunas imágenes y a partir de ahí inicia ya el movimiento, pero si el robot se sale de la trayectoria se le compensa negativamente y si continua durante un tiempo sin salirse del camino se le recompensa de forma positiva para que continue en esa dirección.
- Con los problemas encontrados con la compatibilidad de librerías entre el lenguaje de programación y los sistemas no se ha podido continuar con el reconocimiento de objetos, en concreto de semáforos y señales de tráfico. Esta sería otra de las opciones de trabajo a futuro por las que continuar.
- Por último, en caso de no disponer de un entorno real, se puede investigar la creación de un entorno virtual con las herramientas y sistemas vistos como son ROS, Rviz y Gazebo. Esto permitiría a futuro ahorrar tiempo de ensayo real y que otras personas que no puedan disponer de un entrono real, de una pista de pruebas, puedan investigar y ensayar de forma virtual las redes neuronales o métodos que desarrollen.

ANEXO I - CÓDIGO

En este anexo se expone el código utilizado para recopilar imágenes, mover el robot, entrenar los modelos de redes neuronales y ponerlos en práctica.

Seguimiento de carreteras

En este apartado se muestra el código utilizado para que el robot pueda llegar a seguir una ruta previamente entrenada. Este código está dividido en los siguientes archivos:

- data_collection.ipynb
- telemando.ipynb
- train_model.ipynb
- optimizacion.ipynb
- demo_seguimiento.ipynb

data_collection.ipynb

Importación de librerías para mostrar "widgets", la interfaz de la cámara, los motores y procesar imágenes.

```
# Librerias para mostrar widgets
import ipywidgets
import traitlets
import ipywidgets.widgets as widgets
from IPython.display import display

# Interfaces de la camara y los motores del JetBot
from jetbot import Robot, Camera, bgr8_to_jpeg

# Paquetes basicos de python para la anotación de imagenes
from uuid import uuid1
import os
import json
import glob
import datetime
import numpy as np
import cv2
import time
```

Esta parte del script crea un directorio para guardar las imágenes, en el caso de que ya este creado salta una excepción. Además crea un widget para visualizar la imagen de la cámara del robot y la imagen que se va a guardar, una vez que se ha pulsado en la imagen de la cámara para guardar el punto (x,y) al que el robot va a apuntar.

```
from jupyter_clickable_image_widget import ClickableImageWidget

DATASET_DIR = 'dataset_mejorado'

# Lanzar una excepción en el caso de que ya exista el directorio

try:
    os.makedirs(DATASET_DIR)
except FileExistsError:
    print('Directories not created because they already exist')

camera = Camera()

# crear imagen previa
camera_widget = ClickableImageWidget(width=camera.width, height=camera.height)
snapshot_widget = ipywidgets.Image(width=camera.width, height=camera.height)
traitlets.dlink((camera, 'value'), (camera_widget, 'value'), transform=bgr8_to_jpeg)

# crear widget que vaya contando
count_widget = ipywidgets.IntText(description='count')
# actualizar el conteo al inicio
count_widget.value = len(glob.glob(os.path.join(DATASET_DIR, '*.jpg')))

def save_snapshot(_, content, msg):
    if content['event'] == 'click':
        data = content['eventData']
        x = data['offsetX']
        y = data['offsetY']

        # guardar
        #dataset.save_entry(category_widget.value, camera.value, x, y)
        uuid = 'xy_%03d_%03d_%s' % (x, y, uuid1())
        image_path = os.path.join(DATASET_DIR, uuid + '.jpg')
        with open(image_path, 'wb') as f:
            f.write(camera_widget.value)

        # mostrar la imagen guardada
        snapshot = camera.value.copy()
        snapshot = cv2.circle(snapshot, (x, y), 8, (0, 255, 0), 3)
        snapshot_widget.value = bgr8_to_jpeg(snapshot)
        count_widget.value = len(glob.glob(os.path.join(DATASET_DIR, '*.jpg')))

camera_widget.on_msg(save_snapshot)

data_collection_widget = ipywidgets.VBox([
    ipywidgets.HBox([camera_widget, snapshot_widget]),
    count_widget
])

display(data_collection_widget)
```

Parar la cámara para dejar de recibir imágenes y guardar el conjunto de imágenes como un archivo zip.

```
camera.stop()
```

```
def timestr():
    return str(datetime.datetime.now().strftime('%Y-%m-%d_%H-%M-%S'))

!zip -r -q road_following_{'dataset_mejorado'}.zip {'dataset_mejorado_1'}
```

telemando.ipynb

Importación de la librería para mostrar widget y asignar un índice al mando que controlará el robot.

```
import ipywidgets.widgets as widgets

controller = widgets.Controller(index=0)  # reemplazar el indice por el de tu mando

display(controller)
```

Une los botones del mando (1 y 0) con las acciones de activar el motor izquierdo y activar el motor derecho respectivamente.

```
from jetbot import Robot
import traitlets

robot = Robot()

left_link = traitlets.dlink((controller.axes[1], 'value'), (robot.left_motor, 'value'), transform=lambda x: -x)
right_link = traitlets.dlink((controller.axes[0], 'value'), (robot.right_motor, 'value'), transform=lambda x: -x)
```

train_model.ipynb

Importación de librerías para entrenar el modelo de red neuronal con “torch”.

```
import torch
import torch.optim as optim
import torch.nn.functional as F
import torchvision
import torchvision.datasets as datasets
import torchvision.models as models
import torchvision.transforms as transforms
import glob
import PIL.Image
import os
import numpy as np
```

Carga del conjunto de imágenes, obtiene los puntos X e Y, estandariza el tamaño de las imágenes, realiza una fluctuación del color de las imágenes para obtener una mayor cantidad y diferenciación y da la posibilidad de crear imágenes espejo. Al final estas funciones ayudan a obtener un conjunto mayor de imágenes para mejorar el entrenamiento de la red neuronal.

```python
def get_x(path, width):
    """Obtiene el valor X del archivo de la imagen"""
    return (float(int(path.split("_")[1])) - width/2) / (width/2)

def get_y(path, height):
    """Obtiene el valor Y del archivo de la imagen"""
    return (float(int(path.split("_")[2])) - height/2) / (height/2)

class XYDataset(torch.utils.data.Dataset):

    def __init__(self, directory, random_hflips=False):
        self.directory = directory
        self.random_hflips = random_hflips
        self.image_paths = glob.glob(os.path.join(self.directory, '*.jpg'))
        self.color_jitter = transforms.ColorJitter(0.3, 0.3, 0.3, 0.3)

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        image_path = self.image_paths[idx]

        image = PIL.Image.open(image_path)
        width, height = image.size
        x = float(get_x(os.path.basename(image_path), width))
        y = float(get_y(os.path.basename(image_path), height))

        if float(np.random.rand(1)) > 0.5:
            image = transforms.functional.hflip(image)
            x = -x

        image = self.color_jitter(image)
        image = transforms.functional.resize(image, (224, 224))
        image = transforms.functional.to_tensor(image)
        image = image.numpy()[::-1].copy()
        image = torch.from_numpy(image)
        image = transforms.functional.normalize(image, [0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

        return image, torch.tensor([x, y]).float()

dataset = XYDataset('dataset_mejorado', random_hflips=False)
```

División del conjunto de imágenes en imágenes de entrenamiento e imágenes de test.

```python
test_percent = 0.1
num_test = int(test_percent * len(dataset))
train_dataset, test_dataset = torch.utils.data.random_split(dataset, [len(dataset) - num_test, num_test])
```

```python
train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=8,
    shuffle=True,
    num_workers=0
)

test_loader = torch.utils.data.DataLoader(
    test_dataset,
    batch_size=8,
    shuffle=True,
    num_workers=0
)
```

Definición de la red neuronal que se va a utilizar, realiza la carga en el dispositivo, ya sea “cuda” o “cpu”, definición de los parámetros de entrenamiento y realización del entrenamiento dependiendo del número de épocas.

```
model = models.resnet18(pretrained=True)

model.fc = torch.nn.Linear(512, 2)
device = torch.device('cuda')
model = model.to(device)


NUM_EPOCHS = 70
BEST_MODEL_PATH = 'best_steering_model_xy.pth'
best_loss = 1e9

optimizer = optim.Adam(model.parameters())
#optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

for epoch in range(NUM_EPOCHS):
    #print('Epoch: %d' % (epoch))
    model.train()
    train_loss = 0.0
    for images, labels in iter(train_loader):
        #print('modelo de entrenamiento')
        images = images.to(device)
        labels = labels.to(device)
        #print('optimizacion')
        optimizer.zero_grad()
        outputs = model(images)
        #print('loss')
        loss = F.mse_loss(outputs, labels)
        #loss = F.cross_entropy(outputs, labels)
        train_loss += float(loss)
        loss.backward()
        #print('%f' % (epoch))
        optimizer.step()
        #print('optimizado')
    train_loss /= len(train_loader)

    model.eval()
    test_loss = 0.0
    for images, labels in iter(test_loader):
        #print('modelo de evaluacion')
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        loss = F.mse_loss(outputs, labels)
        #loss = F.cross_entropy(outputs, labels)
        test_loss += float(loss)
        #print('fin modelo')
    test_loss /= len(test_loader)

    print('%d, %f, %f' % (epoch, train_loss, test_loss))
    if test_loss < best_loss:
        torch.save(model.state_dict(), BEST_MODEL_PATH)
        best_loss = test_loss
```

optimizacion.ipynb

Carga del modelo de red neuronal en el dispositivo.

```
import torchvision
import torch

model = torchvision.models.resnet18(pretrained=True)
model.fc = torch.nn.Linear(512, 2)
model = model.cuda().eval().half()

model.load_state_dict(torch.load('jetson_resnet18_datasetmejorado/best_steering_model_xy.pth'))

device = torch.device('cuda')
```

Dependiendo de la red neuronal utilizada se puede llegar a realizar una conversión y optimización del modelo usando la librería "torch2trt". Esta optimización permite una respuesta más rápida en cuanto al procesamiento de imágenes.

```
from torch2trt import torch2trt

data = torch.zeros((1, 3, 224, 224)).cuda().half()

model_trt = torch2trt(model, [data], fp16_mode=True)

torch.save(model_trt.state_dict(), 'jetson_resnet18_datasetmejorado/best_steering_model_xy_trt.pth')
```

demo_seguimiento.ipynb

Carga del modelo de red neuronal optimizado.

```
import torch
from torch2trt import TRTModule
import torch
device = torch.device('cuda')
model_trt = TRTModule()
model_trt.load_state_dict(torch.load('jetson_resnet18_datasetmejorado/best_steering_model_xy_trt.pth'))
```

Carga de librerías de procesamiento de imágenes y definición de la función de preprocesamiento ya que las imágenes con las que hemos entrenado el modelo no coinciden con las de la cámara, por tanto, hay que convertir el formato, normalizar las imágenes, añadir una dimensión al lote de datos y transferir los datos de la CPU a la memoria GPU.

```
import torchvision.transforms as transforms
import torch.nn.functional as F
import cv2
import PIL.Image
import numpy as np

mean = torch.Tensor([0.485, 0.456, 0.406]).cuda().half()
std = torch.Tensor([0.229, 0.224, 0.225]).cuda().half()

def preprocess(image):
    image = PIL.Image.fromarray(image)
    image = transforms.functional.to_tensor(image).to(device).half()
    image.sub_(mean[:, None, None]).div_(std[:, None, None])
    return image[None, ...]
```

El siguiente script carga la instancia de la cámara y el robot, así como los complementos o widgets para ver la cámara en directo y las barras de selección con la que se ajustará la velocidad, el giro, la respuesta y el sesgo de dirección.

```
from IPython.display import display
import ipywidgets
import traitlets
from jetbot import Camera, bgr8_to_jpeg
from jetbot import Robot

camera = Camera()
image_widget = ipywidgets.Image()

traitlets.dlink((camera, 'value'), (image_widget, 'value'), transform=bgr8_to_jpeg)

display(image_widget)

robot = Robot()

speed_gain_slider = ipywidgets.FloatSlider(min=0.0, max=1.0, step=0.01, value=0.10, description='velocidad')
steering_gain_slider = ipywidgets.FloatSlider(min=0.0, max=1.0, step=0.01, value=0.03, description='giro')
steering_dgain_slider = ipywidgets.FloatSlider(min=0.0, max=0.5, step=0.001, value=0.05, description='respuesta kd')
steering_bias_slider = ipywidgets.FloatSlider(min=-0.3, max=0.3, step=0.01, value=0.0, description='sesgo de dirección')

display(speed_gain_slider, steering_gain_slider, steering_dgain_slider, steering_bias_slider)

x_slider = ipywidgets.FloatSlider(min=-1.0, max=1.0, description='x')
y_slider = ipywidgets.FloatSlider(min=0, max=1.0, orientation='vertical', description='y')
steering_slider = ipywidgets.FloatSlider(min=-1.0, max=1.0, description='steering')
speed_slider = ipywidgets.FloatSlider(min=0, max=1.0, orientation='vertical', description='speed')

display(ipywidgets.HBox([y_slider, speed_slider]))
display(x_slider, steering_slider)
```

La siguiente función lo que hace es ejecutar la activación de los motores en base a la diferencia de ángulo que existe entre la posición del robot y el punto a donde se quiere desplazar. Esa diferencia de ángulos es analizada en cada “frame” de forma que se va desplazando de manera continua mientras la cámara esté en funcionamiento.

```
angle = 0.0
angle_last = 0.0

def execute(change):
    global angle, angle_last
    image = change['new']
    xy = model_trt(preprocess(image)).detach().float().cpu().numpy().flatten()
    x = xy[0]
    y = (0.5 - xy[1]) / 2.0

    x_slider.value = x
    y_slider.value = y

    speed_slider.value = speed_gain_slider.value

    angle = np.arctan2(x, y)
    pid = angle * steering_gain_slider.value + (angle - angle_last) * steering_dgain_slider.value
    angle_last = angle

    steering_slider.value = pid + steering_bias_slider.value

    robot.left_motor.value = max(min(speed_slider.value + steering_slider.value, 1.0), 0.0)
    robot.right_motor.value = max(min(speed_slider.value - steering_slider.value, 1.0), 0.0)

execute({'new': camera.value})

camera.observe(execute, names='value')
```

Estas funciones ejecutan las acciones de parar la cámara y el robot.

```
import time

camera.unobserve(execute, names='value')

time.sleep(0.1)  # add a small sleep to make sure frames have finished processing

robot.stop()
camera.stop()
```

Evitar la colisión con obstáculos

En este apartado se muestra el código utilizado para que el robot pueda llegar a evitar obstáculos que se encuentren en su trayectoria. Este código está dividido en los siguientes archivos:

- conjunto_de_datos.ipynb
- entrenamiento.ipynb
- optimizacion_resnet.ipynb
- demo_evita_colisiones.ipynb
- demo_colisiones_resnet.ipynb

conjunto_de_datos.ipynb

Este "script" permite obtener imágenes de la cámara y guardarlas en carpetas diferentes dependiendo de si son imágenes con algún objeto bloqueando el paso o si por el contrario el camino está libre de obstáculos.

Importación de las librerías necesarias para obtener imágenes y mostrar los complementos o "widgets".

```
import traitlets
import ipywidgets.widgets as widgets
from IPython.display import display
from jetbot import Camera, bgr8_to_jpeg


camera = Camera.instance(width=224, height=224)

image = widgets.Image(format='jpeg', width=224, height=224)

camera_link = traitlets.dlink((camera, 'value'), (image, 'value'), transform=bgr8_to_jpeg)

display(image)

blocked_dir = 'dataset/blocked'
free_dir = 'dataset/free'

# Comprobar si el directorio existe y si no crear uno para cada conjuntos de datos
try:
    os.makedirs(free_dir)
    os.makedirs(blocked_dir)
except FileExistsError:
    print('No se crean las carpetas porque ya existen')
```

La siguiente parte, carga los complementos que muestran la cámara, y los botones para seleccionar la imagen como bloqueada o libre de obstáculos.

```
button_layout = widgets.Layout(width='128px', height='64px')
free_button = widgets.Button(description='add free', button_style='success', layout=button_layout)
blocked_button = widgets.Button(description='add blocked', button_style='danger', layout=button_layout)
free_count = widgets.IntText(layout=button_layout, value=len(os.listdir(free_dir)))
blocked_count = widgets.IntText(layout=button_layout, value=len(os.listdir(blocked_dir)))

display(widgets.HBox([free_count, free_button]))
display(widgets.HBox([blocked_count, blocked_button]))
```

Definición de funciones que permiten guardar la imagen y enviarla a la carpeta correspondiente.

```
from uuid import uuid1

def save_snapshot(directory):
    image_path = os.path.join(directory, str(uuid1()) + '.jpg')
    with open(image_path, 'wb') as f:
        f.write(image.value)

def save_free():
    global free_dir, free_count
    save_snapshot(free_dir)
    free_count.value = len(os.listdir(free_dir))

def save_blocked():
    global blocked_dir, blocked_count
    save_snapshot(blocked_dir)
    blocked_count.value = len(os.listdir(blocked_dir))


free_button.on_click(lambda x: save_free())
blocked_button.on_click(lambda x: save_blocked())
```

Muestra los complementos y permiten ser utilizados por el usuario.

```
display(image)
display(widgets.HBox([free_count, free_button]))
display(widgets.HBox([blocked_count, blocked_button]))
```

entrenamiento.ipynb

Importación de las librerías necesarias para entrenar la red neuronal en base al conjunto de imágenes que se han recogido.

```
import torch
import torch.optim as optim
import torch.nn.functional as F
import torchvision
import torchvision.datasets as datasets
import torchvision.models as models
import torchvision.transforms as transforms
```

Carga del conjunto de imágenes, normalización y división del conjunto en imágenes de entrenamiento e imágenes de prueba.

```
dataset = datasets.ImageFolder(
    'dataset',
    transforms.Compose([
        transforms.ColorJitter(0.1, 0.1, 0.1, 0.1),
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
)

train_dataset, test_dataset = torch.utils.data.random_split(dataset, [len(dataset) - 50, 50])

train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=8,
    shuffle=True,
    num_workers=0
)

test_loader = torch.utils.data.DataLoader(
    test_dataset,
    batch_size=8,
    shuffle=True,
    num_workers=0
)
```

Carga de la red neuronal y de sus parámetros, los cuales se van a utilizar para crear el modelo que permite al robot evitar los obstáculos que encuentre en su trayectoria.

```
model = models.alexnet(pretrained=True)
model.classifier[6] = torch.nn.Linear(model.classifier[6].in_features, 2)

device = torch.device('cuda')
model = model.to(device)

NUM_EPOCHS = 30
BEST_MODEL_PATH = 'best_model_new.pth'
best_accuracy = 0.0

optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

for epoch in range(NUM_EPOCHS):

    for images, labels in iter(train_loader):
        images = images.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = F.cross_entropy(outputs, labels)
        loss.backward()
        optimizer.step()

    test_error_count = 0.0
    for images, labels in iter(test_loader):
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        test_error_count += float(torch.sum(torch.abs(labels - outputs.argmax(1))))

    test_accuracy = 1.0 - float(test_error_count) / float(len(test_dataset))
    print('%d: %f' % (epoch, test_accuracy))
    if test_accuracy > best_accuracy:
        torch.save(model.state_dict(), BEST_MODEL_PATH)
        best_accuracy = test_accuracy
```

Si se quisiera aplicar una red neuronal del tipo ResNet18 se tendrían que cambiar estas líneas de código.

```
model = models.resnet18(pretrained=False)
model.fc = torch.nn.Linear(512, 2)
device = torch.device('cuda')
model = model.to(device)
```

optimización_resnet.ipynb

Se peude usar el mismo código que en el seguimiento de carreteras para la optimización del modelo si se ha usado una red neuronal de tipo ResNet.

```
from torch2trt import torch2trt

data = torch.zeros((1, 3, 224, 224)).cuda().half()

model_trt = torch2trt(model, [data], fp16_mode=True)
torch.save(model_trt.state_dict(), 'best_model_trt.pth')
```

demo_evita_colisiones.ipynb

Importación de librerías y carga del modelo en el dispositivo.

```
import torch
import torchvision

model = torchvision.models.alexnet(pretrained=True)
model.classifier[6] = torch.nn.Linear(model.classifier[6].in_features, 2)
model.load_state_dict(torch.load('best_model_new.pth'))
device = torch.device('cuda')
model = model.to(device)
```

Normalización y función de preprocesamiento de las imágenes.

```
import cv2
import numpy as np

mean = 255.0 * np.array([0.485, 0.456, 0.406])
stdev = 255.0 * np.array([0.229, 0.224, 0.225])

normalize = torchvision.transforms.Normalize(mean, stdev)

def preprocess(camera_value):
    global device, normalize
    x = camera_value
    x = cv2.cvtColor(x, cv2.COLOR_BGR2RGB)
    x = x.transpose((2, 0, 1))
    x = torch.from_numpy(x).float()
    x = normalize(x)
    x = x.to(device)
    x = x[None, ...]
    return x
```

Importación e instanciación de librerías de complementos, del robot y de la cámara

```
import traitlets
from IPython.display import display
import ipywidgets.widgets as widgets
from jetbot import Camera, bgr8_to_jpeg
from jetbot import Robot

camera = Camera.instance(width=224, height=224)
image = widgets.Image(format='jpeg', width=224, height=224)
blocked_slider = widgets.FloatSlider(description='blocked', min=0.0, max=1.0, orientation='vertical')
speed_slider = widgets.FloatSlider(description='speed', min=0.0, max=0.5, value=0.0, step=0.01, orientation='horizontal')

camera_link = traitlets.dlink((camera, 'value'), (image, 'value'), transform=bgr8_to_jpeg)

display(widgets.VBox([widgets.HBox([image, blocked_slider]), speed_slider]))

robot = Robot()
```

Ejecución de la red neuronal a través de la función “update”, cuya salida ejecuta la acción de parar el robot si se encuentra un obstáculo.

```
import torch.nn.functional as F
import time

def update(change):
    global blocked_slider, robot
    x = change['new']
    x = preprocess(x)
    y = model(x)

    # Normalizar el vector de salida con dimensión 1
    y = F.softmax(y, dim=1)

    prob_blocked = float(y.flatten()[0])

    blocked_slider.value = prob_blocked

    if prob_blocked < 0.5:
        robot.forward(speed_slider.value)
    else:
        #robot.left(speed_slider.value)
        robot.stop()

    time.sleep(0.001)

update({'new': camera.value})  # we call the function once to initialize
```

demo_colisiones_resnet.ipynb

Si se va a utilizar una red neuronal de tipo ResNet se debe cargar el modelo optimizado mediante este “script”.

```
import torch
import torchvision
from torch2trt import TRTModule

device = torch.device('cuda')
model_trt = TRTModule()
model_trt.load_state_dict(torch.load('best_model_trt.pth'))
```

Y realizar el siguiente preprocesamiento de imagen, el cual se basa en los mismos puntos, pero no se programa de la misma forma al tener una red neuronal diferente y haber entrenado y optimizado de distinta manera.

```
import torchvision.transforms as transforms
import torch.nn.functional as F
import cv2
import PIL.Image
import numpy as np

mean = torch.Tensor([0.485, 0.456, 0.406]).cuda().half()
std = torch.Tensor([0.229, 0.224, 0.225]).cuda().half()

normalize = torchvision.transforms.Normalize(mean, std)

def preprocess(image):
    image = PIL.Image.fromarray(image)
    image = transforms.functional.to_tensor(image).to(device).half()
    image.sub_(mean[:, None, None]).div_(std[:, None, None])
    return image[None, ...]
```

El resto del código es el mismo que el que aparece en el punto anterior "demo_evita_colisiones.ipynb"

Seguimiento de carreteras y evitar la colisión con obstáculos

El siguiente código aúna los dos códigos anteriores de forma que el robot pueda realizar ambas acciones al mismo tiempo.

completemotion.ipynb

Importación de librerías. Se trata de las mismas librerías que se han utilizado en los códigos anteriores.

```
import torch
import torchvision
from torch2trt import TRTModule
import torchvision.transforms as transforms
import torch.nn.functional as F
import cv2
import PIL.Image
import numpy as np
import traitlets
from IPython.display import display
import ipywidgets.widgets as widgets
from jetbot import Camera, bgr8_to_jpeg
import ipywidgets
from jetbot import Robot
import time
```

Carga del modelo de red neuronal que evita los obstáculos en la trayectoria del robot. Si queremos cambiar de tipo de red a una ResNet se utilizarán las líneas comentadas.

```
model = torchvision.models.alexnet(pretrained=True)
model.classifier[6] = torch.nn.Linear(model.classifier[6].in_features, 2)
#model = torchvision.models.resnet18(pretrained=True)
#model.fc = torch.nn.Linear(512, 2)
model.load_state_dict(torch.load('best_model_new.pth'))
```

Carga del modelo de red neuronal para que el robot pueda seguir un camino.

```
model_trt = TRTModule()
model_trt.load_state_dict(torch.load('best_steering_model_xy_trt.pth'))
device = torch.device('cuda')
model = model.to(device)
```

Normalización de las imágenes para los dos modelos de redes neuronales.

```
mean_1 = 255.0 * np.array([0.485, 0.456, 0.406])
stdev_1 = 255.0 * np.array([0.229, 0.224, 0.225])
normalize = torchvision.transforms.Normalize(mean_1, stdev_1)

mean_2 = torch.Tensor([0.485, 0.456, 0.406]).cuda().half()
std_2 = torch.Tensor([0.229, 0.224, 0.225]).cuda().half()
```

Preprocesamiento de imágenes y ejecución de las funciones que dan movimiento al robot y hacen que siga un camino (función “execute”) y la función que detecta obstáculos y hace que se paré el robot para evitar colisionar (función “update”). Para usar el modelo de red neuronal “Alexnet” se utilizará la función “preprocess_1”, mientras que, si se quiere utilizar una red neuronal del tipo ResNet, se usará la función “preprocess_2”. Está última función también se utiliza para preprocesar las imágenes que hacen que el robot siga una ruta.

```
def preprocess_1(camera_value):
    global device, normalize
    x = camera_value
    x = cv2.cvtColor(x, cv2.COLOR_BGR2RGB)
    x = x.transpose((2, 0, 1))
    x = torch.from_numpy(x).float()
    x = normalize(x)
    x = x.to(device)
    x = x[None, ...]
    return x

def preprocess_2(image):
    image = PIL.Image.fromarray(image)
    image = transforms.functional.to_tensor(image).to(device).half()
    image.sub_(mean_2[:, None, None]).div_(std_2[:, None, None])
    return image[None, ...]

def execute(change):
    global angle, angle_last
    image = change['new']
    xy = model_trt(preprocess_1(image)).detach().float().cpu().numpy().flatten()
    #xy = model_trt(preprocess_2(image)).detach().float().cpu().numpy().flatten()
    x = xy[0]
    y = (0.5 - xy[1]) / 2.0

    x_slider.value = x
    y_slider.value = y

    speed_slider.value = speed_gain_slider.value

    angle = np.arctan2(x, y)
    pid = angle * steering_gain_slider.value + (angle - angle_last) * steering_dgain_slider.value
    angle_last = angle

    steering_slider.value = pid + steering_bias_slider.value

    robot.left_motor.value = max(min(speed_slider.value + steering_slider.value, 1.0), 0.0)
    robot.right_motor.value = max(min(speed_slider.value - steering_slider.value, 1.0), 0.0)

def update(change):
    global blocked_slider, robot
    x = change['new']
    x = preprocess_1(x)
    #x = preprocess_2(x)
    y = model(x)


    y = F.softmax(y, dim=1)

    prob_blocked = float(y.flatten()[0])

    blocked_slider.value = prob_blocked

    if prob_blocked < 0.5:
        execute({'new': camera.value})
    else:
        robot.stop()


    time.sleep(0.001)
```

Inicialización de la cámara y del robot. Así como de los diferentes complementos que se usan para visualizar la cámara y los valores de velocidad, giro, respuesta y sesgo de dirección. Además, también se pueden ajustar estos valores mediante una barra de desplazamiento.

```
camera = Camera()
image_widget = ipywidgets.Image()
traitlets.dlink((camera, 'value'), (image_widget, 'value'), transform=bgr8_to_jpeg)
display(image_widget)
#display(image)
robot = Robot()
speed_gain_slider = ipywidgets.FloatSlider(min=0.0, max=1.0, step=0.01, value=0.10, description='velocidad')
steering_gain_slider = ipywidgets.FloatSlider(min=0.0, max=1.0, step=0.01, value=0.03, description='giro')
steering_dgain_slider = ipywidgets.FloatSlider(min=0.0, max=0.5, step=0.001, value=0.05, description='respuesta kd')
steering_bias_slider = ipywidgets.FloatSlider(min=-0.3, max=0.3, step=0.01, value=0.0, description='sesgo de dirección')
display(speed_gain_slider, steering_gain_slider, steering_dgain_slider, steering_bias_slider)
x_slider = ipywidgets.FloatSlider(min=-1.0, max=1.0,  orientation='vertical', description='x')
y_slider = ipywidgets.FloatSlider(min=0, max=1.0, orientation='vertical', description='y')
steering_slider = ipywidgets.FloatSlider(min=-1.0, max=1.0, description='steering',  orientation='vertical')
#speed_slider = ipywidgets.FloatSlider(min=0, max=1.0, orientation='vertical', description='speed')
speed_slider = ipywidgets.FloatSlider(description='velocidad', min=0.0, max=0.5, value=0.0, step=0.01, orientation='vertical')
blocked_slider = ipywidgets.FloatSlider(description='prob. bloqueo', min=0.0, max=1.0, orientation='vertical')
#display(widgets.VBox([widgets.HBox([image_widget, blocked_slider]), speed_slider]))
display(ipywidgets.HBox([x_slider, y_slider, speed_slider, steering_slider, blocked_slider]))
#display(x_slider, steering_slider)

angle = 0.0
angle_last = 0.0
```

Ejecución de la función “update” y de la cámara, la cual enviará a la función las imágenes correspondientes.

```
update({'new': camera.value})  # inicio de la función update
camera.observe(update, names='value')  # actualizar la función update con lo que muestra la cámara
```

Parada de la cámara y del robot.

```
camera.unobserve(execute, names='value')
camera.unobserve(update, names='value')
time.sleep(0.1)  # add a small sleep to make sure frames have finished processing
robot.stop()
camera.stop()
```

ANEXO II - PRESUPUESTO

En este anexo se documenta el presupuesto necesario para poder realizar este proyecto paso a paso y llegar a las conclusiones pertinentes además de poder realizar algunas futuras líneas de trabajo como se comenta en el capítulo 8 de la memoria.

Tipos	Producto	Coste individual	Cantidad	Total
Materiales y herramientas	**Jetson Nano 2GB**	200 €	1	200 €
	Kit Jetbot para Jetson Nano	116 €	1	116 €
	Memoria SD 64 GB	8,86 €	1	8,86 €
	Baterías 18650 mAh 3,7V	7 €	9	63 €
	Cartulinas	0,80 €	9	7,20 €
	Cinta de pintor blanca	2,50 €	2	5 €
	Plastilina	0,60 €	1	0,60 €
	Paquete de palos	0,75 €	1	0,75 €
	Paquete de semáforos	28,47 €	1	28,47 €
	Portátil	1.400,00 €	1	1.400,00 €
Investigación y desarrollo	**Investigación**	40€/h	100h	4.000 €
	Diseño	30€/h	50h	1.500 €
	Fabricación	15€/h	20h	300 €
	Puesta a punto	15€/h	70h	1.050 €
	Documentación	20€/h	60h	1.200 €
TOTAL				**9.879,88 €**

Tabla 32 - Presupuesto

No es un coste muy alto ya que se trata de un proyecto de estudio y de investigación sobre inteligencia artificial y más allá del material que se use prima sobre todo la mano de obra, las horas que se acaban utilizando para poder entrenar un modelo de aprendizaje que sea útil y valido para las tareas que se encomiendan. Se podía haber usado otro tipo de robot u otro tipo de placa del mismo fabricante más caros y más estables, pero como proyecto inicial no compensaba comprar algo que fuese más caro cuando no tienes una base sólida de la que partir.

A partir de ahora, con esta base, sí que merece la pena comprar algo con unas características superiores para obtener mejores resultados.

BIBLIOGRAFÍA

[1] R. García (2022, marzo, 24) Las mejores rivales y alternativas a la Raspberry Pi 4 [online] Available: Alternativas a Raspberry Pi 4: Mejores placas potentes y baratas (adslzone.net)

[2] NVIDIA Corporation (2022) Getting Started with Jetson Nano 2GB Developer Kit [online] Available: Getting Started with Jetson Nano 2GB Developer Kit | NVIDIA Developer

[3] NVIDIA Corporation (2022) Jetson Nano 2GB Developer Kit User Guide [online] Available: Jetson Nano 2GB Developer Kit User Guide | NVIDIA Developer

[4] Waveshare Electronics (2022) JetBot 2GB AI Kit [online] Available: JetBot 2GB AI Kit, AI Robot Based On Jetson Nano 2GB Developer Kit (Optional) (waveshare.com)

[5] NVIDIA Corporation (2022) KITS DE ROBOTS DE IA DE JETBOT DE PARTNERS DE NVIDIA [online] Available: Kits de robots de IA de partners de NVIDIA JetBot | NVIDIA

[6] ORACLE (2022) ¿Qué es la inteligencia artificial? [online] Available: ¿Qué es la inteligencia artificial (IA)? | Oracle España

[7] J. A. Sánchez (2020, agosto, 31) ¿Cómo aprenden las máquinas? Machine Learning y sus diferentes tipos [online] Available: ¿Cómo aprenden las máquinas? Machine Learning y sus diferentes tipos | datos.gob.es

[8] Iberdrola, S.A (2022) 'Deep learning': un concepto clave para llevar la inteligencia artificial al siguiente nivel [online] Available: Deep Learning qué es y por qué es clave para la inteligencia artificial - Iberdrola

[9] Lic. Geol. Francisco Crespo Jimenez y Ing. Aldo Algorry (2019, agosto, 28) ¿Deep Learning… y clasificación de imágenes? [online] Available: ¿Deep Learning… y clasificación de imágenes? – www.idecor.gob.ar

[10] Oscar García-Olalla Olivera (2019, septiembre, 16) Redes Neuronales artificiales: Qué son y cómo se entrenan [online] Available: Redes Neuronales artificiales: Qué son y cómo se entrenan | [site:name] (xeridia.com)

[11] Pherkad (2018, marzo, 31) Primeros pasos con JupyterLab [online] Available: Python 3 para impacientes: Primeros pasos con JupyterLab (python-para-impacientes.blogspot.com)

[12] Amanda Dattalo (2018, agosto, 8) ROS/Introduction [online] Available: ROS/Introduction - ROS Wiki

[13] Automation Ware (2022) Programming a robot with ROS [online] Available: ROS - AutomationWare

[14] Aaron M.R (2014, junio, 21) ROS/Concepts [online] Available: ROS/Concepts - ROS Wiki

[15] Ken_in_JAPAN (2014, Agosto, 10) Obstacle doesn't show on rviz [online] Available: Obstacle doesn't show on rviz [closed] - ROS Answers: Open Source Q&A Forum

[16] Fetch Robotics Inc (2019) Visualizing with RVIZ [online] Available: Tutorial: Visualization — Fetch & Freight Research Edition Melodic documentation (fetchrobotics.com)

[17] FS STUDIO (2021, enero, 19) ROS Gazebo: Everything You Need To Know [online] Available: ROS Gazebo: Everything You Need To Know - Robotic Simulation Services

[18] Louise Poubel (2019, mayo, 7) Open-Source Robotics: Getting Started with Gazebo and ROS 2 [online] Available: Open Source Robotics: Getting Started with Gazebo and ROS 2 (infoq.com)

[19] Automatic Addison (2021, julio, 22) Setting Up the ROS Navigation Stack for a Simulated Robot [online] Available: Setting Up the ROS Navigation Stack for a Simulated Robot – Automatic Addison

[20] ERTRAC (2017, mayo, 29) Automated Driving Roadmap [online] Available: ERTRAC_Automated_Driving_2017.pdf

[21] SAE (2021, abril) SURFACE VEHICLE RECOMMENDED PRACTICE [online] Available: j3016-202104.pdf (brightspotcdn.com)

[22] Ruedas de prensa, S.L. (2022) Conducción autónoma | Niveles y tecnología [online] Available: Conducción autónoma | Los cinco niveles (km77.com)

[23] STRIA (2019, abril) Roadmap on Connected and Automated Transport [online] Available: stria-roadmap_on_connected_and_automated_transport2019-TRIMIS_website.pdf (aecarretera.com)

[24] Fesvial (2018) EL PROYECTO "UN MILLÓN DE KILÓMETROS" EN CONDUCCIÓN AUTÓNOMA ARRANCA EN EL ESCORIAL (MADRID) [online] Available: FESVIAL | EL PROYECTO "UN MILLÓN DE KILÓMETROS" EN CONDUCCIÓN AUTÓNOMA ARRANCA EN EL ESCORIAL (MADRID)

[25] Ministerio de Transportes, Movilidad y Agenda Urbana, Gobierno de España (2021, junio, 25) Proyecto BRAVE sobre conducción autónoma [online] Available: Proyecto BRAVE sobre conducción autónoma | Estrategia de Movilidad Segura, Sostenible y Conectada 2030 (mitma.es)

[26] Mónica Redondo (2022, julio, 4) R3CAV, el proyecto que desarrollará los futuros vehículos autónomos [online] Available: R3CAV, el proyecto que desarrollará los vehículos autónomos (highmotor.com)

[27] Diario de Transporte (2020, enero, 14) EZ10, el primer autobús autónomo universitario de España [online] Available: EZ10, el primer autobús autónomo universitario de España. Vídeo (diariodetransporte.com)

[28] Waveshare Electronics (2022) JetBot AI Kit Assemble Manual [online] Available: JetBot AI Kit Assemble Manual - Waveshare Wiki

[29] Jesús Ultrera Burgal (2018, diciembre, 5) Deep Learning básico con Keras (Parte 4): ResNet [online] Available: Deep Learning básico con Keras (Parte 4): ResNet (enmilocalfunciona.io)

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Microsoft Research (2015, diciembre, 10) Deep Residual Learning for Image Recognition [online] Available: 1512.03385v1.pdf (arxiv.org)

[31] Tawsifur Rahman, Muhammad E. H. Chowdhury, Amith Khandakar,Khandaker R. Islam , Khandaker F. Islam, Zaid B. Mahbub, Muhammad A. Kadir and Saad Kashem (2020, mayo, 6) Transfer Learning with Deep Convolutional Neural Network (CNN) for Pneumonia Detection Using Chest X-ray [online] Available: 2004.06578.pdf (arxiv.org)

[32] Pablo Ruiz (2018, octubre, 8) Understanding and visualizing ResNets [online] Available: Understanding and visualizing ResNets | by Pablo Ruiz | Towards Data Science

[33] Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton (2020, abril, 14) ImageNet Classification with Deep Convolutional Neural Networks [online] Available: ImageNet Classification with Deep Convolutional Neural Networks (neurips.cc)

[34] Prabin Nepal (2020, julio, 30) AlexNet Architecture Explained [online] Available: AlexNet-1.png (960×540) (learnopencv.com)

[35] Great Learning Team (2020, junio, 24) AlexNet: The First CNN to win Image Net [online] Available: AlexNet: The First CNN to win Image Net | What is AlexNet? (mygreatlearning.com)