UNIVERSIDAD POLITÉCNICA DE CARTAGENA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

Aplicación de algoritmos Machine Learning para un vehículo autónomo

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA TELEMÁTICA

Autor: Pedro Antonio Sánchez Romero

Director: Juan Carlos Jacobo Aarnoutse Sánchez

Codirector: Javier Vales Alonso

Cartagena, abril de 2021

AGRADECIMIENTOS

Para comenzar, me gustaría agradecer a los dos profesores que me han guiado en este proyecto, el Profesor Dr. Juan Carlos Jacobo Aarnoutse Sánchez y el Profesor Dr. Javier Vales Alonso por su dedicación y atención a la hora de formarme y aclarar mis dudas.

Por supuesto a la Universidad Politécnica de Cartagena que durante estos años me ha formado como estudiante y me ha preparado para afrontar cualquier proyecto el día de mañana. En especial con el Departamento de Tecnologías de la Información y las Comunicaciones, con el que he tenido más contacto y ha hecho tan cómoda mi etapa universitaria.

Por último, me gustaría agradecer a mis amigos por apoyarme y a mi familia, pues sin ella esto no habría sido posible. Gracias a todos me llevo un cariñoso recuerdo de esta universidad y de mi paso por ella, que además me ha formado como persona.

Muchas gracias.

RESUMEN

Pedro Antonio Sánchez Romero. "Aplicación de algoritmos Machine Learning para un vehículo autónomo", Cartagena, abril de 2021

Desde la creación del vehículo motorizado se ha pensado en la idea de un coche capaz de viajar por sí solo, de manera autónoma. No ha sido hasta hace relativamente pronto que se han podido crear sistemas de conducción autónoma, siendo los más conocidos los vehículos de la marca Tesla.

Esta idea cuenta con la potencialidad de aumentar la seguridad vial, pues un gran porcentaje de los muertos en carretera (40%) son causa de siniestros debido a la salida de la vía de circulación, problema que se podría solucionar si el vehículo fuera capaz de percatarse de dicha salida y corregir de manera automática la dirección. Por otro lado, un sistema de frenado automático permitiría responder ante un obstáculo aparecido repentinamente, como un peatón o una moto.

Por ello, diseñar un vehículo autónomo sencillo para fines didácticos podría incentivar al estudio de estas tecnologías y con ello un desarrollo más acelerado de estos sistemas a gran escala.

Dependemos en gran medida del hardware disponible. Por ejemplo, se va a utilizar un chasis dirigido por una Raspberry para este proyecto. Estamos limitados a la capacidad de dicho chasis, ya sea en cuanto a velocidad, dirección o durabilidad.

Suponiendo que todo funcione correctamente, habremos diseñado un vehículo capaz de seguir una trazada por sí solo, sin intervención humana.

ABSTRACT

Pedro Antonio Sánchez Romero. "Application of Machine Learning algorithms for an autonomous vehicle", Cartagena, April 2021

Since the creation of the motorized vehicle, people have been dreaming of the idea of a car able to travel autonomously. Nevertheless, it was not until very recently that autonomous driving systems have been developed; the best known of them are Tesla vehicles.

This idea has the potential to increase road safety, since a large percentage of road deaths (40%) are cause by road exit accidents, a problem that could be solved if vehicles were capable of notice such exit and automatically correct the direction. The possibility of reacting to an obstacle that appears suddenly, such as a pedestrian or a motorcycle, could be addressed by implementing an automatic braking system.

Therefore, designing a simple autonomous vehicle for educational purposes could encourage the study of these technologies and, with it, a more accelerated development of these large-scale systems.

We are highly dependent on available hardware. For example, a Raspberry-driven chassis will be used for this project. We are limited to the capabilities of that chassis, whether it is speed, steering or durability.

Assuming that everything works correctly, we will have designed a vehicle capable of following a line by itself, independent of human intervention.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

CONTEXTO

Un vehículo autónomo, también conocido informalmente como sin conductor o auto conducido, es un vehículo capaz de imitar las capacidades humanas en control, manejo e interpretación en la circulación por vías públicas, conviviendo con otros vehículos, peatones, señales e imprevistos.

Los vehículos perciben el entorno mediante técnicas complejas como láser, radar, sistema de posicionamiento global y visión computarizada. Generalmente son capaces de recorrer carreteras previamente programadas y requieren una reproducción cartográfica del terreno, con lo cual si una ruta no está recogida por el sistema se puede dar el caso que no pueda avanzar de forma coherente y normal.

A fecha de hoy diversos fabricantes de vehículos ya disponen de modelos autónomos, pero para su implantación definitiva mundial se requiere de un ajuste de varios aspectos relacionados con la seguridad vial y con las compañías de seguros. Son algunas de las dudas que conciernen a una forma de transporte que está cerca de ser realidad en pocos años según empresas involucradas en su desarrollo, como Google, Daimler AG, BMW, Renault, Ford o Volvo, así como Bosch o Delphi, en el área de componentes y electrónica.

Dado que la legislación es diferente en cada país, en algunos ha sido posible ver algunos de estos vehículos circulando por sus vías. Por ejemplo, en agosto de 2016 la empresa estadounidense nuTonomy, filial del MIT, lanzó el primer taxi autónomo del mundo en Singapur. Uber opera también con coches autónomos en las ciudades de Pittsburgh y San Francisco desde finales de 2016. El 19 de marzo de 2018 se produjo el primer atropello mortal por un vehículo sin conductor. Una mujer falleció en Tempe, Arizona, tras ser atropellada por un vehículo sin conductor operado por Uber. Como consecuencia, Uber anunció que suspende las pruebas que se venían llevando a cabo con vehículos autónomos en Tempe, Pittsburgh, Toronto y San Francisco.

Algunos de los beneficios más importantes de la implementación de los vehículos autónomos son:

- Aumento de la seguridad vial respecto a los conductores humanos.
- Aumento de la accesibilidad de las personas que no pueden conducir.
- Aumenta la eficiencia energética y la calidad del aire debido a técnicas para mejorar el tráfico.
- Aumento de la calidad del espacio público, reduciendo la cantidad de espacio requerido para estacionar y aumentando la capacidad de las carreteras debido a la reducción de la distancia de seguridad.

- Aumento de la eficiencia económica, reduciendo los costos de los conductores humanos contratados, reduciendo el número de multas, de policía de tráfico y de los seguros.

En este proyecto se implementará un vehículo capaz de seguir una trazada. Para ello, se hará uso de redes neuronales, aplicando una regresión. Se implementará con la librería *tensorflow*, conocida por sus buenos resultados y su optimización a la hora de crear y entrenar dichas redes. En concreto se trata de un caso de Deep Learning, pues la red contará con más de una capa oculta.

Objetivos

El objetivo final de este proyecto es el desarrollo de un vehículo que de forma autónoma sea capaz de seguir una trazada, marcada en el suelo con una cinta azul. Para ello se implementará una red neuronal profundo, entrando en el terreno del Aprendizaje Profundo o Deep Learning.

Además de este objetivo principal, se desarrollarán una serie de conocimientos relacionados con la titulación cursada para realizar este trabajo, tales como:

- Aptitudes sobre el planteamiento, desarrollo y resultado de un problema.
- Habilidades sobre el Deep Learning tales como diseño de una red, su entrenamiento, evaluación de resultados y posibles mejoras en su diseño.
- Recopilación de datos válidos para conformar un dataset.
- Técnicas de procesado de imagen y de datos.
- Comunicación con un dispositivo vía ssh e intercambio de archivos vía scp.
- Recabar información de interés y actuar en consecuencia a lo estudiado.
- Familiarización con las librerías de Python más famosas, como *numpy, opencv, tensorflow, os, pandas,* etc.

Estructura de la memoria

La memoria se va a organizar en cuatro capítulos:

1. Capítulo 1. Contiene la introducción a la temática, explicada a grandes rasgos, además de su importancia en el presente y el futuro. Incluye la motivación a realizar este proyecto.

2. Capítulo 2. Enfocado a explicar el funcionamiento de los vehículos autónomos existentes, basando esta explicación en estudios contrastados y publicaciones verificadas, más conocido como estado del arte de esta tecnología. Incluirá además los componentes que formarán el vehículo de pruebas, así como su montaje.

3. Capítulo 3. Dedicado a explicar los conceptos usados en el desarrollo del proyecto tales como el Deep Learning, así como la metodología utilizada para obtener el resultado final. Se expondrá dicho resultado y su justificación.

4. Capítulo 4. Con el fin de comentar los resultados obtenidos en el capítulo anterior.

Para acabar esta memoria, se hará una valoración del trabajo realizado así como su conclusión.

1 CAPÍTULO 1. INTRODUCCIÓN

1.1 INTRODUCCIÓN

Cada vez más fabricantes de automóviles implementan dispositivos que ayudan a la conducción, como pueden ser sensores de proximidad, cámaras traseras, detección de vehículos en el punto ciego de los espejos, sistemas que impiden salirse del carril de circulación, etc.

Estos avances tienen como fin conseguir una conducción autónoma del vehículo, de manera que sea capaz de circular por sí mismo sin suponer un peligro para el resto de los conductores y usuarios de la vía. Un ejemplo de este desarrollo es la empresa Tesla, conocida por sus vehículos 100% eléctricos, capaces de circular por cualquier ciudad, respetando tanto señales, como otros vehículos y peatones.

Estos sistemas son de gran utilidad y relevancia, debido a que son capaces de prevenir accidentes ocasionados por una conducción pésima, mala condición del conductor o razones ajenas al mismo. En un futuro, esta tecnología será predominante en el sector automovilístico por las razones mencionadas anteriormente.

Este trabajo guarda gran similitud con un estudio muy conocido ocurrido entre 1986 y 1991 en Pittsburgh, Pennsylvania. Se llevó a cabo por la Universidad de Carnegie Mellon[1] y su nombre fue ALVINN (Autonomous Land Vehicle In a Neural Network)[2]. Este proyecto consistía en el desarrollo de una red neuronal con una capa de neuronas oculta cuyo objetivo era seguir una carretera. Tomando imágenes con una cámara y un láser como entradas de la red, calculaba la dirección que debía tomar el vehículo. En esencia, este trabajo intenta replicar el proyecto ALVINN para fines didácticos.

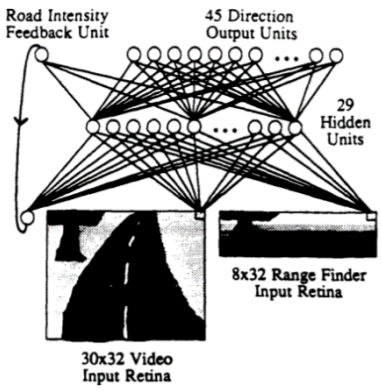


Figura 1. Arquitectura ALVINN

1.2 MOTIVACIÓN

La tecnología usada en la industria automovilística ha estado evolucionando desde la creación del primer coche en 1886 por Karl Friedrich Benz, añadiendo a lo largo del tiempo sistemas de ayuda a la conducción. El final de este largo camino culmina en la creación de automóvil autónomo.

Con una correcta implementación de esta tecnología, se conseguirían rebajar considerablemente el número de accidentes y de fallecidos en la carretera. Según datos de la DGT, en 2019 hubo 1101 fallecidos y 4433 ingresos hospitalarios; en 2020 el número bajó a 870 y 3463 respectivamente, motivado en parte por el confinamiento producido por la pandemia[3].

Por otra parte, se registraron en 2020 en torno a 44800 positivos en controles de alcoholemia, lo que supone un peligro en la carretera tanto para ellos como para el resto de los conductores.

Otra razón son las salidas de la vía, que son la causa del 44% de los siniestros, ya sean debidos a un exceso de velocidad, distracciones o problemas del conductor como puede ser la somnolencia. Según palabras del presidente de la Asociación Española de la Carretera, Juan Francisco Lazcano: "En 2020 ocurrió algo que llama la atención, y es que la salida de vía fue la causa del 44% de los siniestros, cuando la media de los años anteriores era del 40%", señala. "Esta circunstancia se debe habitualmente a las distracciones, pero apuntaría también a los excesos de velocidad. Y unas carreteras más vacías han podido influir en el aumento", añade[4].

Por ello, con un adecuado sistema de conducción autónoma se podría llegar a minimizar estos accidentes, evitando que el vehículo abandone el carril o salidas de la vía, por ejemplo.

2 CAPÍTULO 2. ESTADO DEL ARTE

2.1 HISTORIA

La idea de un vehículo autónomo comienza en 1939, presentada en la feria de muestras Futurama, durante la Exposición Universal en Nueva York, donde se contemplaba la idea de un vehículo sin conductor[5].

Figura 2. Primer vehículo autónomo en Japón [6]

Sin embargo, este sueño no pudo hacerse realidad hasta 1977, en los laboratorios Tsukuba Mechanical Engineering en Japón, donde se creó un vehículo capaz de seguir unas marcas blancas pintadas en la carretera (muy parecido al objetivo de este proyecto) y alcanzar una velocidad máxima de 30 km/h[7].

Unos años más tarde, en 1986, en la Universidad de Múnich se consiguió el primer vehículo capaz de controlar volante acelerador y freno a la vez, alcanzo velocidades de hasta 100 km/h. Esto fue posible gracias a una evaluación en tiempo real de una secuencia de imágenes (de nuevo, la misma idea que este proyecto). Tan solo un año más tarde, este vehículo se probó en un tramo de Autobahn, alcanzando los 90 km/h sin incidentes[5], [8].

Figura 3. Vehículos 'VaMoRs' y 'VaMP' de Ernst Dickmann [9]

Se lanzaron varios prototipos, vistos en octubre de 1994 cerca del aeropuerto de París. Este vehículo fue un Mercedes 500 SEL, al que se le llamó 'VaMP'. Fue capaz de recorrer más de 1000 kilómetros en la carretera que rodea París y llegar a velocidades de hasta 130 km/h.

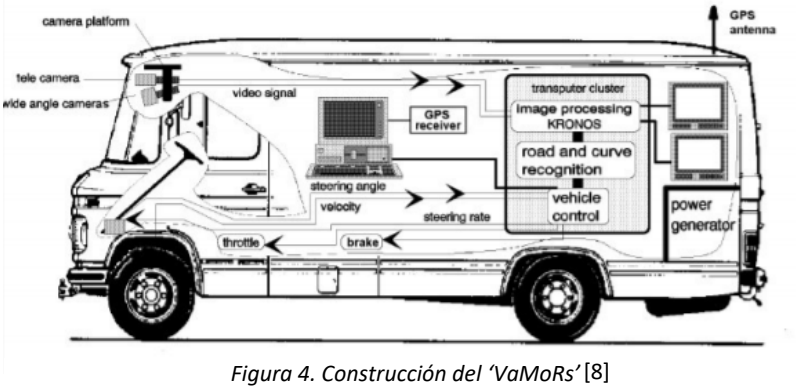


Figura 4. Construcción del 'VaMoRs' [8]

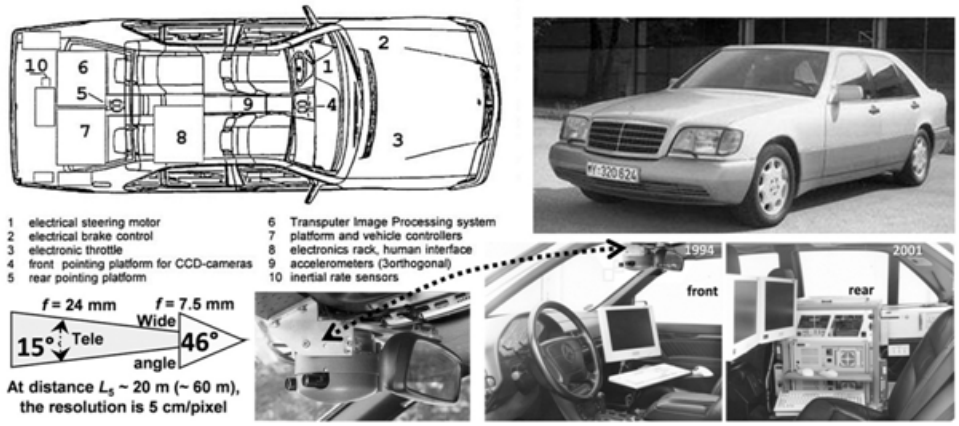


Figura 5. Construcción del 'VaMP' [10]

Estos avances despertaron interés, lo cual dio nacimiento al lanzamiento de PROMETHEUS (PROgraMme for a European Traffic of Highest Efficiency and Unprecedented Safety, 1987-1995)[11] en Europa con el fin de mejorar el tráfico en las carreteras europeas, aumentando la seguridad, la rentabilidad, la comodidad, la eficiencia y la contaminación.

Por otra parte, Estados Unidos desarrollo 11 vehículos automatizados en el laboratorio de la Universidad Carnegie Mellon, que en 1995 viajaron 3000 millas (4828 km) autónomamente el 98% del tiempo[12].

Figura 6. Vehículo 'Navlab 5' de Carnegie Mellon [12]

El siguiente paso relevante ocurrió en 2004, cuando la Agencia de Proyectos de Investigación Avanzada de Defensa (DARPA) de América ofreció una recompensa de un millón de dólares para un vehículo autónomo capaz de conducir durante 150 millas a través del desierto de Mojave en California[12]. El desafío estaba en el terreno escarpado del desierto, muy diferente al asfalto. Ninguno de los concursantes llegó a lograrlo, siendo el 'Sandstorm' de la Universidad de Carnegie Mellon el que recorrió una mayor distancia, de 7.3 millas.

Figura 7. Vehículo 'Sandstorm' de Carnegie Mellon [12]

Desde 2010 muchas compañías y grupos de investigación se han esforzado para desarrollar sistemas y características autónomas a los nuevos vehículos. Un buen ejemplo de esto es la compañía Tesla, con su 'Tesla D', capaz de estacionar sin supervisión, tomar el control en carretera e intervenir si piensa que puede hacer una mejor trabajo a la hora de evitar accidentes[13].

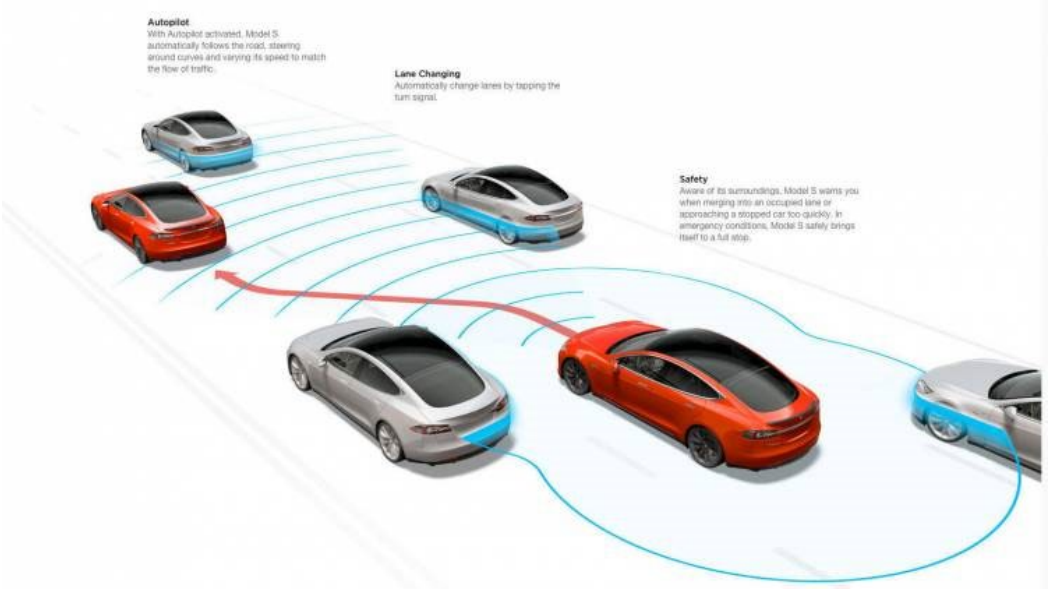


Figura 8. Adelantamiento de un 'Tesla D' usando Autopilot [14]

2.2 NIVELES DE AUTOMATIZACIÓN

Los niveles de automatización de un vehículo indican de qué es capaz el vehículo de realizar por su cuenta. Existen 6 niveles, creados por la Sociedad de Ingenieros Automotrices de los Estados Unidos (SAE). A menor número más implicación del conductor, o dicho de otra manera, menos autónomo es el vehículo. Estos niveles están descritos en el estándar *SAE J3016*[15], publicado en enero de 2014.

Esta clasificación se puede llevar a cabo, siguiendo el estándar *SAE J3016*, en función de cuatro aspectos fundamentales:

- Quién se encarga del movimiento, pudiendo ser este longitudinal (acelerar o frenar) y lateral (la dirección).
- Quién se encarga de la detección y respuesta ante objetos u obstáculos.
- Quién se encarga del respaldo de la conducción, o dicho de otra manera, quién actúa ante situaciones de fallo.
- La capacidad del sistema de conducción de funcionar en situaciones desfavorables (meteorológicas, tráfico, horarias, etc.).

Basándonos en la clasificación NHTSA[16], contamos con 6 niveles de automatización, al igual que la SAE, siendo el nivel 0 el más básico y el nivel 5 la automatización total[17].

2.2.1 Nivel 0: Sin asistencia

El nivel más básico. No ofrece ninguna asistencia al conductor, donde todas las tareas recaen en el piloto, tales como acelerador, freno, luces, etc.

2.2.2 Nivel 1: Asistencia al conductor

El vehículo cuenta con algún sistema de asistencia al movimiento longitudinal o lateral, pero no a la vez. El conductor realiza el resto de las actividades. Un ejemplo podría ser el modo crucero que mantiene la velocidad indicada sin necesidad de que el conductor acelere o frene.

2.2.3 Nivel 2: Automatización parcial

El vehículo cuenta con asistencia al movimiento tanto longitudinal como lateral al mismo tiempo. El resto de las tareas como la respuesta ante un obstáculo o la detección de otros vehículos recaen sobre el conductor. Es el más extendido en la actualidad. Un ejemplo sería el control de crucero mientras se corrige la trayectoria dentro del mismo carril.

2.2.4 Nivel 3: Automatización condicionada

Al igual que en el nivel anterior, el vehículo ofrece sistemas de asistencia al movimiento, tanto longitudinal como lateralmente y además ofrece mecanismos para la detección de obstáculos y eventualidades. El conductor recibe el papel de usuario listo para intervenir en situaciones no contempladas por el sistema. Esta intervención se da solo en ciertas ocasiones.

2.2.5 Nivel 4: Automatización elevada

El vehículo cuenta con asistencia al movimiento longitudinal y lateral, además de detección de obstáculos y su resolución. A diferencia del nivel anterior, el vehículo cuenta con sistemas de respaldo para reaccionar en caso de emergencia sin necesidad de la actuación humana. Aun así, existen ciertas condiciones en las que la figura del conductor es necesaria, aunque sea mínima. Desaparece como tal la figura del conductor.

En la actualidad, la industria está trabajando en desarrollar sistemas con este nivel de automatización. Un caso existente hoy en día es el 'Tesla Model S', que con su software 'AutoPilot 2.0' es capaz de alcanzar este nivel.

2.2.6 Nivel 5: Automatización completa

El vehículo es autónomo por completo. Controla el movimiento longitudinal y lateral, es capaz de detectar obstáculos y sortearlos, en caso de emergencia cuenta con sistemas de respaldo para solucionar dicha emergencia. La figura del conductor desaparece al desaparecer también las situaciones en las que sería necesaria su intervención.

Tanto en este nivel como en el anterior, sería posible fabricar el automóvil sin mecanismos de control (volante, pedales, etc.) puesto que el vehículo sería capaz de viajar sin intervención humana.

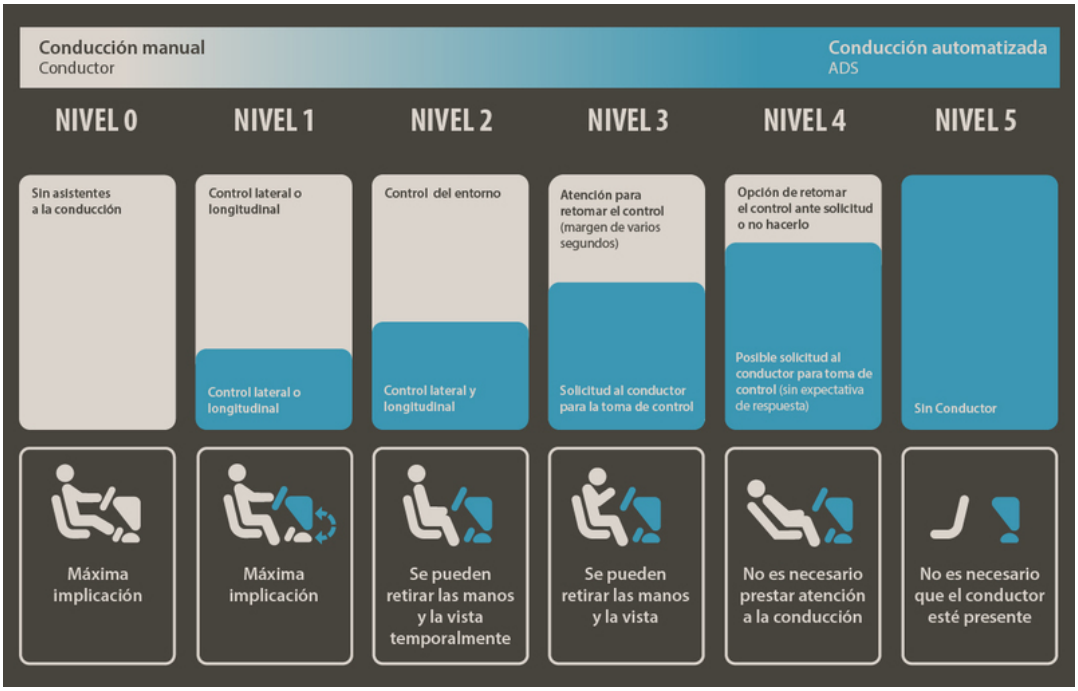


Figura 9. Niveles de automatización [18]

2.3 CIRCUITERÍA

Los vehículos actuales con un nivel de automatización 3 cuentan con un gran abanico de hardware dedicado a la recolección de información y a su procesamiento. Entre otros, se destacan[19]:

- **Ordenador central**: Zona principal de cómputo, donde toda la información recibida de los sensores y cámaras se procesa, determinando en cada momento que decisión es la más acertada (girar, acelerar, frenar, etc.).
- **Radares**: Sistemas utilizados mayoritariamente en el sector naval y aéreo. Su principal función es la de detectar objetivos u obstáculos, por lo que se

adaptaron para su uso en vehículos de cuatro ruedas como mecanismo de detección.

- **Sensores de ultrasonidos**: Su funcionamiento es similar a los radares, con la diferencia que en estos sensores se utilizan ondas sonoras a alta frecuencia, imperceptibles para el oído humano. Alertan de la distancia del vehículo a otro vehículo o a obstáculos. Principalmente son usados en el aparcamiento o la detección de obstáculos a baja velocidad.
- **Cámaras**: Se distribuyen a lo largo de la carrocería del vehículo para generar una imagen en tres dimensiones de lo que rodea al vehículo. Apoyan al resto de sensores ofreciéndoles información del tráfico, marcas viales, etc.
- **LIDAR**: Genera una visión total del entorno del vehículo, proyectando millones de haces de luz. Gracias a su diseño basado en láser, es capaz de detectar formas y crear un mapa 3D con el apoyo de las cámaras.
- **GPS**: Sistema de posicionamiento y localización por satélite. Con la ayuda de la red de satélites desplegada, permite conocer la posición exacta del vehículo en todo el globo terráqueo. También podría usarse otros sistemas como GALILEO, el sistema europeo.

Estos son los sensores que implementan los vehículos de alta gama, sin embargo, nuestro coche solamente contará con el apoyo de una cámara, suficiente para lograr un nivel de automatización 2 según el SAE, ya que será capaz de controlar el movimiento tanto longitudinal como lateral, pero no sabrá reaccionar ante un imprevisto, como por ejemplo, una salida de pista.

Para el desarrollo de este proyecto, se ha utilizado como ordenar central una Raspberry Pi 3B+[20], a la que se le ha conectado una webcam Logitech C920[21] con la que capturar el circuito y un chasis Cherokey 4WD[22]. Para su control, se ha usado un mando de PlayStation 3 vía Bluetooth.

2.3.1 Raspberry Pi 3B+

La Raspberry Pi 3B es un ordenador de tamaño reducido de bajo coste desarrollada en el Reino Unido. Su principal función fue la de conseguir que la informática llegara al mayor número de personas posible y, gracias a su popularidad, acabó usándose incluso para fines ajenos a la robótica.

Es una placa sencilla, aunque con capacidad para montar y ejecutar un sistema operativo basado en Linux, como puede ser Debian.

Cuenta con 1GB de RAM, conexión Ethernet, 40 pines dedicados al GPIO, 4 puertos USB, salida de audio Jack 3.5, salida HDMI, puerto microSD y conexiones

específicas como una entrada para la cámara Raspberry Pi y un puerto para conectar una pantalla táctil.

Su uso es muy amplio y variado, siendo capaz de actuar como servidor web, estación de videoconferencia, sistema de streaming de video y audio (funciones de tv box), entre otros.

En este proyecto, se usará mediante un sistema operativo Raspbian, en el que ejecutaremos ciertos archivos en Python[23] tanto para recoger los datos previos, como a la hora de su conducción autónoma.

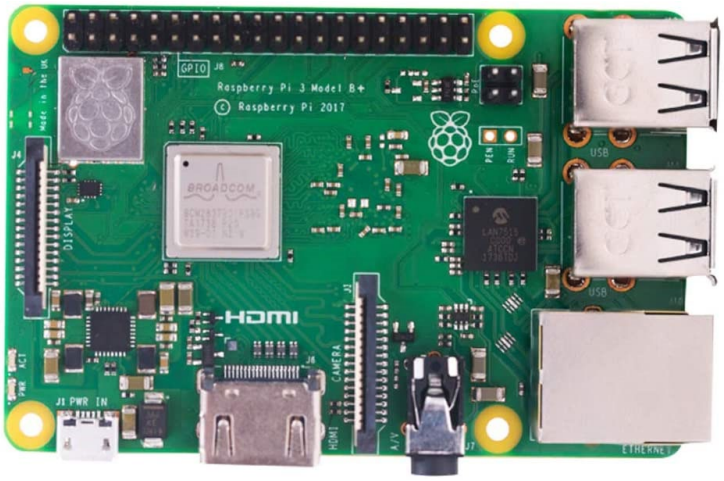


Figura 10. Raspberry Pi 3B+[20]

2.3.2 Webcam Logitech C920

Para este trabajo se ha seleccionado la cámara Logitech C920. Este dispositivo se encargará de captar las imágenes que se usarán tanto para la creación del dataset como para la conducción autónoma. Ofrece una calidad de imagen buena acompañado de un enfoque claro. Es importante recabar información de calidad para que el sistema funcione correctamente.

Figura 11. Logitech C920 [21]

2.3.3 Cherokey 4WD

Se trata de un robot móvil compatible con la mayoría de los microcontroladores como Arduino UNO[24], Arduino MEGA 2560[25], etc.

El chasis consta de 4 motores de corriente continua independientes, conectados cada uno de ellos directamente a una rueda. Cuenta con una segunda base, para así habilitar la opción de expansión a otros sistemas adicionales.

Cuenta con un chip L298P, que es un controlador de doble canal para poder manejar los cuatro motores simultáneamente, siendo capaz de mover ambos ejes de motores con 6-12 Voltios a una corriente máxima de 2 amperios. Es posible sincronizar los motores del mismo eje longitudinal para que giren en el mismo sentido, mecanismo similar al de un tanque de guerra. Este sistema es el que se aplicará para este proyecto.

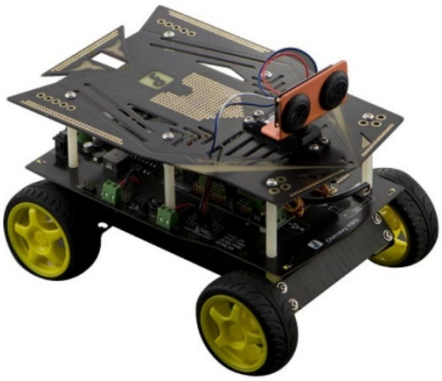


Figura 12. Cherokey 4WD [26]

2.3.4 Mando PlayStation 3

Un mando común de la consola PlayStation 3. Se usará para dirigir al vehículo en la etapa de recolección de imágenes para generar el dataset de entrenamiento. Se conectará a la Raspberry mediante Bluetooth, configurado previamente.

Principalmente se usarán los joysticks, el derecho para controlar los motores del eje derecho y el izquierdo para controlar los motores del eje izquierdo. Su funcionamiento se puede resumir en que las ruedas giran en el mismo sentido vertical que el joystick, es decir, si lo desplazamos hacia el frente, ese lado girará para lograr que el vehículo se mueva hacia delante.

Figura 13. Mando PlayStation 3 [27]

2.4 MONTAJE

Se explicará a grandes rasgos en qué ha consistido el montaje del vehículo:

- Usando la base más elevada del chasis, se ha anclado la Raspberry Pi 3B+. A este chasis se le acoplará además la webcam y las baterías necesarias para su funcionamiento, una que alimentará a la Raspberry y otra acoplada en los bajos que alimentará los motores.
- Usando los pines GPIO disponibles en la Raspberry, concretamente los pines 31, 33, 35 y 37, se comunicará a los pines del Cherokey, suministrándole la señal que indique que debe activar ese motor.
- La webcam se conectará usando uno de los puertos USB disponibles en la Raspberry.
- Se le ha conectado además a los puertos GPIO un led que indica cuando la cámara está activa y un interruptor con el que apagar la Raspberry.
- La Raspberry estará conectada vía wifi. Tiene habilitado un servidor ssh así como un servidor VNC, por lo que es posible conectarse a ella usando un cliente ssh (por ejemplo Putty en Windows, o el comando ssh en Linux) y mediante un cliente VNC (por ejemplo VNC Viewer), que crea una ventana del escritorio de la Raspberry, facilitando así la edición de archivos y la transferencia de estos.

El resultado del montaje puede verse en la Figura 14.

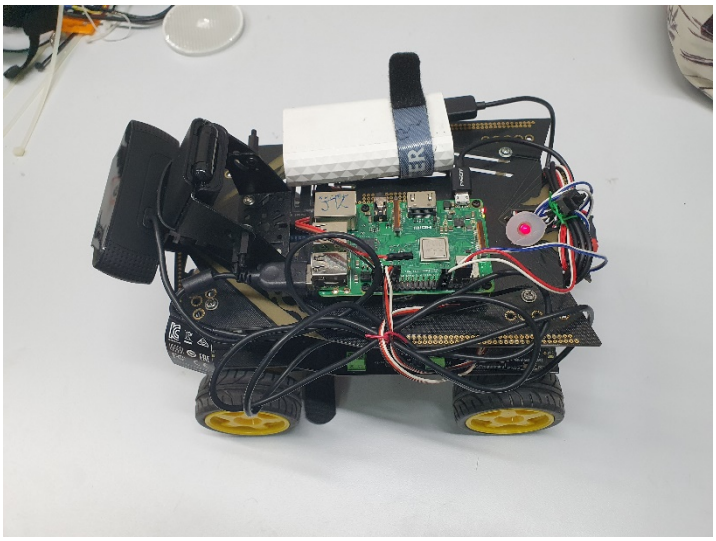


Figura 14. Vehículo de pruebas

Hay que asegurarse de que la webcam siempre capture la misma toma, por lo que deberemos tener alguna referencia en el caso de que mueva, ya sea por un choque o por cualquier causa. Sin una homogeneidad en el método para obtener los datos, no serán válidos para su uso futuro. Es necesario medir de la misma manera en todo momento.

Para ello, se ha diseñado una escala de referencia en un folio en blanco A3. El vehículo debe colocarse sobre unas marcas indicadas en el mismo, quedado en una posición como la que se muestra en la figura 15.

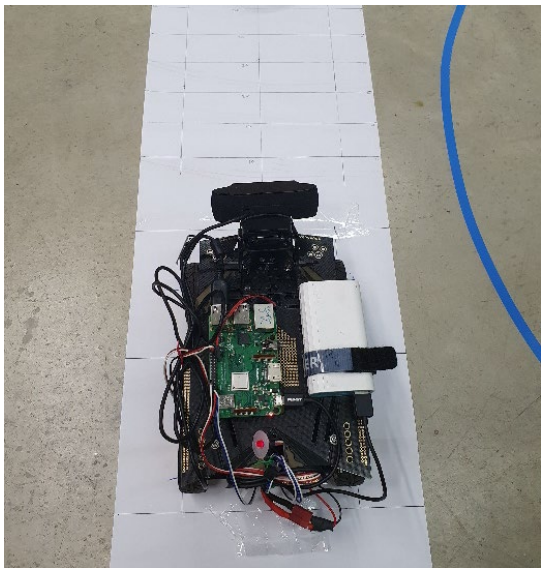


Figura 15. Colocación de la webcam con respecto a medidas

El folio está dividido en rectángulos de 5 centímetros de ancho, simplemente para poder así fijar una elevación de la webcam. El vehículo debe estar bien posicionado, pues adelantar o retrasar la posición de medida afectará negativamente al resultado. El folio además debe estar tenso, no deben quedar arrugas que puedan afectar a la longitud de éste, lo que también afectaría negativamente al resultado.

La figura 16 muestra la imagen que usaremos para fijar la cámara. Se ha usado para la toma de todas las imágenes de muestra, por lo que, si en algún momento la webcam se desvía, se deberá hacer uso del folio de referencia y orientar la cámara hasta capturar una imagen igual a la anterior, que como vemos, alcanza el cuarto rectángulo.

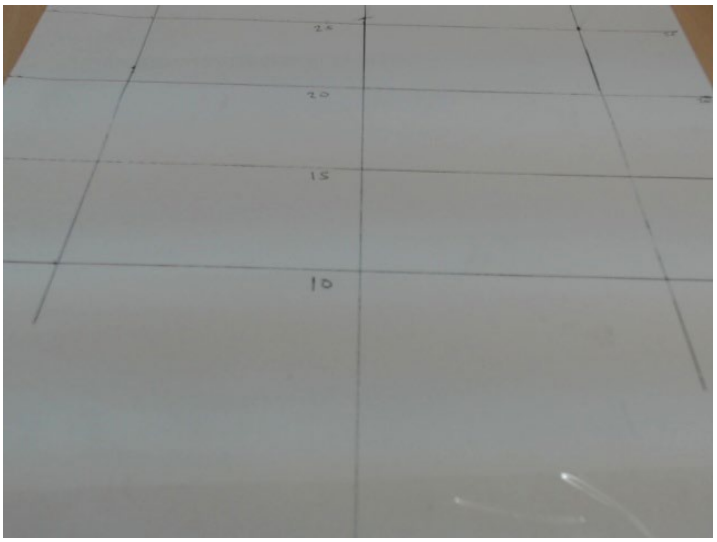


Figura 16. Imagen de referencia captada por la webcam

El vehículo deberá ser capaz de seguir una trazada marcada por una cinta azul, la cual se ha colocado en el suelo dibujando un circuito. Con el fin de obtener mejores muestras y más variedad, se han diseñado dos circuitos, uno con curvas más cerradas y otro con curvas más abiertas.

Figura 17. Circuito 1

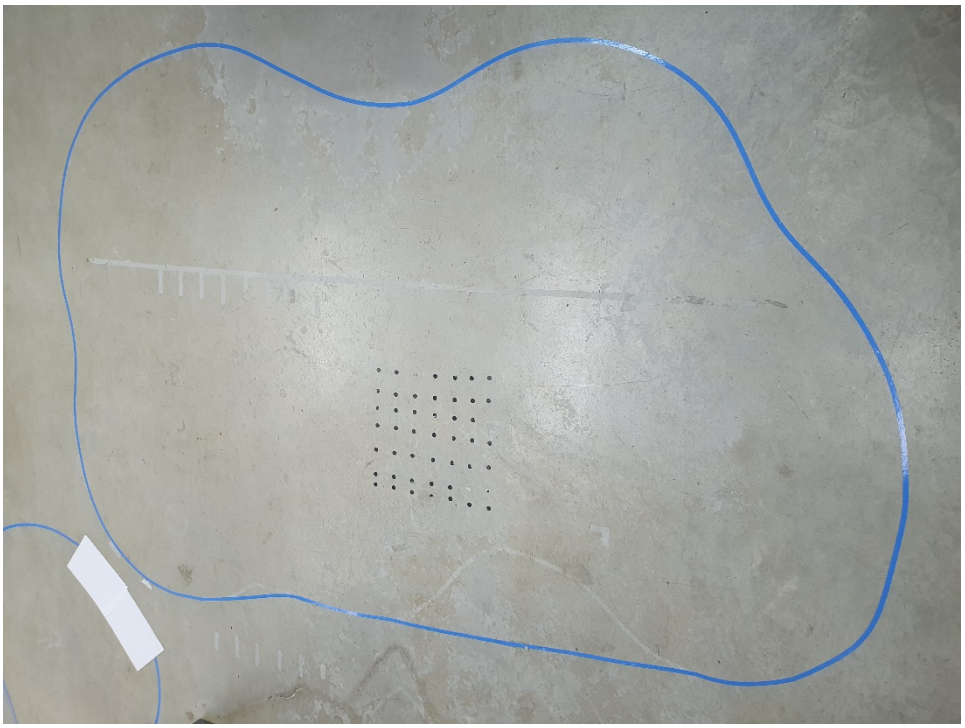


Figura 18. Circuito 2

3 CAPÍTULO 3. METODOLOGÍA

Una vez explicado el trasfondo de este proyecto, además del hardware que se va a utilizar, se explicará el software usado. La idea es usar un lenguaje de alto nivel, es este caso Python, para diseñar primeramente un programa que haga controlable el vehículo con el mando de la PlayStation 3, además de que periódicamente tome fotografías del fragmento de circuito que visualiza la webcam. Más tarde se someterán esos datos a un preprocesado con el fin de construir finalmente el dataset útil para el entrenamiento de la red.

Para la implementación de dicha red, se hará uso de las redes neuronales artificiales, usando para ello la famosa librería de Python *Tensorflow*[28]. Una vez diseñada y entrenada la red, el vehículo deberá ser capaz de reconocer el tramo de circuito que está visualizando, predecir qué valor se ajusta más fielmente a dicho tramo, y actuar sobre los motores para conseguir tanto la velocidad como la dirección necesaria.

3.1 INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL

Según la Real Academia Española[29], la inteligencia es "Capacidad de entender o comprender" y "Capacidad de resolver problemas", por lo que se podría definir la Inteligencia Artificial (IA) como la capacidad de entender, comprender y resolver problemas artificialmente o dicho de otra manera, la inteligencia aplicada a las máquinas.

Aunque también es posible usar la definición que dejó uno de los primeros estudiosos de esta rama, como es Marvin Minsky[30], que definió la Inteligencia Artificial como "La Inteligencia Artificial es la ciencia de construir máquinas para que hagan cosas que, si las hicieran humanos, requerirían inteligencia".

Por lo que para que un sistema pueda considerarse inteligente, éste debe ser capaz de aprender, lo que se conoce como Aprendizaje Autónomo. Se puede resumir en la capacidad de asociar unos valores de entrada a unos valores de salida, como por ejemplo, asociar una imagen en la que aparece un perro con que en esa imagen hay un perro. Puede parecer algo trivial, pero hay que recordar que un sistema no tiene ninguna información ni inteligencia como para saber que en esa imagen existe un perro, por lo que se deberá de entrenar, usando otras imágenes como ejemplo hasta que sea capaz de diferenciar las imágenes donde haya perros.

Uno de los diseños más usados y que mejor funcionan, son las Redes Neuronales Artificiales (RNA), que intentan emular el funcionamiento de una neurona humana, conectándose con otras neuronas para compartir información y poder llegar a un resultado.

3.2 REDES NEURONALES ARTIFICIALES

Una Red Neuronal Artificial está compuesta por la unión de múltiples neuronas, estando éstas distribuidas en capas. Dichas neuronas reciben un número n de entradas, complementadas con unos pesos w, que condicionarán la importancia de cada entrada. Cada entrada tendrá un peso asociado. La neurona está compuesta por un sumador y una función de activación.

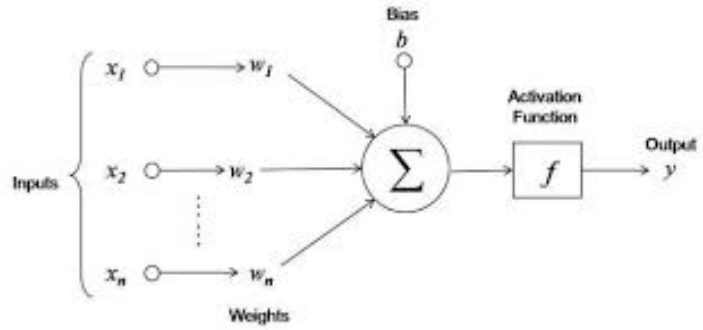
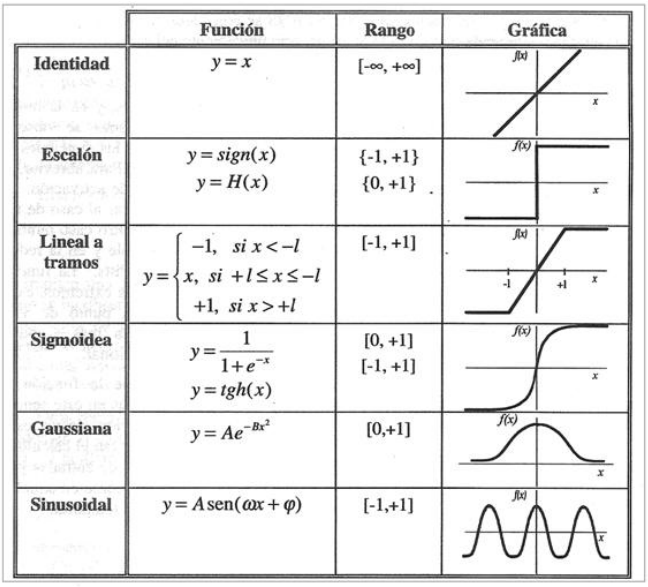


Figura 19. Esquema de una neurona artificial [31]

Además, será necesario la adición de un término independiente, llamado sesgo o bias. Este término sirve para controlar qué tan predispuesta está esa neurona a disparar un 1 o un 0 independiente de los pesos. Se puede reescribir la imagen anterior en forma de ecuación:

$$y = f(\sum_{1}^{n} w_n x_n + b)$$

La función de activación define la salida de una neurona dada una serie de entradas. Existen distintos modelos para la función de activación, aunque los más conocidos son la función lineal, función escalón, función sigmoide, la función gaussiana y la función relu.



	Función	Rango	Gráfica
Identidad	$y = x$	$[-\infty, +\infty]$	
Escalón	$y = sign(x)$ $y = H(x)$	{-1, +1} {0, +1}	
Lineal a tramos	$y = \begin{cases} -1, & si\ x < -l \\ x, & si\ +l \le x \le -l \\ +1, & si\ x > +l \end{cases}$	[-1, +1]	
Sigmoidea	$y = \frac{1}{1+e^{-x}}$ $y = tgh(x)$	[0, +1] [-1, +1]	
Gaussiana	$y = Ae^{-Bx^2}$	[0,+1]	
Sinusoidal	$y = A\,\mathrm{sen}(\omega x + \varphi)$	[-1,+1]	

Figura 21. Funciones de activación habituales [33]

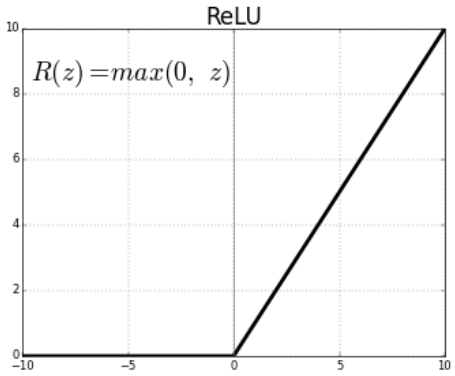


Figura 20. Función de activación Relu[32]

Una vez definida una neurona, se puede ampliar la explicación a una red neuronal. El aspecto general de una red artificial está compuesto por una capa de entrada, una capa intermedia u oculta y una capa de salida. Esta capa intermedia es opcional y se pueden clasificar de dos maneras:

- **Redes Neuronales Monocapa:** Compuestas de una única capa de neuronas, localizadas en la capa de salida. Usadas normalmente para tareas de auto asociación. Aunque su tiempo de cómputo es muy bajo, no son capaces de crear patrones complejos, generando únicamente soluciones a problemas lineales.
- **Redes Neuronales Multicapa:** Compuestas por más de una capa de neuronas, repartidas en las capas intermedias y la capa de salida. Pueden producir patrones complejos, aunque su tiempo de entrenamiento es mayor, generando soluciones a problemas no lineales.

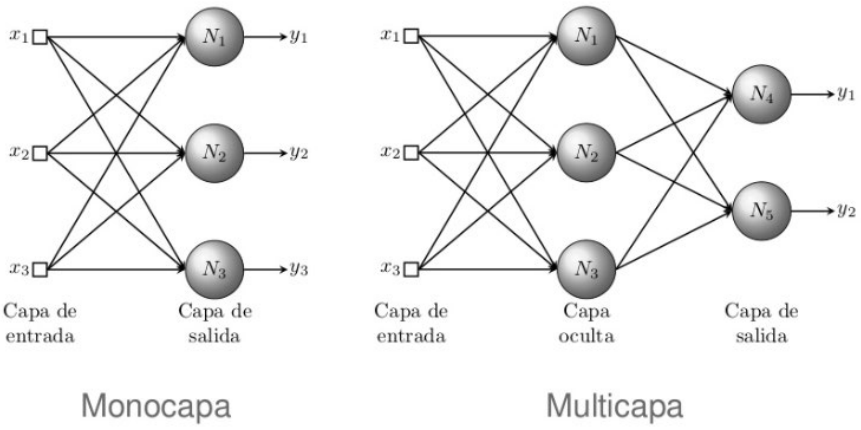


Figura 22. Representación de Red Neuronal Monocapa y Multicapa [34]

3.3 DEEP LEARNING

Partiendo de la idea de que una red neuronal necesita aprender, surge una cuestión, ¿cómo aprende la red? Existen tres distintas filosofías o estilos a la hora de resolver esta cuestión[35]:

- **Aprendizaje supervisado**: Se acompañan los datos de entrada con una etiqueta, indicando el valor de salida asociado a esas entradas. Indicamos a la red que valor deben tener un conjunto de entradas.
- **Aprendizaje no supervisado**: Los datos de entrada no están acompañados de ninguna etiqueta. La red debe ser capaz de diferenciar los datos en áreas o clústeres según su similitud. No se le da a la red ninguna clase de realimentación o feedback.
- **Aprendizaje por esfuerzo**: Es una unión de los dos métodos anteriores. La red usará un aprendizaje no supervisado para dividir las muestras, para más tarde intervenir con un aprendizaje supervisado indicándole si la división se ha realizado correctamente.

Un subcampo de este aprendizaje es el aprendizaje profundo o Deep Learning. Se caracteriza por su capacidad para resolver problemas muy complejos. Necesariamente una red neuronal profunda debe contar con más de una capa intermedia u oculta[36] y necesitan de una gran cantidad de datos de entrada para conseguir un resultado óptimo.

Estas redes cuentan en la mayoría de los casos con múltiples capas ocultas, con un número elevado de neuronas y por lo tanto, un número muy elevado de cálculos. A diferencia de una red tradicional, el coste computacional de una red profunda es notablemente mayor con lo que ello conlleva: un hardware potente y capacidad de almacenamiento. Este problema se subsano en la década de los 90, donde se introdujo la utilización de GPUs para acelerar el cómputo, lo que se mantiene hoy en día.

El aprendizaje sigue un modelo jerárquico, es decir, cada capa oculta toma los datos de la capa anterior, filtrando cada vez patrones más complejos. A mayor profundidad, mayor cantidad de patrones cada vez más complejos y por tanto mejor precisión de la red ante nuevas entradas.

Para este proyecto se va a usar una Red Neuronal Multicapa.

3.4 REDES NEURONALES MULTICAPA

En esencia, es una red neuronal artificial compuesta de múltiples capas, entre ellas una capa de entrada, una capa intermedia como mínimo y una capa de salida, con la capacidad de resolver problemas no lineales. A diferencia de un perceptrón cuya función de activación es de tipo Heaviside (función escalón), una red neuronal multicapa puede tener cualquier función de las mencionadas anteriormente (gaussiana, sigmoidal, tangencial, etc.). Uno de los requisitos para la creación de una de estas redes es que las funciones de activación usadas en las capas deben ser derivables o, al menos, parcialmente derivables, por ejemplo la función sigmoide para el primer caso y la función relu para el segundo caso. Las neuronas de las capas ocultas tendrán la cualidad de aprender distintos patrones ocultos en los datos de entrada, característica común a la subrama del Deep Learning.

Se trata de un aproximador universal, capaz de aproximar cualquier función continua, siempre y cuando se tenga un número elevado de neuronas ocultas.

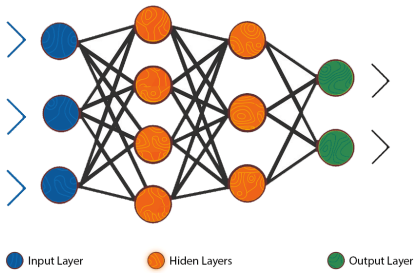


Figura 23. Red Neuronal de tres capas

3.5 DESARROLLO DEL PROYECTO

Como ya se comentó anteriormente, el primer paso para atacar este proyecto fue el montaje del vehículo de pruebas. Posteriormente, se configuró para conectar un mando de PlayStation 3 vía Bluetooth a la central de cómputo con el que se iba a guiar al vehículo a través de la trazada.

Con estos requisitos previos cumplidos, da comienzo la fase de captura de datos. El programa utilizado para controlar el vehículo ya estaba prediseñado, por lo que no se comentará.

Con la ayuda de un piloto, el vehículo recorre la trazada de la mejor manera posible, ya que cuanto más acertada sea la conducción, más sencillo será el entrenamiento de la red. Se controla mediante los joysticks del mando, en el eje

vertical con un rango de valores entre -1 y 1, siendo -1 joystick hacia delante y 1 hacia atrás. Los valores registrados en cada uno de los joysticks del mando se emplean para controlar la velocidad y dirección de giro de las ruedas motoras de cada lado del vehículo (joystick izquierdo para el par de ruedas de la izquierda del vehículo y joystick derecho para las ruedas del lado derecho).

Gracias al software prediseñado, durante la fase de toma de imágenes de muestra se capturan imágenes a una razón de diez por segundo, almacenándolas con la siguiente información:

- Fecha y hora de la captura.
- Valor del joystick izquierdo y derecho.

De esta forma el dataset estará compuesto por el conjunto de imágenes que contienen en el nombre del archivo los datos relativos a la fecha y hora de la imagen, así como los valores de los dos joysticks en ese instante de tiempo. La figura 24 muestra un ejemplo de imagen del dataset.

Figura 24. imagen_2021-02-16 14_38_52.252954_EjeIzda1_-0.041259765625_EjeDcha4_-1.0

Con este procedimiento de toma de datos, se realizaron dos tandas de imágenes con distinto objetivo: en una de ellas únicamente se usó el rango negativo de valores, es decir, el rango de posibles valores otorgados por los joysticks oscila entre 0 y -1; la otra tanda de imágenes contemplaba el rango completo. Esta diferenciación se hizo por puro afán experimental, para comprobar si el vehículo fuera capaz de realizar giros completos usando solo uno de los ejes de potencia.

De esta manera, se han recogido un total de 7571 y 6312 imágenes respectivamente, variando ligeramente la trazada y circulando en ambos sentidos de los circuitos para recoger la mayor cantidad de información posible.

3.5.1 Preprocesado de las imágenes

Una vez creados los datasets, es necesario aplicarles un procesado, eliminando así características que no sirvan o aporten poco valor, haciendo así más liviano el entrenamiento. Las imágenes capturadas tienen un tamaño de 640*480 pixeles haciendo un total de 921600 características de entrada, teniendo en cuenta que se trata de imágenes a color. Esta cantidad de información es excesiva para este problema, por lo que se va a procesar.

1. Aumento del contraste hasta la saturación del fondo.
2. Conversión a escala de grises.
3. Binarizacion de la imagen realzando la trazada.
4. Redimensionado de la imagen a 24*18 pixeles.
5. Normalización de los valores entre 0 y 1.

Con este simple procesado, las variables de entrada se reducen en un 99.95%, conservando la información necesaria, la trazada.

Figura 26. Imagen contrastada

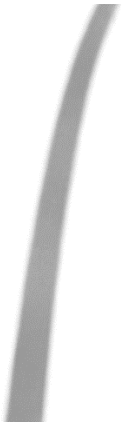


Figura 25. Imagen en escala de grises

Figura 27. Imagen binarizada

Como resultado del procesado, obtenemos un array con tantas filas como número de imágenes disponibles y tantas columnas como número de pixeles en la imagen resultante más dos, pues también se les extrae los valores del eje izquierdo y derecho, almacenándolos junto con el resto de los datos. Se exportará este array en un formato '.csv', para poder ser leídos en la etapa siguiente.

3.5.2 Entrenamiento de la red

Como se ha comentado, la red que se va a usar es una MLP con dos capas ocultas, contará con tantas neuronas en la capa de entrada como número de características tras el procesado (432 características). Se trata de un problema de aprendizaje supervisado, pues las entradas están etiquetadas con los valores que debe tener en la salida, extraídos del nombre de cada una de las imágenes.

El conjunto de datos se ha separado: un 80% para el conjunto de entrenamiento, un 10% para el conjunto de validación y un 10% para el conjunto de test. El conjunto de validación será útil para comprobar el error que produce nuestra red con unas imágenes que no ha visto. Si el error de entrenamiento es mínimo pero el error de validación es alto, significará que la red sufre de sobreajuste, lo que se traduce en una generalización pobre.

Las pérdidas se van a medir mediante la función Joint Square Error, que al igual que el Mean Square Error (MSE), calcula el error cuadrático medio de los valores de salida. A diferencia de los regresores simples, esta red cuenta con dos salidas, por lo que es necesario calcular el error en ambos casos, por ese motivo se usa el Joint Square Error, que se define con la siguiente fórmula:

$$J(w) = \frac{1}{2}\sum_{n=1}^{N} \|y(x_n, w) - t_n\|^2$$

Se ha utilizado como optimizador el Adam, que es una variante del popular Descenso de Gradiente Estocástico (SGD). Mantiene un factor de entrenamiento por parámetro, cada factor de entrenamiento también se ve afectado por la media del momentum del gradiente. Combina las ventajas del AdaGrad y RMSProp[37]:

- **Adaptative Gradient Algorithm (AdaGrad):** mantiene un factor de entrenamiento específico para cada uno de los pesos.
- **Root Mean Square Propagation (RMSProp):** el escalado del factor de entrenamiento se calcula dividiéndolo por la media del declive exponencial del cuadrado de los gradientes.

El entrenamiento de la red puede llevar al sobreajuste, por ello se ha creado un conjunto de validación, que avisará de que la red no ha mejorado su error. Esto es conocido como Early Stopping y es ampliamente utilizado en redes profundas.

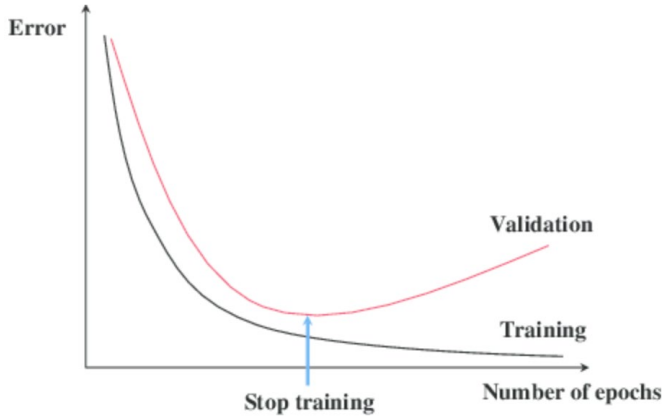


Figura 28. Ejemplo de Early Stopping [38]

Como se puede observar, su función es la de detener el entrenamiento cuando no se mejore el error de validación, lo que significará que nuestra red no mejora la precisión ante la entrada de imágenes no usadas en el entrenamiento, lo que produce sobreajuste u overfitting.

Las capas ocultas se han configurado con una función de activación tipo 'relu', además de agregarle una propiedad 'Dropout' que desactiva una cierta cantidad de neuronas por época. El número de épocas debe ser elevado, pues ya contamos con el Early Stopping para detener el entrenamiento. Un número bajo de épocas puede provocar que no se entrene correctamente, fenómeno llamado underfitting. Dado que es un problema de regresión, se buscará minimizar el error cuadrático medio conjunto, aunque para Python, se seguirá llamando error cuadrático medio (RSE).

La capa de salida cuenta con dos neuronas y una función de activación de tipo lineal, debido a que la red debe predecir dos valores, uno para cada eje del vehículo.

Una vez que el entrenamiento haya concluido, se guardará el modelo y los pesos calculados, para usarlos más adelante en el vehículo.

Por último, hay que mencionar que el entrenamiento se ha ejecutado en un equipo con un procesador de 8 núcleos y 16 hilos, 16GB de RAM y una aceleración por GPU usando una RTX 2060, lo que ha contribuido drásticamente a la velocidad de cómputo, llegando a ser varias veces menor. Gracias a esta paralelización, se pueden probar distintas arquitecturas de red neuronal en un

tiempo reducido, obteniendo así las más adecuada para cada tipo de problema y de datos.

3.5.3 Programación del vehículo

Se trata de un programa sencillo el cual podemos desarrollar rápidamente gracias al código prediseñado para crear los datasets. Deberemos cargar el modelo y los pesos guardados previamente, capturar nuevas imágenes, procesarlas y enviarlas al modelo para que prediga qué valores ocupar en cada eje.

Una vez cargado el archivo de programación usando una consola con ssh o mediante algún software específico (por ejemplo, Filezilla), el vehículo está listo para comenzar su ruta.

En el anexo se detallará todo el código desarrollado así como su fundamento.

4 CAPÍTULO 4. RESULTADOS

En este punto, ya se ha diseñado y desarrollado el proyecto, de manera que se pasará a comentar y analizar los resultados obtenidos. Se había creado dos datasets diferentes para observar la evolución de una misma red ante distintas entradas. En primer lugar comentaremos los resultados en la consola de Python y después como comporta el vehículo con esa configuración.

4.1 COMPORTAMIENTO DE LA RED TEÓRICO

4.1.1 Dataset A

Este dataset corresponde a las imágenes en las que el rango de valores posibles del joystick oscila entre 0 y -1. Se entreno la red numerosas veces, variando el contenido de los conjuntos de datos en cada una de las iteraciones. El mejor valor que se encontró para las pérdidas de validación fue de 0.02097.

Es un resultado bastante bueno, llegando a conseguir predicciones muy precisas, con un margen de error del 4% como se puede ver en este ejemplo:

Eje izquierdo real: -1.0, Eje derecho real: -1.0
Eje izquierdo predicho: -0.9680079, Eje derecho predicho: -0.9914741

Se puede apreciar, que la diferencia es mínima. También podemos visualizar la evolución de las pérdidas de entrenamiento y de validación apoyándonos en estas gráficas:

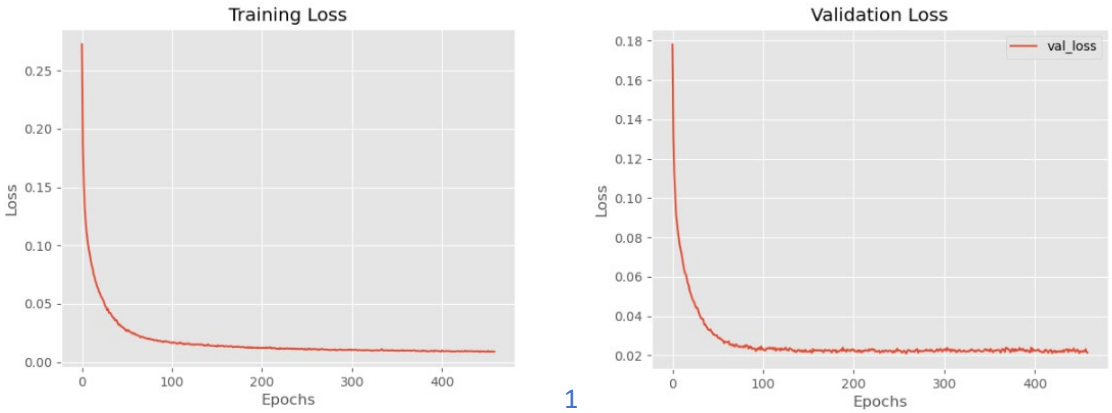


Figura 29. Evolución de pérdidas de validación

Figura 30. Evolución de pérdidas de entrenamiento

Basándonos en estos resultados, se espera que el comportamiento del vehículo sea favorable.

4.1.2 Dataset B

Este otro dataset está compuesto por las imágenes que se tomaron usando el rango completo de los joysticks, es decir, de 1 a -1. Sobre el papel, este comportamiento debería ser mejor ya que el abanico de posibilidades se amplía al añadir este nuevo umbral y dar más libertad de movimiento al vehículo.

Hay que tener en cuenta, que hemos dado más libertad a las salidas pero no hemos incrementado demasiado el número de muestras. Esto perjudica en sí al entrenamiento, pues a más complejo es un sistema, más muestras necesita para aprender. Esto se ve claramente en las pérdidas que obtenemos, repitiendo varias veces el entrenamiento como con el anterior dataset, cuyo mejor resultado da 0.045277, un poco más del doble. Esto puede ser consecuencia de distintas opciones:

- Falta de muestras en el entrenamiento
- Poca calidad en la toma de las muestras
- Poca profundidad de la red neuronal

En cualquier caso, a la hora de predecir cuenta con un error bastante más apreciable que el anterior, en torno al 25% como podemos ver en este ejemplo:

Eje izquierdo real: -1.0, Eje derecho real: -1.0
Eje izquierdo predicho: -0.7521537, Eje derecho predicho: -0.9834996

Esta vez sí que se nota el error. Al dar más libertad a las salidas y no aumentar en consecuencia el dataset, el error se ha visto incrementado, como también podemos ver en estas gráficas:

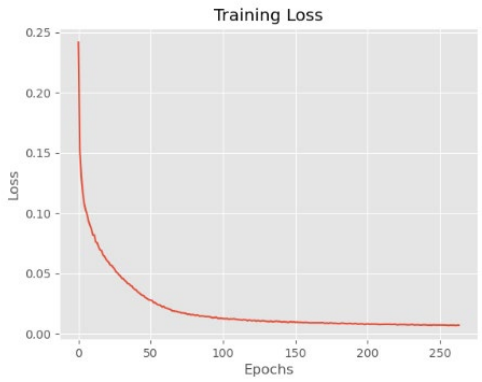


Figura 32. Evolución de pérdidas de entrenamiento

Figura 31. Evolución de pérdidas de validación

Como detalle también podemos observar como la gráfica que representa las pérdidas de validación es mucho más irregular que con el primer dataset.

Para intentar subsanar este error, se ha ampliado el dataset a 16819 imágenes con la esperanza de obtener mejores resultados. Sin embargo, no se observa una gran mejoría con unas pérdidas de validación de 0.043354, una mejoría de 0.002 que en principio, no hará que el desempeño del vehículo varíe mucho.

Para la misma imagen usada anteriormente, se calcula esta salida de la red:

Eje izquierdo real: -1.0, Eje derecho real: -1.0
Eje izquierdo predicho: -0.6811461, Eje derecho predicho: -0.7603459

Lo que a priori podría parecer un resultado peor, en este caso se trata de un ajuste mayor. Al no existir una diferencia entre los valores de los ejes tan ancha como en el caso anterior, el vehículo avanzará más fielmente por la trazada, pues la potencia en ambos ejes es cercana. En el caso anterior, el vehículo se vería desplazado a la izquierda por una potencia notablemente mayor en el eje derecho.

También se puede comprobar la evolución de las pérdidas de entrenamiento y validación que para la segunda, es mucho más abrupta:

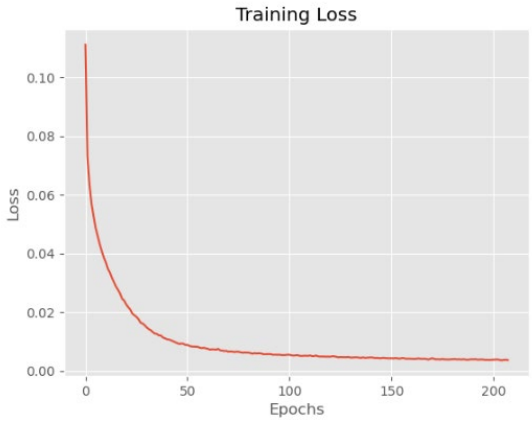


Figura 33. Evolución de pérdidas de entrenamiento

Figura 34. Evolución de pérdidas de validación

Se va a intentar solucionar este error una vez más, en este caso aumentando aún más el número de muestras, variando la conducción a la hora de formar el dataset y añadiéndole un filtro final que suavice las decisiones tomadas.

El nuevo dataset consta de 20056 imágenes, donde las curvas han sido trazadas en su mayoría por la zona interior, consiguiendo así que la cámara capte siempre el circuito.

Con esta nueva configuración, se ha conseguido rebajar el error notablemente, de 0.043354 a 0.02947, una mejora de 0,01388. Si hacemos la misma prueba que en los casos anteriores obtenemos lo siguiente:

Eje izquierdo real: -1.0, Eje derecho real: -1.0
Eje izquierdo predicho: -0.9012349, Eje derecho predicho: -0.7857656

En la predicción, la diferencia ha aumentado 0.035, sin embargo los valores son más fieles a los reales, aunque la diferencia sea un poco mayor. Este resultado nos indica un mejor rendimiento del vehículo en el circuito. Como comentario final, se puede ver como la curva que representa las pérdidas de validación es mucho más suave que en el caso anterior:

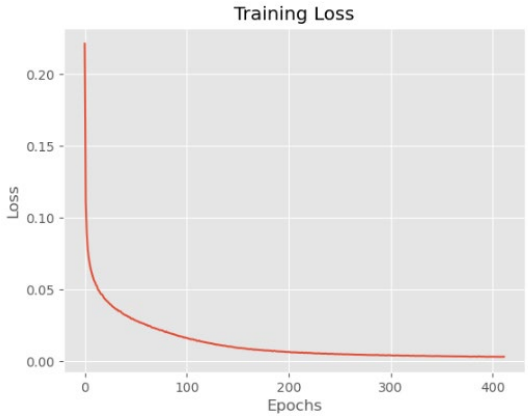


Figura 36. Evolución de pérdidas de entrenamiento

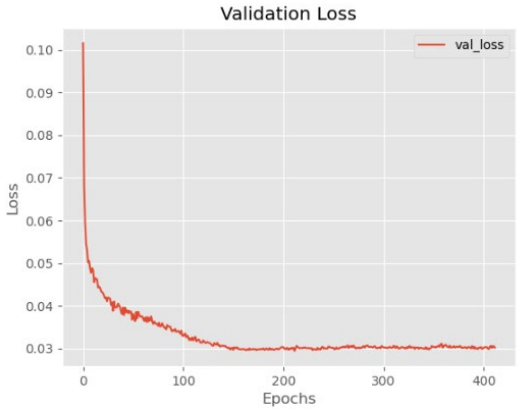


Figura 35. Evolución de pérdidas de validación

4.2 COMPORTAMIENTO DE LA RED PRÁCTICO

4.2.1 Dataset A

En la práctica, el comportamiento de este dataset no era en absoluto malo. Funciona mejor en circuitos abiertos sin curvas cerradas como el visto en la Figura 17, pues la capacidad de giro es limitada utilizando únicamente un eje para girar, aunque es un resultado satisfactorio.

4.2.2 Dataset B

Lo que a priori parece que no funcionará correctamente, sorprende en la práctica, siendo capaz de realizar giros cerrados gracias a la incorporación de su otro eje para rotar sobre sí mismo. La primera versión era capaz de realizar alguna curva, aunque se perdía rápidamente. Con el dataset ampliado, es capaz de recorrer y encadenar algunas curvas con mayor o menor ángulo, aunque no es capaz de terminar el circuito completo. Aun así, consideramos éste como un buen resultado.

Con el último intento se ha mejorado por completo el vehículo, tanto la velocidad del vehículo como su funcionamiento en curvas, siendo este capaz de recorrer el circuito de la Figura 17. Este último modelo es el más realista y el que consideraremos como resultado final de este trabajo.

5 CONCLUSIONES

Durante el desarrollo de este Proyecto Fin de Grado, se han adquirido una serie de conocimientos y capacidades, pudiendo concluir que:

- A la hora de abordar un proyecto técnico, éste se ha de estudiar detenidamente, buscando qué solución es la óptima para ese problema y abordarla siguiendo una serie de pautas ordenadas y estructuradas.
- El uso de Redes Neuronales Artificiales está en auge por su sencillez a la hora de programarlas y por los buenos resultados que ofrecen, con una tasa de error que, con los datos correctos, puede rondar en el 5%.
- El Deep Learning resulta ser una poderosa herramienta cuando se busca abordar un problema complejo. A pesar de su tiempo de cómputo, ofrece resultados muy satisfactorios.
- Es necesario tener una base de datos de calidad y variada para entrenar una red neuronal correctamente. Un mal conjunto de datos puede provocar underfitting, lo que conlleva el comportamiento nefasto de la red.

5.1 CONOCIMIENTOS ADQUIRIDOS

En el proceso de diseño de la red y del proyecto en general, el alumno ha adquirido los siguientes conocimientos:

- Cómo abordar un proyecto técnico, la forma de estudiarlo y de encontrar soluciones.
- Familiarización con el lenguaje de programación Python, entendiendo por que es uno de los más usados para el Machine Learning.
- Familiarización con las librerías de alto nivel dedicadas al Machine Learning, como es el caso de *Tensorflow* y *Scikit-Learn*.
- La importancia de la paralelización del proceso de entrenamiento de la red usando GPUs.
- La importancia de un correcto procesamiento de los datos antes de que sean usados.

6 BIBLIOGRAFÍA

[1] C. M. University, «Homepage - CMU - Carnegie Mellon University». http://www.cmu.edu/index.html (accedido abr. 28, 2021).

[2] «ADA218975.pdf». Accedido: abr. 28, 2021. [En línea]. Disponible en: https://apps.dtic.mil/sti/pdfs/ADA218975.pdf.

[3] «Los accidentes de tráfico se cobran la vida de 870 personas durante el año pasado». https://www.dgt.es/es/prensa/notas-de-prensa/2021/Los_accidentes_de_trafico_se_cobran_la_vida_de_870_personas_durante_el_ano_pasado.shtml (accedido abr. 21, 2021).

[4] «Las muertes en carretera en 2020 y la trampa de una hoja de Excel | Actualidad | Motor EL PAÍS». https://motor.elpais.com/actualidad/pandemia-accidentes-carretera-fallecidos/ (accedido abr. 21, 2021).

[5] C. G. Oliva, «Esta es la historia del coche autónomo, y ojo porque no es tan nuevo como lo pintan…». https://www.autonocion.com/historia-coche-autonomo/ (accedido abr. 21, 2021).

[6] «Essay». http://selfdrivingcars.weebly.com/essay.html (accedido abr. 21, 2021).

[7] «Alonso - EL COCHE FANTÁSTICO, ¿UN SUEÑO.pdf». Accedido: abr. 21, 2021. [En línea]. Disponible en: https://riuma.uma.es/xmlui/bitstream/handle/10630/13659/Conferencia.pdf?sequence=1.

[8] «Echtzeitanalyse_von_Landstrassen.pdf». Accedido: abr. 21, 2021. [En línea]. Disponible en: http://www.inf.fu-berlin.de/lehre/SS05/Autonome_Fahrzeuge/Echtzeitanalyse_von_Landstrassen.pdf.

[9] «Fig.5 “VaMoRs” and “VaMP” vehicles by Ernst Dickmann.», *ResearchGate*. https://www.researchgate.net/figure/VaMoRs-and-VaMP-vehicles-by-Ernst-Dickmann_fig3_338412717 (accedido abr. 21, 2021).

[10] «Figure 4. Demonstration vehicle VaMP of UniBwM (top right, a twin to...», *ResearchGate*. https://www.researchgate.net/figure/Demonstration-vehicle-VaMP-of-UniBwM-top-right-a-twin-to-the-Daimler-VITA-2-top-left_fig1_341211118 (accedido abr. 21, 2021).

[11] M. Williams, «PROMETHEUS-The European research programme for optimising the road transport system in Europe», en *IEE Colloquium on Driver Information*, dic. 1988, p. 1/1-1/9.

[12] P. Russell, *How Autonomous Vehicles Will Profoundly Change The World*. 2015.

[13] C. Ziegler, «Tesla’s autopilot isn’t special (but it’s still cool)», *The Verge*, oct. 17, 2014. https://www.theverge.com/2014/10/17/6982289/tesla-autopilot-is-a-non-revolution-for-self-driving-cars (accedido abr. 21, 2021).

[14] «Vídeo: 4 funciones increíbles de Tesla Autopilot que no hacen que tu coche sea autónomo (pero casi)», *article*. https://www.diariomotor.com/2016/01/26/tesla-model-s-4-funciones-autopilot/ (accedido abr. 21, 2021).

[15] «SAE J3016 automated-driving graphic». https://www.sae.org/site/news/2019/01/sae-updates-j3016-automated-driving-graphic (accedido abr. 21, 2021).

[16] «Automated Vehicles for Safety», *NHTSA*, sep. 07, 2017. https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety (accedido abr. 21, 2021).

[17] M. J. M. Ayora, «Diseño de un sistema de conducción autónoma para vehículo Renault Twizy», p. 64.

[18] «Conducción autónoma | Los cinco niveles». https://www.km77.com/reportajes/varios/conduccion-autonoma-niveles (accedido abr. 21, 2021).
[19] J. A. Valero-Matas y A. D. la Barrera, «El Coche autónomo: ¿Un futuro mejor?», *Sociol. Technoscience*, vol. 10, n.º 1, Art. n.º 1, ene. 2020.
[20] T. R. P. Foundation, «Buy a Raspberry Pi 3 Model B+», *Raspberry Pi*. https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/ (accedido abr. 21, 2021).
[21] «Cámara web HD PRO Logitech C920, video 1080p con audio estéreo». https://www.logitech.com/es-mx/products/webcams/c920-pro-hd-webcam.html (accedido abr. 21, 2021).
[22] «Cherokey_4WD_Mobile_Platform__SKU_ROB0102_-DFRobot». https://wiki.dfrobot.com/Cherokey_4WD_Mobile_Platform__SKU_ROB0102_ (accedido abr. 21, 2021).
[23] «Welcome to Python.org», *Python.org*. https://www.python.org/ (accedido abr. 21, 2021).
[24] «Arduino Uno Rev3 | Arduino Official Store». https://store.arduino.cc/arduino-uno-rev3 (accedido abr. 21, 2021).
[25] «Arduino Mega 2560 Rev3 | Arduino Official Store». https://store.arduino.cc/arduino-mega-2560-rev3 (accedido abr. 21, 2021).
[26] «Plataforma Móvil Compatible con Arduino iOS Cherokey 4WD». https://www.robotshop.com/us/es/plataforma-movil-compatible-con-arduino-ios-cherokey-4wd.html (accedido abr. 21, 2021).
[27] «Sony Dual Shock 3 Negro PS3 - DiscoAzul.com». https://www.discoazul.com/dual-shock-3-negro-ps3.html (accedido abr. 21, 2021).
[28] «TensorFlow», *TensorFlow*. https://www.tensorflow.org/?hl=es-419 (accedido abr. 22, 2021).
[29] «Inicio», *Real Academia Española*. http://www.rae.es/inicio (accedido abr. 22, 2021).
[30] J. Sampedro, «Análisis | Marvin Minsky, cerebro de la inteligencia artificial», *El País*, Madrid, ene. 26, 2016.
[31] «TFG_JAVIER_MARTINEZ_LLAMAS.pdf». Accedido: abr. 22, 2021. [En línea]. Disponible en: http://oa.upm.es/53050/1/TFG_JAVIER_MARTINEZ_LLAMAS.pdf.
[32] «Redes Neuronales». https://ml4a.github.io/ml4a/es/neural_networks/ (accedido abr. 22, 2021).
[33] «CONCEPTOS BÁSICOS». http://grupo.us.es/gtocoma/pid/pid10/RedesNeuronales.htm (accedido abr. 22, 2021).
[34] matallanas, «IRIN clase 140509», 09:09:02 UTC, Accedido: abr. 22, 2021. [En línea]. Disponible en: https://es.slideshare.net/matallanas/irin-clase140509.
[35] «Rouhiainen - 2018 - Inteligencia artificial 101 cosas que debes saber.pdf». Accedido: abr. 22, 2021. [En línea]. Disponible en: https://static0planetadelibroscom.cdnstatics.com/libros_contenido_extra/40/39308_Inteligencia_artificial.pdf.
[36] «What is the difference between Deep Learning and Machine Learning? ⋆ Quantdare», *Quantdare*, ago. 01, 2019. https://quantdare.com/what-is-the-difference-between-deep-learning-and-machine-learning/ (accedido abr. 22, 2021).
[37] L. Velasco, «Optimizadores en redes neuronales profundas: un enfoque práctico», *Medium*, abr. 26, 2020. https://velascoluis.medium.com/optimizadores-en-redes-neuronales-profundas-un-enfoque-pr%C3%A1ctico-819b39a3eb5 (accedido abr. 22, 2021).
[38] B. Chen, «A Practical Introduction to Early Stopping in Machine Learning», *Medium*, ago. 03, 2020. https://towardsdatascience.com/a-practical-introduction-to-early-stopping-in-machine-learning-550ac88bc8fd (accedido abr. 22, 2021).

7 ANEXO

Introducción

En este notebook voy a documentar el proceso de desarrollo del proyecto en cuanto a código se refiere. Nuestra intención es lograr que un vehiculo equipado con una Raspberry Pi 3 sea capaz de seguir una trazada autónomamente. Para lograr esto, primeramente se han recogido muestras de la trazada con la webcam que incluye este vehiculo. Estas muestras van a pasar por un preprocesamiento previo para reducir las componentes, para más tarde usar estas componentes en una red neuronal multicapa o MLP.

Preprocesado

En esta sección nuestra intención es la de preparar los datos de entrada a la red con el fin de hacerlos lo más sencillos posible de procesar por nuestra red. Esto se traducirá en una binarización de la imagen, aislando de esta manera la información útil, es decir, la trazada a seguir. Para este caso, se han tomado imágenes con una webcam conectada al vehiculo, que nos proporciona capturas de 640 de ancho y 480 de alto. Esto nos daría un total de 921600 entradas a la red, excesivo en cualquier caso. Dado que el número de datos de entrenamiento aumenta con el número de entradas, necesitaríamos una cantidad inmensa de muestras, con todo lo que eso conlleva: elevado tiempo de cómputo, red compleja, etc. Por lo mencionado anteriormente, el preprocesado de los datos de entrada es de carácter obligatorio. Las imágenes usadas se componen de un fondo gris con la línea azul a seguir, como se muestra en este ejemplo:

In [1]:
```
921600 import cv2
import numpy as np

img = cv2.imread('imagen_2021-02-16 14_47_49.661111_EjeIzda1_-0.762908935546875_EjeDch
cv2.imshow('Imagen Original', img)
cv2.waitKey(0)
```

Out[1]: -1

Como se puede apreciar, hay una gran cantidad de píxeles que no aportan apenas información pero sí que obstaculizan el entrenamiento. Estas imágenes pasaran por una serie de procesos hasta obtener una imagen binarizada, redimensionada y normalizada. Se ha usado un dataset de 6312 imágenes, suficientes para producir un bajo error.

Desarrollo del código

En un fichero llamado preprocesar.py se ha diseñado un código simple pero que ejecuta a la perfección esta necesidad. En primer lugar, importamos los paquetes necesarios, estos son: *os*, *cv2*, *numpy* y *re*.

In [2]:
```
import os
import cv2
import numpy as np
import re
```

La idea es la de "muestrear" la imagen y convertirla en un vector fila con todos los valores de los pixeles finales. Para ello se crean las variables usadas como índices del array que contendrá dichos valores. A continuación, se recupera el número de imágenes disponibles en el directorio.

In [ ]:
```
indice_x = 0
indice_y = 0
```

```
dir = 'D:/imagenes'

path, dirs, files = next(os.walk(dir))
file_count = len(files)
```

"Escaneamos" el directorio en busca de las imágenes, que renombramos como *imagenes*. Creamos una matriz vacía en la que guardaremos los valores de los pixeles seleccionados. Como se puede observar, tiene tantas filas como elementos en el directorio y tantas columnas como número total de puntos que queramos o dicho de otra manera, el número final de píxeles a los que reduciremos la imagen añadiendole dos columnas extra, dedicadas a los valores de salida para ese conjunto.

In [ ]:

```
with os.scandir(dir) as imagenes:
    valores = np.empty((file_count, 24 * 18 + 2))
```

Prosigue el bucle principal, donde se van a realizar las operaciones. Para poner en contexto, los archivos estan nombrados de manera que aparezca la fecha y hora de la toma, así como los valores que tenían los joysticks del mando. Será necesario extraer estos últimos dos valores, pues se tratan de nuestras salidas. Sabiendo esto, se ha usado una *expresion regular*. Para este caso, usando el paquete *re* y su función *findall* se encuentran todos los números existentes en el nombre. Continúa cargando la imagen y comenzando el procesado:

- Aumento del contraste
- Conversión a escala de grises
- Binarización
- Redimensionado
- Normalización entre 0 y 1

Se extraen los valores del eje izquierdo y derecho y comienza el "muestreo" de la imagen.

Esta operación puede llevar unos minutos de cómputo, dependiendo del número de imágenes y el tamaño final deseado. En un equipo de 8 núcleos y 16 hilos y 16GB de memoria RAM tarda aproximadamente menos de un minuto.

In [5]:

```
    for imagen in imagenes:
        valoresSalida = [float(s) for s in re.findall(r'-?\d+\.?\d*', imagen.name)]
        img = cv2.imread(dir + '/' + imagen.name)
        contrast_img = cv2.addWeighted(img, 2.3, np.zeros(img.shape, img.dtype), 0, 0)
        gray_img = cv2.cvtColor(contrast_img, cv2.COLOR_BGR2GRAY)
        ret, thresh1 = cv2.threshold(gray_img, 200, 255, cv2.THRESH_BINARY_INV)
        imagen_re = cv2.resize(thresh1, (24, 18))
        final_img = imagen_re / 255
        valores[indice_x, indice_y] = valoresSalida[7]
        valores[indice_x, indice_y + 1] = valoresSalida[9]
        for i in range(final_img.shape[0]):
            for j in range(final_img.shape[1]):
                valores[indice_x, indice_y + 2] = final_img[i, j]
                indice_y = indice_y + 1
        indice_x = indice_x + 1
        indice_y = 0

    valores = np.round(valores, 3)
```

El resultado de aplicar los dintintos procesamientos se muestran a continuación:

Cargar imagen

In [13]:
```
img = cv2.imread('imagen_2021-02-16 14_47_49.661111_EjeIzda1_-0.762908935546875_EjeDch
cv2.imshow('Imagen Original', img)
cv2.waitKey(0)
```

Out[13]: -1

Aumento del contraste

In [3]:
```
contrast_img = cv2.addWeighted(img, 2.3, np.zeros(img.shape, img.dtype), 0, 0)
cv2.imshow('Imagen Contrastada', contrast_img)
cv2.waitKey(0)
```

Out[3]: -1

Conversión a escala de grises

In [4]:
```
gray_img = cv2.cvtColor(contrast_img, cv2.COLOR_BGR2GRAY)
cv2.imshow('Imagen en Gris', gray_img)
cv2.waitKey(0)
```

Out[4]: -1

Binarización

In [5]:
```
ret, thresh1 = cv2.threshold(gray_img, 200, 255, cv2.THRESH_BINARY_INV)
cv2.imshow('Imagen Binarizada', thresh1)
cv2.waitKey(0)
```

Out[5]: -1

Redimensionado

In [12]:
```
resize_img = cv2.resize(thresh1, (24, 18))
cv2.imshow('Imagen Redimensionada', resize_img)
cv2.waitKey(0)
```

Out[12]: -1

Para finalizar, guardamos el array con los valores capturados en formato *csv* para poder usarlo posteriormente.

In [ ]:
```
np.savetxt('data.csv', valores, delimiter=',')
```

Entrenamiento

En la sección anterior se ha creado un dataset listo para ser usado por una red neuronal. En esta, se verá la creacion de la misma, los resultados obtenidos y el por qué de las decisiones tomadas. Se ha decidido usar una Red Neuronal Artificial (ANN), más concretamente una Multilayer Perceptron (MLP) por su gran desempeño y facilidad de creación. Dado que no son valores discretos, como podría ser la pertenencia a una clase o etiqueta, sino que son valores continuos, estamos ante un problema de regresión. El fin de la red será dado un vector, ser capaz de entregar de la manera más fiel posible los valores que un piloto usaría con el mando ante esa misma situación. Se obtendrán valores entre 0 y -1, aunque no se tratan de límites estrictos. Para ello, se ha usado la librería *tensorflow* que incluye *keras*. Más adelante se comentará con mayor detalle.

Desarrollo del código

En un fichero llamado "entrenar.py" se ha diseñado la red. Comenzamos importando todas las funciones necesarias, estas son:

- Sequential: Agrupa una pila de capas a un modelo
- Dense: Capa de neuronas
- Dropout: Desactiva un tanto por ciento de neuronas
- Adam: Optimizador del modelo
- train_test_split: Divide un dataset aleatoriamente en conjunto de entrenamiento y conjunto de test
- EarlyStopping: Para automáticamente cuando no se mejora el error de validación
- mean_squared_error: Usado para calcular el error entre el conjunto de test y las predicciones del modelo

Usando el dataset generado en la sección anterior, cargamos los valores que se van a trabajar.

In [ ]:

```
from pandas import read_csv
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.metrics import mean_squared_error
import os
import matplotlib.pyplot as plt


dataframe = read_csv("data.csv", header=None, sep=',')
dataset = dataframe.values
```

El dataset cargado se divide en en valores de entrada "X" y valores de salida "Y". A su vez, estos dataset se dividen usando la función anteriormente mencionada, creando un conjunto de entrenamiento del 80% de las muestras, dejando un 10% para el conjunto de validación y otro 10% para el conjunto de test.

In [ ]:

```
X = dataset[:, 2:]
Y = dataset[:,0:2]

lr = 0.001
filas_entrada = dataset.shape[0]
columnas_entrada = dataset.shape[1]-2
batch_size = 32

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)
X_test, X_val, Y_test, Y_val = train_test_split(X_test, Y_test, test_size=0.5)
```

Siguiendo con lo anterior, se puede dar comienzo al diseño de la red neuronal. Para ello, creamos un objeto tipo *Sequential*, que será el plano donde se añadirán las capas de neuronas. Será una red de 3 capas, una capa de entrada y 2 capas ocultas, por lo que se puede considerar una red profunda. Este es el diseño que mejor resultado ha obtenido para este trabajo. Continuamos añadiendo la primer capa densa, con tantas neuronas como características de entrada (de aquí la importancia de contar con un buen procesado de datos), con una función de activación tipo "relu". A esta capa le añadiremos un *Dropout* del 40%. Agregamos la primera capa oculta, con un tercio de neuronas que la capa de entrada, cuya función de activacion es la misma, al igual que el *Dropout*. La segunda capa oculta cuenta con un quinto de neuronas que la capa de entrada, manteniendo la función de activacion y bajando la probabilidad del *Dropout* al 30%. Terminamos el diseño con la capa de salida, que consta de 2 neuronas ya que son necesarios dos valores para el funcionamiento del vehiculo. Dado que se trata de un problema de regresión, la función de activación de la capa de salida debe ser lineal. Nótese que lo que se busca con este diseño es un embotellamiento, comenzando por una mayor cantidad de neuronas para ir reduciendo el número capa a capa. Dado que son capas densas, están interconectadas entre ellas, o dicho de otra manera, es una red full conected.

In [ ]:

```
red = Sequential()

red.add(Dense(units=columnas_entrada, activation='relu'))
red.add(Dropout(0.4))

red.add(Dense(units=int(columnas_entrada/3), activation='relu'))
red.add(Dropout(0.4))

red.add(Dense(units=int(columnas_entrada/5), activation='relu'))
red.add(Dropout(0.3))

red.add(Dense(units=2, activation='linear'))
```

Una vez diseñada la red, debemos definir qué se quiere medir, cómo se va a mejorar y qué medida juzgará el desempeño. Dado que es un problema de regresión, las pérdidas se mediran con el error cuadrático medio o mse, se optimizará o actualizará el valor de los pesos con *Adam* y observaremos la evolución del error cuadrático medio. También iniciamos el *EarlyStopping*, simplemente indicando cuantas épocas deben pasar sin mejoría para detener el entrenamiento. Además, le indicamos que una vez finalizado el entrenamiento, recupere la mejor configuración que haya sido capaz de encontrar. Puede parecer que el número de épocas sin mejoría es alto, pero se ha indicado un learning rate bajo, por lo que los pesos variarán lentamente. De esta manera se asegura un resultado competitivo. Por último, comienza el entrenamiento. Será necesario indicar el conjunto de entrenamiento de entrada con sus salidas correspondientes, un batch_size de 32 muestras, los datos para la validación, creados anteriormente y un callback, donde añadiremos el *EarlyStopping*.

Esta operación puede llevar tiempo de procesamiento, dependiendo de la capacidad de la máquina usada, el tamaño de los datos, el batch_size, etc.

In [ ]:

```
red.compile(loss='mean_squared_error', optimizer=Adam(lr=lr),
            metrics=['MeanSquaredError'])

early_stopping = EarlyStopping(patience=200, restore_best_weights=True)

resultado = red.fit(x=X_train, y=Y_train, batch_size=batch_size,
            epochs=10000, validation_data=(X_val, Y_val),
            verbose=2, callbacks=[early_stopping])
```

A modo de comprobación, se puede mostrar el mejor valor del error de validación, el que podemos comparar con el error que se genera entre la predicción y el conjunto de test. En la gran mayoria de casos, este segundo error debe ser mínimamente mayor que el error de validación, lo que nos indica que es capaz de predecir correctamente. Se generan dos gráficas pertenecientes a la evolución de las pérdidas de entrenamiento y de validación.

In [ ]:
```
print(min(resultado.history['val_loss']))

Y_pred = red.predict(X_test)

mse = mean_squared_error(Y_test, Y_pred)

print(mse)

plt.style.use("ggplot")
plt.figure()
plt.plot(resultado.history['loss'], label="train_loss")
plt.title("Training Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.figure()
plt.plot(resultado.history['val_loss'], label="val_loss")
plt.title("Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

Para finalizar esta sección, creamos un directorio donde guardar la configuración calculada por la red y guardamos tanto el modelo como los pesos, que usaremos más adelante.

In [ ]:
```
dir = 'modelo'

if not os.path.exists(dir):
    os.mkdir(dir)
red.save('modelo/modelo.h5')
red.save_weights('modelo/pesos.h5')
```

Test

Antes de probar estos resultados en el vehiculo, se va a simular el comportamiento del modelo entrenado. Para ello, se creará una funcion que realice el preprocesado de la imagen, y con una imagen que no se haya usado para el entrenamiento, se probará. Se carga el modelo y sus pesos correspondientes.

In [5]:
```
import numpy as np
import cv2
from tensorflow.keras.models import load_model
import re

modelo = 'modelo.h5'
pesos = 'pesos.h5'
red = load_model(modelo)
red.load_weights(pesos)
```

```
WARNING:tensorflow:Sequential models without an `input_shape` passed to the first laye
r cannot reload their optimizer state. As a result, your model isstarting with a fresh
ly initialized optimizer.
```

La función *predict* será la encargada de dicha tarea. Es el mismo procesado que sufren las imágenes para generar el dataset.

In [6]:
```
def predict(file):
```

```
    indice = 0
    valoresSalida = [float(s) for s in re.findall(r'-?\d+\.?\d*', file)]
    img = cv2.imread(file)
    valores = np.empty((1, 24*18))
    contrast_img = cv2.addWeighted(img, 2.3, np.zeros(img.shape, img.dtype), 0, 0)
    gray_img = cv2.cvtColor(contrast_img, cv2.COLOR_BGR2GRAY)
    ret, thresh1 = cv2.threshold(gray_img, 200, 255, cv2.THRESH_BINARY_INV)
    resize_img = cv2.resize(thresh1, (24, 18))
    final_img = resize_img / 255
    for i in range(final_img.shape[0]):
        for j in range(final_img.shape[1]):
            valores[0, indice] = final_img[int(i), int(j)]
            indice = indice + 1


    respuesta = red.predict(valores)
    print('Eje izquierdo real: ',valoresSalida[7],
          ', Eje derecho real: ', valoresSalida[9])
    print('Eje izquierdo predicho: ',respuesta[0,0],
          ', Eje derecho predicho: ', respuesta[0,1])
```

Hacemos uso de la función, y como se puede ver, el error entre ambos valores es mínimo, por lo que se puede esperar que el funcionamiento del vehiculo en las pruebas sea satisfactoria.

In [7]:

```
predict('imagen_2021-02-16 14_47_49.661111_EjeIzda1_-0.762908935546875_EjeDcha4_-0.979
```

```
Eje izquierdo real:  -0.762908935546875 , Eje derecho real:  -0.979400634765625
Eje izquierdo predicho:  -0.7943145 , Eje derecho predicho:  -1.0023944
```

Out[7]:

```
array([[-0.7943145, -1.0023944]], dtype=float32)
```

Prueba en el vehiculo

Ahora que hemos comprobado que el regresor funciona correctamente, podemos cargarlo en el vehiculo y comenzar las pruebas reales. En primer lugar cargamos los paquetes necesarios, el modelo y los pesos calculados en el entrenamiento.

In [ ]:

```
import RPi.GPIO as GPIO
import numpy as np
from tensorflow.keras.models import load_model
import cv2


# Cargamos el modelo y los pesos de la red
modelo = 'modelo.h5'
pesos = 'pesos.h5'

red = load_model(modelo)
red.load_weights(pesos)
```

Configuramos los pines del puerto GPIO de la Raspberry Pi, describiendolos como salidas que proporcionarán un pulso PWM a los motores para así regular la velocidad. Hacemos esto para el eje derecho e izquierdo. Finalmente, comenzamos a capturar imagenes con la cámara.

In [6]:

```
directionRight = 31 #Conectar a D4
speedRight = 33 # PWM pin connected Right motor #Conectar a M1 -> D5
speedLeft = 35 # PWM pin connected Left motor #Conectar a M2 -> D6
directionLeft = 37 #Conectar a D7

GPIO.setwarnings(False) #disable warnings
GPIO.setmode(GPIO.BOARD) #set pin numbering system # Referencia a la posición física.
```

```
GPIO.setup(directionLeft,GPIO.OUT)
GPIO.setup(directionRight,GPIO.OUT)

GPIO.setup(speedLeft,GPIO.OUT)
pwmLeft = GPIO.PWM(speedLeft,100) #create PWM instance with frequency #En el código de
pwmLeft.start(0) #start PWM of required Duty Cycle

GPIO.setup(speedRight,GPIO.OUT)
pwmRight = GPIO.PWM(speedRight,100) #create PWM instance with frequency
pwmRight.start(0) #start PWM of required Duty Cycle

cam = cv2.VideoCapture(0)
```

Entramos en un bucle infinito, pues el coche no para de moverse hasta que sea interrumpido. Leemos la imagen capturada y se le aplica el procesado explicado anteriormente. Extraemos los valores de la predicción y comprobamos su polaridad. Si el resultado es mayor que cero significa que las ruedas deberán girar en sentido contrario, es decir, marcha atras. Por otro lado, si es menos que cero, girarán en el sentido de la circulación. Para regular la potencia, definimos la variable *pot* que ocupará un valor entre 0 y 100. Este valor, multiplicado con la salida de la red indicará el tamaño del pulso PWM, en otras palabras, el tiempo que estará encendido el motor en un ciclo del pulso. Para terminar, liberamos la cámara cuando se detenga el código.

In [ ]:

```
while (True):
    ret, img = cam.read()
    indice = 0
    valores = np.empty((1, 24*18))
    contrast_img = cv2.addWeighted(img, 2.3, np.zeros(img.shape, img.dtype), 0, 0)
    gray_img = cv2.cvtColor(contrast_img, cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(gray_img, 200, 255, cv2.THRESH_BINARY_INV)
    resize_img = cv2.resize(thresh, (24,18))
    final_img = resize_img / 255

    for i in range(final_img.shape[0]):
        for j in range(final_img.shape[1]):
            valores[0, indice] = final_img[i, j]
            indice = indice + 1

    output = red.predict(valores)
    Eje_izq = output[0,0]
    Eje_dch = output[0,1]

    if Eje_izq < 0:
        GPIO.output(directionLeft, True)
    else:
        GPIO.output(directionLeft, False)

    if Eje_dch < 0:
        GPIO.output(directionRight, True)
    else:
        GPIO.output(directionRight, False)

    pot = 50 # JK pot = 100 va a toda velocidad.
    pwmRight.ChangeDutyCycle(int (abs(Eje_dch)*pot))
    pwmLeft.ChangeDutyCycle(int (abs(Eje_izq)*pot))

cam.release()
```