Universidad Complutense de Madrid
Facultad de Informática

Técnicas inteligentes para su integración en un vehículo autómata

Intelligent techniques for integration into an automaton vehicle

Guillermo Cortina Fernández

Trabajo de Fin de Grado en Ingeniería del Software

Curso: 2019 - 2020

Director: Gonzalo Pajares Martinsanz

Resumen

Es evidente que los vehículos autónomos de futuro constituyen un nicho de interés tecnológico elevado. La dotación de inteligencia a los mismos es, a su vez, un reto. Es bajo esta perspectiva sobre la que se plantea el presente trabajo, si bien a muy pequeña escala.

En efecto se propone el desarrollo de un prototipo de vehículo autónomo dotado de inteligencia, esto es, un autómata inteligente. El modelo de prototipo, en línea con desarrollos de este tipo, se plantea considerando una plataforma dotada con los elementos necesarios para navegación autónoma, incluyendo ruedas, motores, alimentación, por supuesto sensores de visión (cámara) y de proximidad (ultrasonidos) y naturalmente un computador local basado en una *Raspberry Pi* que gestiona y controla las acciones del vehículo. El sistema también consta de un computador remoto, en perfecta comunicación y sincronía con el local donde se realizan los procesos con alta carga computacional y donde está instalada la parte inteligente del sistema.

El computador local envía datos (imágenes y distancias) al remoto, iniciándose en éste su procesamiento. En el caso de las imágenes mediante la aplicación de técnicas de segmentación para localizar una plataforma con una base rectangular negra conteniendo figuras geométricas (cuadrado, círculo, rectángulo, triángulo) de color en su interior, y que constituye el objetivo del vehículo hacia el que se tiene que dirigir. Con tal finalidad, se extraen regiones candidatas, que pueden contener la plataforma, aplicando técnicas de umbralización y etiquetado de componentes conexas, para proceder a recortar dichas regiones sobre la imagen original. Estos recortes se pasan a una Red Neuronal Convolucional, previamente re-entrenada a partir del modelo AlexNet, que proporciona un valor de probabilidad para determinar si el recorte puede o no considerarse como la plataforma. Un proceso posterior, basado en la identificación de las figuras geométricas de la plataforma refuerza o penaliza la probabilidad dada por la red, procediendo a decidir si la región analizada es o no finalmente la plataforma. En el caso de las distancias, proporcionadas por el sensor de ultrasonidos, simplemente se trata de verificar la proximidad del vehículo a un objeto o a la plataforma, para confirmar la llegada al objetivo. Esta información se transmite a la *Raspberry Pi*, que se convierte en acciones sobre el vehículo para girar, avanzar y seguir o no enviando más datos desde el vehículo.

Los experimentos realizados han verificado la validez de la propuesta, destacando los aspectos relacionados con el sistema inteligente y la navegación autónoma del vehículo autómata.

Palabras clave: Vehículo autómata, Visión Artificial, Aprendizaje Profundo, Red Neuronal Convolucional, Entrenamiento, Clasificación, Raspberry Pi.

Abstract

It is obvious the autonomous vehicles of the future constitute a niche of high technological interest. The endowment of intelligence to them is, in turn, a challenge. It is from this perspective that the present work is proposed, albeit on a very small scale.

Indeed, the development of a prototype of an autonomous vehicle endowed with intelligence is proposed, i.e. an intelligent automaton. The prototype model, in line with developments of this type, is proposed considering a platform equipped with the necessary elements for autonomous navigation, including wheels, motors, power supply, of course vision-based sensors (camera) and proximity (ultrasound) and obviously a local computer based on a *Raspberry Pi*, which handles and controls the actions of the vehicle. The system also consists of a remote computer, in perfect communication and synchrony with the local computer, where processes with high computational load are carried out and where the intelligent part of the system is installed.

The local computer sends data (images and distances) to the remote, starting its processing there. In the case of images by applying segmentation techniques to locate a platform with a black rectangular base containing geometric colored figures (square, circle, rectangle and triangle) inside, and which constitutes the objective of the vehicle towards it has to be guided. For this purpose, candidate regions, which may contain the platform, are extracted, by applying thresholding techniques and labeling of connected components, in order to clip these regions on the original image. These clipping are passed to a Convolutional Neural Network, previously re-trained from the AlexNet model, which provides a probability value to determine whether or not the clipping can be considered as the platform. A subsequent process, based on the identification of the geometric figures of the platform, reinforces or penalizes the probability given by the network, proceeding to decide whether or not the analyzed region is finally the platform. In the case of distances, provided by the ultrasound sensor, it is simply a matter of verifying the proximity of the vehicle to an object or to the platform, to confirm arrival at the target. This information is transmitted to the *Raspberry Pi*, which is converted into actions on the vehicle to turn, advance and continue or not sending more data.

The experiments carried out have verified the validity of the proposal, highlighting aspects related to the intelligent system and autonomous navigation of the automated vehicle.

Keywords: Automata Vehicle, Artificial Vision, Deep Learning, Convolutional Neural Network, Training, Classification, Raspberry Pi.

Tabla de contenidos

1.- Introducción

1.1 Antecedentes

Es bien conocido que las grandes compañías del sector del automóvil se encuentran inmersas en múltiples procesos competitivos en una carrera imparable hacia el desarrollo de vehículos autónomos. Son conocidos por el despliegue publicitario algunas como Tesla, Uber, Google, sin descartar las grandes empresas del sector tradicional del Automóvil. Por otra parte, existen centros tecnológicos y de investigación desarrollando proyectos bajo la misma filosofía, y desde hace ya bastante tiempo, tal es el caso del prototipo de la Universidad Libre de Berlín liderado por el profesor Rojas (2020), el proyecto Autopía (2020) del Consejo Superior de Investigaciones Científicas (CSIC) español o el proyecto CAV (2020) del Instituto Universitario de Investigación del Automóvil (INSIA) de la Universidad Politécnica de Madrid para integración de sistemas cooperativos en tráfico compartido. Todos ellos comprometidos con el futuro para su integración en las ciudades inteligentes (Citilogs, 2020; Tecnalia, 2020) donde los vehículos autónomos y conectados formarán parte de los paisajes urbanos, sin lugar a dudas.

En estos desarrollos se combinan los sistemas físicos con modelos inteligentes, donde el aprendizaje profundo está llamado a jugar un papel relevante. Es bajo esta perspectiva sobre la que se plantea el presente trabajo, motivado por el interés en desarrollar un prototipo inteligente a pequeña escala, pero incorporando tecnologías de los prototipos a gran escala como los desarrollados por las mencionadas compañías automovilísticas. Así, el vehículo navega hacia un objetivo (plataforma de aproximación) guiado por un sistema de visión, que envía imágenes a un computador remoto, de forma que mediante una red neuronal convolucional se identifica el objetivo. La ubicación del objetivo permite realizar las maniobras de guiado, ayudándose de un sistema de ultrasonidos. Se establece la conveniente comunicación entre el procesador a bordo del vehículo, en este caso una *Raspberry Pi*, y el computador central (remoto) donde se realizan los procesos con alto coste computacional, como son el procesamiento de la imagen y la clasificación de los objetos mediante la red neuronal.

De esta forma se plantea un prototipo de coche inteligente (HackerCar, 2020) que va más allá de la mera simulación, que por otra parte es algo esencial en todo desarrollo inicial, y por tanto entra en problemas reales como puede ser la captura de imágenes en condiciones de iluminación adversas, el giro de las ruedas o el avance de los motores, por citar sólo algunas. La propuesta que se formula en este proyecto está en línea con el modelo de Tian (2019) basado en una *Raspberry Pi*, un SunFounder PiCar como sensor de

ultrasonidos para evitar obstáculos, y una unidad Google's Edge TPU, para procesamiento en las proximidades del sensor (*edge*) mediante una unidad TPU (Tensor Processing Unit), que utiliza como herramientas software el lenguaje Python, las librerías OpenCV para procesamiento de imágenes y Tensorflow como marco específico para procesamientos basados en aprendizaje profundo. A nivel conceptual, la diferencia que se propone en este trabajo con respecto a la anterior estriba en la comunicación entre sistemas, de forma que existe un trasvase de datos entre sistemas (procesadores), teniendo en cuenta que en los sistemas conectados debe imperar este tipo de planteamientos con intercomunicaciones entre los vehículos conectados y un necesario Centro de Control. Los procesos se llevan a cabo mediante los correspondientes *toolboxes* (Image Processing, Computer Vision y Deep Learning) de Matlab (2020) para el procesamiento de imágenes y aprendizaje profundo. Alguna otra aproximación en la misma línea es la propuesta en Self-Driving (2020) que utiliza Google Colab (Martín de la Fuente, 2019), como herramienta en la nube para ejecutar código Python, que en este caso utiliza un sensor tipo LIDAR (*Light Detection and Ranging* o *Laser Imaging Detection and Ranging*) para derivar la estructura tridimensional (3-D) de las escena.

Por otra parte, son diversos los grupos nacionales e internacionales, en ocasiones bajo el auspicio de determinadas organizaciones, que promueven el desarrollo de prototipos de esta índole, a veces con fines de competición (Make, 2020). En el desarrollo de prototipos de esta naturaleza es frecuente utilizar computadores simples del tipo *Raspberry Pi* (2020), según se indica en Upton (2020) o Horsey (2016).

1.2 Objetivos

El objetivo general es desarrollar un prototipo de vehículo inteligente a escala, con su correspondiente equipamiento hardware/software.

a) **Hardware**: *Raspberry Pi 3B*+, cámara, sensor de ultrasonidos, motores, controladores de motores, chasis, batería.
b) **Software**: procesamiento de imágenes, aprendizaje profundo (*deep learning*).

El conjunto constituye una plataforma integrada, para navegación autónoma, de forma que mediante la captura de imágenes por la cámara y gestionadas por la placa computadora *Raspberry Pi 3B*+, se envían a un procesador central donde se aplican técnicas de segmentación para la extracción de potenciales regiones, en este caso una plataforma como objetivo a alcanzar por el vehículo, que se reconocen mediante técnicas de aprendizaje profundo. La localización de la plataforma permite aplicar técnicas para controlar los motores y dirigir el vehículo hacia la plataforma.

Se proporciona de este modo una solución de concepto a pequeña escala, dentro de los vehículos inteligentes, que constituyen un potencial de futuro en la aplicación de las nuevas tecnologías en los futuros vehículos conectados autónomos.

El planteamiento anterior, da lugar a los siguientes objetivos específicos:

- Definir y configurar los distintos componentes hardware que configuran el vehículo.
- Puesta en marcha de los dispositivos, integración y definición de los componentes de comunicación y transferencia de datos.
- Desarrollar técnicas basadas en técnicas de segmentación de imágenes para extraer potenciales regiones, que identifican la plataforma.
- Desarrollar técnicas de transferencia de conocimiento mediante Redes Neuronales Convolucionales (RNC) pre-entrenadas y re-entrenadas, dentro del paradigma de lo que se conoce como aprendizaje profundo.
- Definir los protocolos y procedimiento de las órdenes necesarias para activar los motores que hacen girar convenientemente las ruedas del vehículo.
- Desarrollar un método de corrección de la trayectoria del vehículo para su guiado hacia la plataforma.
- Integrar las técnicas y módulos anteriores para desarrollar el vehículo autónomo final.

Las contribuciones realizadas en el marco del presente proyecto coinciden exactamente con la consecución de los objetivos planteados.

El plan de trabajo contiene exactamente el mismo número de tareas que objetivos, coincidiendo también en el nombre, con una distribución equitativa entre las semanas que comprende el curso completo.

1.3 Organización de la memoria

Tras la introducción en las secciones 1 y 2, que incluye la visión general del proyecto junto con la definición de los objetivos, en la sección 3 se describen los contenidos teóricos que sustentan el planteamiento de la propuesta. En la sección 4 se describe el diseño del prototipo del vehículo, explicando con detenimiento los componentes principales que lo constituyan, la forma en la que se implementa su circuito y la información referente al entrenamiento de la RNC. En la sección 5 se proporciona una visión general de funcionamiento y comunicación entre el vehículo y el ordenador principal, así como la explicación del sistema global del vehículo para luego detallar los procesos y resultados que se llevan a cabo en el ordenador principal donde se realiza el procesamiento relevante de

las imágenes y su clasificación mediante la RCN. Finaliza la sección con una serie de ejemplos de errores identificados durante la clasificación de regiones con la RNC. En las secciones 6 y 7 se exponen las conclusiones finales del proyecto así como posibles mejoras o avances a partir del sistema ya implementado que pudieran realizarse más adelante. Finalmente, se incluye un Anexo, que contiene el manual de usuario para probar el sistema en su conjunto.

2. - Introduction

2.1 Preliminary

It is well known that large companies in the automotive sector are involved in multiple competitive processes in an unstoppable race towards the development of autonomous vehicles. Some as Tesla, Uber, Google are known for the advertising roll-out, without ruling out large companies in the traditional automobile sector. On the other hand, there are technological and research centers developing projects under the same philosophy, and for quite some time, such is the case of the prototype of the Free University of Berlin led by professor Rojas (2020), the Autopía project (2020) of the Spanish Higher Council for Scientific Research (CSIC) or the CAV (2020) project of the Institute for Automobile Research (INSIA) of the Polytechnic University of Madrid for the integration of cooperative systems in shared traffic. All of them committed to the future for their integration in smart cities (Citilogs, 2020; Tecnalia, 2020) where autonomous and connected vehicles will be part of urban landscapes, without a doubt.

In these developments, physical systems are combined with intelligent models, where deep learning is called to play a relevant role. It is under this perspective on which the present work arises, motivated by the interest in developing a small-scale intelligent prototype, but incorporating technologies coming from large-scale prototypes such as those developed by the aforementioned automobile companies. Thus, the vehicle navigates towards a target (platform) guided by a vision system, which sends images to a remote computer, so that the target is identified through a convolutional neural network. The location of the target allows guiding maneuvers, using a complementary ultrasound system. Convenient communication is established between the on-board processor in the vehicle, in this case a Raspberry Pi, and the central (remote) computer where processes with high computational cost are carried out, such as image processing and object classification, through the neural network.

In this way, a prototype of an intelligent car is proposed (HackerCar, 2020) that goes beyond a mere simulation, which on the other hand is essential in all initial development, and therefore enters into real problems such as image acquisition in adverse lighting conditions, the turning of the wheels or the advance of the motors, to name just a few. The proposal formulated in this project is in line with Tian's (2019) model based on a Raspberry Pi, a SunFounder PiCar as an ultrasonic sensor to avoid obstacles, and a Google's Edge TPU unit, for processing close to the sensor (edge) through a TPU unit (Tensor Processing Unit), which uses, as software tools, the Python language, OpenCV libraries for image processing and Tensorflow as a specific framework for deep learning-based processing. At the conceptual level, the difference proposed in this work, with respect to the previous one, lies in communication between systems (processors), so that there is a transfer of data between systems, taking into account that in connected systems this type of approach must prevail with intercommunications between connected vehicles and a required Control Center. The processes are carried out through the corresponding toolboxes (Image Processing, Computer Vision and Deep Learning) from Matlab (2020) for image processing and deep learning. Some other approach along the same lines is the one proposed in Self-Driving (2020) that uses Google Colab (Martín de la Fuente, 2019), as a tool in the cloud to execute Python code, which, in this case, uses a LIDAR (Light Detection and Ranging or Laser Imaging Detection and Ranging) based sensor to derive the three-dimensional (3-D) structure of the scenes.

On the other hand, national and international groups are diverse, sometimes under the auspices of certain organizations, which promote the development of prototypes of this nature, sometimes for competition purposes (Make, 2020). In the development of this kind of prototypes, it is common to use Raspberry Pi (2020) computers, as indicated in Upton (2020) or Horsey (2016).

2.2 Objectives

The general objective is to develop a prototype of an intelligent vehicle at scale, with its corresponding hardware / software equipment.

a) **Hardware**: Raspberry Pi 3B +, camera, ultrasound sensor, motors, motor controllers, chassis, power supply.

b) **Software**: image processing, deep learning.

All these elements constitute an integrated platform for autonomous navigation, so that by capturing images trough the camera and managed by the *Raspberry Pi* computer, they are sent to a central processor where image segmentation techniques are applied to extract

potential regions in order to identify the platform through deep learning techniques, which is the target to be achieved. The location of the platform allows applying techniques to control the engines and guide the vehicle towards the platform.

In this way, a small-scale concept solution is provided, within smart vehicles, which constitute a future potential in the application of new technologies in future autonomous connected vehicles.

The above approach provides the following specific objectives:

- Define and configure the different hardware components that make up the vehicle.
- Start-up the devices, integration and definition of the communication and data transfer components.
- Develop techniques based on image segmentation techniques to extract potential regions, which identify the platform.
- Develop knowledge transfer techniques through pre-trained and re-trained Convolutional Neural Networks, within the paradigm of what is known as deep learning.
- Define required protocols and procedures to activate the motors that make the vehicle wheels turn conveniently.
- Develop a vehicle path correction method for guiding it towards the platform.
- Integrate the previous techniques and modules to develop the final autonomous vehicle.

The contributions made, within the framework of this project, match exactly with the achievement of these objectives.

3.- **Descripción de métodos aplicados**

Tal y como se describe en los objetivos, se plantea el desarrollo de un prototipo de vehículo autónomo a escala, que mediante el reconocimiento de imágenes identifique un objeto, en este caso, una plataforma para su identificación y localización con el fin de navegar de forma autónoma hasta ella. Como se ha indicado previamente, el prototipo consta de una serie de componentes hardware y software. En esta sección se describen los algoritmos inteligentes encuadrados dentro de estos últimos, dejando el resto en la descripción del prototipo, que se aborda en la sección cuatro.

Son tres las técnicas inteligentes las que se describen a continuación, a saber: a) segmentación de imágenes, b) etiquetado de regiones y mejora de las mismas mediante operaciones morfológicas y c) reconocimiento mediante técnicas de aprendizaje profundo basadas en Redes Neuronales Convolucionales (RNC).

3.1 Segmentación de imágenes por color

En las imágenes aparecen ciertas áreas o zonas caracterizadas por el hecho de que constituyen agrupaciones de píxeles conectados entre sí, pero además de la conexión, dichos píxeles presentan propiedades o características comunes. Dichas áreas son las regiones. Una de tales propiedades es precisamente el color. Tomando como base esta propiedad, para la plataforma de aproximación del vehículo que se plantea, se propone el modelo mostrado en la figura 1, de suerte que sobre una base con fondo negro se incluyen cuatro figuras geométricas con otros tantos colores: rojo (cuadrado), verde (triángulo), azul (círculo), naranja (rectángulo).

Figura 1. *Plataforma de aproximación.*

Este diseño obedece a una serie de criterios lógicos en relación al entorno de navegación. Por un lado se plantea una base de color negro, con las figuras y sus colores indicados sobre ella. El fondo negro permite identificar sobre el escenario, con predominio de colores claros, regiones de esta naturaleza en base a la propiedad del color. Lo mismo es aplicable a la identificación de las figuras, si bien en el diseño que se presenta, esto no es necesario, ya que éstas sirven como elementos diferenciadores para la RNC. En cualquier caso, es el color la propiedad diferenciadora en esta etapa del procesamiento de la imagen.

Desde el punto de vista del análisis del color, resulta muy conocido el hecho de que la luz solar que atraviesa un prisma de cristal se descompone en los siete colores básicos del espectro visible. Los colores que el ojo humano percibe de un objeto están determinados

por la naturaleza de la luz reflejada por dicho objeto. Un cuerpo que absorbe la luz de todas las longitudes de onda se muestra como negro al observador. Sin embargo, un cuerpo que favorece la reflectancia en un rango limitado de longitudes de onda en el espectro visible exhibe un determinado color. Por ejemplo, los objetos verdes reflejan la luz con longitudes de onda en el rango de 500 a 570 nm y absorben gran cantidad de energía en otras longitudes de onda.

Tal y como se describe en Pajares y Cruz (2007), muchos modelos de color en uso hoy día están orientados, bien hacia el hardware (monitores en color e impresoras) o hacia aplicaciones donde la manipulación del color es un objetivo (creación de gráficos en color para animación). Los modelos más comunes orientados al hardware usados en la práctica son el RGB (rojo, verde, azul) para monitores en color y una amplia gama de vídeo cámaras; el CMY (cyan, magenta, amarillo) para impresoras en color; y el YIQ para TV en color. En el modelo YIQ, la coordenada Y corresponde a la reflectancia, e I y Q son dos componentes cromáticas llamadas *infase* y *cuadratura* respectivamente. En procesamiento de imágenes en general, se utilizan con bastante frecuencia los modelos RGB, YIQ, HSV (matiz, saturación, valor), HSI (matiz, saturación, intensidad). En el presente trabajo se considera el modelo RGB como el más simple y apropiado para el objetivo que se plantea, si bien para la extracción de regiones para su umbralización se utiliza el modelo HSI, de forma que es la imagen de intensidad, I, la más apropiada para tal fin.

En el modelo RGB cada color aparece en sus componentes espectrales primarias: rojo, verde, azul. Este modelo está basado en el sistema de coordenadas cartesianas. El subespacio de color de interés es el tetraedro mostrado en la figura 2. En el cual los valores RGB se sitúan sobre tres vértices; cyan, magenta y amarillo se ubican en los otros tres vértices, el negro corresponde al origen y el blanco en el vértice más alejado del origen. En este modelo, la escala de grises se extiende desde el negro al blanco a lo largo de la diagonal que une esos dos puntos, y los colores son puntos dentro del tetraedro, definidos por vectores desde el origen. Por conveniencia, se asume que todos los vectores han sido normalizados, de modo que el tetraedro de la figura 2 es el tetraedro unitario, es decir, todos los valores de R, G y B están en el rango [0,1]. Las imágenes en este modelo se forman por la combinación en diferentes proporciones de cada uno de los colores primarios RGB.

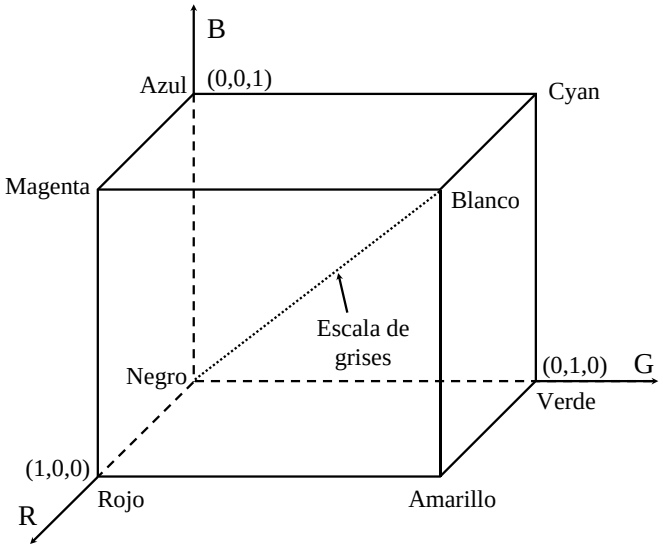


Figura 2. *Tetraedro de color en el modelo RGB.*

Como se ha indicado previamente, para la identificación de la plataforma el color base de la plataforma es el negro, cuyas característica esencial es que sus valores en las tres componentes se sitúan alrededor del origen del tetraedro de la figura 2, constituyendo así el criterio diferenciador para su identificación. Por otra parte, sobre el mismo tetraedro se identifican las componentes dominantes de los distintos colores, que por ejemplo en el caso del color rojo puro, éste posee valores RGB = (1,0,0) o equivalentemente (255, 0, 0). Consecuentemente, los colores Verde y Azul puros tienen respectivamente valores (0,255,0) y (0,0,255). Finalmente, el color naranja se identifica con los valores (255,128,0).

Para identificar las regiones con posibilidades de ser la base de la plataforma se establece el criterio de que cada píxel pose unos valores de componentes de color, R,G y B próximos a los valores espectrales que definen el color negro. Para ello se aplica el método de multi-umbralización basado en el algoritmo de Otsu (1979) de forma que se obtienen diferentes niveles de umbral y aquellos próximos a cero, so los que determinan el color de la base de la plataforma.

3.2 Etiquetado de componentes conexas

Los píxeles con unos niveles espectrales R, G y B próximos a los valores que definen el negro se ponen a cero, y el resto a uno. De esta manera, se genera una nueva imagen donde los píxeles potencialmente negros aparecen con dicho valor, constituyendo agrupaciones de píxeles que definen las potenciales regiones que definen la plataforma de aproximación del vehículo.

Para el etiquetado de las regiones así identificadas, se parte por tanto del hecho de que la imagen se encuentra binarizada (imagen de ceros y unos). A partir de este momento el objetivo consiste en etiquetar las componentes conexas de forma que el resultado final sea una región donde todos los píxeles estén unidos entre sí. El concepto de componentes conexas es el siguiente: a todos los píxeles que tienen valor binario “1” y están conectados entre sí por un camino o conjunto de píxeles todos ellos a su vez con el valor binario “1” se les asigna la misma etiqueta identificativa, que debe ser única de la región a la cual pertenecen los píxeles y constituye su identificador.

Con tal propósito se aplican algoritmos específicos de etiquetado de componentes conexas, que en esencia consiste en agrupar píxeles de la misma región asignándoles la misma etiqueta.

Estos algoritmos se fundamentan en el tipo de conectividad, distinguiéndose dos tipos, conectividad-4 y conectividad-8. Durante el proceso de exploración de los píxeles para su etiquetado y agrupamiento en regiones, en la conectividad-4 intervienen los píxeles situados en dirección horizontal a la izquierda y derecha del píxel central y en dirección vertical por arriba y por abajo con referencia al píxel central. En el caso de la conectividad-8 intervienen todos los que rodean al píxel central y por tanto a los cuatro anteriores más los otros cuatro diagonales, sumando un total de 8, de ahí su nombre. En general, los algoritmos procesan una fila de la imagen en cada instante y asignan nuevas etiquetas al primer píxel de cada componente e intentan propagar la etiqueta de un píxel a sus vecinos según la conectividad establecida (Haralick y Shapiro, 1992), consideremos la imagen *A* de la figura 3, cuyo resultado final tras el etiquetado es la imagen *B* de la misma figura.

$$A \equiv \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 0 & 1 \end{bmatrix} \quad B \equiv \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 3 \\ 1 & 1 & 0 & 2 & 0 & 0 & 3 \\ 0 & 0 & 0 & 2 & 0 & 0 & 3 \\ 0 & 0 & 2 & 2 & 2 & 0 & 3 \end{bmatrix}$$

Figura 3*. Ejemplo de etiquetado de A en las etiquetas de B.*

3.3 Operaciones morfológicas

Las operaciones morfológicas han sido ampliamente utilizadas en imágenes binarias con el fin de perfilar y mejorar los resultados obtenidos en la segmentación. Para su definición, en primer lugar es necesario determinar el origen de la imagen desde el punto de vista de las coordenadas (*x*,*y*). El origen se identifica gráficamente con un punto situado en uno de los píxeles.

Con la imagen dada y el origen fijado se construye un conjunto cuyos elementos son los pares de coordenadas (*x*,*y*) que indican las posiciones de los "*unos*" lógicos. A modo de ejemplo, considérese la imagen siguiente con su origen marcado por el punto dado, figura 4.

$$X = \begin{array}{c} \\ y \\ \downarrow \end{array} \begin{array}{c} \begin{array}{ccccc} & x \rightarrow & & & \\ -1 & 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & \bullet 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array} \begin{array}{c} \\ -1 \\ 0 \\ 1 \\ 2 \\ 3 \end{array}$$

Figura 4. *Imagen morfológica.*

A partir de ella se construye el conjunto según sigue,

$$X = \{(0,-1),(1,-1),(2,-1),(0,0),(1,0),(2,0),(1,1),(2,1),(1,2),(2,2)\} \quad (1)$$

Una transformación morfológica ϕ viene dada por la relación de la imagen (conjunto de puntos *X*) con otro pequeño conjunto de puntos *B*, llamado operador o elemento estructural. *B* se expresa también con respecto a un origen local *O* (llamado punto representativo o elemento director). Un elemento estructural es el que se muestra en la figura 5.

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & \bullet 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Figura 5. *Elemento estructural.*

..

La transformación morfológica de la **dilatación** $\oplus$ combina dos conjuntos utilizando la adición de vectores (o adición de conjuntos de Minkowski). La dilatación $X \oplus B$ es el conjunto de puntos de todas las posibles adiciones vectoriales de pares de elementos, uno de cada conjunto *X* y *B*.

$$X \oplus B = \left\{ \mathbf{d} \in E^2 : \mathbf{d} = \mathbf{x} + \mathbf{b} \text{ para cada } \mathbf{x} \in X \text{ y } \mathbf{b} \in B \right\} \quad (2)$$

La transformación morfológica de la erosión $\otimes$ combina dos conjuntos utilizando la sustracción de vectores.

$$X \otimes B = \left\{ \mathbf{d} \in E^2 : \mathbf{d} + \mathbf{b} \in X \text{ para cada } \mathbf{b} \in B \right\} \quad (3)$$

Esta expresión indica que cada punto **d** del conjunto *X*, que para nosotros será la imagen, es comprobado; el resultado de la erosión está dado por los puntos **d** para los

cuales todos los posibles **d** + **b** están en *X.* La erosión y dilatación son transformaciones no invertibles. Si una imagen es erosionada y luego dilatada, la imagen original no se recupera. En efecto, el resultado es una imagen más simplificada y menos detallada que la imagen original.

La erosión seguida de una dilatación crea una transformación morfológica importante llamada *apertura*. La apertura de una imagen *X* por un elemento estructural *B* se denota por $X \circ B$ y se define como,

$$X \circ B = (X \otimes B) \oplus B \tag{4}$$

La dilatación seguida de una erosión crea una transformación morfológica llamada *cierre*. El cierre de una imagen *X* por un elemento estructural *B* se denota por $X \bullet B$ y se define como

$$X \bullet B = (X \oplus B) \otimes B \tag{5}$$

3.4 Redes Neuronales Convolucionales

Una Red Neuronal Convolucional (RNC) es un tipo específico de red neuronal, cuyo fundamento son las denominadas capas de convolución, de ahí su nombre. El objetivo, consiste en *aprender* determinados valores o pesos a partir de las imágenes proporcionadas durante la fase que se conoce como entrenamiento. Una de las RNC ampliamente utilizadas es AlexNet (ImageNet, 2019; Russakovsky y col., 2015; Krizhevsky y col., 2012), que ganó la competición ILSVRC-2012 con una tasa de error de un 15.3% frente al segundo competidor que obtuvo un 26.2%. La figura 6 muestra la arquitectura de este modelo.

Se trata de una red con veinticinco capas, previamente pre-entrenada, capaz de clasificar mil objetos diferentes. Los tamaños y estructuras de las diferentes capas se establecen mediante las operaciones correspondientes que se especifican seguidamente. En cada capa se aplican operaciones de convolución con núcleos de distintos tamaños según la nomenclatura *m*×*m*. Los indicadores *k*, *p* y *s* que aparecen en las diferentes capas, hacen referencia respectivamente al número de filtros (con su correspondiente núcleo de convolución) aplicados, al número de ceros añadidos (*padding*) y unidades de desplazamiento del núcleo (*stride*). La entrada resulta ser una imagen de dimensión 227×227×3.

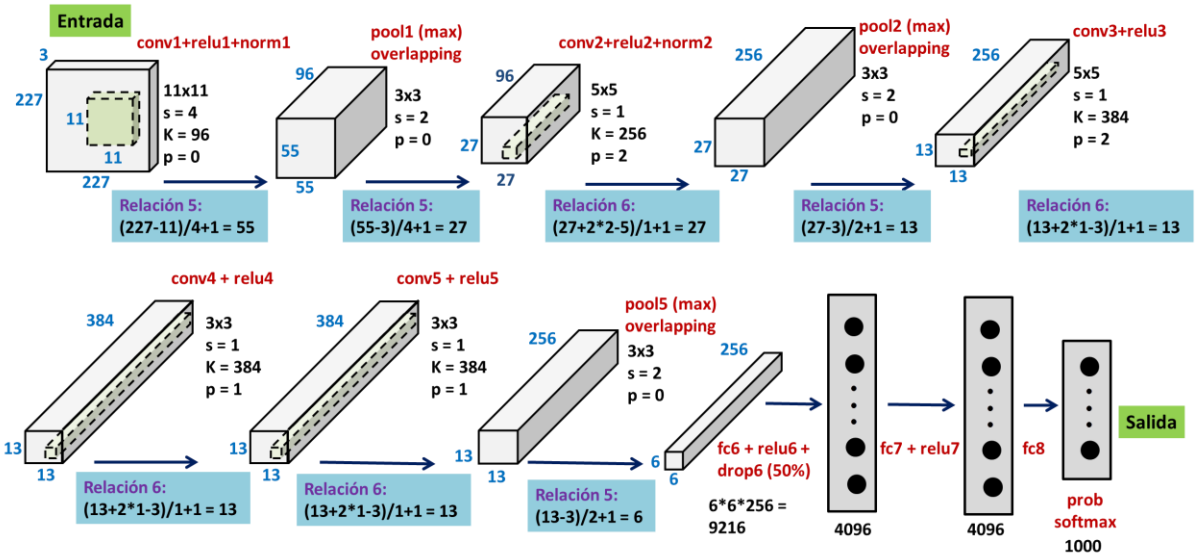


Figura 6. *Esquema de la red AlexNet*

Convolución (conv), esta operación matemática combina dos funciones para generar una tercera, consiste básicamente en superponer una función sobre otra modificada para obtener una serie de características. Se define según la expresión dada en la ecuación (6).

$$S(i,j) = (I * K)(i,j) \tag{6}$$

donde el argumento *I* se conoce como entrada (*input*), que es un vector o matriz multidimensional. El segundo argumento *K*, conocido como filtro (*kernel*) de convolución, es una matriz de pesos que se obtienen durante el proceso de aprendizaje. Ambas entradas son estructuras conocidas como tensores. La salida S se define generalmente como mapa de características (*feature map*).

ReLU (relu), es una función de activación, de forma que, para cada resultado generado por la convolución, se aplica una función de transformación que rectifica linealmente el valor de entrada devolviendo 0 para valores negativos y el propio valor de entrada para valores positivos, definida como $f(x) = max(0, x)$, con x siendo la entrada a la neurona, y cuya representación gráfica se muestra en la figura 7.

Normalización (norm), la operación de normalización sirve para acelerar la convergencia durante el proceso de aprendizaje. Actualmente se usa la normalización por lotes que se define como sigue,

$$b^i_{x,y} = a^i_{x,y} \Big/ \left(k + \alpha \sum_{j=max(0,i-n/2)}^{min(N-1,i+n/2)} \left(a^i_{x,y} \right)^2 \right)^{\beta} \tag{7}$$

donde N es el número de filtros de la capa dada, $a^i_{x,y}$ es la actividad de la neurona y k, α, n, β son hiperparametros. En Krizhevsky (2012) se propone *k*=2, *n*=5, $\alpha = 10^{-4}$, β =0.75.

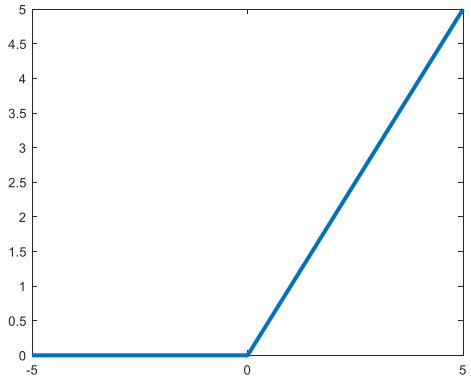


Figura 7. *Función ReLU*

Pooling (pool), la función de pooling modifica la capa de salida agrupando en una porción más pequeña las características, de tal forma que dada una pequeña traslación en la entrada no afecte a los valores de las salidas. Se define como se indica en la ecuación (2.10), teniendo en cuenta que las variables involucradas representan la dimensión de entrada (*i*), el tamaño del núcleo (*kernel*) de convolución o de *pooling* (*k*), el desplazamiento del núcleo a lo largo de la imagen (*s*, *stride*).

$$O = \left[\frac{i-k}{s} \right] + 1 \tag{8}$$

Dropout (drop), sirve para no sobresaturar la red neuronal generando soluciones de polinomios de grado demasiado alto. Para ello se propone anular determinado tipo de neuronas de forma aleatoria mediante un hiperparámetro que indique la probabilidad de supervivencia de cada neurona. Las operaciones de este tipo aparecen indicadas como *drop,* correspondiendo en ambos casos al 50% de las neuronas anuladas.

Softmax, se aplica la función exponencial a cada elemento del vector de entrada y se normalizan sus valores por la suma de las exponenciales, lo que asegura que los valores del vector de salida se sitúen en el rango [0,1], proporcionando así la probabilidad de pertenencia a cada una de las clases para una imagen de entrada dada. Su función se define según la siguiente expresión.

$$softmax(\mathbf{x})_i = \frac{exp(x_i)}{\sum_{j=1}^{n} exp(x_j)} \tag{9}$$

Gradiente descendente. Técnica de minimización, que se usa también para el ajuste de los pesos de la red durante el proceso de retro-propagación. La función a minimizar se conoce como función *objetivo* o *criterio* y cuando se está minimizando, se le denomina también *cost function, loss function* o *error function*.

Todos los procedimientos basados en RNC, consisten en extraer determinadas características de las imágenes mediante las operaciones definidas previamente, de forma que se obtienen los valores que definen los núcleos de convolución. Estas características pueden ser bordes, colores, formas u otras estructuras subyacentes en las imágenes. En las primeras capas se usan las funciones de convolución, rectificación lineal para destacar las características importantes y normalización para acelerar la convergencia. Para conectar cada neurona con su capa oculta se utiliza una matriz de pesos, que se modificará según la red neuronal vaya "aprendiendo", y un sesgo, a este par se le conoce como filtro con su correspondiente núcleo. Cada núcleo se genera en función de las características de la imagen a destacar. Seguidamente se usa una capa de *pooling* para crear una versión condensada de las características derivadas de la información recogida por la capa convolucional, cabe destacar que existirán tantas capas de convolución como de agrupación.

Tras varias etapas de convolución-agrupación se llega a las conexiones de tipo "*Fully Connected*" o totalmente conectadas, identificadas como *fc*, concretamente en AlexNet se corresponde con la sexta etapa. Esta etapa se caracteriza por realizar la función de conectar todas las neuronas de entrada con todas las neuronas de salida, la función ReLU y la función dropout que permiten reducir el sobreajuste de un modelo, que ocurre por la existencia de diversas capas ocultas con múltiples conexiones, y que generan un error de tipo aleatorio por el hecho de destacar detalles irrelevantes. Por último, se aplica la función *softmax* definida previamente para obtener la probabilidad para cada una de las clases que queramos analizar.

Para que la red tenga resultados lo más correctos posibles debe ser reentrenada para reajustar los pesos de las capas convolucionales, con respecto a los valores que posee en su modelo original, de forma que las salidas sean las deseadas. En el caso del presente trabajo se han establecido dos clases diferentes de salida (Plataforma y Otros objetos) por lo que existirá como resultado de la red, después de la operación softmax, un vector cuyos valores corresponden a esas dos clases [y0, y1]. En el caso de la red AlexNet los pesos iniciales se heredan del modelo pre-entrenado, dando como resultado una salida, probablemente distante con respecto a la deseada. El objetivo es ajustar (minimizar) la función *de pérdida* durante el entrenamiento, de forma que una salida deseada, por ejemplo [1,0], que correspondería a la plataforma se corresponda con la que se obtiene para una entrada dada. Este reajuste se realiza mediante retro-propagación del error aplicando la técnica del gradiente descendente.

La red debe reentrenarse con el mayor número de ejemplos (imágenes) disponibles para conseguir el mejor mínimo posible de la mencionada función de pérdida.

4.- Prototipo

En esta sección se muestran y definen los componentes que integran el prototipo de vehículo a escala, su funcionamiento y el diseño global de dicho vehículo, con especial énfasis en el computador de a bordo (Raspberry Pi 3B+), el sensor de ultrasonidos (HC-SR04) y la cámara de visión, como elementos esenciales.

4.1 Prototipo y componentes principales

En la figura 8 se muestra el prototipo visto desde su parte lateral derecha y los componentes visibles desde esta perspectiva. Los elementos sobre la misma son:

a) Raspberry Pi 3B+, cuyos detalles se describen en la sección 4.2.
b) Motor DC derecho, que controla la rueda derecha.
c) Sensor de proximidad HC-SR04, descrito en la sección 4.3.
d) Cámara o sistema de visión, descrito en la sección 4.4.

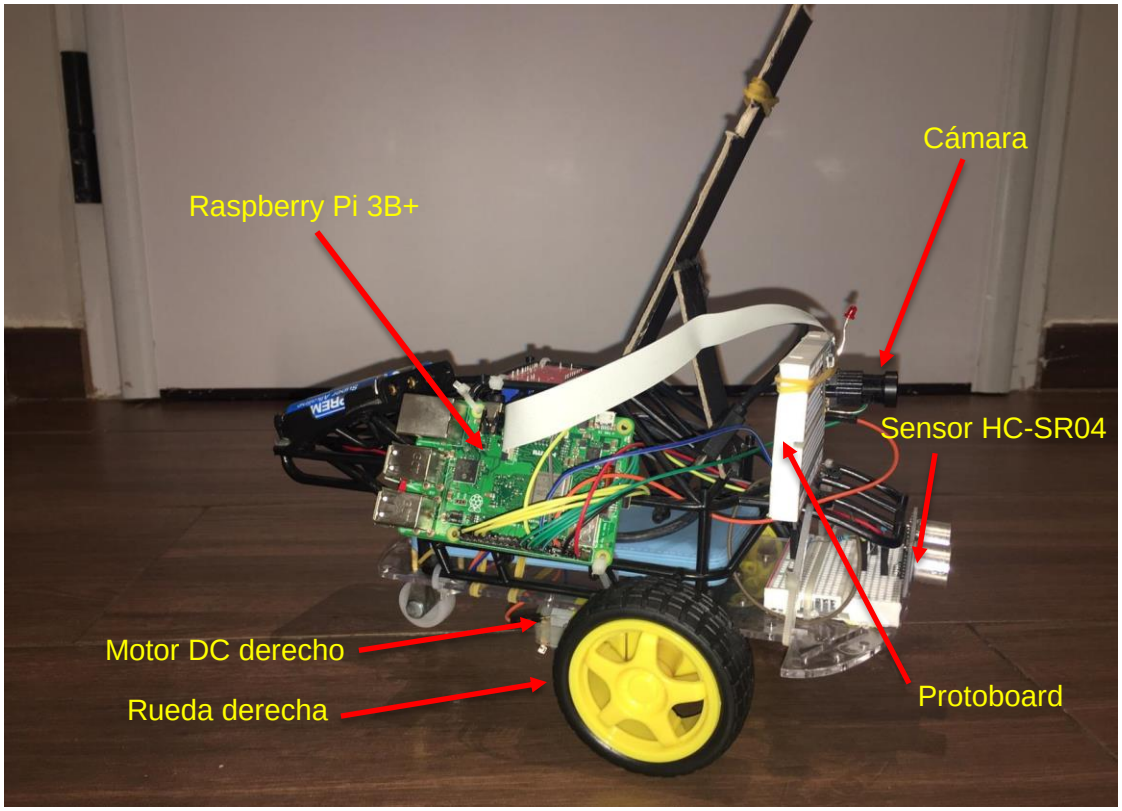


Figura 8. *Prototipo y componentes, lado derecho*

En la figura 9 se muestra el prototipo visto desde su parte lateral izquierda, junto con sus componentes visibles desde esta perspectiva. Desde esta perspectiva aparecen elementos comunes a los mostrados en la figura 8, los cuales se excluyen del listado.

a) Chasis del vehículo formado por una placa de metacrilato y una estructura plástica de la carrocería reutilizada de otro coche rc, que se utiliza como sostén de algunos componentes.
b) Controlador de motores L298N descrito en la sección 4.5.
c) Soporte elevado de cámara.
d) Powerbank de 10A para alimentar a la *Raspberry Pi*.

El resto de componentes del vehículo se enumeran en el apartado 4.6.

En el diseño del prototipo se ha tenido especial cuidado en realizar un sistema fácilmente mantenible y de fácil acceso a sus componentes, con la idea de que se pueda desmontar en poco tiempo preservando la estructura original en la medida de lo posible. Componentes como la cámara y el sensor *HC-SR04* no tienen una posición fija, si no que se pueden mover dependiendo de la posición que sea más razonable según la función que tenga que realizar el vehículo, las cuales se describen más adelante.
Por ultimo cabe destacar que este diseño flexible admite la posibilidad de añadir nuevos componentes y modificar la estructura con la facilidad que se pretende para pruebas futuras.

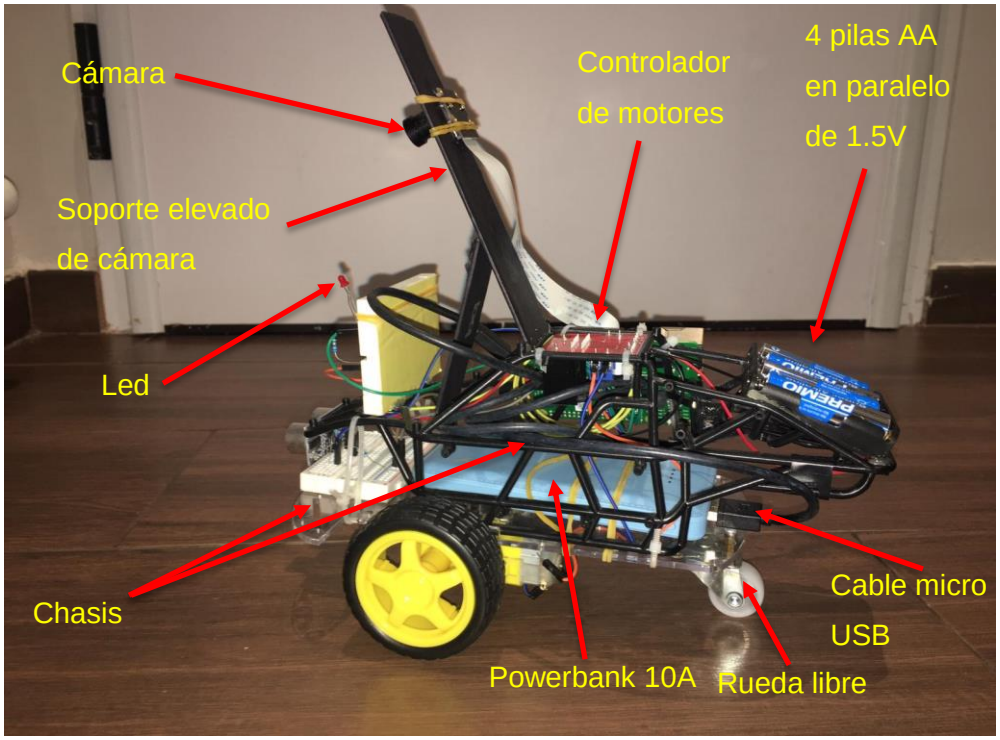


Figura 9. *Prototipo y componentes, lado izquierdo*

En la imagen de la figura 10 se muestra el vehículo en su conjunto, con una vista frontal en la parte izquierda y visto desde arriba en la parte derecha de la imagen, donde vuelven a aparecer visibles los componentes situados en estas partes.

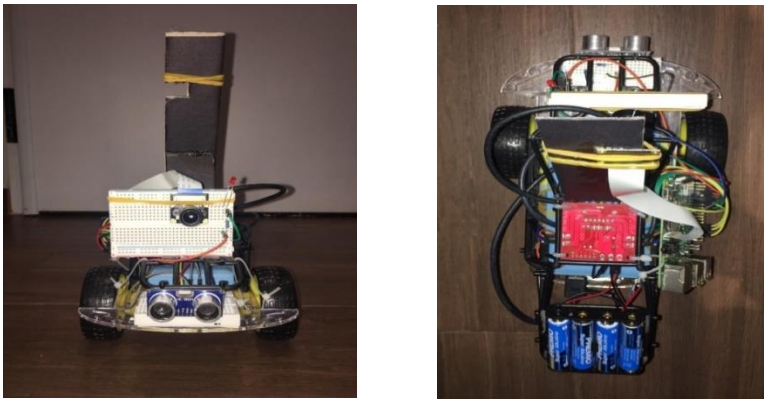


Figura 10. *Parte frontal y superior del vehículo.*

4.2 *Raspberry Pi 3B+*: ordenador principal a bordo del vehículo

El componente que procesa y envía la información a los demás periféricos del prototipo es una *Raspberry Pi 3B+*, que es un ordenador de tipo SBC (Single Board Computer). Realmente se trata de un micro-ordenador que contiene todos sus componentes primarios en una placa de tamaño reducido, ideal para realizar pequeños proyectos de este tipo.

También se comunica con un ordenador externo que realiza todo el procesamiento pesado y envía datos a la *Raspberry* debido a que ésta tiene una potencia de cómputo limitada con lo cual no tiene capacidad para llevar a cabo algunas funcionalidades importantes con alta carga computacional o su procesamiento sería mucho más lento. En la figura 11 se muestra en la parte izquierda la placa Raspberry Pi propiamente dicha y en la parte derecha información más detallada de sus pines de conexión.

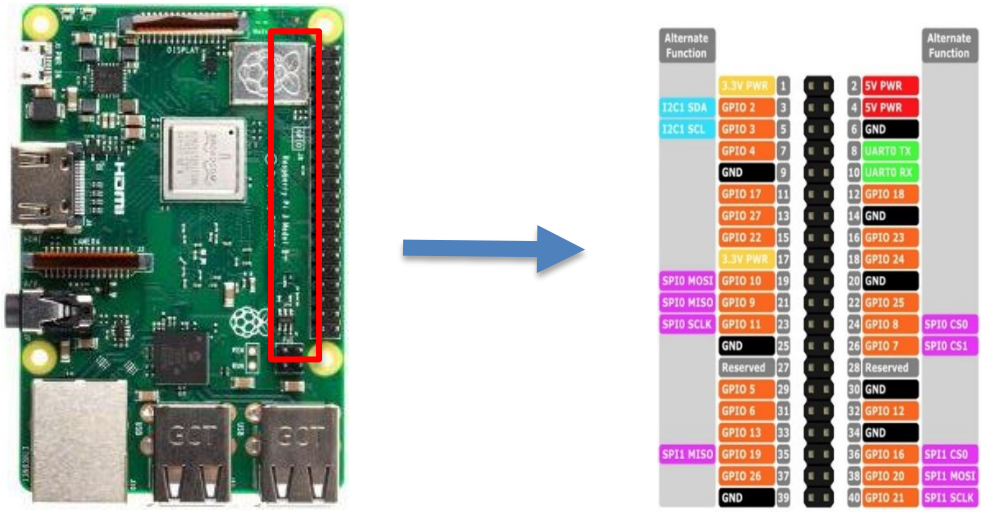


Figura 11. *Raspberry Pi 3 B+ e información de sus pines.*

4.3 Sensor ultrasónico *HC-SR04*: medición de distancia a objetos

El componente que se utiliza para medir la distancia a objetos próximos, con el fin de detectar y esquivar en su caso, obstáculos, es el sensor ultrasónico *HC-SR04*, que consta de dos transductores, uno que emite la onda en el rango de frecuencias correspondientes los ultrasonidos y el otro que se encarga de recibirla, este sensor tiene un alcance de 4.5m y posee un ángulo de actuación efectivo aproximado de 15 grados, en la figura 12(*a*) se muestra el aspecto físico del sensor, con el transmisor situado en su parte izquierda y el receptor en la derecha. En la figura 12(b) se muestra el esquema de funcionamiento de dicho sensor, consistente en el envío de una onda original por parte del emisor, que al impactar sobre un objeto se produce un rebote de la onda (reflejada), que capta el receptor y calcula el tiempo transcurrido entre la emisión y la recepción, de suerte que conociendo la velocidad de propagación de la onda ultrasónica en el aire es posible calcular la distancia a la que se encuentra el objeto. Como se puede observar en la figura 12(b) la distancia a medir, *d*, es en realidad la mitad de la distancia total en la ida y vuelta del viaje de la onda, por lo que para el cálculo de la distancia solamente se considera la mitad del tiempo total, *t*, transcurrido desde el envío de la onda original y la llegada de la onda reflejada. De forma que conociendo la velocidad de propagación de las ondas ultrasónicas, *v*, que a nivel del mar resulta ser de aproximadamente 343 m/s, o equivalentemente 34300 cm/s, resulta que la distancia $d = v \times t/2 \approx 17150*t$ (medida en cm). En alturas diferentes a la del nivel del mar, el sistema necesita una calibración, no obstante, a los efectos de navegación por parte del vehículo esto no se ha considerado necesario, ya que el objetivo es evitar obstáculos por la detección de éstos en las proximidades del sensor, incluso asumiendo errores en el cálculo de las distancias de hasta 5 cm.

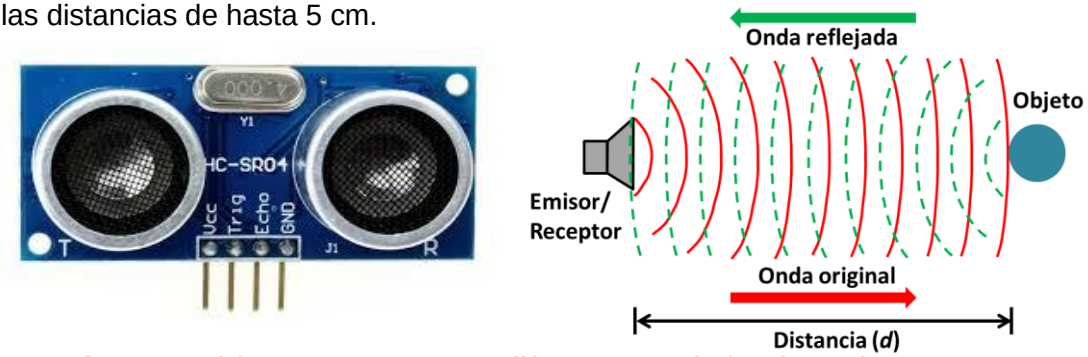


Figura 12. (*a*) *Sensor HC-SR04;* (*b*) *esquema de funcionamiento.*

4.4 Sistema de Visión del vehículo

Esta es sin duda la parte más importante del vehículo, ya que las decisiones de navegación a tomar están basadas en el procesamiento de las imágenes obtenidas por la cámara y por ende en la identificación de la plataforma mediante la RNC. En este caso se

ha utilizado una cámara de 5Mpx equipada con dos luces led infrarrojas por lo que podría utilizarse con poca luz o en total oscuridad, aunque en las pruebas realizadas no se ha utilizado la capacidad de visión infrarroja ya que se han llevado a cabo en condiciones de iluminación natural en un entorno de interior y en condiciones de luz compatibles. De hecho las imágenes utilizadas para el re-entrenamiento de la RNC, son de estas características. No obstante, conviene reseñar en este sentido que nada impediría realizar un re-entrenamiento con imágenes infrarrojas de tal forma que se incrementara la capacidad de detección de la plataforma bajo estas condiciones.

La cámara puede situarse en un soporte en altura para mantener una posición elevada o en la *protoboard* central, además, dada una determinada altura puede variarse su posición horizontalmente sin más que realizar el correspondiente desplazamiento, procediendo a su fijación por un sistema de simples gomas, todo ello dependiendo del uso que se le quiera dar a la cámara. Así, para la identificación y seguimiento de la plataforma la cámara se sitúa en la *protoboard*. De esta forma, el soporte de cámara elevado permite una adaptación del sistema de visión cuando es necesaria una visión más cercana y a baja altura de los objetivos a detectar, ya que con el ángulo del soporte, la cámara enfoca preferentemente al suelo que tiene cercano. De esta forma se puede utilizar para el seguimiento de líneas en el suelo, por ejemplo, como se indica en la sección 6, apartado 4 de la memoria, donde se detectan las líneas de la carretera con la transformada de Hough.

En la figura 13 se muestran las dos posiciones de la cámara, se observa cómo en la parte izquierda (*protoboard*) se puede mover la cámara horizontalmente y verticalmente si fuera necesario, además de poderla situar con gran precisión debido a que los números de la *protoboard* se sitúan justo a la altura de unos orificios que tiene la base de la cámara para colocar los tornillos de fijación. Sobre el soporte elevado también se pueden hacer ligeras modificaciones para poder situar la cámara en el lugar más apropiado en función de la finalidad que se pretenda llevar a cabo con las imágenes captadas.

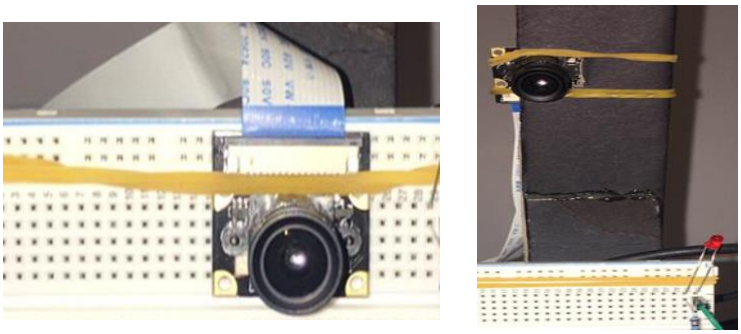


Figura 13. *Cámara y sus posibles posiciones.*

4.5 Guiado: dirección y giros del vehículo

Para el movimiento del vehículo, para su guiado hacia el objetivo, se utilizan dos motores de corriente continua y para controlar su funcionamiento se usa el controlador de motores *L298N* ya que gracias a los dos *'puente H'* que lleva incorporados, es posible cambiar el sentido de giro del motor. El hecho de disponer de esos dos puentes es porque con este módulo se pueden hacer girar los dos motores a la vez.

Por otra parte también es posible modificar la velocidad de cada motor, disponiendo de un regulador de voltaje para tal fin y para cuando es necesaria su aplicación.

Por tanto, en base a los sistemas y dispositivos descritos previamente, cuando el vehículo necesite realizar un giro a la derecha, el motor izquierdo rotará a más velocidad que el derecho y viceversa, para producir el giro global del vehículo en el sentido deseado. El mecanismo es similar al que aplican los vehículos dotados de sistemas de cadenas, tales como los vehículos equipados con cadenas, también conocidos como orugas.

En la figura 14 se muestra el esquema de controlador de motores *L298N* utilizado con sus correspondientes entradas y salidas, pudiéndose observar que en este caso está preparado el acoplamiento de 2 motores.

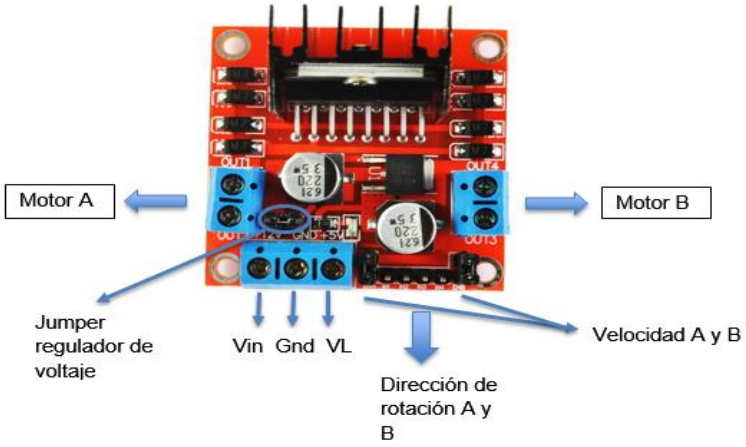


Figura 14. *Controlador L298N.*

4.6 Otros componentes

El equipamiento del prototipo se completa con el conjunto de accesorios que se indica a continuación:

- 2 motores de DC con reductora
- Led rojo
- Cable micro USB
- Base de metacrilato para los motores
- Estructura de otro coche RC para la parte superior del vehículo
- Batería de 4 pilas AA en paralelo de 1.5V, en total 6V para alimentar a los motores
- PowerBank de 10A para alimentar a la *Raspberry Pi*
- 2 protoboards
- Cables y resistencias
- Interruptor para los motores
- Soporte elevado casero para cámara

4.7 Esquema de conexiones del circuito

El esquema del circuito y sus correspondientes conexiones entre los distintos componentes se muestra en la figura 15.

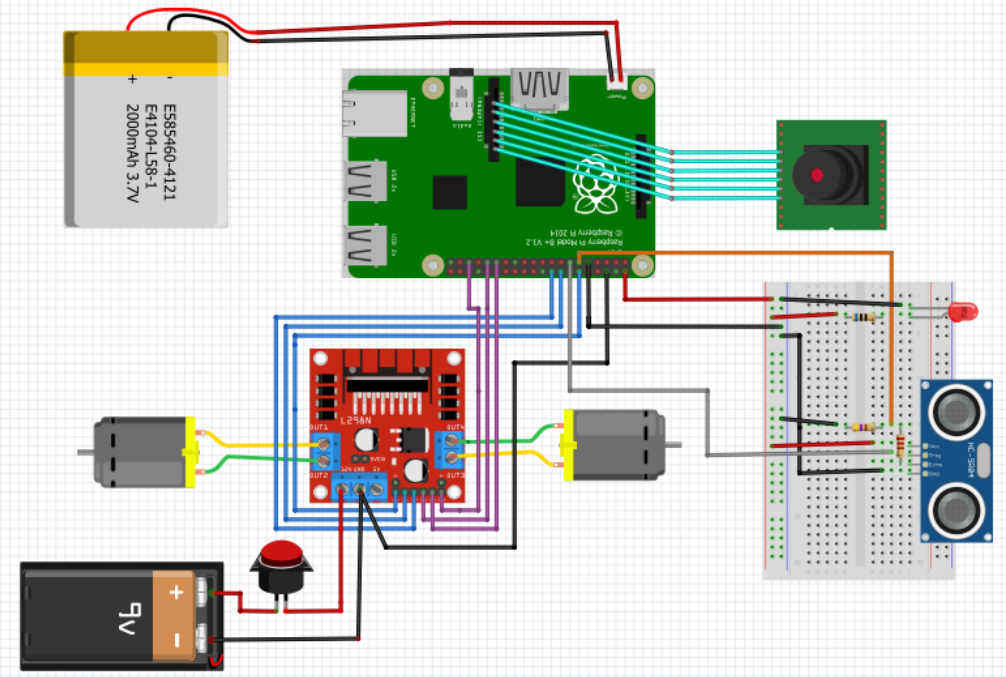


Figura 15. *Esquema del circuito y conexiones utilizando el programa frizting.*

En el prototipo final, en vez de la pila de 9V se utilizaran 4 pilas AA en paralelo generando un total de 6V. La batería que alimenta la *Raspberry Pi* es una *powerbank* de 10A.

4.8 Entrenamiento de la RNC

Como se ha indicado previamente, la parte verdaderamente inteligente del prototipo reside en la utilización de la RNC para la identificación de la plataforma. Así, pues se requiere realizar un proceso de re-entrenamiento con el modelo de red AlexNet, para la que se ha configurado la capa de salida de forma que contenga las dos clases (Plataforma y Otros objetos) utilizadas en este trabajo en lugar de las mil que contiene la configuración original pre-entrenada. No obstante, lo que resulta cierto, es que la red preserva información relevante de las características de los 1000 objetos con los que fue originalmente entrenada, ajustando sus pesos según las nuevas características de los nuevos objetos.

El número de imágenes utilizadas para cada clase se proporcionan en la tabla 1, junto con un ejemplo ilustrativo para cada clase. Obsérvese, cómo para cada clase se han utilizado las 46 imágenes para la identificación de la plataforma y 89 para los objetos que son distintos a ella, con resultados satisfactorios, aunque siempre mejorables, como se verá posteriormente. En este sentido conviene poner de manifiesto la potencialidad de este tipo de redes y la capacidad de ingerir características generales, como las procedentes de las imágenes con las que ha sido entrenada, que son útiles en aplicaciones con un reducido número de ejemplos de entrenamiento. Hay que tener en cuenta que la red AlexNet se ha entrenado con la base ImageNet que contiene del orden de 1,2 millones de imágenes de 1000 clases diferentes. Por tanto, las 135 imágenes utilizadas en el presente trabajo representan un porcentaje mínimo de muestras, con también un número reducido de clases (2 clases frente a 1000).

Clase	Ejemplo representativo	Número de imágenes
Plataforma		46
Otros objetos		89

Tabla 1. Clases de entrenamiento con ejemplos ilustrativos de cada clase

Como puede observarse, y se ha mencionado previamente, en cada uno de los ejemplos mostrados, las imágenes correspondientes a cada clase contienen mayoritariamente el elemento que representan, aunque en la imagen perteneciente a una clase pueden aparecer parcialmente elementos de la otra, si bien, a la hora de decidir se tomará como válida la imagen en función de la probabilidad que proporciona la red.

En la figura 16 se muestra el progreso relativo al proceso de entrenamiento, observándose en cada *epoch* e iteración los resultados relativos a la precisión y al ajuste relativo a la función de coste o *loss function*. Los parámetros y configuraciones establecidos en relación al proceso de entrenamiento son los que se describen a continuación:

1. *Muestras de entrenamiento y validación*. Indican respectivamente, como su nombre indica, el número de muestras utilizadas para el entrenamiento y la validación, obtenidas a partir del total de muestras disponible. En este trabajo, del total de 135 muestras disponibles, el 70% (95 imágenes aproximadamente) se utilizan para el entrenamiento y el 30% para la validación (40).
2. *Epochs e iteraciones*. Representa el paso de todas las muestras disponibles para el entrenamiento y por tanto las 95 reseñadas previamente. Se han establecido 10 *epochs* con 12 iteraciones por *epoch*, resultando un total de 120 iteraciones.
3. *Precisión en la validación* (*validation accuracy*). Precisión en el conjunto de imágenes utilizadas para validación (40 en total).
4. *Dimensión por lotes* (*mini-batch size*). Define la cantidad de imágenes que se utilizan en cada iteración.
5. *Razón de aprendizaje* (*learning rate*). Determina la rapidez con la que la red aprende según el método de actualización de parámetros que constituyen el objetivo del aprendizaje.
6. *Patience*. Indica la detención del proceso tras el número de *epochs*, fijado por dicho parámetro, para el que no se consiguen mejoras relevantes en la precisión.

La tabla 2 define los valores de estos parámetros para el proceso de aprendizaje llevado a cabo en uno de los experimentos realizados, que son suficientemente representativos respecto del conjunto de pruebas de entrenamiento llevadas a cabo.

Parámetros de aprendizaje	Valores
Muestras de entrenamiento	95 (70%)
Muestras para validación	40 (30%)
Método de optimización	gradiente descendente
Razón de aprendizaje inicial	1e-4
Disminución de la razón de aprendizaje	0.2 cada 5 epochs
Número máximo de epochs	10
Dimensión por lotes	12

Tabla 2. Parámetros de aprendizaje y valores

En la figura 17 se muestra el resultado final del entrenamiento, en este caso se ha obtenido una precisión en la validación del 100%, en 3 minutos y 20 segundos, que puede considerarse ciertamente exitoso, teniendo en cuenta que el aprendizaje se ha realizado bajo un computador con un procesador Intel Core i7 2.0 GHz (4th *generation*) con 8 GB de RAM y Windows 10 como sistema operativo de 64-bits. Se trata de un tiempo suficiente para el propósito que se pretende, teniendo en cuenta que el entrenamiento no es crítico desde el punto de vista operativo en este tipo de sistemas, situándose dentro de los límites razonables para un proceso *off-line* de tal naturaleza. Obsérvese cómo mucho antes de completar la primera *epoch* los resultados correspondientes a la validación y la precisión en relación al ajuste son aceptables, en el sentido de que ya consiguen el 100% de precisión. De estos resultados se deduce que tanto el diseño y modelo de red utilizado como el número de muestras empleadas son adecuados en el contexto de los experimentos llevados a cabo.

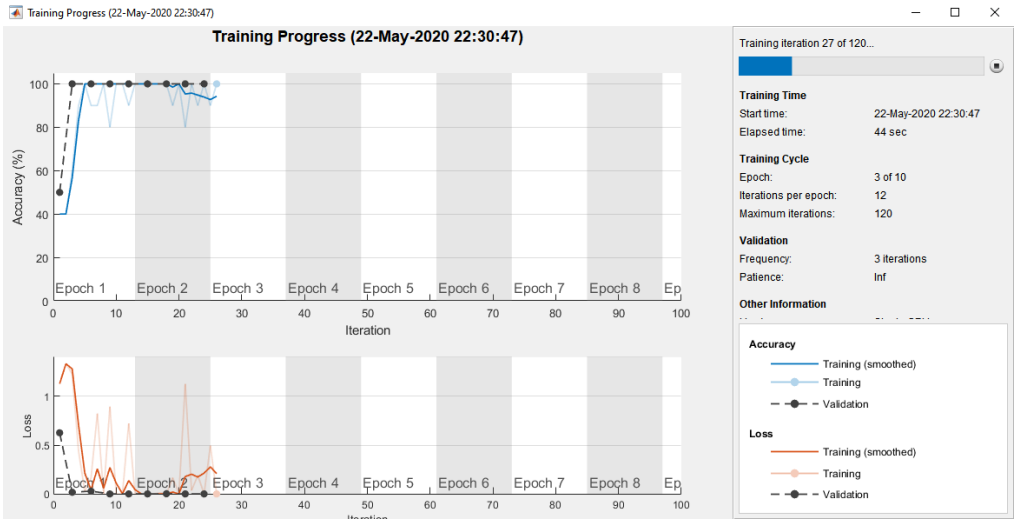


Figura 16. Progreso del proceso de entrenamiento

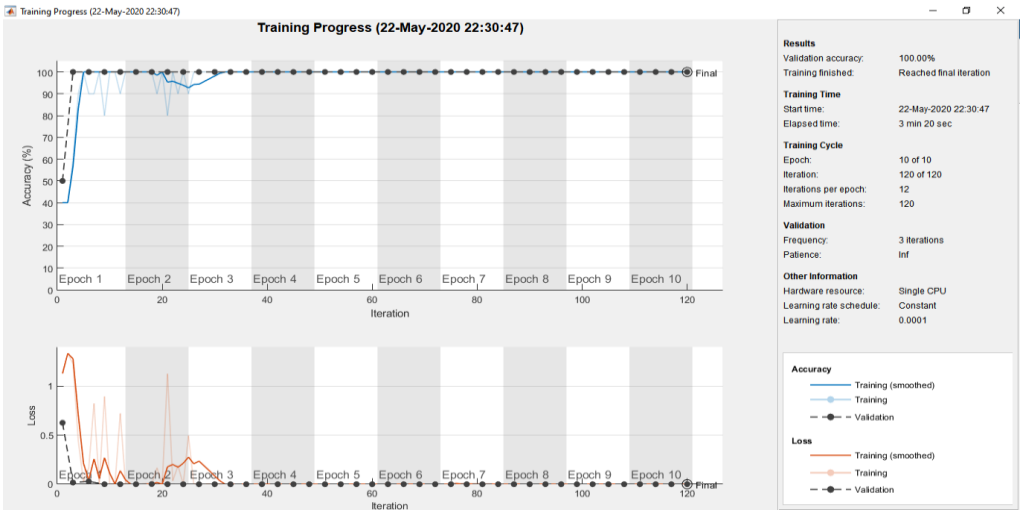


Figura 17. Resumen final del proceso de entrenamiento

La figura 18 muestra los resultados obtenidos tras el aprendizaje en la primera capa de convolución (*conv1* en la figura 6) para una arquitectura de red del tipo AlexNet. Se muestran los 96 filtros correspondientes a esta capa, según se ha definido en la mencionada figura 6. Cada filtro posee tres canales, tal y como también se especifica en dicha figura, de ahí su visualización en color. Esta representación coincide de forma muy aproximada con la representación de los conocidos filtros o funciones de Gabor (Daugman, 1985; Goodfellow y col., 2016). De esta forma, como era de esperar, se establece la conocida similitud con los modelos neuro-computacionales establecidos como base de la justificación de este tipo de redes y su alto rendimiento a la hora de describir las características de los objetos de cara a su reconocimiento.

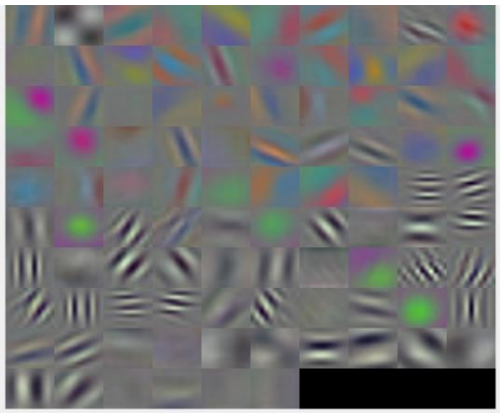


Figura 18. Representación de los pesos aprendidos en la primera capa de convolución (conv1)

A modo de ejemplo, también con carácter ilustrativo, en la tabla 3 se muestran tres núcleos de convolución para la capa correspondiente identificada como *conv5* en la figura 6, que posee 384 núcleos (*K*) de dimensión 3×3. Estos filtros no tienen, en principio una interpretación clara relacionada con aspectos neurobiológicos. En cualquier caso, poseen la capacidad de generar a este nivel lo que se conoce como mapas de características, que contribuyen eficientemente a la caracterización de la imagen de entrada, tanto para la plataforma como para el resto de objetos.

Núcleos	Valores
K(:,:,3,1)	-0.0124 -0.0065 0.0073 -0.0331 -0.0168 0.0055 -0.0240 -0.0017 0.0142
K(:,:,192,1)	-0.0285 -0.0179 -0.0177 -0.0159 -0.0193 -0.0110 -0.0141 -0.0201 -0.0104
K(:,:,192,256)	-0.0161 -0.0193 0.0081 0.0015 -0.0235 -0.0242 0.0047 0.0142 -0.0108

Tabla 3. Pesos aprendidos en la capa de convolución cinco (conv5)

Por último tras realizar el entrenamiento se genera el modelo de red *redTrained* re-entrenado, que luego es exactamente el modelo que se utiliza para realizar la clasificación de los recortes de las imágenes para identificar la plataforma.

5.- Diseño y análisis de la aplicación

En esta sección se explica, el diseño de la aplicación general, teniendo en cuenta que parte de ella se encuentra instalada en el computador *Raspberry Pi 3B+* a bordo del vehículo y otra parte en el computador principal donde se localiza la red neuronal bajo ejecución Matlab. Además se definen los protocolos de comunicación entre los dos procesadores con el correspondiente intercambio de datos. En cada uno de los casos se analizan los resultados obtenidos.

5.1 Funcionamiento general y comunicación

En primer lugar para el desarrollo de proyecto general, mencionar que se necesita un ordenador externo al vehículo, de mayor rendimiento para que realice todo el procesamiento pesado de datos presentes en el sistema, siendo principalmente las imágenes en relación a su segmentación para extraer las regiones y la clasificación de éstas para identificar la plataforma mediante la RNC.

La *Raspberry Pi* funciona de manera totalmente autónoma, de forma que tras el inicio, a los pocos segundos, se lanza el script *car_init.sh*, el cual se encarga de gestionar varios programas que se ejecutan de manera concurrente realizando sus tareas correspondientes, descritas más adelante, este script está alojado en el *crontab**,*** que es un fichero que almacena comandos para ejecutar en segundo plano, es propio de distribuciones *Linux*, ya que la *Raspberry Pi* utiliza el sistema operativo *Raspbian*.

Para el resto de este sistema, que se encarga de gestionar el script de inicio, se dispone de varios scripts programados en Python, siendo los siguientes:

- Vehiculo.py: encargado de gestionar la cámara y el sensor de proximidad *HC-SR04*, obteniendo los datos correspondientes, que guarda convenientemente para que posteriormente el ordenador principal pueda acceder a ellos de forma remota.
  También se encarga de recibir un archivo que contiene las órdenes dirigidas a los motores del vehículo, procediendo a su procesamiento para enviar dicha información a los motores.

- Archivos.py: el cual se encarga de borrar una serie de archivos basura que se crean en la fase anterior durante la ejecución del sistema, así se eliminan antes de reiniciar el sistema de nuevo.

En el ordenador principal y bajo Matlab se dispone de una serie de archivos que procesan las imágenes y la distancia estimada a su objetivo (plataforma). Concretamente se dispone del modelo de la RNC (*redTrained*) para detectar la plataforma y los archivos relacionados necesarios para el procesamiento de imágenes, explicados en la sección 5.2.

A la hora de iniciar la aplicación, se puede encender el vehículo o ejecutar el programa en Matlab en el orden que se quiera, ya que siempre esperan a que el computador de la *Raspberry Pi* esté conectado. El programa de Matlab debe ser ejecutado manualmente mientras que el prototipo solo necesita ser encendido y el proceso comienza a ejecutarse de forma automática, como se ha indicado previamente.

A) COMUNICACIÓN

Para establecer la comunicación entre procesadores se utiliza la instrucción *rosdevice* de Matlab con la que se permite la conexión a otro dispositivo de manera remota mediante el protocolo SSH (*Secure Shell*) de manera que tras una conexión exitosa se pueden enviar archivos y obtener archivos del dispositivo conectado, en este caso la *Raspberry Pi*. Para ello, es necesario introducir la dirección IP, el administrador y la contraseña según la siguiente instrucción de Matlab,

```
ipaddress = '192.168.17.128';
d = rosdevice(ipaddress,'user','password')
```

Para enviar datos desde el ordenador principal al prototipo se utiliza la instrucción *putFile* de Matlab, pasando el objeto correspondiente a la *Raspberry Pi*, objeto de origen y la carpeta de destino del prototipo,

```
putFile(mypi,'C:\Work\.profile','/home/pi/.profile');
```

A la hora de obtener los datos necesarios procedentes de los sensores, cámara y ultrasonidos, a través de la *Raspberry Pi*, esto es la imagen tomada y la distancia más próxima a un objeto se utiliza la función de Matlab *getFile*, similar a la *putFile*, si bien con el objeto origen situado en la *Raspberry Pi*,

```
getFile(mypi,'/home/pi/*.png','C:\Users\myusername\desktop',);
```

Tanto el ordenador principal como la *Raspberry Pi* están conectados a la misma red Wi-Fi.

B) FUNCIONAMIENTO BÁSICO

Al igual que se describe en el apartado anterior de comunicación, se puede iniciar tanto el programa Matlab en el computador principal o el de la *Raspberry* indistintamente.

La *Raspberry Pi* toma una imagen y la distancia del sensor ultrasónico para después crear un archivo de texto que indique al ordenador principal que es su turno.

El ordenador principal obtiene la imagen y la distancia, si la distancia al objetivo es menor que 40cm entonces se crea un archivo que indica el fin del programa y se envía a la *Raspberry* para que también finalice su ejecución, procediendo a reiniciar el sistema. En el caso de que la distancia sea mayor que el mencionado, entonces la imagen es procesada en Matlab, de forma que utilizando la RNC se puede determinar si en la imagen se encuentra el objetivo buscado. En caso de que la probabilidad de que sea el objetivo sea menor que cierto valor, es decir que el objetivo buscado no se encuentre en la imagen, se envía al vehículo, en un archivo de texto, la información relativa al movimiento que tiene que realizar próximamente, que consistirá en aplicar una rotación hasta que encuentre el objetivo buscado, en caso de que se encuentre el objetivo entonces se determina la posición de la plataforma con respecto al centro de la imagen tomada y según la diferencia calculada entre el centro de la imagen y el de la plataforma se decide si el vehículo debe girar hasta que el centro de la plataforma se sitúe en las proximidades del centro de la imagen, hecho lo cual el vehículo continúa en línea recta hacia la plataforma. Las órdenes de giro a los motores se envían a través de un archivo de texto desde el computador principal hasta la *Raspberry*.

Cuando el script *Vehículo.py* recibe la información del ordenador principal, la procesa y realiza el movimiento según la orden recibida, cuando acaba continua este mismo script, el cual se encarga nuevamente de tomar una imagen y obtener la distancia para continuar su avance hacia el objetivo. El proceso completo se describe esquemáticamente en la figura 19.

El sistema implementado en Matlab en el computador principal recibe la imagen capturada por la cámara en el vehículo y la procesa en busca de la plataforma de aproximación mostrada en la figura 1.

Una vez obtenida una imagen, el siguiente paso consiste en buscar las posibles áreas candidatas a ser plataforma para pasar las correspondientes regiones de la imagen original a la RCN, de forma que se pueda identificar la plataforma o seguir la búsqueda de la misma haciendo girar el vehículo.

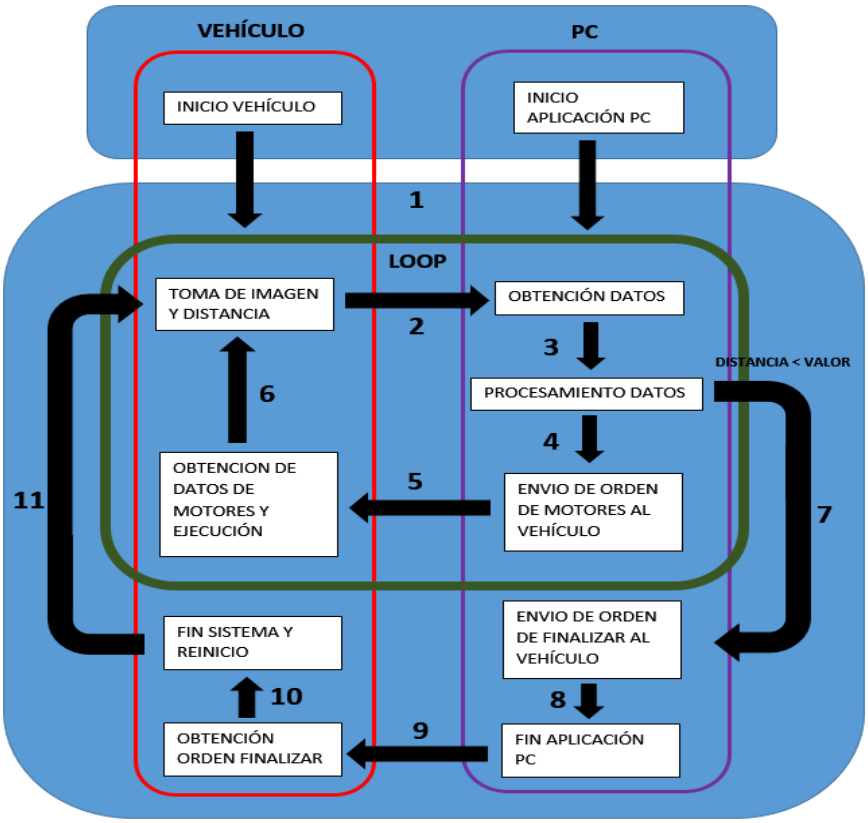


Figura 19. Procesamiento de imágenes y script principal en Matlab

También se dispone de un programa en Matlab encargado de encontrar las figuras existentes en la plataforma, las cuales consisten en un cuadrado (rojo), un triángulo (verde), un círculo (azul) y un rectángulo (naranja), que se sitúan próximos a las esquinas de la plataforma con fondo de color negro. Con ello es posible también determinar en qué zona de la imagen se sitúa la plataforma identificando las figuras existentes y su posición. También, dependiendo del número de figuras encontradas se puede obtener una probabilidad de que la región procesada sea plataforma o no, que complementa la probabilidad de detección proporcionada por la RNC, para así obtener una mayor precisión en la identificación de la plataforma.

El color de fondo de la cartulina es negro debido a que la zona de pruebas tiene las paredes de color blanco, consiguiendo así un mayor contraste y por tanto una mayor facilidad para su localización, máxime teniendo en cuenta que la calidad de las imágenes es más bien baja.

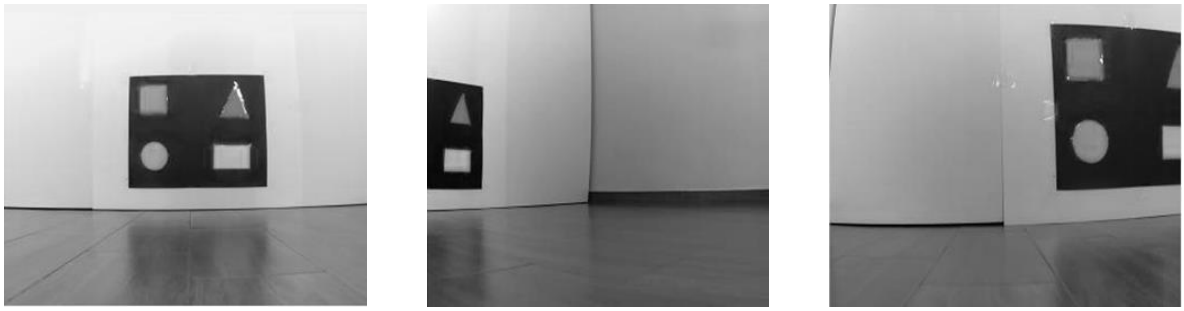
En la figura 20 se muestran tres imágenes originales ilustrativas de los ejemplos que gobiernan la explicación siguiente.

(*a*) (*b*) (*c*)

Figura 20. *Imágenes iniciales sin procesar.*

5.2 Procesamiento de la imagen y clasificación mediante la RNC

En primer lugar, tras recibir la imagen, *ImagenRGB*, ésta se transforma del modelo de color RGB al HSV para binarizar la imagen de intensidad, V, ya en escala de grises. Para ello se aplica la función *rgb2hsv(ImagenRGB)*. Como se ha indicado en la sección tres, el objetivo es identificar zonas con valores de intensidad próximos al cero (negro), buscando la base de la plataforma. Previamente, los valores de V se transforman mediante una función lineal al rango [0,1], según la figura 2. Esta operación se realiza con la función *mat2gray(V)*. En la figura 21 se muestran las imágenes de intensidad normalizadas al rango [0,1] correspondientes a las imágenes mostradas en la figura 20.



(*a*) (*b*) (*c*)

Figura 21. *Imágenes en escala de grises.*

El objetivo siguiente consiste en separar las zonas de la imagen de grises con valores de intensidad próximos a cero, dado que son los que representan la base negra de la plataforma, como se ha indicado previamente. No obstante, debido a que el negro absoluto es imposible de conseguir, lo que se hace es buscar valores lo más próximos posibles al cero, para ello se utiliza la función de umbralización *multithresh(V,7)*, basada en el método de Otsu (1979), que se aplica sobre la imagen de gris, V, para obtener 7 valores de umbral. A continuación, para cada imagen se selecciona el menor de ellos, de forma que todos los píxeles con valores por debajo del menor valor constituyen las regiones candidatas, asignándoseles el valor lógico ‘1’, que se corresponde con el blanco y al resto el ‘0’ que se corresponde con el negro. A modo de ejemplo, en la tabla 4, se muestran siete niveles de

umbral para la imagen de la izquierda en la figura 21, obteniendo como menor valor el de nivel 1, que es el utilizado para obtener la correspondiente imagen binarizada.

Niveles	1	2	3	4	5	6	7
Umbrales	0.1333	0.2627	0.3882	0.4980	0.6275	0.7255	0.8275

Tabla 4. Valores de los siete niveles de umbral

El problema que surge tras el proceso de binarización es la aparición de pequeñas regiones espurias debido a la existencia de ruidos y zonas de sombra aleatorias debido a cambios de iluminación variables, que no se corresponden con la plataforma. Para su eliminación se aplica la operación morfológica de erosión *bwmorph(Imagen,'erode',1)* con un elemento estructural de 3×3, la operación sólo se aplica una vez, para evitar que la eliminación sea muy agresiva. La figura 22(*a*) muestra una imagen binaria y en (*b*) la misma imagen tras la aplicación de la erosión, observándose la desaparición de áreas no relevantes, como se pretendía.

(*a*) (*b*)

Figura 22. (*a*) Imagen binaria sin erosionar; (*b*) la misma imagen erosionada.

En la figura 23 se muestran las tres imágenes binarizadas y erosionadas, identificadas genéricamente como *ImgBin*, a partir de las correspondientes imágenes de gris mostradas en la figura 21. Son las que constituyen el punto de partida para proceder a los pasos siguientes para la identificación de la plataforma.

(*a*) (*b*) (*c*)

Figura 23. Imágenes binarizadas y erosionadas.

El siguiente paso del proceso consiste en aplicar el método de etiquetado de componentes conexas descrito en la sección tres. Para ello se aplica la función *bwlabel(ImgBin)* sobre la correspondiente imagen *ImgBin*, donde las áreas no significativas

fueron eliminadas, como se ha indicado previamente. En la figura 24 se muestra el resultado del etiquetado sobre una imagen concreta, con las regiones numeradas con el correspondiente valor de la etiqueta de 0 a 9. Se observa cómo a la plataforma se le ha asignado la etiqueta 5.

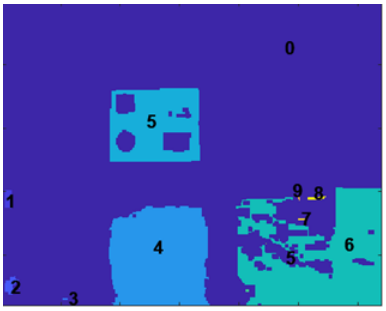


Figura 24. Regiones etiquetadas.

En las figura 25 se muestran bajo distintas tonalidades las regiones etiquetadas para las imágenes de la figura 23.

(*a*) (*b*) (*c*)

Figura 25. Regiones etiquetadas a partir de las imágenes de la figura 23.

Cada una de las regiones etiquetadas posee unas propiedades determinadas, entre las que se encuentran el área (A), expresada en píxeles y el mínimo rectángulo que contiene cada región, denominado *bounding box*, expresado como H×W, donde H es el alto y W el ancho, ambos también en píxeles. De entre todas las regiones se seleccionan como candidatas exactamente aquellas que poseen un valor A > 500. Así, con todas las áreas que superan este valor, se procede a trasladar el *bounding box* a la imagen original, procediéndose a aplicar un recorte de imagen sobre la imagen original con las mismas dimensiones H×W del *bounding box* y recuperando los tres canales R, G y B correspondientes a dicho recorte. A continuación se procede a aplicar un redimensionado del recorte para dejarlo en las dimensiones requeridas por el modelo AlexNet para las imágenes de entrada, esto es 227×227×3, tal y como se refleja en la figura 6. Este recorte de imagen redimensionado se pasa al modelo re-entrenado *redTrained*. La red devuelve los correspondientes valores de probabilidad para esa imagen de entrada en función del tipo de clase (Plataforma y Otros objetos) en la que haya sido clasificada, p_1 es la probabilidad de que la figura sea Plataforma.

Una vez superada esta fase de clasificación se procede a identificar el número, N, de figuras (cuadrado, triángulo, círculo y rectángulo) dentro de la región clasificada, de forma que en función de este número se obtiene un valor de confianza, de la forma p_2=N/4, variando de 0 a 1 cuando no se reconoce ninguna figura o las cuatro respectivamente. Finalmente se obtiene un valor de probabilidad total, $p = (p_1 + p_2)/2$. Así, dada una imagen, de entre todas las regiones clasificadas se selecciona la de máxima probabilidad p, para esa imagen dada, que se asocia con la plataforma. El hecho de seleccionar una única región de entre las posibles se debe a que en el entorno sólo existe una única plataforma.

5.3 Detección de las formas geométricas de la plataforma

Como se ha mencionado previamente, la probabilidad p_2 se calcula identificando las figuras geométricas que se supone existen en la plataforma con el fin de reforzar o penalizar la probabilidad de detección obtenida mediante la RNC. La idea subyacente es verificar o descartar una determinada región en su candidatura a ser plataforma, ya que entre otras cosas es posible que la verdadera plataforma aparezca parcialmente en la imagen (por la proximidad del vehículo o por no situarse completamente en el campo de vista), lo cual puede originar que ésta obtenga una baja probabilidad en la clasificación por la RNC, mientras puede verse reforzada por la presencia de las figuras, y viceversa, una región con alta probabilidad en la RNC puede no corresponderse con la plataforma.

El proceso se realiza mediante análisis de color, de forma que para cada uno de los colores representados en las figuras se estiman los valores promedio de los mismos a partir de una muestra significativa de píxeles representativos de las regiones. Esos valores medios de cada color se establecen para la comparación posterior.

A partir del recorte de imagen para cada región, obtenido como se ha indicado previamente, se traslada este mismo recorte sobre la imagen de intensidad, V, en el modelo de color HSV, según se ha indicado previamente. Sobre este recorte en V se vuelve a aplicar el método de Otsu (1979), con la función *multithresh*, en este caso para obtener un único valor de umbral, con el que se consigue la binarización de la imagen recortada. La figura 26 muestra tres ejemplos significativos en los que aparecen las figuras aisladas dentro de la plataforma.

Figura 26. Resultado final con la plataforma encontrada.

El siguiente paso consiste en aplicar el algoritmo de etiquetado de componentes conexas sobre los recortes binarizados, por el mismo procedimiento aplicado previamente y descrito en la sección tres, también mediante *bwlabel*. La figura 27 muestra un ejemplo de las etiquetas asignadas numeradas de 0 al 6. En la figura 28 se muestran las regiones obtenidas sobre los tres ejemplos de recortes mostrados en la figura 26.

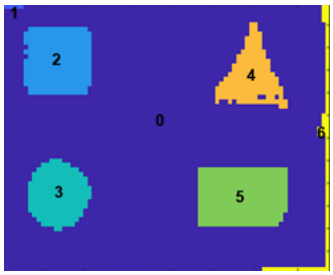


Figura 27. *Regiones del recorte correspondiente a la plataforma.*

Figura 28. *Regiones existentes en la plataforma.*

Con las regiones de los recortes etiquetadas, cada una de estas regiones se analizan según su color en el recorte RGB de la imagen original, para ello se computa el valor promedio del color de la región, comparándose con los valores medios obtenidos previamente. La comparación se establece calculando la distancia *euclidea* entre estos valores promedios y los estimados según se ha indicado previamente, de forma que valores mínimos de estas distancias, y por debajo de un determinado valor, establecido en 10 unidades, determinan la presencia de las N regiones presentes en la plataforma por su color, lo que permite finalmente obtener la probabilidad p_2 buscada. Este proceso se lleva a cabo mediante la función *EncontrarFiguras(recorte)*, que tiene como parámetro de entrada el recorte de cada región detectada sobre la imagen original RGB.

En la figura 29 se muestra gráficamente la probabilidad de clasificación de la región clasificada como plataforma de la figura 30(*a*) en la que p = 0.98 para la plataforma remarcada, mientras que p_1 = 0.96 y p_2 = 1.0, en este último caso una vez se han identificado las cuatro figuras geométricas dentro de la plataforma. Se observa en este caso, un refuerzo de la probabilidad p_1, proporcionada por la RNC que se incrementa en un 0.2, mejorando el resultado de la red. En términos generales, este es el comportamiento

habitual del proceso, mientras que también se observa, a partir de los resultados, cómo en caso contrario, también se producen penalizaciones.

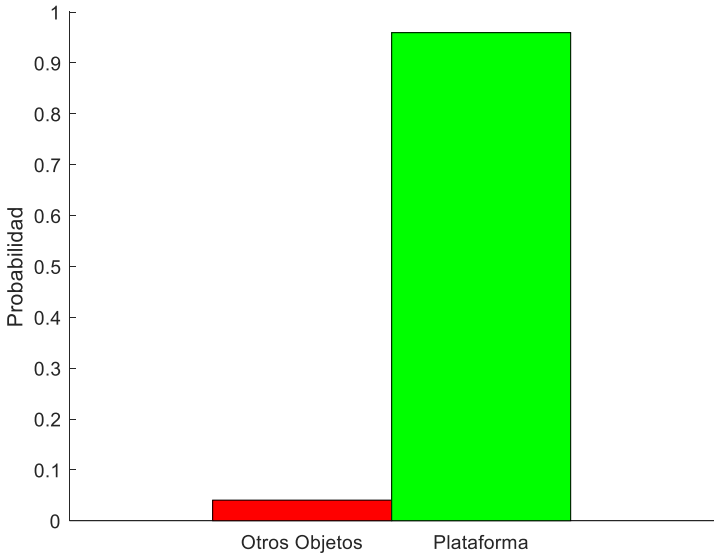


Figura 29. Probabilidad de clasificación.

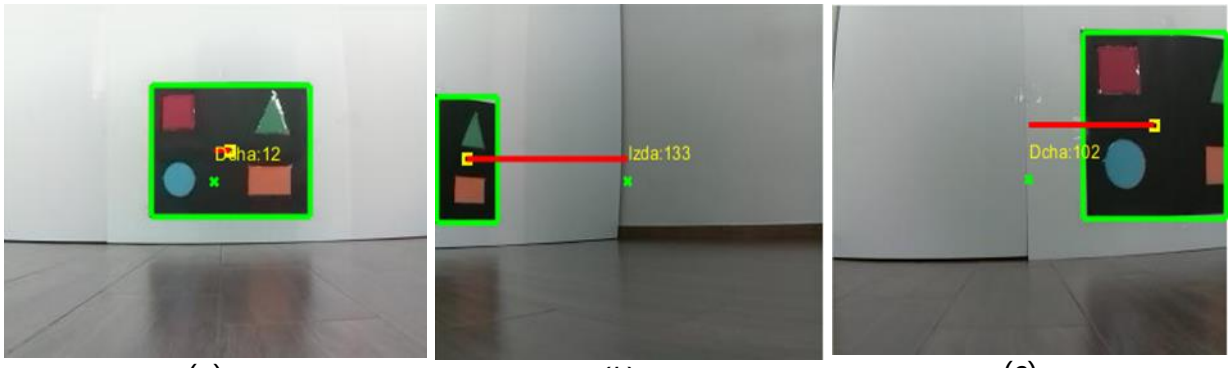
Desde el punto de vista del proceso de clasificación, se han seleccionado 120 imágenes aleatorias. Con ellas se ha determinado que en 98 se ha identificado correctamente la plataforma (Verdadero Positivo, VP), en 10 de ellas se han clasificado regiones como plataforma que no lo eran (Falso Positivo, FP). Con estos valores la precisión resulta ser Precisión = VP/(VP + FP) = 0.91. En 12 de las 120 imágenes ninguna región se ha clasificado como plataforma, que por otra parte ha sido correcto. A la vista de estos resultados, se deduce que el sistema de identificación de la plataforma obtiene resultados aceptables.

5.4 Determinación de la desviación del vehículo respecto de la imagen

Como se ha indicado previamente, en función de la máxima probabilidad, *p*, obtenida para una imagen dada, y siempre que ésta sea superior a 0.5, se determina que la región cumpliendo esta restricción es la plataforma identificada como tal. El siguiente paso es dirigir el vehículo hacia ella. Lo ideal es que la plataforma esté centrada en la imagen, de forma que el vehículo avance en línea recta hacia ella. En caso de que esta situación no se dé el objetivo consiste en determinar cuál es la desviación para reorientar el vehículo.

Para ello, se determina el centro geométrico de la región que define la plataforma, cuadro amarillo en las imágenes de la figura 30, y el centro geométrico de la imagen, cruz verde en las imágenes de la misma figura. El centro de la plataforma se obtiene calculando los valores medios del ancho y alto de su *bounding box.* La diferencia de ambos centros en la dirección horizontal determina la posición del centro de la plataforma con respecto al centro de la imagen. En la figura 30 estas diferencias están marcadas por la línea roja, expresadas

en píxeles e indicando la desviación (izquierda o derecha), de forma que en función de estas desviaciones deberá girar el vehículo para posicionar el centro de la plataforma lo más próximo posible al centro de la imagen. Estas diferencias se transforman en órdenes de giro a los motores a través de la *Raspberry Pi*.



(*a*) (*b*) (*c*)

Figura 30. Desviaciones del centro de la plataforma respecto del centro de la imagen.

5.5 Medición aproximada de distancias

Como se ha mencionado previamente, para determinar la distancia que separa el vehículo de la plataforma objetivo se utiliza el sensor de proximidad HC-SR04. Esto es posible ya que la plataforma se sitúa en una pared, e incluso aunque estuviera colocada en otro lugar sin ningún objeto solido a su alrededor tiene el tamaño suficiente como para que el sensor la pueda detectar como objeto sólido con el fin de poder medir la distancia del vehículo hasta ella. El cómputo de esta distancia por parte del sensor de proximidad es directo.

No obstante, se ha desarrollado un procedimiento alternativo, basado en el sistema de visión, para calcular la distancia del vehículo a la plataforma. Una vez identificada la plataforma y a partir del *bounding box* de la región que la define se obtienen las cuatro esquinas de este rectángulo. Si el centro de la plataforma se sitúa más a la derecha que el centro de la imagen se seleccionan los dos puntos de sus esquinas situados a la izquierda del rectángulo, obteniéndose la recta que los une, o los dos puntos de las esquinas situados a la derecha, en caso contrario. Según sea el tamaño de la recta correspondiente, y previo calibrado del sistema de visión, se determina la distancia a la que se encuentra la plataforma del vehículo. La figura 31 muestra sendos ejemplos de ubicación de las mencionadas líneas rectas según la posición de la plataforma respecto del centro de la imagen.

Durante la fase experimental del proyecto, se ha determinado que esta medición de la distancia sólo funciona con garantía hasta una distancia mínima de 1 metro de la plataforma, ya que a distancias menores la plataforma no es visible en su totalidad,

cometiendo errores graves en el cálculo de la distancia ya que se obtendrían distancias mayores al objetivo, cuando en realidad son menores, lo que aumenta el riesgo de colisiones del vehículo contra la propia plataforma.

Figura 31. Cálculo de la distancia a la plataforma según las rectas de color rojo laterales.

Aunque para medir la distancia aproximada este procedimiento plantea los problemas indicados, se trata de una técnica alternativa o complementaria con el sensor, a la vez que de interés en el caso de que haya que detectar objetos aislados de pequeño tamaño como podría ser una señal de tráfico a escala del vehículo, la cual sería difícil de detectar por el sensor ultrasónico en ciertas situaciones ya sea por tamaño o por la zona de su ubicación, que pudiera estar fuera del radio de acción de dicho sensor, como se puede ver en la sección 6, apartado 4 de la memoria, donde se tiene que medir la distancia del vehículo a la señal de Stop, pero esta es demasiado pequeña y el sensor de proximidad puede fallar a la hora de detectarla con precisión.

5.6 Errores de clasificación de la RNC y situaciones destacables

Aunque el sistema funciona aceptablemente bien en términos de precisión, como se ha indicado previamente, en algunas situaciones extremas se pueden producir fallos en la clasificación de la plataforma, que merece la pena considerar y analizar. Para ello se proporcionan tres ejemplos significativos en los que se generan objetos de cierta similitud a la plataforma original, bien modificando la imagen de esta plataforma original o generando una totalmente nueva. A continuación se describen los fallos con su correspondiente explicación.

1) <u>Ejemplo 1</u>

Las imágenes de este ejemplo se muestran en la figura 32, sobre una imagen de la plataforma real, se modifica el número de figuras interiores de la plataforma para determinar cómo puede afectar esto a la clasificación de la RNC, en la imagen (a) se han suprimido

todas las figuras de la plataforma, en la imagen (b) se suprimen las dos figuras de la izquierda (cuadrado rojo y círculo azul) y en la (c) se eliminan todas excepto el rectángulo naranja de la esquina inferior derecha.

(a) (b) (c)

Figura 32.Ejemplo eliminando figuras de la plataforma.

En la figura 32 se puede observar mediante la localización del rectángulo que encuadra la plataforma, cómo la existencia o no de las figuras geométricas en la plataforma afecta directamente a la correcta clasificación de ésta. En la imagen (a) la RNC la clasifica como no plataforma (otros objetos) mientras que las imágenes (b) y (c) sí que son clasificadas de forma correcta como plataformas ya que se encuentra en un caso similar al de la figura 33 que se muestra a continuación, en la que sólo es visible la mitad derecha de la plataforma y por lo tanto sólo se ven dos figuras, esta consideración es la misma para la imagen (c) que se asemeja al ejemplo anterior ya que sólo se puede ver la esquina inferior derecha de la plataforma por su situación relativa con respecto al vehículo.

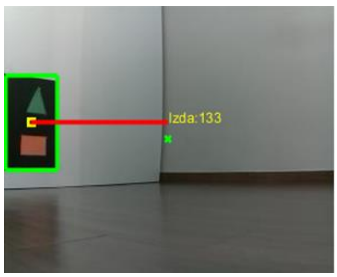


Figura 33. Ejemplo de plataforma incompleta clasificada correctamente.

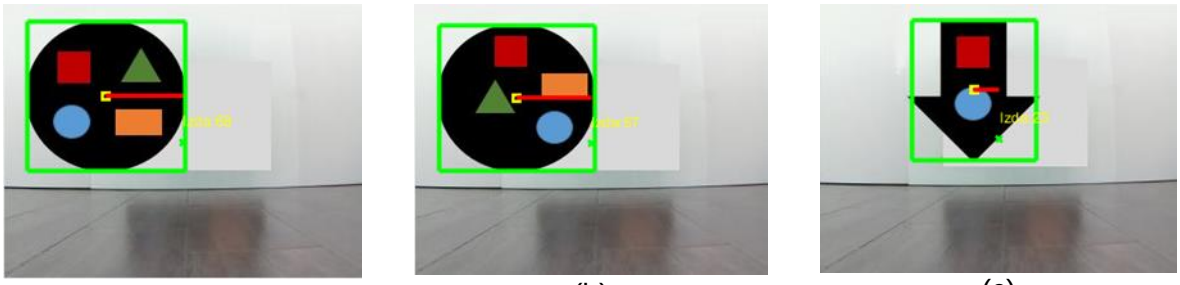
Para evitar que las imágenes (b) y (c) no fueran clasificadas como plataforma, habría que establecer algún tipo de restricción, de forma que en base a las figuras que sean visibles se establezca una dimensión máxima del tamaño de la plataforma visible, es decir, en la imagen (a) sólo se ven las dos figuras de la derecha pero la plataforma tiene un tamaño demasiado grande para contener únicamente esas dos figuras.

Hay que pensar que si se establece algún tipo de restricción del tipo citado en el párrafo anterior, la proporción de tamaño de la plataforma debe ser siempre la misma, el sistema no funcionaría correctamente si la plataforma tuviera una proporción de tamaño distinta, pudiéndose dar el caso de encontrarse con una plataforma en mal estado, en la cual las

figuras se reconocieran peor y por tanto con una pérdida de efectividad. Si bien, esto no es un error como tal y puede servir de utilidad mantener el sistema de esta forma.

2) Ejemplo 2

Las imágenes que aparecen figura 34, son varios ejemplos de plataformas modificadas con una forma distinta a la original y en el caso de la imagen (b) también tiene las figuras distribuidas en distinta posición a la original.



(a) (b) (c)

Figura 34. Ejemplo de imágenes clasificadas como plataforma con forma similar.

Tanto la imagen (a) como la imagen (b) tienen la misma forma esférica, con una forma y proporción similares a la plataforma original aunque con su forma esférica. El diseño de este ejemplo se fundamenta en gran parte a que a la hora de re-entrenar la RNC las imágenes disponibles se tienen que redimensionar a 227×227 píxeles, lo que en ocasiones conduce a que las imágenes de entrenamiento tienden a modificarse con dimensiones ancho y alto similares y por ello este tipo de figuras pueden generar falsos positivos. En la imagen (b), las figuras están dispuestas en un orden distinto al original con el fin de determinar que esto no afecta considerablemente a la clasificación, lo cual verifica la tolerancia del sistema ante la posible aparición de figuras ficticias, ello a pesar de que un número mayor de figuras no impide que la plataforma siga clasificándose como tal, a pesar de que la probabilidad p se vea penalizada por p_1. En el caso de la imagen (c) se presenta un ejemplo similar a los del *Ejemplo 1* con una forma próxima a la de la figura 33, aunque la forma de flecha inferior es diferente, tampoco difiere mucho de la original.

3) Ejemplo 3

En las imágenes mostradas en la figura 35, las diferencias de la plataforma son más acusadas que en los casos anteriores.

(a) (b)

Figura 35. Ejemplo de imágenes diferenciadas clasificadas como plataforma.

En (a) se tiene una plataforma con forma muy alargada y redondeada, de forma que como ocurre en *Ejemplo 1* y *Ejemplo 2* no se realiza ninguna comprobación de relación entre la altura y la anchura de la plataforma. En (b), aparece un modelo con dos figuras, una flecha apuntando hacia abajo y una figura esférica unida a la flecha en su parte derecha, este caso se puede solucionar re-entrenando la RNC, introduciendo en la clase '*Otros Objetos*' ejemplos similares a esta imagen.

Como conclusión general en relación a lo expuesto hasta aquí, para evitar este tipo de situaciones mostrados en los ejemplo, particularmente en el *Ejemplo 3* se podría re-entrenar la RNC añadiendo más ejemplos a la clase '*Otros Objetos*' para evitar situaciones en las que se detecten plataformas que difieran considerablemente en forma con la original, aunque, como se ha mencionado anteriormente, manteniendo el sistema como está ahora no se requeriría modificaciones sustanciales, incluso ante cambios bruscos de iluminación que pudieran afectar a los colores, y por lo tanto a la región objeto de clasificación.

En una situación real sería muy difícil que se dieran los factores necesarios para que la RNC produjera falsos positivos, ya que la plataforma tiene unas características muy diferentes a otros posibles objetos.

6.- **Conclusiones y trabajo futuro**

Se ha desarrollado un prototipo de vehículo autónomo dotado de inteligencia. El modelo de prototipo, en línea con desarrollos similares, se plantea considerando una plataforma dotada con los elementos necesarios para navegación autónoma, incluyendo ruedas, motores, alimentación, por supuesto sensores de visión (cámara) y de proximidad (ultrasonidos) y naturalmente un computador local basado en una *Raspberry Pi* que gestiona y controla las acciones del vehículo. El sistema también consta de un computador remoto, comunicado y sincronizado con el computador local donde se realizan los procesos con alta carga computacional y donde está instalada la parte inteligente del sistema.

El computador local envía datos (imágenes y distancias) al remoto para su procesamiento. Se han desarrollado técnicas de segmentación y de aprendizaje profundo para identificar y localizar una plataforma diseñada específicamente con una base rectangular negra conteniendo figuras geométricas (cuadrado, círculo, rectángulo, triángulo) de color en su interior, y que constituye el objetivo del vehículo hacia el que se dirige.

Las técnicas de procesamiento de imágenes incluyen umbralización y etiquetado de componentes conexas, que permiten recortar dichas regiones sobre la imagen original. Estos recortes son procesados por una Red Neuronal Convolucional, previamente re-entrenada a partir del modelo AlexNet, que proporciona un valor de probabilidad para

determinar si el recorte puede o no considerarse como la plataforma. Se ha diseñado un proceso posterior, basado en la identificación de las figuras geométricas de la plataforma, que refuerza o penaliza la probabilidad dada por la red, permitiendo decidir si la región en cuestión es o no finalmente la plataforma. En el caso de las distancias, proporcionadas por el sensor de ultrasonidos, simplemente se trata de verificar la proximidad del vehículo a un objeto o a la plataforma, para confirmar la llegada al objetivo. Esta información se transmite a la *Raspberry Pi*, que convierte en acciones sobre el vehículo para girar, avanzar y continuar o no con el envío de más datos desde el computador local.

Los experimentos realizados han verificado la validez de la propuesta, destacando los aspectos relacionados con el sistema inteligente y la navegación autónoma del vehículo autómata.

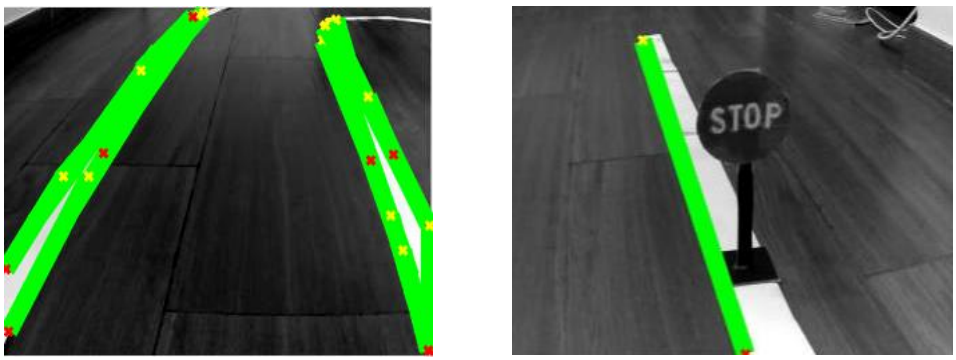
Como propuestas de futuro se han identificado las siguientes:

1) Experimentar con otros modelos de redes diferentes a AlexNet, entre las que destacan: VGG-16 y VGG-19 siguiendo las pautas establecidas en Russakovsky y col., (2015) y Simonyan y Zisserman (2014), así como con GoogleLeNet (Szegedy y col., 2014) o ResNet (He y col., 2016).
2) Utilizar técnicas de segmentación semántica para identificar los colores de la plataforma.
3) La migración del sistema para implementarlo con otra tecnología en python y OpenCV para aumentar la velocidad de proceso, con respecto a Matlab, permitiendo así procesar video en tiempo real. De esta forma se evita que el vehículo tenga que detenerse para tomar una imagen, procesarla y volver a moverse.
4) Desarrollar técnicas de seguimiento de líneas en el suelo, simulando las líneas que definen las carreteras y añadiendo reconocimiento de señales de tráfico. Esta cuarta propuesta ha sido implementada parcialmente, sin haberse concluido satisfactoriamente por falta de tiempo. Para el seguimiento de las líneas del suelo se utilizan unas franjas blancas de papel que simulan en todo caso las líneas de la carretera, tal y como muestra la figura 36, en ella se pueden observar dos tipos de imágenes, en la imagen (a) aparece una larga recta con una curva a la derecha y en la imagen (b) se muestra la franja blanca del carril derecho de la carretera y además de una señal de stop. Las franjas delimitadoras de la carretera resaltan mucho con el suelo oscuro permitiendo una mejor binarización y algún posible objeto de color muy claro que en procesos posteriores se pueden eliminar o descartar por su forma y tamaño.

(a) (b)

Figura 36. Imágenes de la carretera y una señal de stop.

Ahora en la figura 37 podemos ver el resultado después de realizar lo anteriormente dicho y aplicar la transformada de Hough (Gonzalez y col., 2004) con la que se obtienen las rectas que aparecen en la imagen para posteriormente guiar el vehículo según la diferencia entre las pendientes de las ecuaciones de las rectas obtenidas de las franjas izquierda y derecha de la carretera con la de la recta que pasa por el centro de la imagen. Se puede observar cómo en la imagen (a) se obtienen varias rectas diferentes en las que las marcas en forma de aspas rojas son el inicio y las amarillas el fin de las rectas. En (b) se obtiene solamente una recta porque detecta menos imperfecciones entre la línea y el suelo, ya que se sitúa más próxima a la cámara y además se trata de un tramo más corto y con menos imperfecciones. Este ejemplo sirve de base para el inicio de las mejoras previstas en este sentido.



(a) (b)

Figura 37. Detección de líneas mediante la transformada de Hough.

Una vez se detectan las líneas de la carretera, el siguiente paso consistiría en procesar la imagen con técnicas de umbralización y etiquetado de regiones similares a las aplicadas para la detección de la plataforma, para ello se tendría que re-entrenar de nuevo la RNC creando una nueva clase con los objetos a detectar, en este caso, la señal de Stop.

En esta propuesta también resulta interesante utilizar las metodologías mencionadas en la *sección 5.5* del proyecto para medir la distancia a la que se sitúa la señal de Stop en este caso, que por su altura, pequeño tamaño y el ángulo que forma con respecto al sensor *HC-*

SR04 es necesario calcular para detener el vehículo cuando éste se sitúe en las proximidades de la señal y en la posición correcta.

7.- **Conclusions and future work**

A prototype of an autonomous vehicle equipped with intelligence has been developed. The prototype model, in line with similar developments, is proposed considering a platform equipped with the mandatory elements for autonomous navigation, including wheels, motors, power supply, of course vision (camera) and proximity (ultrasound) sensors, and obviously a local computer based on a Raspberry Pi that handles and controls actions in the vehicle. The system also consists of a remote computer, connected and synchronized with the local computer where processes with high computational load are carried out and where the intelligent part of the system is installed.

The local computer sends data (images and distances) to the remote one for processing. Segmentation and deep learning techniques have been developed to identify and locate a specifically designed platform with a black rectangular base containing colored geometric figures (square, circle, rectangle and triangle) inside, which constitutes the target of the vehicle towards it is guided.

Image processing techniques include thresholding and labelling of connected components, which allow these regions to be clipped onto the original image. These clippings are processed by a Convolutional Neural Network, previously re-trained from the AlexNet model, which provides a probability value to determine whether or not the clipping can be considered as the platform. A subsequent process has been designed, based on the identification of the geometric figures of the platform, which reinforces or penalizes the probability given by the network, allowing deciding whether or not the region in question is finally identified as the platform. In the case of distances, provided by the ultrasound sensor, it is simply a matter of verifying the proximity of the vehicle to an object or to the platform, to confirm arrival at the target. This information is transmitted to the Raspberry Pi, which converts actions on the vehicle to turn, advance and continue or not sending more data from the local computer.

The experiments carried out have verified the validity of the proposal, highlighting aspects related to the intelligent system and the autonomous navigation of the automated vehicle.

The following have been identified as future proposals:

1. Experiment with other network models other than AlexNet, among which stand out: VGG-16 and VGG-19 following the guidelines established in Russakovsky et al. (2015) and Simonyan and Zisserman (2014), as well as with GoogleLeNet (Szegedy et al., 2014) or ResNet (He et al., 2016).
2. Use semantic segmentation techniques to identify the colors of the platform.
3. The migration of the system for its implementation under other technologies, such as python and OpenCV to increase the speed of the system, because the processing of images with OpenCV is faster than with Matlab and instead of processing images we would proceed to process video in real time, to avoid the vehicle stops to capture an image, process it and move again.
4. Develop ground tracking techniques on the ground, simulating the lines defining borders on roads and also for traffic signs recognition. This proposal was partially implemented but could not be completed before the final delivery deadline. To follow the lines on the ground, white strips of paper are used, as one can see in figure 36(a) a long straight line with a curve to the right is displayed and in (b) the white strip of the right lane of the highway and a stop sign. The delimiting stripes of the road stand out a lot with the dark ground, so the image is binarized, leaving only the lines of the road visible and some possible very light-colored object that would later be ruled out due to its shape and size. Figure 37 displays the result after applying the Hough transform (Gonzalez et al., 2004) with which the lines are obtained in order to guide the vehicle according to the difference between the slopes of the lines obtained from the left and right strips of the road with the line that passes through the center of the image. In image (a) one can see that several different lines are obtained in which the blades red and yellow determine start and end points respectively. In (b) only one line is obtained because fewer imperfections are detected on the road, as it is closer to the camera and with a shorter section. As mentioned before, this is a basic example that would have to be changed and significantly improved.

Once the road lines are detected, the image is to be processed by applying similar techniques (segmentation and labelling) to those used for detecting the platform. To do that, the CNN must be re-trained by creating a new class with appropriate images; here, the stop sign.

In this proposal, it is interesting to use the methodologies mentioned in *section 5.5* in order to measure the distance between the vehicle and the stop sign for determining its location, considering its height, small size and the angle it makes with respect to the sensor *HC-SR04* sensor to stop the vehicle when it is correctly positioned with respect the stop sign.

9.- **Bibliografía**

1. Autopía (2020). Conducción conectada y automatizada de vehículos. Disponible on-line: https://autopia.car.upm-csic.es/ (Accedido Febrero 2020).

2. CAV (2020). Integración de sistemas cooperativos para vehículos autónomos en tráfico compartido. Disponible on-line: http://insia-upm.es/portfolio-items/proyecto-cav/. (Accedido Febrero 2020).

3. Citilogs (2020). Smart Cities: Optimización de tráfico. Disponible on-line: http://www.citilog.com/es-cities. (Accedido Febrero 2020).

4. Daugman, J.G. (1985). Uncertainty relations for resolution in space, spatial frequency, and orientation optimized by two-dimensional visual cortical filters. Journal of the Optical Society of America A, 2, 1160-1169.

5. Gonzalez, R.C., Woods, R.E., Eddins, S.L. (2004). Digital Image Processing with Matlab. Prentice Hall, Upper Saddle River, NJ, USA.

6. Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep learning. MIT Press. USA. Disponible on-line: https://www.deeplearningbook.org/ (accedido Marzo 2020).

7. HackerCar (2020). ¿Qué es un coche inteligente- Smart-Car?. Disponible on-line: hhttps://hackercar.com/faq/coche-inteligente-smart-car/ (Accedido Febrero 2020).
8. Haralick, R. M., Shapiro, L.G. (1992). Computer and Robot Vision, Volume I, Addison-Wesley, 1992, pp. 28-48.

9. He, K., Zhang, X., Ren, S., Sun, J. (2016). Deep residual learning for image recognition. In Proc. of the IEEE conference on computer vision and pattern recognition, pp. 770-778.

10. Horsey, J. (2016). Raspberry Pi Self Driving Car (video). Disponible on-line: https://www.geeky-gadgets.com/raspberry-pi-self-driving-car-29-01-2016/ (Accedido Octubre 2019).
11. ImageNet (2019). Disponible on-line: http://www.image-net.org (Accedido Octubre 2019).
12. Krizhevsky, A., Sutskever, I., Hinton, G.E. (2012) ImageNet Classification with Deep Convolutional Neural Networks. In Proc. 25th Int. Conf. on Neural Information Processing Systems (NIPS'12), vol. 1, pp. 1097-1105.
13. Make (2020). Build an Autonomous R/C Car with Raspberry Pi. Disponible on-line: https://makezine.com/projects/build-autonomous-rc-car-raspberry-pi/ (Accedido Octubre 2019).
14. Matlab (2020). MATLAB para inteligencia artificial. Disponible on-line: https://es.mathworks.com/. (Accedido Octubre 2019).
15. OpenCV (2020). Open Source Computer Vision. Disponble on-line: https://opencv.org/ (Accedido Febrero, 2020).
16. Otsu, N., (1979). A Threshold Selection Method from Gray-Level Histograms, IEEE Trans. Systems, Man, and Cybernetics, 9(1), 62-66.

17. Pajares, G., de la Cruz, J.M. (2007). Visión por Computador: imágenes digitales y aplicaciones. RA-MA, Madrid.

18. Raspberry Pi (2020). Disponble on-line: https://www.raspberrypi.org/ (Accedido Febrero, 2020).

19. Rojas, R. (2020). Autonomous Cars (2006-2015). Disponible on-line: http://dcis.inf.fu-berlin.de/rojas/autonomous-cars-2006-2015/ (Accedido Febrero 2020).

20. Russakovsky, O., Deng, J., Su, H. Krause, J., Satheesh, S., Ma, S. Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV). 115(3), 211–252.

21. Self-Driving (2020) Build a Self-Driving RC Car using Raspberry Pi and Machine Learning using Google Colab. Disponible on-line: https://techwithsach.com/build-a-self-driving-rc-car-using-raspberry-pi-and-machine-learning-using-google-colab/ (Accedido Marzo, 2020).

22. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A. (2014). Going Deeper with Convolutions. Computing Research Repository. arXiv:1409.4842 [cs.CV].

23. Simonyan, K., Zisserman, A. (2014). Two-stream convolutional networks for action recognition in videos. In Advances in Neural Information Processing Systems, pages 568–576.

24. Tecnalia (2020). Future Smart Cities. Disponible on-line: https://www.tecnalia.com/images/stories/Catalogos/catalogo_AP-FutureCities-SmartCities-ES.pdf. (Accedido Febrero, 2020).

25. TensorFlow (2020). An end-to-end open source machine learning platform. Disponble on-line: https://www.tensorflow.org/ (Accedido Febrero, 2020).

26. Tian, D. (2019). DeepPiCar- Part 1: How to Build a Deep Learning, Self Driving Robotic Car on a Shoestring Budget. Disponible on-line: https://towardsdatascience.com/deeppicar-part-1-102e03c83f2c (Accedido Febrero 2020).

27. Upton, L. (2016). Self-driving car. Disponible on-line: https://www.raspberrypi.org/blog/self-driving-car/ (Accedido Febrero 2020).

ANEXO

1.- Repositorio de código y video de funcionamiento

En los dos siguientes enlaces tenemos acceso al repositorio de Google Drive donde se encuentra alojado el código y un enlace directo al video de funcionamiento del sistema que se encuentra en YouTube.

- Enlace de Google Drive
  https://drive.google.com/drive/folders/11jI_SWmwLfnr5t3UH88vGWlNd3xDFaqh?usp=sharing
- Video de funcionamiento
  https://www.youtube.com/watch?v=qhVf_37l-I4

2.- Manual de usuario

A continuación se proporcionan las instrucciones necesarias, estructuradas en dos pasos, a modo de manual de usuario para ejecutar todas las funcionalidades del sistema al completo:

A) PASO 1 (vehículo)

1. Conectar el cable micro USB de la *powerbank* a la *Raspberry Pi*, tras lo cual el vehículo se iniciará automáticamente.
2. Conectar una pantalla al puerto HDMI de la *Raspberry Pi* y un teclado y ratón USB.
3. Una vez estamos en la pantalla de inicio de la *Raspberry Pi*, buscamos en la esquina superior derecha el símbolo de red Wi-Fi y se pulsa sobre él, aparecerán las redes Wi-Fi disponibles, nos conectamos a la misma red que nuestro ordenador principal.
4. Hay que conseguir la dirección IP de la *Raspberry Pi* por lo que abrimos la línea de comandos y escribimos *ifconfig* y nos guardamos la dirección IP que está en wlan0 que corresponde a la red Wi-Fi, como se ve en el recuadro rojo de la figura A.1.
5. Por ultimo reiniciamos la Raspberry Pi, pudiendo desconectar todos los periféricos.

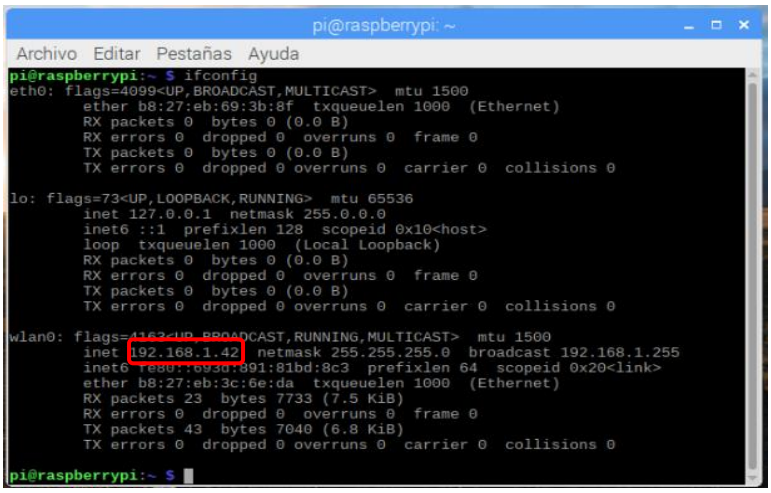


Figura A.1. Ejemplo para obtener dirección IP de la *Raspberry Pi.*

Nota: este proceso hay que realizarlo sólo cuando cambiamos de lugar de trabajo y por lo tanto de red, es la configuración inicial. Una vez realizado este proceso, para el resto de las veces sólo hay que conectar el cable micro USB a la *Raspberry Pi*.

B) PASO 2 (ordenador principal)

Una vez completado el Paso 1, sólo hay que iniciar Matlab, buscar la carpeta del proyecto y lanzar el script *CNNClasificacionPlataforma.m*, si bien cambiando *ipaddress* por la IP de la *Raspberry Pi*, indicada en el Paso 1 como se indica a continuación.

```
ipaddress = '192.168.17.128';
```

```
d = rosdevice(ipaddress,'user','password')
```

Por ultimo sólo es necesario ejecutar *CNNClasificacionPlataforma.m* y el vehículo comenzará a moverse inmediatamente.

Cuando se alcance la plataforma o se termine el programa antes de tiempo, no hay que realizar ningún proceso en el vehículo ya que se empezaría a ejecutar todo nuevamente por lo que la única acción a realizar consiste en ejecutar de nuevo *CNNClasificacionPlataforma.m* las veces necesarias.

Importante: con la configuración inicial del vehículo (Paso 1), si queremos entrar en el sistema de la *Raspberry Pi*, se puede hacer sin tener que usar todos los periféricos del Paso 1. En este caso, Windows 10 tiene una aplicación llamada *conexión a escritorio remoto*, que como su nombre indica, permite la conexión de forma remota a otro ordenador conocida su dirección IP, usuario y contraseña. Para ello se introduce la dirección IP obtenida en el Paso 1, y aparecerá la pantalla que se muestra en la figura A.2, que solicita usuario y contraseña, pudiéndose encontrar éstas en el script *CNNClasificacionPlataforma.m*, justo en la instrucción *rosdevice* que del Paso 2. Se trata de una herramienta de gran utilidad ya que no se necesita ni la pantalla, ni el ratón ni el teclado.

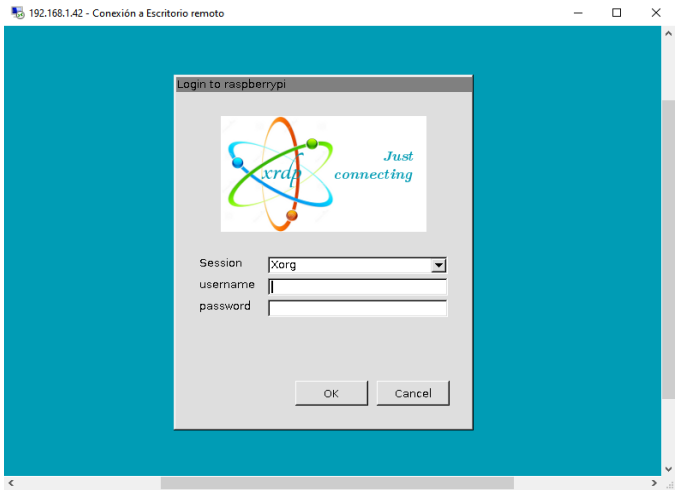


Figura A.2 Pantalla para acceder a la *Raspberry Pi* de forma remota.