Universidad Zaragoza
1542

Trabajo Fin de Grado

Grado Universitario en Ingeniería Electrónica y Automática.

Navegación de un robot móvil en entorno urbano basado en visión por computador y odometría

Navigation of a mobile robot in an urban environment based on computer vision and odometry

Autor

Hugo Salbidea Presa

Directora

Rosario Aragüés Muñoz

Universidad de Zaragoza Escuela de Ingeniería y Arquitectura
Departamento de Informática e Ingeniería de Sistemas2021

Según The New York Times, solo 10.000 personas en el mundo "cuentan con la educación, la experiencia y el talento necesarios para construir los complejos y a veces misteriosos algoritmos matemáticos que dirijan esta nueva raza de inteligencia artificial"

Navegación de un robot móvil en entorno urbano basado en visión por computador y odometría

Resumen

En la robótica actual, que hace ya varios años dejó de ser exclusiva para la industria y se instaló en acciones cotidianas, existe una tendencia creciente de usar inteligencias artificiales para realizar acciones en las que antes se necesitaba el desarrollo de un software muy complejo y de gran coste computacional. Estas inteligencias favorecen al usuario, realizan un trabajo eficiente, concreto y fiable sobre todo en robots móviles. Un claro ejemplo de los dos factores mencionados es el fabricante de robots iRobot, popular por su aspirador Roomba el cual ha acabado recurriendo a la inteligencia artificial en sus modelos de última generación.

Es por esto que en este proyecto se abordan aspectos fundamentales de la inteligencia artificial como son las redes neuronales en un robot móvil para modelar sus movimientos en función del entorno exterior. Más concretamente el objetivo del proyecto se centra en la conducción autónoma de un vehículo, simulando el vehículo con un una plataforma controlada por una Raspberry Pi y construyendo un entorno urbano real a escala para hacer circular el robot a través de él. Para ello se solicita al usuario un destino final, se calcula la ruta a seguir eligiendo la que menos giros implique y se sigue dicha ruta. Por el camino el robot va interactuando con las señales de tráfico del entorno urbano y recalculando la ruta si alguna señal restringiera el movimiento planeado.

A lo largo de esta memoria se detalla en profundidad cada una de las partes más fundamentales del movimiento del robot porque el aprendizaje automático juega un papel fundamental pero no ocupa toda la base del trabajo. La localización del robot mientras se mueve por el entorno urbano, representada por sus parámetros odométricos, está basada en los encoders situados en cada una de las ruedas de tracción del robot. Es en el reconocimiento de elementos que forman parte del entorno urbano donde se usan las redes neuronales, más concretamente, se usan para clasificar señales de tráfico captadas por la cámara del robot.

Por último, además de simular el comportamiento de un vehículo con movilidad autónoma, se analizan y se comparan las prestaciones del experimento real frente a las prestaciones teóricas obtenidas en simulación.

Agradecimientos

A mis padres Koldo y Esther y a mi hermano Diego, por darme la oportunidad de llegar hasta aquí y ser siempre mi máximo apoyo, un apoyo incondicional hasta cuando no lo merecía. Ni en diez vidas podría devolveros todo lo que habéis hecho por mí.

A Nils, Joaquín y Nicolás por ser más que unos amigos de clase, capaces de ser los mejores profesores particulares. Os llevo conmigo para toda la vida.

A Luis, por su apoyo incondicional y hacerme reír en los peores momentos.

A Alba, por confiar siempre en mí y estar ahí antes de que te lo pidiera, todo es más fácil si tienes a alguien así a tu lado.

A mi familia de la Residencia Goya que juntos hemos descubierto Zaragoza y me han abierto las puertas de sus casas siempre.

A mis amigos de toda la vida y para toda la vida, un placer hacer este viaje junto a vosotros.

Agradecer, por supuesto, a mi tutora, Rosario Aragüés, magnífica profesora. Siempre buscando un hueco para mí, siempre comprensiva y siempre proporcionándome lo que necesitaba.

A todos ellos y más gente, gracias por hacerme ser lo que soy, estoy orgulloso de mí mismo pero más de todos vosotros.

Índice

1. Introducción

1.1. Motivación y contexto

En la sociedad actual el incesante progreso de las tareas robotizadas que facilitan nuestra rutina diaria ha provocado que lo más corriente sea encontrar robots realizando labores que en otros tiempos llevaban a cabo personas. Este incremento en la aparición de robots que ayudan y facilitan en nuestra vida cotidiana ha traído una nueva forma en el ámbito de estudio y desarrollo de la robótica. Hace unos pocos años la robótica se centraba prácticamente en su totalidad en el sector industrial, y, sin embargo hoy en día se pueden encontrar robots diseñados para realizar tareas más allá del alcance de la robótica industrial. Quizás la característica más interesante dentro del panorama actual sea el movimiento autónomo de los robots cuyo desarrollo ha aumentado considerablemente en la última década. Sin ir más lejos, se puede utilizar como ejemplo las imágenes de la Fig. 1 que ilustran como la característica que mencionábamos anteriormente ha propulsado la comercialización de robots móviles para campos de acción con características muy dispares.

(a) iRobot Roomba

(b) Omron LD Mobile Robot

(c) iRobot Warrior

(d) Rover ExoMars

Figura 1. Robots con movilidad autónoma: (a) Tareas domésticas. (b) Logística de almacén. (c) Sector militar y de defensa. (d) Exploración de otros planetas.

Se puede notar la aparición de proyectos cada vez más ambiciosos basados en la movilidad de robots. Por lo tanto es muy fácil figurarse un escenario cotidiano donde robots móviles y personas interactúen entre ellos en espacios donde dicha movilidad autónoma suponga una facilidad para llevar a cabo una tarea (conducir, limpiar una casa o trabajos de investigación). No basta con resaltar el hecho de facilitar una acción, sino que también se produce un incremento en la eficiencia y una reducción en el tiempo de ejecución de esta.

Se ha mencionado en el párrafo anterior una interacción entre robot móvil y persona, siendo esto un escenario imaginable en un futuro muy cercano. Entonces, se ha visto motivada la idea de trabajar con un robot móvil que interactúe con un ser humano y su

entorno para este trabajo. Con el contenido de esta memoria se consigue el control y la generación de rutas para un robot móvil y la construcción de un entorno urbano donde se moverá el robot.

1.2. Objetivos

Por todo lo mencionado anteriormente se ha querido desarrollar un sistema de navegación autónoma que sea capaz de identificar señales de tráfico y tomar una decisión en base a las mismas. Por lo tanto el objetivo principal del proyecto sería la correcta circulación del robot en un entorno urbano conociendo en todo momento sus parámetros odométricos. Por parámetros odométricos entendemos la localización en plano 2D con sus coordenadas *x* e *y*, y, además su orientación respecto al origen.

Los objetivos principales de este proyecto son el desarrollo de un sistema capaz de mantener el robot localizado en el mapa en tiempo real, un sistema de planificación y re planificación de rutas y por último el entrenamiento de una red neuronal ligera pero eficiente capaz de reconocer señales de tráfico a través de una cámara incluida en la Raspberry. Además, se hará un pequeño estudio de la materia relacionada con los robots móviles y se desarrollará un sistema de interacción del robot con el usuario.

Para el satisfactorio desarrollo del proyecto se ha construido un entorno urbano a escala para la experimentación y simulación de la conducción autónoma del robot. Dicha experimentación pretende ajustarse, en la medida de lo posible, a un entorno lo más parecido a la realidad, contemplando así un amplio abanico de posibilidades tanto en los requerimientos de usuario como con el vehículo en conducción. Los experimentos aportarán suficientes datos para realizar un análisis y una comparación posteriores en función de las distintas velocidades del vehículo, los diferentes tipos de redes neuronales y la complejidad de la ruta a seguir así como las modificaciones de dicha ruta que irán surgiendo a medida que se recorre el mapa.

Cabe resaltar también que un objetivo de este proyecto es lograr una conducción autónoma eficiente y de bajo coste computacional y económico, factor que puede añadir alguna problemática que se tendrá en cuenta en la realización del software y en la elaboración de la memoria.

1.3. Alcance

Dentro de este proyecto se ha llevado a cabo el desarrollo de programas informáticos que permiten el movimiento del robot tanto en línea recta hacia adelante y hacia atrás como en curvas de noventa grados a izquierda y derecha. Estos movimientos van a construir la base del proyecto que como ya se ha mencionado es el movimiento autónomo de un vehículo. Una vez funcionando estos programas, se incorporó al robot una cámara para gestionar la toma de decisiones en base a las imágenes recibidas por la cámara después de ser sometidas a un método de procesamiento y predicción.

Por lo tanto el alcance real del trabajo es conseguir llevar a cabo un sistema autónomo de conducción que interactúe con el exterior y satisfaga los requerimientos del usuario.

1.4. Entorno de trabajo y herramientas utilizadas

El trabajo desarrollado se compone de un medio físico que simulará el entorno urbano y varias herramientas software para el control del robot.

- **Robot diferencial modelo Pi Robot**.

    1. Raspberry Pi 2 Model B V1.1: Es la placa de un ordenador simple compuesto por un SoC, CPU, memoria RAM, cuatro puertos de entrada y salida para periféricos de bajo nivel, salida de audio, puerto HDMI, ranura SD para almacenamiento, reloj, conexión para cámara y para pantalla, cuarenta pines de entrada/salida y un puerto micro USB para la alimentación. En este proyecto se usan los cuatro puertos USB para el ratón, el teclado, un adaptador de red Wi-Pi que permitirá a la raspberry conectarse a la red local y el último puerto se usa como salida para alimentar los motores a través de un mini-driver que elevará la tensión.

    2. Motores marca "Dagu": Son pequeños motores de 5V, que en Python pueden recibir una entrada comprendida entre 0-255, siendo 255 el equivalente a 5V. Estos motores tienen una relación de transformación de 48:1 y son controlados por la Raspberry Pi. Ésta es la encargada de mandarle las acciones que son entradas escalón.

    3. Ruedas: El robot viene con tres ruedas, dos de las cuales son de 3,15 centímetros y van una a cada lado del robot. La otra es una bola loca situada en la parte delantera que sirve como apoyo y como pivote, está basada en una rótula.

    4. Encoder: Se trata de dos sensores fotoeléctricos (uno para cada rueda) los cuales funcionan según la cantidad de luz que llega al receptor. El haz de luz se enfoca a un disco con 20 ranuras por lo tanto si el recptor recibe la señal del emisor (que está siempre dando señal) significa que la luz está pasando a través de una ranura, y, por lo tanto, si el receptor no recibe señal, el haz de luz está chocando contra el disco. Por lo tanto medimos el avance cada vez que el receptor cambie de estado, la distancia entre cada ranura equivale a unos 0,9895 centímetros.

    5. Pi Camera: Es un sensor de imagen diseñado a medida para la placa Raspberry con una lente de enfoque fijo.

    6. Adaptador de red Wi-Pi: Es el encargado de permitir la conexión de la raspberry a una red wifi ya que el modelo que estamos usando no está habilitado para esta función.

    7. Batería: Se cuenta con una power bank para alimentar el robot durante todo su escenario de movilidad. La fuente tiene una capacidad de 10.400 mAh/38,4 Wh con entrada y salida de 5V/2A de continua.

- **Entorno físico:** Para la experimentación se ha creado, en la medida de lo posible, un escenario que intenta simular lo que sería un entorno urbano, teniendo en cuenta las principales señales que influyen en el comportamiento del robot como por ejemplo las que prohíben realizar un giro o las que obligan a ir en un sentido. No se han contemplado señales como las de STOP o ceda el paso ya que son señales que no influyen en la dirección, sino en un tiempo de espera.

- **Software:** Para la generación del código usaremos Python. Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código. Se trata de un lenguaje de programación multiparadigma, ya que soporta parcialmente

la orientación a objetos, programación imperativa y, en menor medida, programación funcional [33]. Para este trabajo hemos utilizado la versión 3.7.0 que es la que nos permite incorporar la biblioteca Keras para implementar el uso de redes neuronales en la raspberry. Como módulos necesarios se han importado para su uso: numpy, keras, math, opencv, networkx y tensorflow [26].

1.5. Estructura de la memoria

Esta memoria consta de varias partes bien diferenciadas y estructuradas.

El primer capítulo (y en el que nos encontramos ahora mismo) es una introducción del proyecto que se va a realizar, detallando el por qué de la elección de este contenido, las herramientas utilizadas (tanto software como hardware) y los objetivos que se pretenden lograr.

En el capítulo 2 se explica cómo se ha logrado localizar el robot a lo largo de todo el escenario y qué parámetros han sido necesarios calcular. Parte de la información relativa al robot se incluye en el apéndice A, tanto el funcionamiento de los robots móviles como el de los robots diferenciales. El funcionamiento y aplicación de las redes neuronales, así como las diferentes redes probadas y sus prestaciones se profundiza durante el capítulo 3. Después se destina el capítulo 4 a detallar el escenario construido, qué dimensiones tiene y en qué arquitectura se ha basado. En el capítulo 5 se explica sin profundizar, la implementación algorítmica que se puede ver con más detalle en el apéndice B. El capítulo 6 está dedicado únicamente al análisis de resultados.

Por último se dedica el capítulo 7 a las conclusiones, posibles aplicaciones de esta tecnología creada y futuras mejoras e implementaciones que tienen cabida en el proyecto. Y el capítulo 8 para las observaciones o datos de interés derivados de la elaboración de este proyecto.

Como ya se ha mencionado, antes de citar las fuentes y referencias, se incluyen dos apéndices donde se explica con más detalle los temas abordados en los capítulos 2 y 5.

2. Tecnología móvil del robot

El proceso más básico para la navegación de un robot móvil se basa en el sistema de propulsión. Este sistema es el que permite al robot moverse dentro de un determinado entorno. Uno de los sistemas más usuales se basa en el uso de ruedas de tracción diferencial, el cual es un sistema poco complejo y adecuado para la navegación en algunos entornos de desarrollo típicos de actividades humanas. En este caso en particular, se va a experimentar la conducción autónoma con un robot de tracción diferencial.

Como ya se ha comentado en la introducción de esta memoria, el proyecto se basa en la movilidad de un robot de manera autónoma. Por ello es importante analizar el tipo de robot móvil con el cual se va a desarrollar el trabajo y sus opciones de diseño pero para una lectura y comprensión más llevadera, dichos temas se explican en el apéndice A.

2.1. Localización del robot

Como en todo sistema robótico, la localización de los robots es fundamental a la hora de poder realizar un control fiable de éste. Por ello se pretende desarrollar un modelo matemático basado en la información recogida por los encoders para localizar al robot dentro del campo de actuación. Para ello es necesario recordar que cualquier punto ubicado en espacio está perfectamente localizado mediante seis variables puesto que goza de seis grados de libertad (tres desplazamientos y tres giros) como se ilustra en la Fig. 2. Como este es un proyecto que simula el comportamiento de un vehículo tres de esos grados de libertad están restringidos físicamente puesto que no hay movimiento en el eje z y sólo hay giro entorno e ese mismo eje.

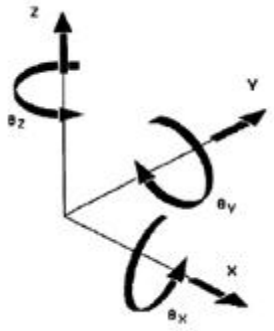


Figura 2. Orientación y sentido de giro de los ejes

Por lo tanto para conocer la localización completa del robot necesitamos tres variables: el movimiento en *x*, el movimiento en *y* y el giro en torno al *eje z*. Vamos a analizar el movimiento independiente de cada rueda y las ecuaciones que lo modelan. Como la actualización de la localización es un proceso que se estará ejecutando en paralelo al programa principal se realizará de forma periódica con un período de muestreo lo suficientemente pequeño como para no perder ningún cambio de estado en la variable asociada al encoder. Esto significa que se puede realizar una muy buena estimación de la localización sumando el incremento de la variable al valor acumulado. Las ecuaciones que lo modelan son:

$$\begin{cases} x_k = x_{k-1} + \Delta x_k \\ y_k = y_{k-1} + \Delta y_k \\ \theta_k = \theta_{k-1} + \theta y_k \end{cases}$$

Siendo x_k, y_k y θ_k los valores de esas variables en el instante *k* y x_{k-1}, y_{k-1} y θ_{k-1} los valores en el instante justo anterior. Por lo tanto, se podrá conocer, por ejemplo el valor de la variable *x* si se conoce el valor de esa variable justo en el instante anterior y se conoce el incremento que ha sufrido esa variable. Por lo tanto, la siguiente cuestión a resolver es cómo calcular el incremento de una variable en un período de muestreo.

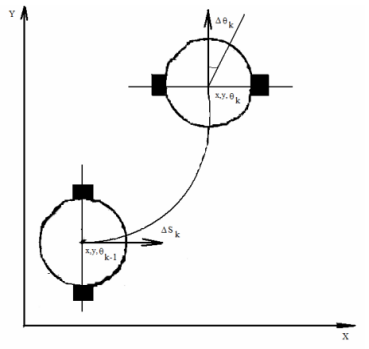


Figura 3. Localización del robot

Para la localización del robot como en la Fig. 3, que ya habíamos dicho que se consigue con tres parámetros, es necesario definir un origen en el espacio de actuación del robot. Por lo tanto todas las variables serán cero cuando el robot esté alineado con el eje x y situado en el primer nudo del mapa. De esta manera a medida que avance tanto horizontal como verticalmente lo hará de forma positiva.

Se va a poner el foco en el cambio de localización en un intervalo de muestreo puesto que de esta manera se puede calcular el incremento de cada variable gracias al incremento del centro del robot en ese intervalo de tiempo. Dicho incremento se representa como Δs_k por lo tanto Δs_d y Δs_i.serán los incrementos de las ruedas derecha e izquierda respectivamente. Por lo tanto a partir de ahora la distancia recorrida por el centro del robot queda representada con la letra *s*, el radio de las ruedas de tracción como *r*, y la distancia entre esas dos ruedas como *L*. Teniendo en cuenta que el incremento en el centro del robot se puede calcular como:

$$\Delta s = \frac{\Delta sd + \Delta si}{2} \quad (1)$$

y el incremento de la orientación se puede calcular como:

$$\Delta\theta = \frac{\Delta sd - \Delta si}{L} \quad (2)$$

porque de esta manera será positivo siempre que gire en sentido anti horario (tal y como se había definido anteriormente) ya es posible calcular Δx y Δy. Resulta muy sencillo calcular dichos incrementos atendiendo a la trigonometría y teniendo en cuenta la manera en que se ha definido el ángulo θ (Fig. 4) como el incremento del centro del robot por el coseno de dicho ángulo para el caso del Δx y multiplicado por el seno de dicho ángulo para el caso del Δy.

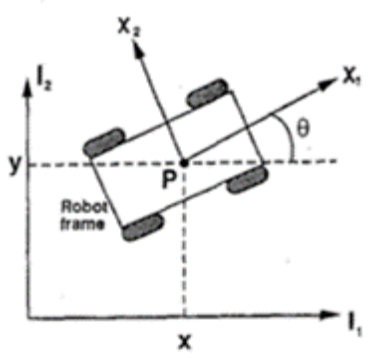


Figura 4. Parámetros odométricos para la localización

Aquí se presenta un pequeño problema porque se está asumiendo que el ángulo θ no presenta variación alguna en ese intervalo de tiempo por lo tanto, si queremos hacer una buena aproximación se debe incluir ese incremento en las ecuaciones quedando estas de la siguiente manera:

$$\Delta x = \Delta s \cdot \cos\left(\theta + \frac{\Delta\theta}{2}\right)$$

$$\Delta y = \Delta s \cdot \operatorname{sen}\left(\theta + \frac{\Delta\theta}{2}\right)$$

Una vez obtenidas las ecuaciones que modelan como se puede obtener la localización y la orientación del robot en todo momento es necesario destacar que una vez estas ecuaciones se implementen en la raspberry se estarán actualizando constantemente y en paralelo con el período de muestreo adecuado como se explica con detalle en el apéndice B.

Por último, enfocando estas ecuaciones a nuestro modelo de robot se hace necesario exponer cómo se consigue Δs_d y Δs_i para poder calcular los incrementos de *x, y* y θ como se ha explicado anteriormente. Para medir los incrementos de cada rueda se dispone de dos encoders. Cada uno de los encoders dispone de un emisor, un receptor y una rueda con 20 marcas. El emisor, que irá alimentado a través de la raspberry, genera un haz de

$$\Delta s_d = 0{,}9895 \cdot \textit{nº marcas (Para la rueda derecha)}$$

luz que atravesando alguna de las 20 marcas llegará hasta el receptor, este, configurado como entrada en la raspberry devolverá la tensión a la placa que la codificará como un "1" en ese pin. Cuando el haz de luz no consiga llegar al receptor (porque no estará atravesando ninguna marca) este no llevará tensión al pin de entrada de la raspberry y será codificado como un "0", este cambio es el que se contabilizar para calcular los incrementos de cada rueda. Para calcular a qué distancia equivale cada marca basta con tener en cuenta que una vuelta completa de la rueda son $2 \cdot \pi \cdot r$ siendo r el radio de la rueda y por lo tanto, una vuelta completa habrá pasado por las 20 marcas generando cambios en el receptor, así pues: $\frac{2 \cdot \pi \cdot r\ (centímetros)}{20\ (marcas)}$ el factor de conversión resulta 0,9895 centímetros cada marca. Si contabilizamos todos los cambios de estado en el receptor, es decir las marcas que se recorren, que tienen lugar en un período de muestreo y las multiplicamos por su factor de conversión obtenemos directamente el Δs de esa rueda. Analíticamente se representa como:

El factor de conversión se representa en las unidades de centímetros/marca porque es

$$\Delta s_i = 0{,}9895 \cdot \textit{nº marcas (Para la rueda izquierda)}$$

la unidad que mejor se adecúa a las dimensiones del escenario construido (capítulo 4). El código para la implementación se explica y analiza con detalle en el apéndice B.

3. Reconocimiento de imágenes

En este trabajo, aparte del uso de la localización geométrica, el robot basa su funcionamiento en el reconocimiento artificial de imágenes. El reconocimiento visual de imágenes es la sensación interior de conocimiento aparente que resulta de un estímulo o impresión luminosa registrada en nuestros ojos donde el ser humano buscará dar significado a la información y esta agrupará y organizará cualquier señal recibida para que se parezca a algo ya conocido. Como ya se ha explicado a lo largo de la memoria se trata de simular, con su aproximación más exacta posible, un vehículo circulando en un entorno urbano en el cual lo único que se conoce con anterioridad es la disposición de las calles, los giros restringidos o permitidos se conocen en cada intersección del mapa.

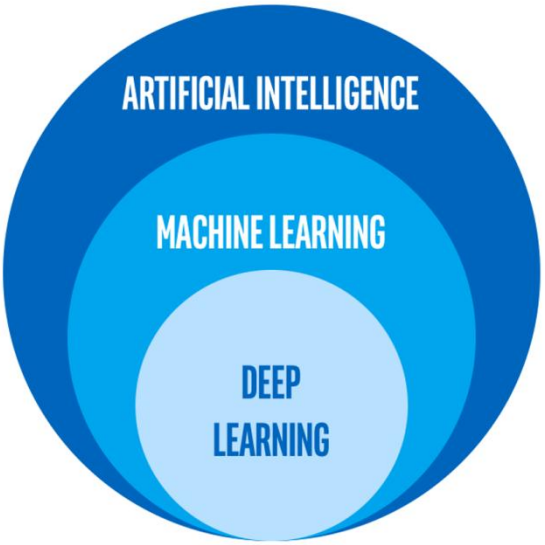


Figura 5. Jerarquía de la inteligencia artificial.

Por lo tanto, queda claro que es necesario un reconocimiento de imágenes a través de la cámara conectada a la raspberry para que sea un proceso totalmente automatizado e independiente al ser humano. Para el reconocimiento de imágenes se ha optado por el aprendizaje automático dentro del cual existen varias alternativas tales como *los K vecinos más cercanos, la regresión lineal o las redes neuronales.* Existen diversas razones por las que se ha elegido hacer el reconocimiento usando redes neuronales basadas en *Deep Learning* (Fig. 5) entre las que destacan: simplicidad en la generación de código, eficiencia y fiabilidad en las decisiones, es un campo en desarrollo donde resulta fácil encontrar mucha información y además, se ha trabajado con más profundidad durante la carrera. El objetivo en este proyecto es por lo tanto conseguir una red que sea capaz de identificar y clasificar señales de tráfico dentro de un entorno urbano al mismo tiempo que recorre una ruta para llegar a un punto final.

3.1. Fundamento de las redes neuronales

Antes de empezar con el análisis de las prestaciones de las diferentes redes que se han utilizado para el proyecto es preciso explicar qué es y cómo funciona una red neuronal. Una red neuronal es un conjunto de neuronas artificiales interconectadas masivamente con una

organización jerárquica trabajando conjuntamente las cuales tratan de interactuar con los objetos del mundo real de la misma manera que lo hace el sistema biológico. La unidad mínima de una red neuronal es, como resulta obvio, una neurona artificial. Una neurona artificial se puede definir como una unidad de cálculo que intenta modelar el comportamiento de una neurona "natural" como las que constituyen el cerebro humano. Se puede modelar dichas neuronas artificiales como un programa de cálculo que recibe unas entradas (bien sean externas o de otras neuronas), las combina y produce una salida.

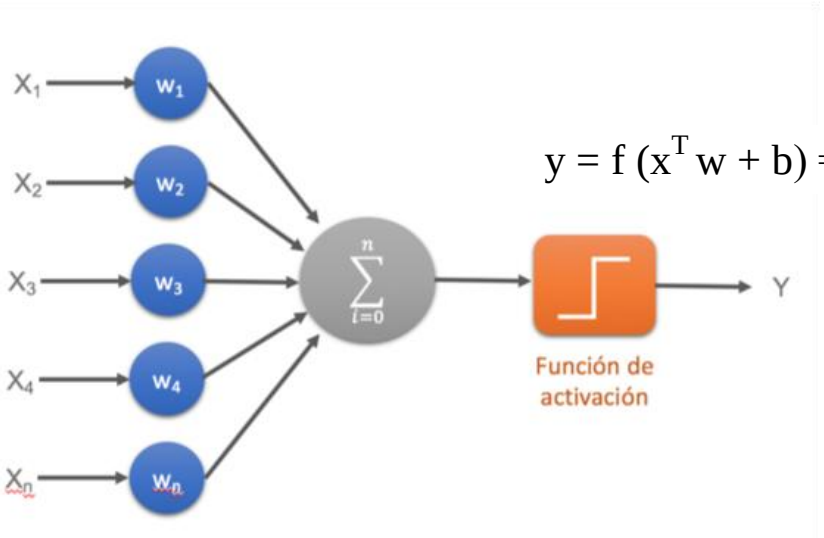


Figura 6. Matemática de una unidad neuronal.

Atendiendo a la Fig. 6 donde:

- x_n son cada una de las entradas a la neurona
- w_n el peso por el que se multiplica a cada entrada
- f es la función de activación
- b es el sesgo de esa función
- y es la salida de la neurona

Por consecuencia, cada neurona realiza una suma ponderada de sus entradas y, mediante una función de activación, se calcula la salida.

La función de activación puede ser una función lineal, una función umbral o una función no lineal que simula con mayor exactitud las características de las neuronas biológicas [3]. Para estas funciones se buscan que sus derivadas sean más simples para reducir el coste computacional. Se dispone de un amplio abanico de posibilidades de funciones de activación: Sigmoid, tanh, ReLU, Leaky ReLU, Maxout, Softmax, Softmax loss, etc.

Para que una red neuronal funcione debe tener mínimo dos capas: una capa de entrada y una capa de salida. Lo más usual es que cualquier tipo de red presente capas intermedias a las que se denominan capas ocultas (Fig. 7).

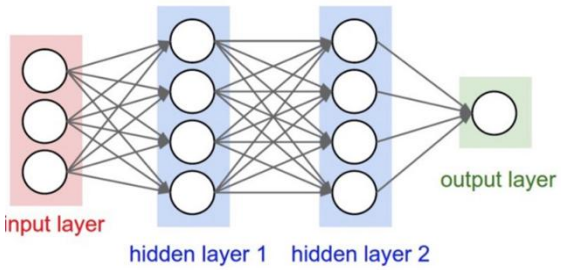


Figura 7. Arquitectura de las redes.

En la Fig. 6 se puede ver un ejemplo de red neuronal con dos capas intermedias pero esto no significa que en todas las redes existentes estén conectadas todas las neuronas entre sí, sino que es un tipo de arquitectura de estas. Esta arquitectura se conoce como red densamente conectada donde todas las neuronas de una capa están conectadas con todas las neuronas de la capa siguiente. Para el caso de la capa de entrada y particularizado al tratamiento de imágenes cada neurona de la capa de entrada estaría conectada a cada píxel de la imagen.

A parte de este tipo de redes existen otras muchas configuraciones como por ejemplo las recurrentes (más útiles para textos, sonido o video), las de base radial (utilizadas sobre todo en series temporales o procesamiento de voz) o las convolucionales (con buena respuesta para reconocimiento de imágenes).

En el siguiente apartado se analiza la estructura la red construida para este proyecto en la cual se han utilizado tanto capas convolucionales como capas densamente conectadas. Las capas convolucionales [16], que como ya se ha comentado están formadas por neuronas con una función de activación y un sesgo, basan su funcionamiento en varios filtros uno detrás de otro los cuales almacenan las características más importantes (aristas, gotas de color, etc.) donde cada filtro se suele encargar de aprender a detectar un patrón. Es por esta razón que se ha elegido la configuración de red convolucional para la construcción de la red neuronal que se analiza en el capítulo siguiente. Los mapas donde se almacena esa información suelen tener información redundante y es por eso que se reduce su tamaño para ahorrar costes computacionales, a esta operación se la conoce como *"Pooling"*.

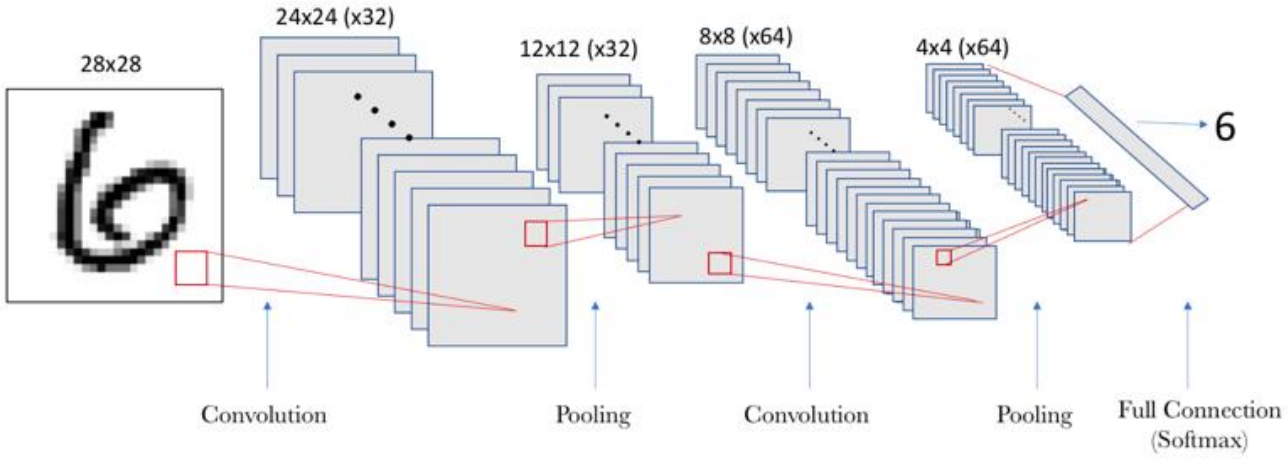


Figura 8. Funcionamiento de las capas convolucionales.

Las redes neuronales artificiales, al igual que las biológicas, aprenden por repetición. El entrenamiento de estas redes se lleva a cabo tomando una serie de datos, imágenes de señales de tráfico en nuestro caso, y actualizando los pesos de las neuronas que forman la red. Durante la fase de entrenamiento de la red hay que tener un cuidado especial ya que si se entrena por demás se puede producir el sobre overfitting o sobre entrenamiento que es básicamente que los pesos de las neuronas de ajustan tanto a los datos de entrenamiento que la red no tiene un comportamiento aceptable cuando trabaja con unos datos distintos a los de entrenamiento. Por el contrario, también se puede producir en underfitting o infra entrenamiento donde los pesos de las neuronas no han llegado a ajustarse correctamente y la red tampoco presenta un comportamiento válido. Una vez se ha acabado el entrenamiento y con otros datos distintos se procede a la validación, momento en que obtendremos las características (fiabilidad, coste, etc.) de la red.

Antes de comenzar el análisis de las prestaciones de las diferentes redes seleccionadas para el proyecto se va a realizar una breve explicación de los diferentes términos que se utilizan para el estudio del rendimiento de las redes neuronales.

Dentro de la biblioteca keras, que es con la que se ha trabajado principalmente, existen una serie de métodos estandarizados que nos facilitan este estudio porque resultan extremadamente útiles a la hora de hacer un seguimiento del rendimiento del Deep Learning.

Antes de comenzar el análisis de las prestaciones de las diferentes redes seleccionadas para el proyecto se va a realizar una breve explicación de los diferentes términos que se utilizan en dichas métricas para el estudio de las redes neuronales [19][21].

- **True Positives (TP):** cuando la clase real del punto de datos era 1 (Verdadero) y la predicha es también 1 (Verdadero)
- **True Negatives (TN):** cuando la clase real del punto de datos fue 0 (Falso) y el pronosticado también es 0 (Falso).
- **False Positives (FP):** cuando la clase real del punto de datos era 0 (False) y el pronosticado es 1 (True).
- **False Negatives (FN):** Cuando la clase real del punto de datos era 1 (Verdadero) y el valor predicho es 0 (Falso).

Estos conceptos se ilustran fácilmente con un ejemplo aplicado del proyecto: El robot va determinar si una señal que procesa por la cámara pertenece a la clase de obligatorio girar a la derecha. Entonces será:

Verdadero Positivo (True Positive) si es una señal de obligatorio girar a la derecha y se clasifica como tal.

Falso Positivo (False Positive) si no es una señal de obligatorio girar a la derecha pero sin embargo sí que se clasifica como una señal de obligatorio girar a la derecha.

Verdadero Negativo (True Negative) si no es una señal de obligatorio girar a la derecha y el robot no lo clasifica como señal de obligatorio girar a la derecha.

Falso Negativo (False Negative) si es una señal de obligatorio girar a la derecha y el robot no lo clasifica así.

- **Confusion Matrix (Matriz de confusión):** ilustra de una manera muy gráfica los

Figura 9. Forma de la matriz de confusión.

conceptos anteriores. Las columnas de la matriz representan las clases predichas por la red y las filas representan las clases reales de esas imágenes. Por lo tanto cuantos más valores se sitúen en la diagonal más fiable será esa red.

- **Precision (Precisión):** Esta métrica se define como la cantidad de casos verdaderos positivos sobre la cantidad total de todo lo que predijo que era positivo. Es decir, de todo lo que el algoritmo predijo como positivo, se evalúa cuánto de eso era cierto.

$$precision = \frac{TP}{TP + FP}$$

Siguiendo con el ejemplo de antes, la precisión sería cuántas señales de las que ha clasificado como obligatorio girar a la derecha son realmente señales de obligatorio girar a la derecha y no de otro tipo. Se suele medir en porcentaje.

- **Accuracy (Exactitud):** se define como la cantidad de casos verdaderos positivos y verdaderos negativos sobre la cantidad total de datos, es decir, el porcentaje de aciertos sobre el total de imágenes a clasificar. Se debe tener especial cuidado porque es una métrica que puede llevar a engaño ya que puede tener un valor muy alto pero no ser una red muy fiable.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Por ejemplo si se tratara de predecir qué señales pertenecen a la clase obligatorio girar a la derecha de un conjunto de 100 señales de las cuales 80 son de obligatorio girar a la derecha y la red predijera que las 100 señales son de obligatorio girar a la derecha se tendría una exactitud del 80%. Esto se soluciona con un conjunto de datos bien balanceados donde haya una cantidad parecida de imágenes de cada clase como es el caso de este proyecto, por este motivo es correcto fiarse del valor de la accuracy.

- **Recall (exhaustividad):** Se compara la cantidad de casos clasificados como verdaderos positivos sobre todo lo que realmente era positivo, es decir, informa de la cantidad de imágenes que la red es capaz de clasificar.

$$recall = \frac{TP}{TP + FN}$$

El ejemplo más claro para esta métrica se ilustra con un tema muy actual como son las pruebas de detección del COVID-19. Lo ideal para estas pruebas sería tener un recall

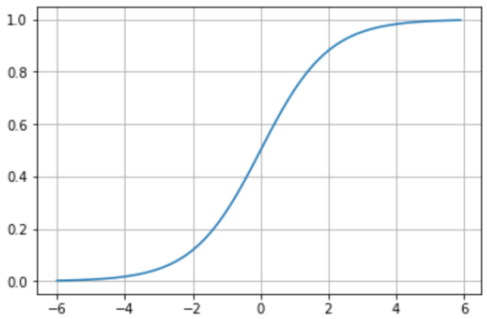
de 1 porque de esta manera una persona portadora del virus siempre sería detectada como positivo, sin embargo, si el recall no fuera del 100% una persona infectada del virus podría resultar negativo en una prueba de detección (falso negativo) y seguir contagiando el virus.

Por lo tanto, dependiendo los datos y el objetivo que se tenga con la predicción de la red es más conveniente atender a una métrica o a otra.

3.2. Particularización y prestaciones

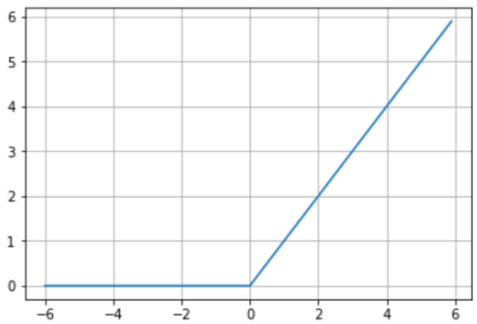
En este apartado se profundiza y se analizan las prestaciones de la red neuronal construida para este proyecto en la cual se han usado las funciones de activación ReLU y Softmax que tienen la siguiente forma:

Softmax (normalized exponential)



$$f(x) = \frac{e^{x_j}}{\sum_k e^{x_k}}$$

Rectified Linear Unit (ReLU)



f (x) = max {0, x}

Figura 10. Gráficas correspondientes a las funciones que se mencionan

La función ReLU, que se usa porque presenta un muy buen comportamiento con imágenes, anula los valores negativos y deja los positivos tal y como entran. Mientras que la función Softmax devuelve la distribución de probabilidad sobre clases de salida mutuamente excluyentes [3]. La salida de esta función se suele definir como la probabilidad pero no es así exactamente, sino que dicha salida es el peso que tiene cada neurona de esa capa. En este proyecto y, como suele ser usual, la función de activación Softmax se usará en la capa de salida para obtener las salidas de manera ponderada y valorar la fiabilidad de la predicción. Como en las capas ocultas se ha usado la función ReLU, en la capa de salida quedarán únicamente valores comprendidos entre 0 y 1.

Para el entrenamiento y validación de nuestra red se dispone de un conjunto de imágenes de señales de tráfico obtenidas en *kaggle.com* que es un servidor web donde se pueden encontrar diversas bases de datos subidas por los usuarios útiles para entrenar las redes neuronales. Se dispone también de un conjunto de imágenes tomadas directamente del entorno físico para facilitar el entrenamiento y la posterior predicción de la red, clasificadas por carpetas en función de la señal a la que pertenezcan. Dependiendo del entrenamiento se ha usado una u otra base de imágenes como se explica posteriormente.

La red construida se ha implementado haciendo uso de la biblioteca keras [28]. Esta biblioteca permite construir redes de manera secuencial mediante la orden *Sequential*. La

arquitectura de esta red consta de:

1. Una capa de entrada: donde las imágenes tendrán un tamaño de 224x224 píxeles y tres canales de color (verde, rojo y azul).

Layer (type)	Output Shape	Param #
module_wrapper (ModuleWrappe	(None, None, None, None)	0
conv2d (Conv2D)	(None, None, None, 254)	7112
conv2d_1 (Conv2D)	(None, None, None, 254)	580898
max_pooling2d (MaxPooling2D)	(None, None, None, 254)	0
conv2d_2 (Conv2D)	(None, None, None, 128)	130176
conv2d_3 (Conv2D)	(None, None, None, 128)	65664
max_pooling2d_1 (MaxPooling2	(None, None, None, 128)	0
conv2d_4 (Conv2D)	(None, None, None, 32)	4128
conv2d_5 (Conv2D)	(None, None, None, 32)	1056
max_pooling2d_2 (MaxPooling2	(None, None, None, 32)	0
dropout (Dropout)	(None, None, None, 32)	0
flatten (Flatten)	(None, None)	0
dense (Dense)	(None, 128)	4224
dense_1 (Dense)	(None, 6)	774

Total params: 794,032
Trainable params: 794,032
Non-trainable params: 0

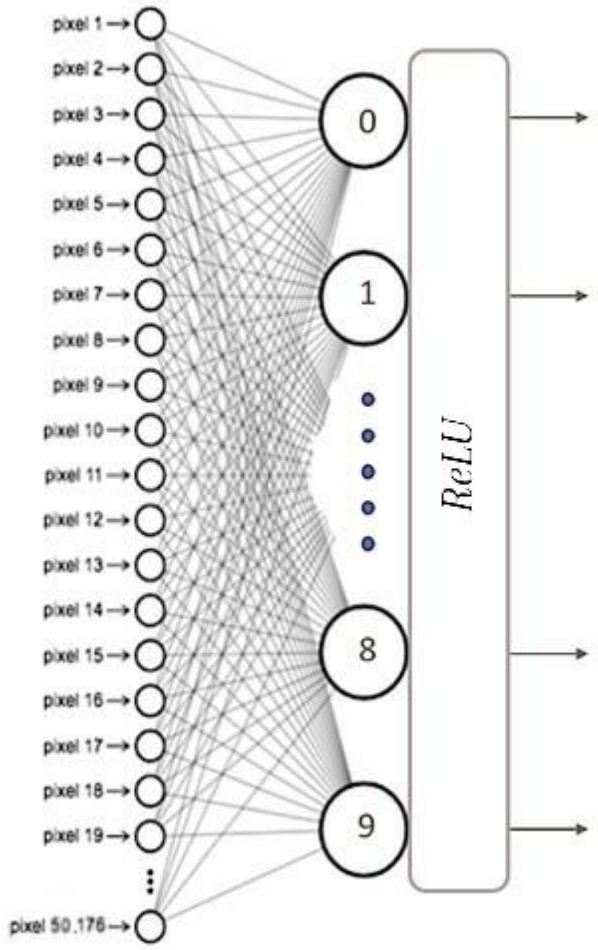


Figura 11. Sumario de la red y conexiones de la capa de entrada

2. Varias capas convolucionales ocultas: se agregan mediante la orden `model.add(Conv2D(254, 3,3 ,activation= 'relu'))`. En este caso se está añadiendo una primera capa de 254 filtros de tamaño 3x3 píxeles y con función de activación ReLU. Después de cada capa convolucional se agrega un *Pooling* cuyo tamaño será diferente en función del tamaño del filtro de la capa que le precede.
3. Una capa oculta de neuronas densamente conectadas: que se agrega mediante la orden `model.add(Dense(128, activation='relu'))`. Aquí se han añadido 128 neuronas con activación ReLU también
4. Una capa de salida de neuronas densamente conectadas con 6 neuronas donde cada una corresponde a un tipo de señal.

Cabe destacar que, como se ve en el sumario, en el código para la construcción de la red se han usado estrategias como el Dropout que consiste en desactivar un porcentaje de neuronas aleatoriamente durante el entrenamiento para aumentar la eficacia y la robustez de la red.

Una vez realizado el entrenamiento resulta de interés analizar qué prestaciones tiene esta red y si fuera lo suficientemente fiable para incluirla en el proyecto. Para el análisis de esta red se van a tener en cuenta las imágenes tomadas con la cámara de la raspberry directamente del circuito físico que se ha construido ya que se aproxima a un comportamiento más real. Las imágenes son de 2.592 píxeles de ancho por 1.944 píxeles de alto procesadas y convertidas en imágenes de 224x224 píxeles. Por esa misma razón el entrenamiento de esta red se ha realizado con imágenes del circuito obtenidas con la cámara incluida en el robot móvil. La siguiente gráfica muestra como ha ido mejorando la precisión de la red durante el

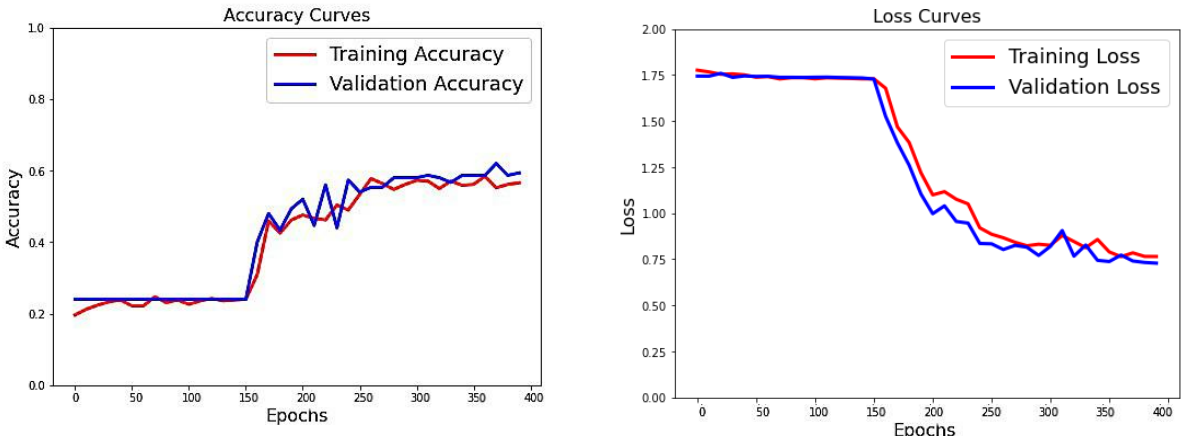


Figura 12. Gráficas de la evoluvión a lo largo del entrenamiento de accuracy y loss.

entrenamiento y como ha ido descendiendo la función *Loss*. La función Loss o función de pérdida, es una función que evalúa la desviación entre las predicciones realizadas por la red neuronal y los valores reales de las observaciones utilizadas durante el aprendizaje. Cuanto menor es el resultado de esta función, más eficiente es la red neuronal. Su minimización, es decir, reducir al mínimo la desviación entre el valor predicho y el valor real para una observación dada, se hace ajustando los distintos pesos de la red neuronal.

Por lo tanto, atendiendo a la Fig. 12, se observa una precisión en torno al 60% y una

- Label Obligatorio derecha: Acc: 0.933, Prec: nan, Rec: 0.000
- Label Obligatorio izquierda: Acc: 0.773, Prec: 0.514, Rec: 1.000
- Label Prohibido derecha: Acc: 0.820, Prec: 1.000, Rec: 0.036
- Label Prohibido izquierda: Acc: 0.820, Prec: 0.509, Rec: 1.000
- Label Prohibido recto: Acc: 1.000, Prec: 1.000, Rec: 1.000
- Label Obligatorio recto: Acc: 0.840, Prec: nan, Rec: 0.000
- Average: Acc: 0.593, Prec: nan, Rec: 0.506

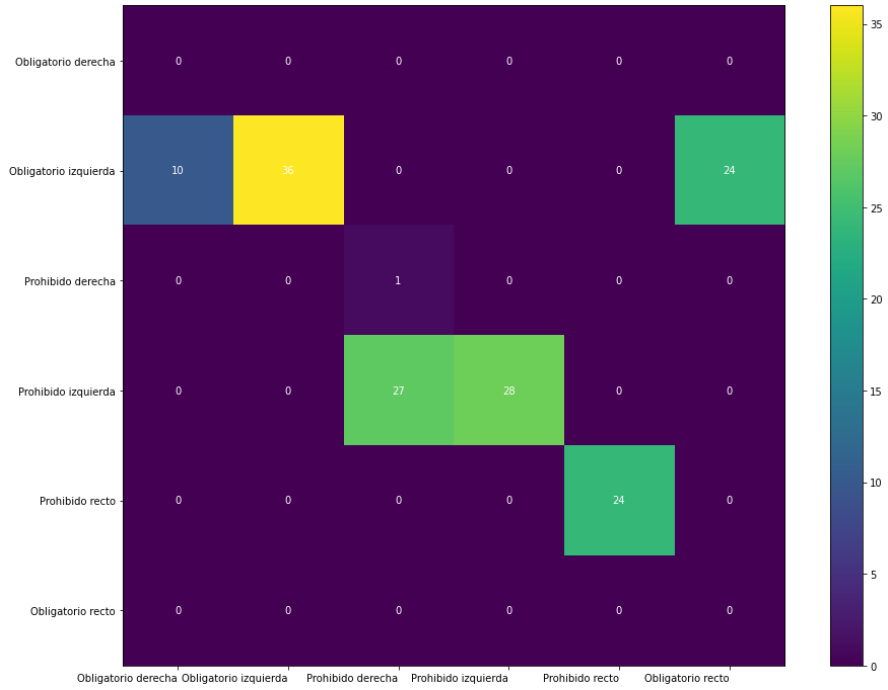


Figura 13. Métricas y matriz de confusión de la red creada.

función de pérdida de unos 0.72. Son valores no muy fiables si lo que se pretende es una red capaz de clasificar señales de tráfico de forma correcta. Se muestra a continuación las estadísticas de cada clase, es decir, de cada tipo de señal. Se incluye también la matriz de confusión para facilitar el análisis y representarlo de una manera más gráfica.

Se puede observar como de las señales de obligatorio derecha y obligatorio recta no diferencia ninguna por lo tanto no es una red válida para el proyecto.

Las presaciones de esta misma red en base al entrenamiento realizado con el banco de imágenes de señales obtenido de *kaggle.com* son algo mejores que las obtenidas con las imágenes tomadas con la PiCamera ya que era una base de datos de 9.235 imágenes, bastante más grande en relación a las 583 imágenes del entrenamiento anterior, y por lo tanto los pesos de las neuronas se ajustan de una manera más precisa aportanddo una mayor robustez. Pero el problema viene al usar la red con las imágenes del circuito y resulta una red poco fiable también. Por ello se prueba y se desarrolla a continuación utilizando redes pre-entrenadas para la detección de imágenes.

3.3. Comparativa y elección de redes pre-entrenadas

El entrenamiento de una red suele ser un proceso largo que requiere de muchos recursos a nivel de cómputo, principalmente si hablamos de redes para el reconocimiento de imágenes y de una alta complejidad a la hora de construir la arquitectura de la red. Es por esto que existen a disposición del usuario, gracias a la biblioteca keras, redes ya entrenadas, es decir, redes donde no haya que esperar un largo tiempo para que los pesos se vayan actualizando y ajustando sino que ya estén pre ajustados y el entrenamiento que haya que hacer sea un proceso mucho más corto y llevadero además de tener la arquitectura ya construida. Algunas de esas redes con las que se ha trabajado son [12][14]:

a) VGG: VGGNet es una arquitectura de red neuronal convolucional propuesta por Karen Simonyan y Andrew Zisserman de la Universidad de Oxford en 2014 [13]. Existen dos variantes de esta arquitectura de red, VGG16 y VGG19 cuya diferencia reside en las capas ocultas que posee cada una. Las imágenes de entrada, tras el preprocesamiento, pasan por las diferentes capas de la red como se ilustra en la Fig. 14. Las imágenes de entrenamiento pasan por una pila de capas de convolución. Hay un total de 13 capas convolucionales y 3 capas totalmente conectadas en la arquitectura VGG16. VGG tiene filtros más pequeños (3*3) con más profundidad en lugar de tener filtros grandes. Ha acabado teniendo el mismo campo receptivo efectivo que si sólo tuviera una capa convolucional de 7 x 7. Tanto la VGG16 como la VGG19 son redes que no tienen muchas capas ocultas, pero sí que son de gran tamaño. Como se ve en el sumario de la red VGG16 esta tiene cerca de 140 millones de parámetros, valor que supera la red VGG19. Su eficacia según [14] supera el 90% pero son redes muy pesadas: más de 500 MB.

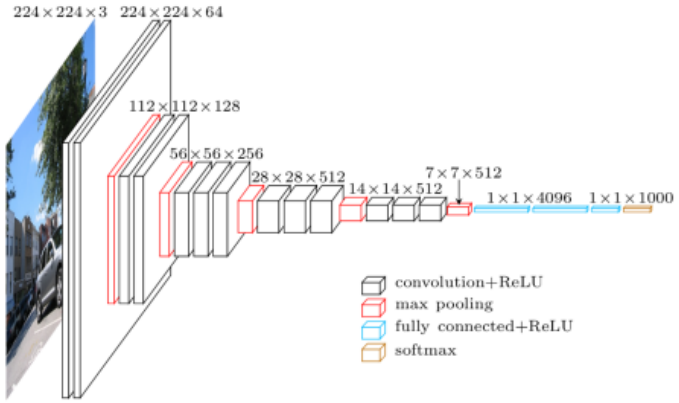


Figura 14. Funcionamiento de la arquitectura VGG.

b) MobileNet: Presenta dos opciones: la versión 1 y la versión 2. En este proyecto se utiliza la segunda versión por ser más algo más ligera y tener una mayor tasa de precisión. La arquitectura de MobileNet v2 (Fig. 15) se basa en una estructura residual invertida en la que la entrada y la salida del bloque residual son capas finas de cuello de botella, al contrario que los modelos residuales tradicionales que utilizan representaciones expandidas en la entrada. MobileNet v2 utiliza convoluciones ligeras en profundidad para filtrar las características en la capa de expansión intermedia. Además, se han eliminado las no linealidades en las capas estrechas para mantener la potencia de representación [29].

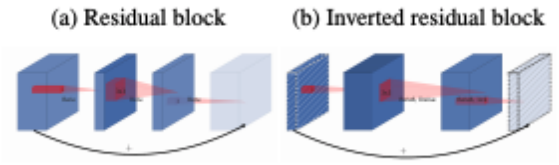


Figura 15. Arquitectura MobileNet

c) NasNet: Única red neuronal que no ha sido creada por el ser humano, sino que ha sido creada con Auto ML. Auto ML es una inteligencia artificial creada por Google para diseñar modelos de aprendizaje automático. La red NasNet es la que mejor eficiencia presentaba de entre todas las redes existentes hasta que apareció EfficientNet, se ha entrenado utilizando las más de 60.000 imágenes de CIFAR-10 que es un banco de imágenes de todo tipo de elementos. Esta red se ha combinado con un modelo convolucional y es computacionalmente menos exigente que otras. Existen dos configuraciones de la arquitectura NasNet que son la Large y la Mobile, en este caso se ha utilizado la segunda por ser menos pesada y por lo tanto más rápida a nivel computacional.

d) EfficientNet: La escala del modelo se estudia sistemáticamente para equilibrar cuidadosamente la profundidad, el ancho y la resolución de la red, lo que puede conducir a un mejor rendimiento. Se propone un coeficiente compuesto efectivo para escalar uniformemente todas las dimensiones de profundidad, ancho, resolución. EfficientNet ha sido obtenida a partir de la arquitectura neuronal de la red anterior, es decir, de tipo NAS. Se dispone de 7 redes diferentes de tipo

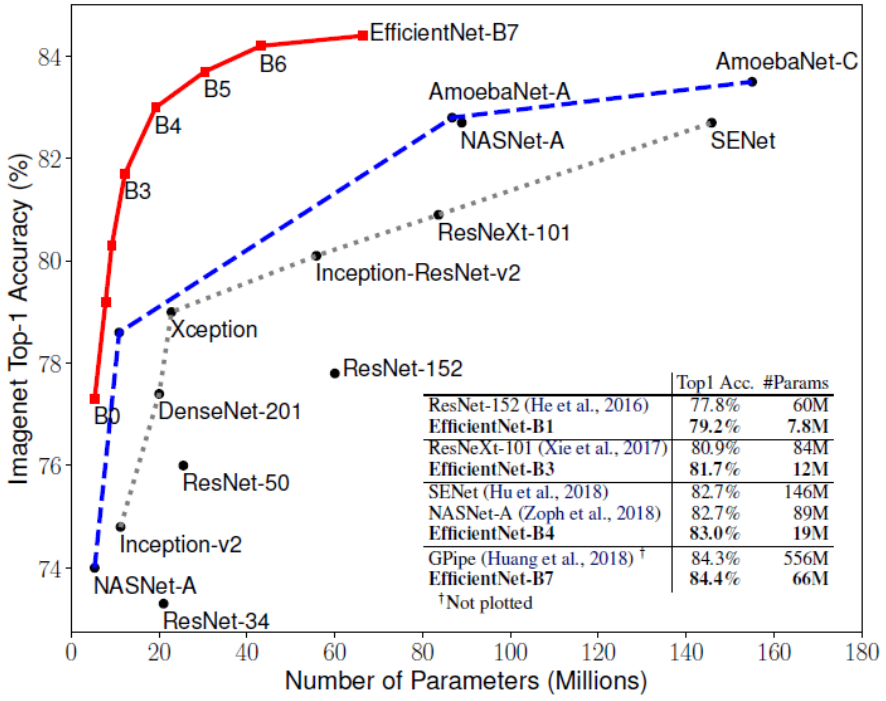


Figura 16. Comparativa entre redes pre-entrenadas

EfficientNet, desde la B0 hasta la B7 donde la diferencia es el número de parámetros que tiene cada red, desde los 5.330.571 que presenta la B0 hasta los 66.658.687 que posee la B7 [31].

e) ResNet: En general, en una red neuronal convolucional profunda, se apilan varias capas y se entrenan para la tarea en cuestión. La red aprende varias características de nivel bajo/medio/alto al final de sus capas. En el aprendizaje residual, en lugar de tratar de aprender algunas características, tratamos de aprender algunas residuales. El residuo puede entenderse simplemente como la sustracción de la característica aprendida de la entrada de esa capa. ResNet hace esto utilizando conexiones de acceso directo (conectando directamente la entrada de la capa n a la capa (n+x)). Se ha demostrado que el entrenamiento de esta forma de redes es más fácil que el entrenamiento de redes neuronales profundas simples y también se resuelve el problema de la degradación de la precisión. Al igual que las otras arquitecturas de red, ResNet cuenta con varios tipos como son ResNet50, ResNet101, ResNet152, ResNet50V2, ResNet101V2 y ResNet152V2. Aplicando el mismo criterio que antes, en este proyecto se trabaja con ResNet50 por ser la más ligera de todas. Aunque sea la que menor precisión tiene no es una gran diferencia (1.4% con respecto a la ResNet que mayor precisión posee) [32].

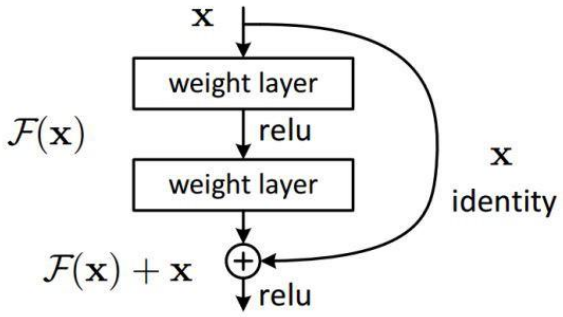


Figura 17. Arquitectura ResNet

Después de conocer la arquitectura de estas redes, procede analizar las prestaciones de cada tipo de red de la misma manera que se ha ilustrado en la Fig. 12 en el apartado anterior con la red neuronal construida únicamente para este proyecto.

a) Gráficas de VGG16

Figura 18. Gráficas de evolución de accuracy y loss de VGG16.

Label Obligatorio derecha: Acc: 0.973, Prec: 0.800, Rec: 0.800, F1 0.800.
Label Obligatorio izquierda: Acc: 0.973, Prec: 0.943, Rec: 0.943, F1 0.943.
Label Prohibido derecha: Acc: 0.879, Prec: 1.000, Rec: 0.357, F1 0.526.
Label Prohibido izquierda: Acc: 0.879, Prec: 0.609, Rec: 1.000, F1 0.757.
Label Prohibido recto: Acc: 1.000, Prec: 1.000, Rec: 1.000, F1 1.000.
Label Obligatorio recto: Acc: 1.000, Prec: 1.000, Rec: 1.000, F1 1.000.
Average: Acc: 0.852, Prec: 0.892, Rec: 0.850, F1 0.838.

Figura 19. Métricas de la red VGG16.

b) Gráficas MobileNetV2

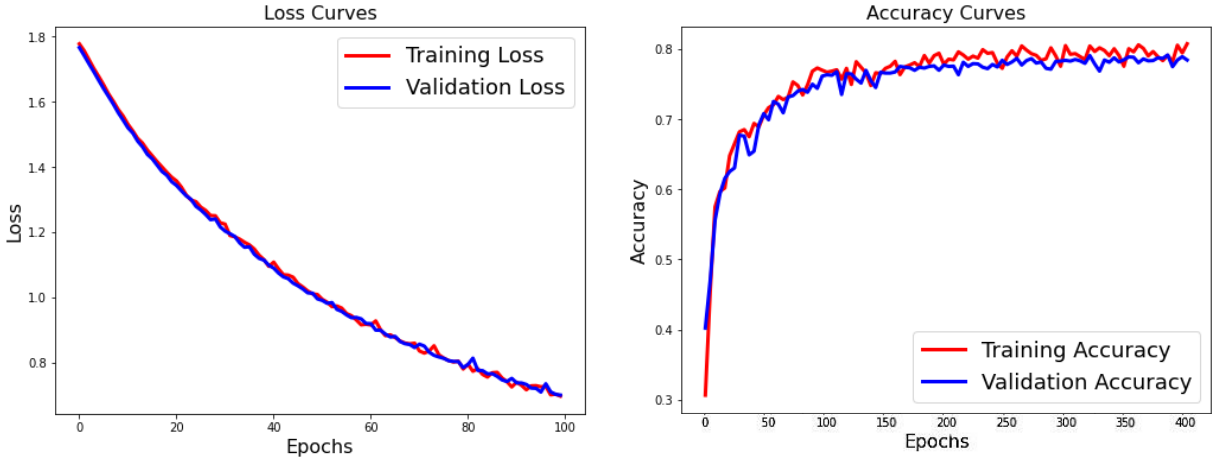


Figura 20. Gráficas de evolución de accuracy y loss de MobileNetV2.

Label Obligatorio derecha: Acc: 0.868, Prec: 0.588, Rec: 0.857, F1 0.698.
Label Obligatorio izquierda: Acc: 0.891, Prec: nan, Rec: 0.000, F1 nan.
Label Prohibido derecha: Acc: 0.924, Prec: 0.505, Rec: 0.974, F1 0.665.
Label Prohibido izquierda: Acc: 0.925, Prec: nan, Rec: 0.000, F1 nan.
Label Prohibido recto: Acc: 0.998, Prec: 1.000, Rec: 0.993, F1 0.996.
Label Obligatorio recto: Acc: 0.961, Prec: 0.891, Rec: 0.989, F1 0.938.
Average: Acc: 0.784, Prec: nan, Rec: 0.636, F1 nan.

Figura 21. Métricas de la red MobileNetV2.

c) Gráficas NasNet

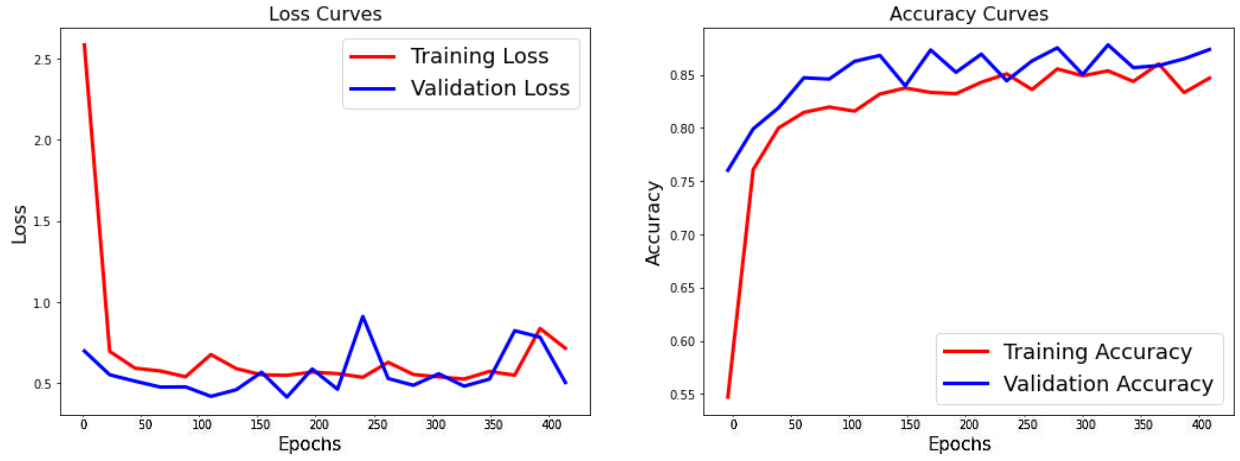


Figura 22. Gráficas de evolución de accuracy y loss de NasNetMobile.

Label Obligatorio derecha: Acc: 0.940, Prec: 0.771, Rec: 0.942, F1 0.848.
Label Obligatorio izquierda: Acc: 0.961, Prec: 0.860, Rec: 0.763, F1 0.809.
Label Prohibido derecha: Acc: 0.941, Prec: 0.600, Rec: 0.723, F1 0.656.
Label Prohibido izquierda: Acc: 0.941, Prec: 0.630, Rec: 0.489, F1 0.551.
Label Prohibido recto: Acc: 0.994, Prec: 0.997, Rec: 0.979, F1 0.988.
Label Obligatorio recto: Acc: 0.971, Prec: 0.986, Rec: 0.914, F1 0.949.
Average: Acc: 0.874, Prec: 0.807, Rec: 0.802, F1 0.800.

Figura 23. Métricas de la red NasNet.

d) Gráficas EfficientNetB0

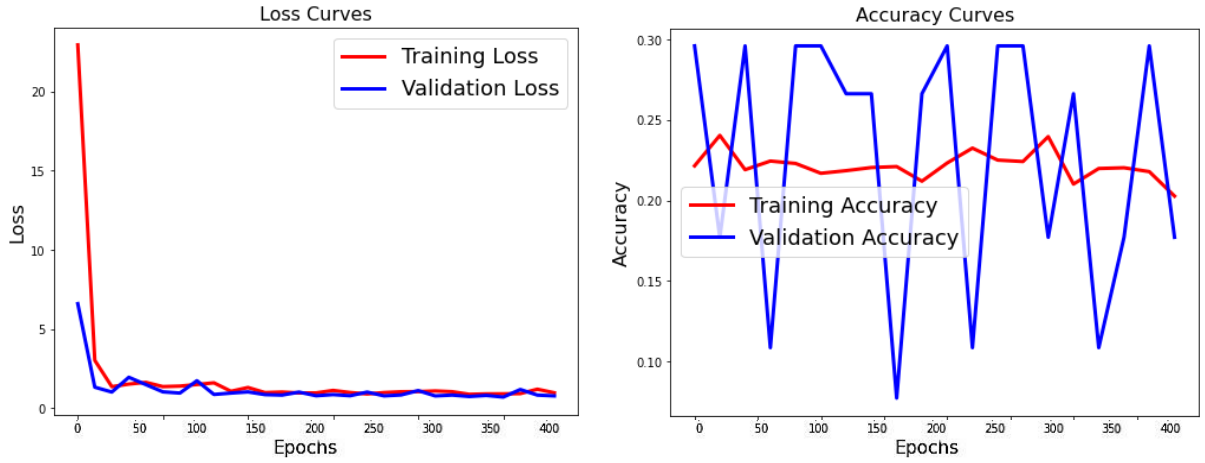


Figura 24. Gráficas de evolución de accuracy y loss de EfficientNetB0.

Label Obligatorio derecha: Acc: 0.177, Prec: 0.177, Rec: 1.000, F1 0.301.
Label Obligatorio izquierda: Acc: 0.891, Prec: nan, Rec: 0.000, F1 nan.
Label Prohibido derecha: Acc: 0.923, Prec: nan, Rec: 0.000, F1 nan.
Label Prohibido izquierda: Acc: 0.925, Prec: nan, Rec: 0.000, F1 nan.
Label Prohibido recto: Acc: 0.734, Prec: nan, Rec: 0.000, F1 nan.
Label Obligatorio recto: Acc: 0.704, Prec: nan, Rec: 0.000, F1 nan.
Average: Acc: 0.177, Prec: nan, Rec: 0.167, F1 nan.

Figura 25. Métricas de la red EfficientNetB0.

e) Gráficas ResNet50

Figura 26. Gráficas de evolución de accuracy y loss de ResNet50.

Label Obligatorio derecha: Acc: 0.844, Prec: 0.949, Rec: 0.125, F1 0.221.
Label Obligatorio izquierda: Acc: 0.784, Prec: 0.308, Rec: 0.792, F1 0.444.
Label Prohibido derecha: Acc: 0.925, Prec: 0.818, Rec: 0.046, F1 0.087.
Label Prohibido izquierda: Acc: 0.899, Prec: 0.420, Rec: 0.952, F1 0.583.
Label Prohibido recto: Acc: 0.931, Prec: 0.805, Rec: 0.978, F1 0.883.
Label Obligatorio recto: Acc: 0.870, Prec: 0.913, Rec: 0.620, F1 0.738.
Average: Acc: 0.627, Prec: 0.702, Rec: 0.586, F1 0.493.

Figura 27. Métricas de la red ResNet50.

Después de tener entrenadas y conocer las prestaciones de todas las posibles redes que se podían implementar para lograr el objetivo del proyecto se procede a explicar cuál ha sido la red elegida para ser usada desde la raspberry. El entrenamiento se ha realizado con el banco de imágenes obtenidas con la cámara de la raspberry puesto que era la opción más lógica ya que las imágenes eran más reales. Teniendo en cuenta que hay que encontrar un equilibrio entre fiabilidad y rapidez a la hora de clasificar fotos la primera de las opciones, la VGG16, se descarta porque la memoria de la CPU de la raspberry no es lo suficientemente grande y salta un error si se intenta cargar el modelo. De aquí se puede confirmar la conclusión que se había presupuesto antes del proceso experimental donde se decía que una red muy pesada no iba a ser funcional para este proyecto puesto que había que tener en cuenta las limitaciones de la Raspberry Pi 2. La siguiente red en ser descartada es la de arquitectura tipo EfficientNet porque no se ha conseguido llevar el entrenamiento hacia una clasificación fiable, de hecho es bastante pobre en ese aspecto pues como se aprecia no supera el 20% de accuracy. De entre las otras tres candidatas, la de tipo ResNet no presenta una accuarcy aceptable por lo tanto se descarta también. La de tipo MobileNet sí que llega a un nivel admisible de accuarcy pero que en comparación con la red NasNet se queda corta. Por lo tanto, como ninguna de estas presenta problemas al ser importadas y gestionadas con la raspberry se ha elegido la red NasNetMobile como la red neuronal que se implementará en la raspberry y será la encargada de gestionar la clasificación de las señales de tráfico.

Para una mayor robustez, se han juntado los dos bancos de imágenes (las obtenidas en *kaggle.com* representadas en la Fig. 28 y las obtenidas con la cámara del robot directamente desde el circuito urbano representadas en la Fig. 29) para realizar un nuevo entrenamiento de la red. Las prestaciones, como se ve en la Fig. 30, no son las esperadas puesto que la precisión de la red, lejos de aumentar, ha disminuido considerablemente. Consecuentemente, la red seleccionada para el proyecto es la NasNetMobile con el entrenamiento realizado únicamente con el banco de imágenes obtenidas con la cámara del robot, directamente del escenario construido.

Figura 28. Imágenes de *kaggle.com*

Figura 29. Imágenes tomadas por la cámara del robot

```
Label Obligatorio derecha: Acc: 0.947, Prec: 0.667, Rec: 0.400, F1 0.500.
Label Obligatorio izquierda: Acc: 0.900, Prec: 0.839, Rec: 0.722, F1 0.776.
Label Prohibido derecha: Acc: 0.820, Prec: 0.511, Rec: 0.857, F1 0.640.
Label Prohibido izquierda: Acc: 0.827, Prec: 0.625, Rec: 0.179, F1 0.278.
Label Prohibido recto: Acc: 0.993, Prec: 0.960, Rec: 1.000, F1 0.980.
Label Obligatorio recto: Acc: 0.940, Prec: 0.727, Rec: 1.000, F1 0.842.
Average: Acc: 0.713, Prec: 0.721, Rec: 0.693, F1 0.669.
```

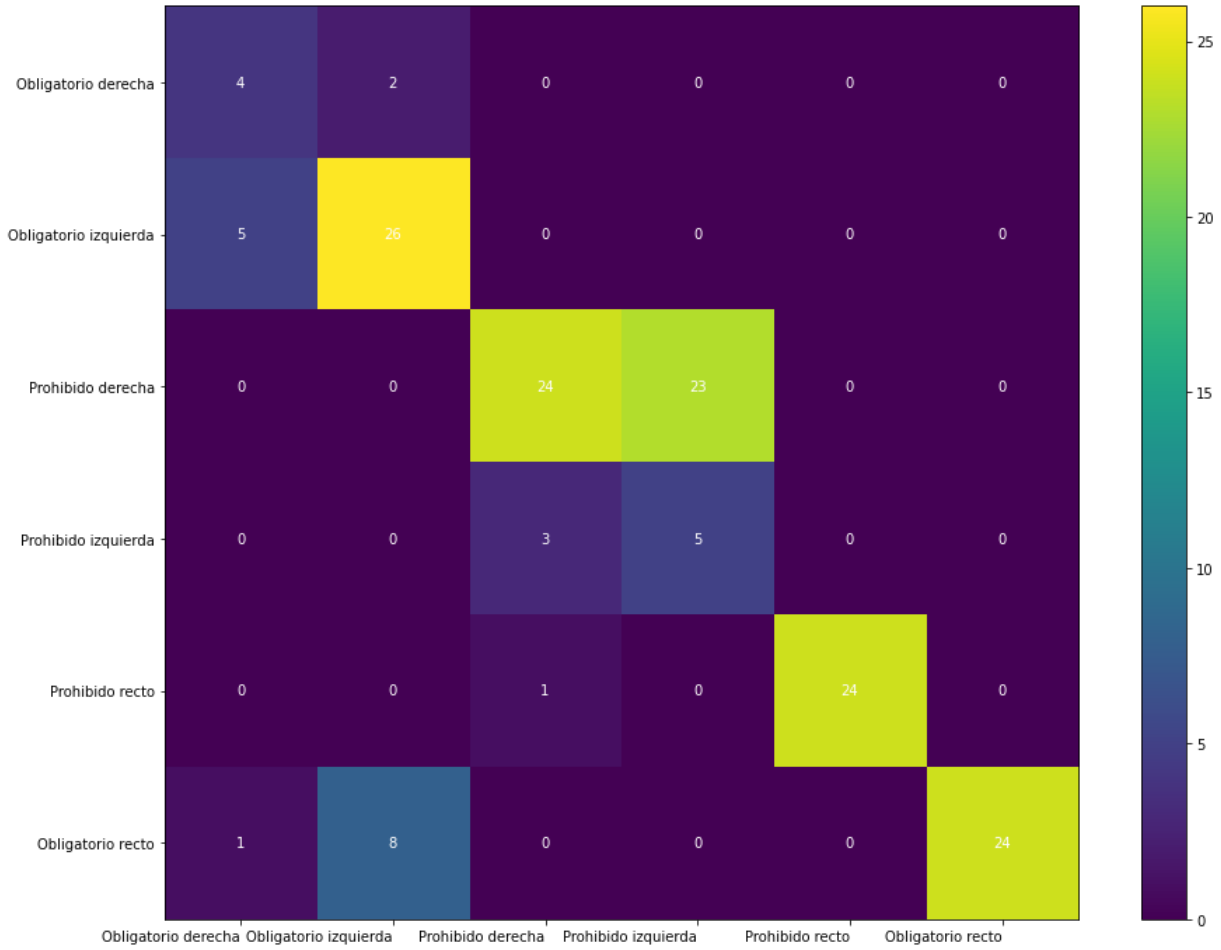


Figura 30. Matriz de confusión y métricas de la red entrenada con imágenes combinadas

4. Construcción del entorno

Seguidamente se va a detallar cómo se ha planteado y llevado a cabo el diseño del entorno por donde se moverá el robot. El objetivo de este proyecto es simular el comportamiento de un vehículo en un entorno urbano. Dicho entorno será conocido, es decir, al robot será necesario pasarle la topografía del entorno en un formato que pueda codificar.

4.1. Arquitectura del entorno

El medio físico a través del cual se va a mover el robot constituye una de las decisiones más importantes del proyecto ya que todos los movimientos del robot se pueden ver condicionados por el entorno. Para ajustar el proyecto todo lo posible a la realidad, se ha creído conveniente hacer la simulación en un entorno urbano existente.

Figura 31. Vista aérea del distrito del Ensanche en Barcelona

Figura 32. Vista aérea de Ciudad Rodrigo

Cabe diferenciar dos tipos (según su arquitectura) de ciudades existentes en la actualidad: las ciudades planificadas y las no planificadas. Las ciudades planificadas son ciudades creadas en un terreno previamente no urbanizado, con un propósito determinado y de acuerdo con un plan urbanístico global. Su desarrollo depende, por lo tanto, de una decisión administrativa y no del movimiento natural de la población [24]. Mientras que las ciudades no planificadas se identifican sobre todo con los cascos antiguos de ciudades o ciudades en otras épocas más amuralladas. Las características de cada tipo de arquitectura están bien diferenciadas de la otra como se aprecia en la Fig. 31 y Fig. 32, mientras que en las ciudades planificadas las calles forman paralelas y perpendiculares entre ellas, en las no planificadas son irregulares y no siguen ningún patrón. La elección de implantar la arquitectura de ciudad planificada como escenario para la simulación no viene únicamente de la sencillez para la generación del software, sino por la facilidad de extrapolarlo a otros escenarios similares (bastaría con adaptar el mapa para usar el mismo software) y por la tendencia de evolución en la sociedad actual donde cada vez circulan menos vehículos por el centro de las ciudades (donde las calles son más irregulares por pertenecer al casco antiguo de estas) y las nuevas construcciones presentan una arquitectura de ciudad planificada. Como aliciente para esta elección de escenario también ha tenido influencia que la idea de movilidad autónoma puede darse lugar en grandes naves industriales utilizadas para almacenes (que suelen tener una organización similar a la de una ciudad planificada) o incluso en hostelería como por ejemplo restaurantes o grandes eventos (bodas, bautizos, etc.).

Por lo tanto, el mapa del proyecto se va a particularizar, en la medida de lo posible, para el entorno urbano de Miranda de Ebro, Burgos (Fig. 33). Se elige este escenario por ser conocido y tener así una representación mucho más visual.

Figura 33. Vista aérea Miranda de Ebro.

Para acabar este apartado conviene destacar que aunque se represente y se vaya a realizar la movilidad sobre un escenario real y conocido, las direcciones de circulación no van a coincidir con las direcciones reales, principalmente porque interesa, en términos de robustez del software, probar diferentes configuraciones de las señales de tráfico que son las que determinan el sentido de circulación. Además, se dota al sistema de un mecanismo reactivo ante cambios puntuales en el sentido de circulación, como los debidos a obras en la calzada.

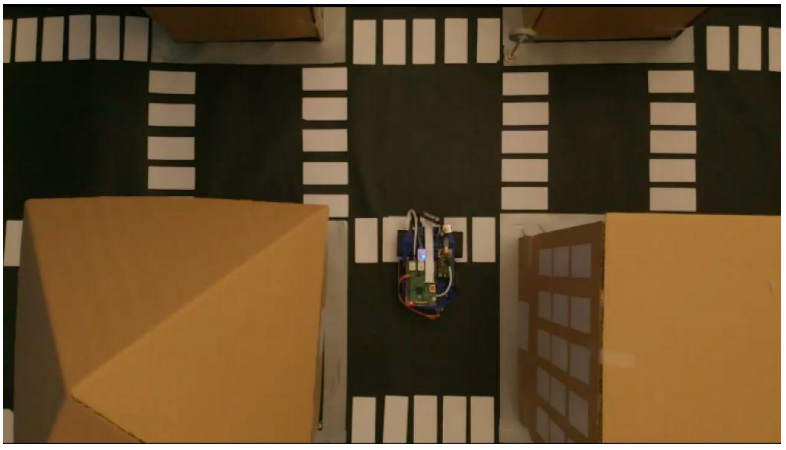


Figura 34. Vista aérea del robot en el escenario construido.

Para la construcción del entorno se han usado cajas de cartón, un papel con una fricción suficiente para impedir que las ruedas del robot deslicen alterando así la medida del encoder y por tanto la odometría, pegatinas de color blanco para simular pasos de cebra y ventanas de los edificios para darle una perspectiva más real y, por último, se han construido 6 señales de tráfico que se podrán mover fácilmente de una intersección a otra y comprobar fácilmente el buen funcionamiento del robot. El mapa por lo tanto tendrá 4 calles horizontales de aproximadamente 3 metros de largo y 5 calles verticales de unos 2,4 metros de largo resultando un total de 20 intersecciones contando las de las esquinas.

En la Fig. 35 a la izquierda se ilustra como quedó el escenario donde se añadieron elementos típicos de la ciudad aportando un enfoque más real (Fig. 35 derecha). Cabe destacar que el escenario a parte de intentar ser lo más ajustado a la realidad posible, tiene las dimensionas lo más grandes posibles.

Figura 35. Escenario final por donde se moverá el robot.

4.2. Implementación y restricciones del algoritmo

Después de haber elegido el tipo de escenario para la simulación del robot, el siguiente paso lógico es la codificación e implementación de ese mapa en la raspberry. Para dotar al proyecto de una mayor flexibilidad, se comenta en el apartado anterior que aparte de poder aplicar el software en diferentes ciudades (planificadas) también podría resultar útil en otras configuraciones como por ejemplo almacenes, oficinas, restaurantes, etc. En definitiva, será útil para modelos de mapa que presenten un patrón regular o simétrico que sea fácilmente codificable.

Es por esto que se hace necesario un método que no sea muy complejo para poder trasladar el mapa real a la unidad de control que en este proyecto es la raspberry. La decisión que se ha tomado es representarlo en forma de matriz donde los elementos de esa matriz sean las intersecciones entre calles. Primeramente, se debe conocer el número de nudos tanto horizontales como verticales del mapa que se quiere codificar. La primera suposición que se hace es que todas las calles paralelas son similares en anchura y longitud. Existe la posibilidad de que no todas las calles sean de igual longitud o anchura, pero no causaría problema ninguno. Lo primero porque bastaría con diferenciar esas calles dentro de la parte del software donde se suministra potencia a los motores de las ruedas que es donde se tiene en cuenta la distancia que debe avanzar el robot, es decir, aunque en la matriz que representemos todos los nudos estén a la misma distancia en la realidad podría no ser así. Como esto no es un factor importante a la hora de extraer

```
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0
```

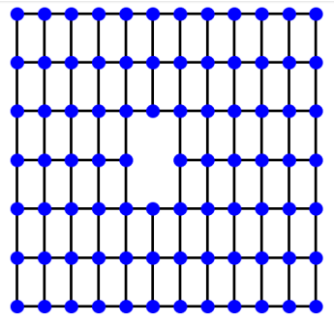


Figura 36. Mapa codificado.

conclusiones en el proyecto se van a considerar todas las calles de igual medida. La segunda razón para no considerar esto como un problema es que la matriz de nudos que se construye se está construyendo teniendo en cuenta la calle más pequeña, por lo tanto habrá nudos en la matriz que no se van a corresponder con ninguna intersección en la realidad (luego se analiza con más detalle este concepto).

Después de saber el número de nudos que hay, es necesario escribirlos en un editor de texto y guardar el archivo como *.txt* para que se pueda procesar correctamente. En este archivo los nudos van a aparecer representados con "0" y el propio programa los conectará con sus vecinos. Se entiende por vecino de un nudo los nudos que están inmediatamente a su izquierda, a su derecha, encima y debajo, lo que se conoce como 4-vecindad. Queda visto entonces, que el programa construirá una matriz de nudos a partir de un fichero *.txt* donde estos deben aparecer representados como ceros. De esta manera se ve reflejado en la Fig. 36 como se codifica un mapa.

En esta figura también se puede ver de qué forma se tendrían en cuenta elementos que es usual encontrarse en los centros urbanos y que no representan calles ni intersecciones, por ejemplo plazas o parques. Con este método se deja explicado cómo se construiría un mapa de las dimensiones que se desee y con las intersecciones que se desee. Volviendo a la figura se aprecia que, siendo el nudo (0,0) el de la esquina superior izquierda, el nudo (3,5) no existe por lo tanto el programa no puede calcular una posible ruta usando ese nudo. Se detalla en el apartado siguiente cómo gestiona el programa un mapa donde no todos los elementos se corresponden con nudos.

Cuando se habla de patrón regular en la arquitectura de las calles es necesario aclarar que no se refiere a únicamente a un diseño de calles rectangulares donde todas ellas formen entre sí ángulos de noventa grados. Estas calles pueden presentar diagonales como se ilustra en la Fig. 37. Bastaría con considerar como vecinos los de las diagonales, lo que se conoce como 8-vecindad y habilitar conexiones entre esos nudos. Ya que el software seguiría siendo prácticamente el mismo.

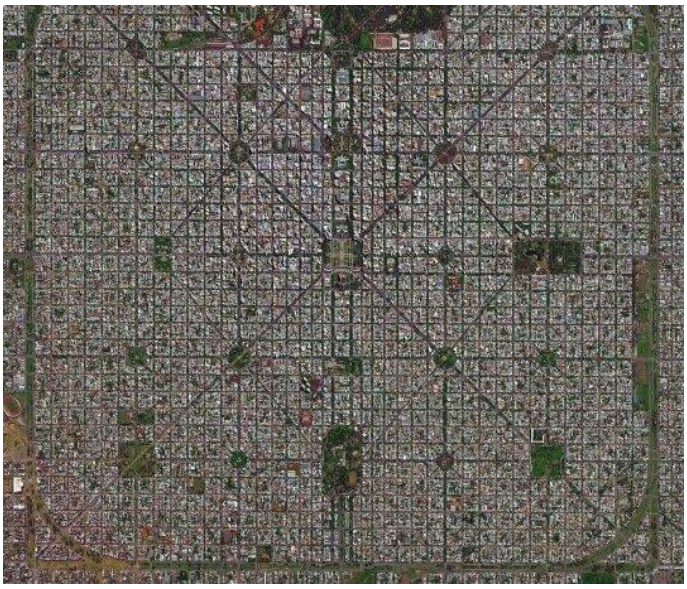


Figura 37. Vista aérea de La Plata, Argentina.

Particularizando el mapa que se ha usado en el proyecto, se ilustra en la Fig. 38 como sería el mapa real con la matriz de nudos superpuesta. En la imagen se aprecia como existen nudos en el mapa que en la realidad no son intersecciones. En estos nudos especiales, además de no existir ninguna señal de tráfico, se habrá inhabilitado mediante

código la conexión con otros nudos con los que no existe conexión real a través de una calle, en este caso conexiones horizontales.

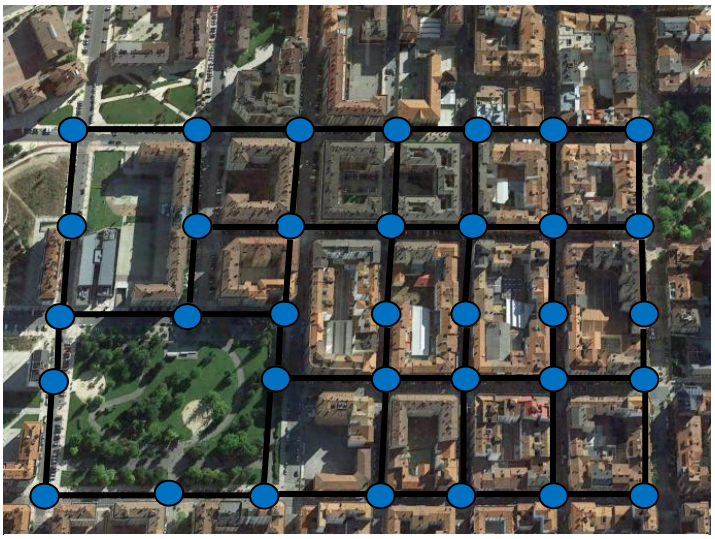


Figura 38. Mapa codificado superpuesto al real.

Por último, volviendo al capítulo 2 donde se habla de la localización del vehículo y teniendo en cuenta como se ha modelado el mapa y cuáles son los elementos más importantes de él: los nudos. En cada nudo el robot, a través de su cámara, comprobará si el movimiento que tiene planeado según su ruta (esto se explica con más detalle en el capítulo siguiente) está permitido, bien sea avanzar recto o realizar un giro, mirando las señales de tráfico que se encuentran en esa intercepción. Por lo tanto, sabiendo que el robot en cada nudo va a hacer uso de la cámara y va a realizar un reconocimiento de imágenes sería lógica la idea de implantar marcas en cada intersección para comprobar si los parámetros odométricos son correctos. Pero como la raspberry tiene una capacidad de cómputo no muy potente y la odometría es bastante precisa no se ha llegado a implantar ese sistema de localización externa. Sería un método útil si el terreno fuera irregular o las ruedas deslizaran, de ahí esta breve explicación. Consecuentemente los únicos elementos de este proyecto con los que el robot interactuará estarán en las intersecciones y serán señales de tráfico.

5. Diseño de la algoritmia

Una vez construido el escenario y elegida la red neuronal con la que se va a trabajar se desarrolla el software que gobernará el proyecto. Detallado con profundidad en el Anexo B se explicarán los módulos principales en este apartado de la memoria.

Lo primero, por ser también la base del proyecto, es generar el código necesario para la localización del robot. El software, que se ha generado con programación orientada a objetos, distingue dos clases: una para la localización y el movimiento del robot y otra para todo lo relacionado con el mapa. Para el cálculo de los parámetros odométricos basta con implementar las ecuaciones detalladas en el capítulo 2 e ir actualizándolos a medida que se mueva el robot. Esto se realiza en un proceso paralelo al programa main. Una vez conseguida la correcta localización basada en la medida del encoder, el siguiente paso es cargar el entorno de simulación.

Para que el robot codificara el mapa se trasforma el fichero *.txt* de nudos en una matriz a través de la cual el robot calculará la ruta a seguir para llegar al destino final. Si ninguna señal o nudo inexistente impidieran el correcto recorrido de la ruta, no es necesario recalcularla. Cuando sí que es necesario hacer un recalculo de la ruta es cuando a través de la cámara se procesa y clasifica una señal de tráfico que impida el movimiento que el robot iba a realizar para seguir la ruta. También se recalculará la ruta si el robot intentara acceder a un nudo que no existiera en la realidad. El proceso de recalcular la ruta sucede una vez el robot ya haya realizado el movimiento de ir a un nudo cercano respecto al nudo que intentó moverse.

Por último, se han desarrollado varias funciones que ayuden al usuario a elegir la velocidad de movimiento, a conocer el nudo en el que se encuentra el robot o incluso, que blinden determinados movimientos como por ejemplo acceder a un nudo marcha atrás. En la evaluación experimental, la única función que se ha usado ha sido la que permite elegir la velocidad ya que se han realizado las trayectorias con diferentes velocidades para ofrecer una mejor comparativa de la precisión del robot. No obstante, las demás funciones se incluyen, como el resto del software, en los anexos.

6. Evaluación experimental

El último paso de este proyecto es llevar toda la teoría desarrollada en la memoria al escenario físico para analizar y extraer conclusiones del proyecto llevado a cabo en este tiempo. Para ello se van a realizar una serie de pruebas donde se monitorizarán los parámetros más importantes como los parámetros odométricos, el tiempo total invertido en realizar el movimiento completo, la representación de la trayectoria recorrida, etc.

6.1. Movimientos base

En primer lugar, se empiezan con los movimientos más simples como son la línea recta o los giros de noventa grados. Es aquí donde se realizan las pruebas con las tres diferentes velocidades posibles a elegir por el usuario. Estas velocidades son lenta, media y rápida. La velocidad rápida se corresponde con la máxima potencia que puede suministrar el motor Dagu que son 5V y se representa con 8 bits (255 unidades). Cuando se habla de velocidad media se proporciona la mitad de potencia que para velocidad rápida, lo que serían 127 unidades y, por último, la velocidad lenta es la mitad de la velocidad media, o sea, 63 unidades. Haciendo un ensayo en vacío (sin fricción en las ruedas), es decir, con el robot suspendido en el aire recorre 85 centímetros en unos 6 segundos lo que resulta unos 14 cm/s de velocidad media.

```
Modo de funcionamiento: velocidad estandar
PID:  1452
yendo recto
tiempo: 6.247447490692139
TH: 1.5707963267948966
Y: 0.859556746019836
X: -0.031467977618128674
```

Figura 39. Parámetros después del movimiento en línea recta

Como se aprecia en la Fig. 39 (donde la *x* y la *y* vienen expresadas en metros, la *th* en radianes y el tiempo en segundos), el avance en línea recta se ha hecho de manera positiva en la coordenada *y*, por lo tanto no resulta difícil deducir que el ángulo inicial es pi/2 que equivale a 1,57. La coordenada en *x* que debería ser cero si fuera una línea recta perfecta vale -0.03 metros lo que significa que el robot se ha desviado 3 centímetros durante el trayecto. Si el robot sufre un desvío lo más lógico es pensar que el ángulo también debería estar desviado con respecto al inicial pero no es así porque, como se

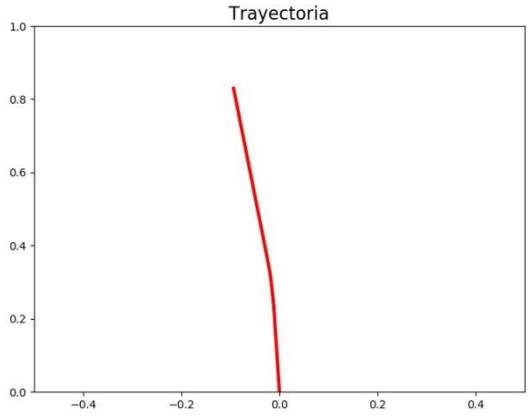


Figura 40. Trayectoria del robot en el movimiento de línea recta

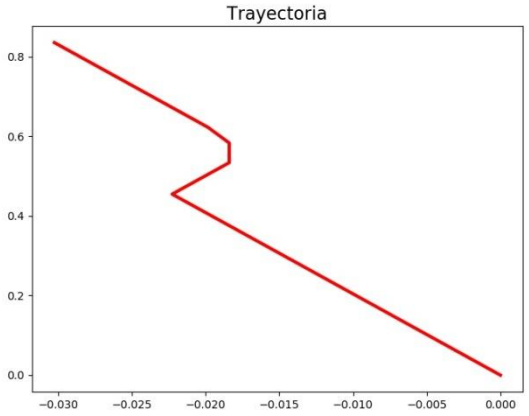


Figura 41. Trayectoria ampliada

ilustra en la Fig. 40 y Fig. 41, el regulador de velocidad interno de los motores hace una pequeña re dirección y el robot vuelve a llevar una trayectoria recta con la orientación adecuada. Esta escena se aprecia muy bien en el vídeo: Línea recta (3 m.) a velocidad estándar.

Las siguientes imágenes muestran los mismos parámetros pero a diferentes velocidades. Como se puede ver, a velocidades más altas el robot sufre un mayor desvío de todos los parámetros (Línea recta (3 m.) velocidad rápida). A velocidades más lentas (Línea recta (3 m.) a velocidad lenta) tiene mucha mayor precisión (como es obvio) pero hay que encontrar un equilibrio entre precisión y tiempo de ejecución porque si se realizara todo el recorrido a esa velocidad tardaría mucho y sería el proyecto muy poco eficiente. Por lo tanto la velocidad por defecto se establece la velocidad media, es decir, a 14 cm/s frente a los menos de 9 cm/s de la velocidad lenta. Aún así, se dejará al usuario poder elegir cualquiera de las tres velocidades disponibles.

```
Modo de funcionamiento: velocidad rapida
PID:  1584
yendo recto
tiempo: 4.917536497116089
TH: 1.9625190990721244
Y: 0.8712794011872046
X: -0.20180545872436084
```

Figura 42

```
Modo de funcionamiento: velocidad lenta
PID:  1780
yendo recto
tiempo: 9.58420467376709
TH: 1.4728656337255897
Y: 0.8545460435890843
X: 0.0137974978787180l6
```

Figura 43

Una vez establecida la velocidad se muestran las diferentes gráficas y parámetros que resultan al monitorizar los movimientos de giro a la derecha y giro a la izquierda como se muestra en los vídeos. Para estos giros también se ha tenido en cuenta las tres posibles velocidades pero se llegó a la misma conclusión.

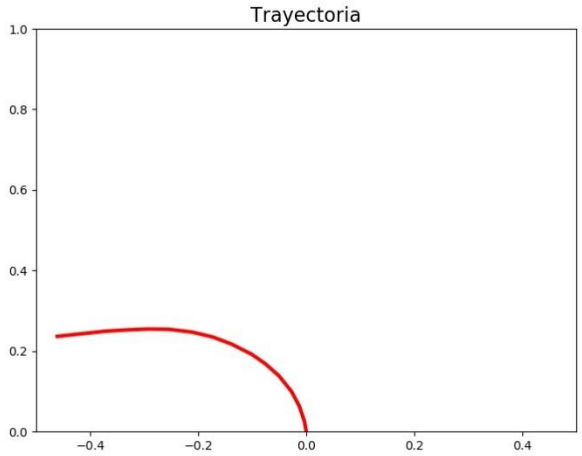


Figura 44. Trayectoria curva

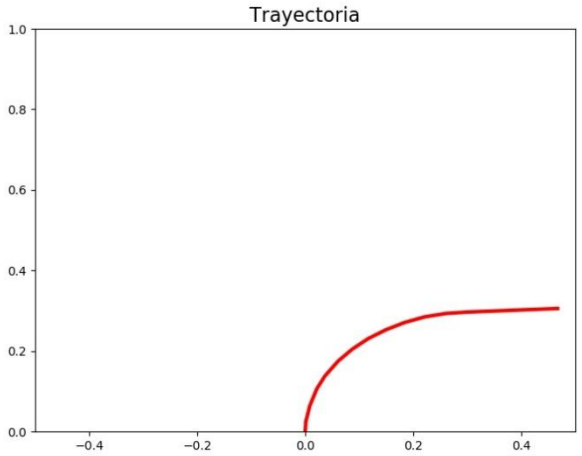


Figura 45. Trayectoria curva

```
Modo de funcionamiento: velocidad estandar
PID:  1915
1.5707963267948966
girando a la izquierda
tiempo: 4.836966276168823
TH: -3.0475671982064694
Y: 0.2319279568788356
X: -0.4902562833779146
```

Figura 46

```
Modo de funcionamiento: velocidad estandar
PID:  2048
1.5707963267948966
girando a la derecha
tiempo: 5.0869481563568115
TH: 0.0039052376859857463
Y: 0.3069170243879016
X: 0.4968130281089631
```

Figura 47

En referencia a la Fig. 46 y 47 cabe explicar todos los elementos que aparecen en la imagen. En la primera línea se muestra la velocidad que el usuario ha decidido escoger, en la segunda línea aparece PID que son las siglas de *Process IDentifier* y es un número utilizado por la mayoría de sistemas operativos para identificar de forma exclusiva un proceso activo (en este caso la actualización en paralelo de la odomería), el número puede utilizarse como parámetro en varias llamadas a funciones, lo que permite manipular los procesos, por ejemplo, ajustando la prioridad del proceso o matándolo por completo. En la tercera línea aparece el valor de θ antes de iniciar el movimiento. En la cuarta línea aparece el tipo de movimiento que se está realizando y en la quinta el tiempo que ha tardado en realizar ese movimiento. Por último se muestra el valor final de los tres parámetros odométricos al finalizar el movimiento. Enlace a vídeos de los movimientos base:

- Trayectoria recta:
  - 5 calles:
    - Velocidad lenta: Línea recta (3 m.) a velocidad lenta.
    - Velocidad media: Línea recta (3 m.) a velocidad estándar.
    - Velocidad rápida: Línea recta (3 m.) velocidad rápida.
  - 1 calle: Recorre una calle a velocidad estándar
- Curva:
  - Giro a la derecha:
    - Velocidad lenta: Giro derecha velocidad lenta
    - Velocidad media: Giro derecha velocidad estándar
    - Velocidad rápida: Giro derecha velocidad rápida
  - Giro a la izquierda:
    - Velocidad lenta: Giro izquierda velocidad lenta
    - Velocidad media: Giro izquierda velocidad estándar
    - Velocidad rápida: Giro izquierda velocidad rápida

6.2. Ruta completa

Finalmente, después de analizar los movimientos básicos y ver su correcto funcionamiento procede hacer la ruta completa y comentar sus parámetros más relevantes. Para el experimento se han realizado tres rutas distintas y se ha grabado desde diferentes cámaras para aportar un análisis más completo. Los primeros experimentos han sido sin hacer el reconocimiento de imágenes garantizando así que el robot es lo adecuadamente autosuficiente para calcular una ruta y llegar al nudo definido como lugar final por el usuario. Así lo muestran los siguientes vídeos:

- **Ruta 1:**

- Lugar de inicio: [2,1]
- Orientación: - π/2 rad
- Lugar final: [0,0]
- Ruta calculada por el robot: [2,1] [2,0] [1,0] [0,0]
- Enlace al video: Ruta 1

- **Ruta 2:**
  - Lugar de inicio: [1,1]
  - Orientación: π/2 rad
  - Lugar final: [2,3]
  - Ruta calculada por el robot: [1,1] [1,2] [1,3] [2,3]
  - Enlace al video: Ruta 2

- **Ruta 3:**
  - Lugar de inicio: [2,1]
  - Orientación: 0 rad
  - Lugar final: [1,2]
  - Ruta calculada por el robot: [2,1] [2,2] [1,2]
  - Enlace al vídeo: Ruta 3

Después de calcular y realizar las rutas correctamente sin hacer uso de la cámara, se va a mostrar dos rutas donde, haciendo uso de la cámara, el robot tenga que recalcular o no ruta en función de las señales de tráfico que procese a través de la cámara. La manera de ilustrarlo será igual que la anterior pero se añadirá una fila más que proporcionará la ruta que finalmente ha seguido el robot si es que hubiera tenido que recalcular.

- **Ruta 4:**
  - Lugar de inicio: [1,1]
  - Orientación: π/2 rad
  - Lugar final: [0,2]
  - Ruta calculada por el robot: [1,1] [1,2] [0,2]
  - Ruta realizada por el robot: [1,1] [1,2] [1,3] [0,3] [0,2]
  - Enlace al vídeo: Ruta 4 Se incluye un vídeo con una cámara situada a la altura del robot: Visión robot
  - Señal detectada**:**

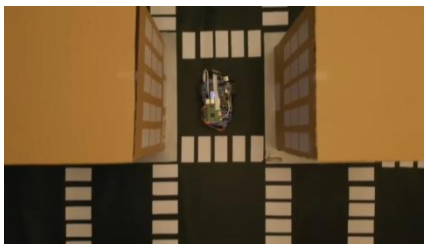
Figura 48. Señal de obligatorio recto detectada

- **Ruta 5:**
  - Lugar de inicio: [2,1]
  - Orientación: - π/2 rad
  - Lugar final: [3,3]
  - Ruta calculada por el robot: [2,1] [3,1] [3,2] [3,3]
  - Ruta realizada por el robot: [2,1] [3,1] [3,2] [2,2] [2,3] [3,3]
  - Enlace al vídeo: Ruta 5. Se incluye un vídeo con una cámara situada a la altura del robot: Visión robot
  - Señal detectada**:**

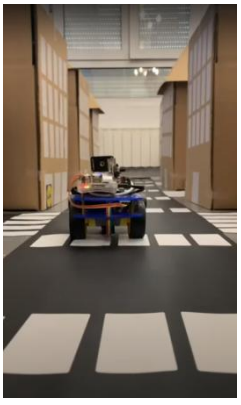
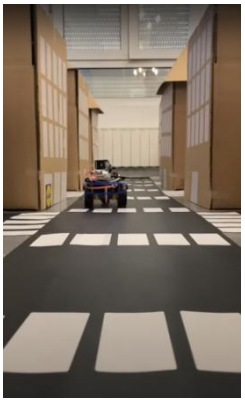
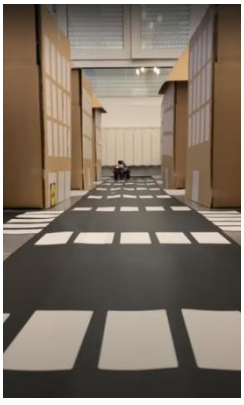
Figura 49. Señal de prohibido detectada

En el video: Ruta 6 se puede ver de qué manera queda resuelto el problema planteado en el anexo B4.4 donde el robot, ante la imposibilidad de ir hacia atrás, se movía al nudo de la izquierda o de la derecha y recalculaba la ruta entendiendo como nudo inicial el nudo al cual se había desplazado.

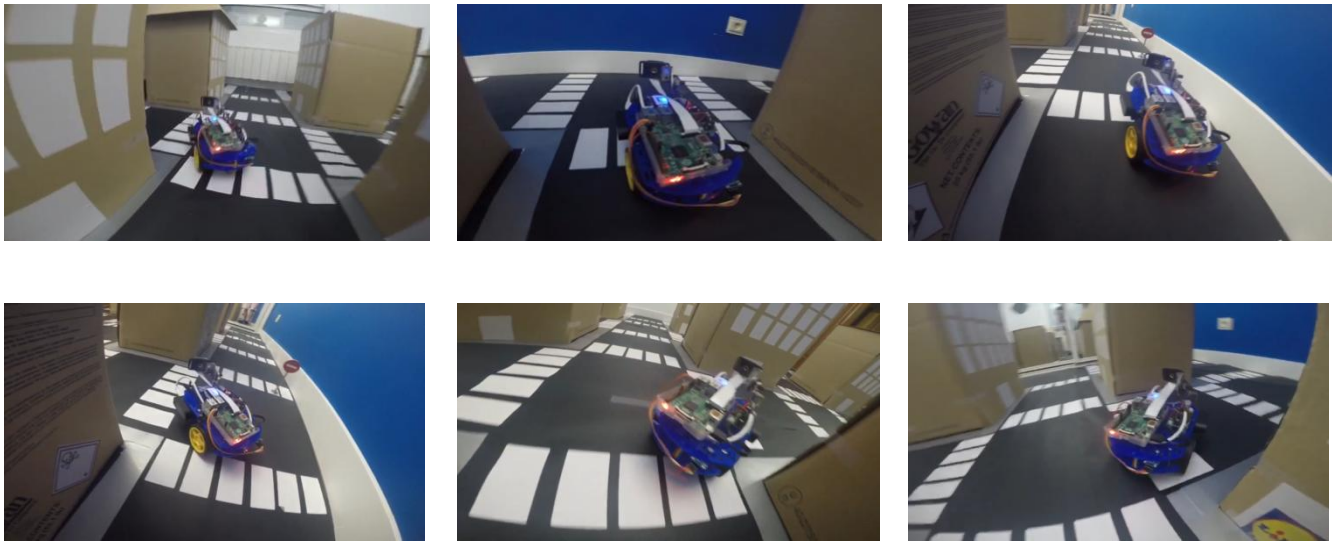
Se adjunta una pequeña galería de imágenes de la Ruta 2, de la Ruta 5 y del movimiento de línea recta:



Figuras 50, 51 y 52. Ruta 2 (sin señales)

Figuras 53, 54, 55 y 56. Línea recta (3 m.)



Figuras 57 a 62. Ruta 5 (con señales) vista desde el robot.

Se observa como, en la Ruta 5, se rectifica la ruta calculada por haber detectado una señal de prohibido el paso. En este caso el robot hubiera avanzado recto una calle para llegar al nudo final ([3,3]). Al impedir la señal este movimiento, el robot recalcula la ruta, con el mismo método que había sido calculada al principio del movimiento pero tomando como nudo inicial el nudo en el que se encuentra, es decir, el nudo donde se detecta la señal de prohibido el paso.

7. Conclusiones y trabajo futuro

Como conclusiones de este proyecto se consideran cumplidos los objetivos propuestos antes de comenzar el desarrollo del trabajo. Se ha conseguido realizar una simulación completa de navegación dentro de un entorno urbano real satisfaciendo los requerimientos solicitados por el usuario.

Se ha logrado localizar al robot a lo largo de todo el escenario de una manera bastante exacta (como se ilustra en el capítulo 2) utilizando únicamente un encoder de 20 marcas. Se conocen del robot en todo momento sus coordenadas y su orientación con un pequeño error (como se ve en el capítulo 6) que no influye en el objetivo principal del robot que es llegar a un destino solicitado por el usuario. Es preciso destacar el buen funcionamiento del encoder atendiendo al período de muestreo de la raspberry y en especial a las buenas condiciones del entorno en el que se ha desarrollado la simulación ya que el suelo no permitía el deslizamiento de las ruedas ni tenía irregularidades que alteraran el buen funcionamiento del encoder.

También se ha logrado construir una red neuronal con prestaciones altamente fiables y eficientes a nivel de cómputo porque la problemática de este método de clasificación de imágenes residía principalmente en el tiempo que tardaba en clasificar una imagen. Se solucionó esta problemática haciendo la clasificación con otra red neuronal más ligera. Esta quizás sea la parte que más ha retrasado la finalización del proyecto ya que era el campo más desconocido para mí y precisaba de bibliotecas con versiones muy recientes e incompatibles con el sistema operativo que se estaba utilizando. Finalmente se destaca la dificultad que existe a la hora de entrenar una red neuronal y lograr una precisión fiable ya que se hace necesaria una base de datos muy extensa con todo tipo de situaciones en las que puede encontrarse una señal de tráfico.

Por último, y aunque menos importante a nivel de objetivos, cabe destacar la construcción a escala de un entorno urbano real, en este caso Miranda de Ebro (Burgos) para hacer una simulación más aproximada a la realidad. Dicho escenario construido a escala se ha adaptado a las medidas del robot que va a simular y al espacio disponible habilitado para el proyecto. Se entiende que se desarrolla un entorno urbano, aparte de por ser más útil, por ser un entorno más difícil que entornos interurbanos donde hay menos giros y menos factores externos.

En conclusión, se logra localizar en todo momento y a bajo coste un robot de tracción diferencial (que a su vez también constituye la forma más barata de construcción de un robot móvil). Se logra la simulación de una conducción autónoma en un entorno urbano y la interacción del robot con elementos usuales de ese entorno basada en visión por computador y Deep Learning.

Referente a lo personal se destaca la grata sensación de haber abordado un proyecto tan amplio y que pertenece a un campo de estudio que se encuentra en total desarrollo y expansión porque cada vez en más operaciones se usan robots móviles y decisiones en base al Deep Learning. Se ha aprendido a realizar un proyecto desde cero, sin directrices ni objetivos concretos, donde se empezó desmontando el robot para analizar el funcionamiento de cada elemento y se puso fin con un robot que es capaz de, de manera totalmente autónoma, calcular rutas, distinguir señales de tráfico e interactuar con elementos externos, recalcular dicha ruta si fuera necesario y realizar el mismo movimiento a diferente velocidad según la requerida por el usuario.

Para finalizar, en referente a los objetivos de la simulación del experimento, se concluye que no sólo estos sistemas de movilidad autónoma han llegado para quedarse sino que actualmente no se ha puesto en escena más que una pequeña parte de todo el potencial que poseen estos sistemas robóticos autónomos. Porque si una sola persona como soy yo (sin conocimientos excesivamente profundos) ha conseguido un robot con movilidad autónoma y capaz de aprender a diferenciar señales de tráfico en un curso donde se ha tenido que alternar con asignaturas, prácticas en empresa y el resto de factores externos que pueden influir en el desarrollo de un proyecto, qué no puede hacer un grupo de personas suficientemente cualificadas totalmente centradas en el proyecto de conducción autónoma.

Como opinión personal, el único problema que puede retrasar la instalación definitiva de esta conducción es la toma de decisiones en situaciones de, por ejemplo accidente inminente, donde entran capacidades personales como la moral o el instinto.

A lo largo de la memoria se describen tanto la alteración de la odometría que pueden producir los encoders como el error acumulado durante el movimiento, esto nos sirve para comentar el trabajo futuro para mejorar este proyecto. El error acumulado se produce porque los motores de las ruedas no suministran exactamente la misma potencia, el deterioro de las ruedas (se ve en los vídeos la holgura que presentan al moverse) tampoco es el mismo por lo tanto es usual que, por ejemplo al seguir una trayectoria recta, se produzca un ligero error que se irá haciendo más grande a medida que se recorre más distancia. Es por esto que se propone un control de trayectorias basado en la cámara donde se calcule la potencia necesaria a suministrar a cada rueda en función de lo centrado que esté el robot en la carretera. Se valora también, como alternativa más sencilla, la sobre escritura de los parámetros odométricos en función de visión por computador y marcas ArUco (o similares) puestas en cada intersección.

Se propone también un sistema multi-robot donde varios robots circulen simultáneamente por el mismo entorno urbano interactuando entre ellos. Este método, viendo el desarrollo actual, sería interesante desarrollarlo con Deep Learning por ser una herramienta tremendamente útil y que facilita en gran medida la generación de código.

Por ende, espero sirva este trabajo como base e inspiración futura a los posteriores alumnos que tengan la oportunidad de trabajar con el proyecto PiRobot o similares ya que considero que es un campo con mucha capacidad de desarrollo y mejora.

8. Observaciones

Como finalización del proyecto se va a desarrollar brevemente los detalles o sucesos más destacables que han formado parte de este TFG y el posible trabajo a realizar sobre la base de este funcionamiento para mejorar la eficiencia y prestaciones del robot.

El primer acto destacable en cuanto a la problemática se produce al comienzo del proyecto cuando el objetivo a corto plazo residía en realizar movimientos rectilíneos con el robot contar las marcas del encoder. Las marcas no actualizaban y no había manera de medir la distancia. El problema residía en la actualización de variables así que en el código para contar las marcas del encoder no aparece ninguna variable asociada sino que aparece directamente el pin de entrada, por ejemplo *GPIO.input(20)*. Una vez aclarado por qué aparece la entrada directamente, resulta de interés aclarar los cambios de versión del sistema operativo de la raspberry.

El proyecto comenzó desarrollándose en Raspbian 8 (Jessie) y Python 3.4 pero los primeros problemas con esta versión aparecieron con la instalación de las bibliotecas keras, tensorflow y pip (que es un instalador de paquetes) puesto que se requerían las versiones 2.3, 2.2 y 19.0 respectivamente. Como con la instalación de estos paquetes surgieron bastantes errores de instalación se decidió actualizar toda la imagen de la raspberry instalando la última versión del sistema operativo: Raspbian 10 (Buster). Una vez aquí, se instalaron las versiones correspondientes de keras, tensorflow y pip sin problema y ya se pudo trabajar con redes neuronales para clasificación de imágenes.

Como última observación se destaca que, a día 20 de mayo de 2021, sigue habiendo una situación social complicada con una pandemia que limita considerablemente el uso de instalaciones y servicios públicos. Es por ello que la construcción del entorno físico se ha construido en una habitación de casa viéndose limitado su espacio. El mapa consta de 3x3 nudos y en ese escenario ha funcionado correctamente teniendo en cuenta todas las posibles situaciones. Extrapolando a un mapa más grande, se concluye que funcionaría correctamente porque el programa calcula sin problema la ruta independientemente de las dimensiones de la matriz de nudos por lo tanto no habría problema alguno para el robot en completar una ruta más larga más allá del error acumulado.

Apéndices

Apéndice A. Tecnología del robot móvil

A1. PiRobot como robot diferencial

Dentro de todas las posibles configuraciones y con las herramientas y el entorno de trabajo disponible, era lo más sensato el diseño de un robot de tracción diferencial frente a robots de diseño sincronizado (robots de tres ruedas donde todas ellas son de dirección y motrices y para realizar un cambio de trayectoria precisa girar todas sus ruedas de manera simultánea) o robots de diseño de coche (muy similar al anterior pero las ruedas direccionales necesitan estar unidas). Con la elección de tracción diferencial se consiguen hacer trayectorias rectas, curvas y giros sobre sí mismo sin necesidad controlar el giro de las ruedas.

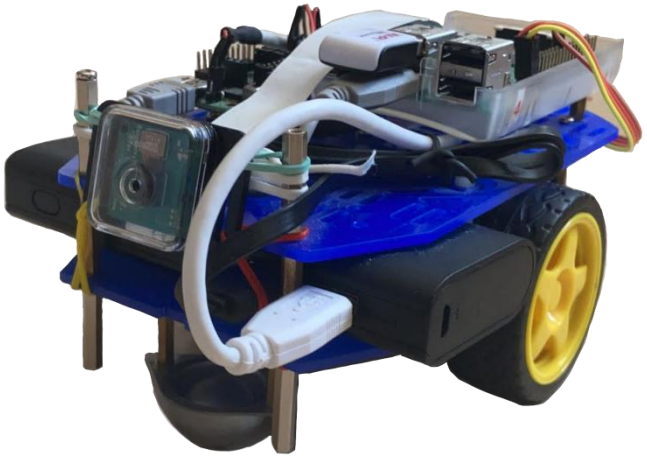


Figura A1. PiRobot con el que se ha realizado el proyecto.

Las configuraciones de tracción diferencial son muy populares y permiten calcular la posición del robot a partir de las ecuaciones geométricas, que surgen de la relación entre los motores y de la información obtenida de los encoders que normalmente llevan asociados a las ruedas. Este método de localización se conoce como estimación odométrica (Odometric Dead-Reckoning) en la que después profundizaremos.

Este tipo de robots pueden tener un único apoyo o dos. En este caso al ser un robot ligero es suficiente con un único apoyo formando los tres apoyos una distribución triangular. La otra distribución posible sería romboidal con dos apoyos pero al no tener un reparto de pesos que

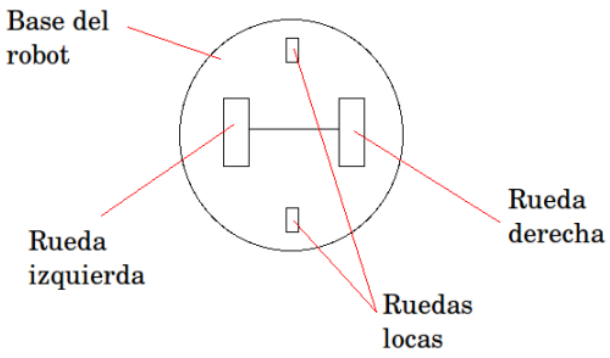


Figura A2. Planta de un robot de tracción diferencial

cause algún problema no se tendrá en cuenta ya que podría causar problemas de inadaptación al terreno si este fuera irregular.

Por lo tanto nuestro robot móvil queda construido con un diseño de tracción diferencial, capaz de satisfacer todos los requerimientos del proyecto y siendo suficiente para simular el comportamiento de un vehículo.

Como se aprecia en la Fig. A2 está formado por dos ruedas de 3,15 centímetros de diámetro alimentadas a través de dos motores (uno para cada rueda) que reciben la tensión de un elevador de tensión. La información de estas ruedas la recoge un encoder y es transmitida a la raspberry gracias a los pines de entrada y salida GPIO. El apoyo se realiza en la parte delantera con una rueda de apoyo de metal que facilitará el movimiento cuando el robot tenga velocidad angular a la vez que sirve de apoyo y distribuye el peso.

A2. Raspberry Pi como unidad de control

Una vez teniendo la configuración de diseño elegida se procede a realizar un pequeño análisis de la unidad de control elegida para el PiRobot.

Puesto que ya se ha hecho referencia de dicha unidad de control, no cuesta adivinar que se trata de una Raspberry Pi. En este caso se trata de una Raspberry Pi 2 Model B V1.1 que pertenece a la segunda generación de Raspberry Pi. Una Raspberry es un ordenador de placa única porque posee prácticamente todos los elementos de un ordenador excepto una placa de disco duro. En la Fig. A3 aparece una foto de la placa que se ha usado para el proyecto con sus conexiones diferenciadas.

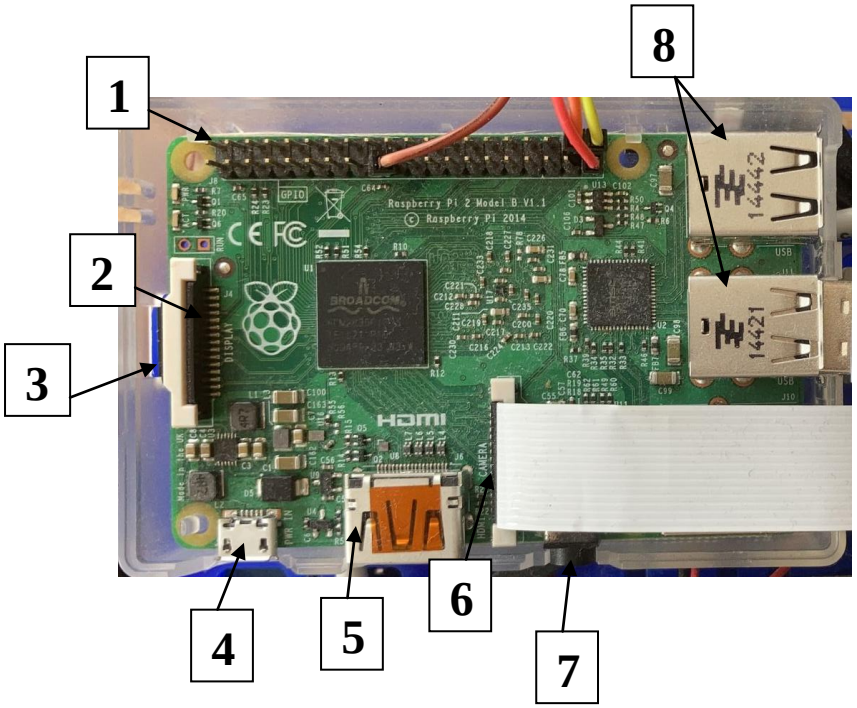


Figura A3. Placa de Raspberry Pi 2.

1. Puertos GPIO: “General Purpose Input/Output” como su nombre indica son pines de comportamiento general que se puede controlar con código y que sirven tanto de entrada como de salida. Son una de las principales diferencias respecto a un ordenador, ya que esto es una característica más propia de los micro controladores. En este modelo de placa los pines GPIO, que son pines macho, tienen dos modos de funcionamiento configurables vía software como

son BCM y BOARD. La opción BOARD especifica los pines por número del enchufe, es decir, el primero sería P1 y el último, en nuestro caso que tenemos 40, sería P40. La opción BCM especifica la referencia de los pines por el número del "Broadcom SOC channel". En la Fig. A4 se describe gráficamente. Para este trabajo usamos los pines GPIO20 y GPIO21 para leer las entradas del encoder y gestionar la localización del robot (capítulo 2.1).

Figura A4. Configuraciones posibles de los GPIO

2. Display: Es un puerto que permite la conexión entre la placa y una pequeña pantalla LCD o táctil con la que puede interactuar el usuario. No se usará en este proyecto.

3. Espacio para tarjeta micro SD: No se encuentra visible en la foto, sino que está en la parte de debajo de la placa. Esta tarjeta es la que contiene el sistema operativo o los archivos que guardemos porque recordamos que una característica de la raspberry era que no tenía placa de disco duro para almacenamiento. En este proyecto, siguiendo las recomendaciones se ha usado una tarjeta micro SD de más de 4 GB y clase 10. El proyecto se ha realizado con la última versión del sistema operativo para Raspberry Pi OS instalado con Raspberry Pi Imager que es la forma más fácil de instalar el sistema operativo. Cabe destacar que cuando se empezó el proyecto la versión del sistema operativo no era la más actual. Dicha actualización de la versión del sistema operativo se debe a la necesidad de instalar paquetes como Keras o Tensorflow que se comentará en el capítulo 3.

4. Micro USB: Es utilizado para la alimentación de la Raspberry. En el proyecto, como ya se ha mencionado se usa una batería portátil incluida en el robot para asegurar la perfecta movilidad por todo el escenario.

5. Conexión HDMI: Con un cable HDMI se puede conectar la raspberry a un monitor y trabajar en entorno gráfico, es decir se trabajará con una versión de escritorio. En el proyecto se ha instalado la versión de escritorio para trabajar en la generación de software a través del monitor. Para ejecutar el código se accede a la raspberry de manera remota ya que si estuviera conectada al monitor su movilidad se ería muy limitada.

6. Cámara: De igual manera que se dispone de una conexión para un Display también está habilitada la conexión para una cámara específica. Para poder usar la cámara debe estar activada la opción de uso de la cámara. No sería necesaria la instalación de ningún paquete más. En nuestro caso para tomar imágenes con la cámara y procesarlas se ha usado OpenCV [27] que es una biblioteca de código abierto para visión por computador.

7. Salida de audio: La raspberry dispone de una salida de audio que se realiza mediante un jack de 3,5 milímetros.
8. Cuatro puertos USB: En este proyecto se usan los cuatro puertos. Como ya se ha comentado la manera más frecuente de funcionar de la raspberry es con un monitor conectado y trabajando en el entorno gráfico por lo tanto es necesario el uso de un teclado y un ratón que ocupan dos puertos. Los otros dos están ocupados por un adaptador Wi-Pi que permitirá la conexión de la raspberry a internet y por la placa de alimentación de los motores de las ruedas que es un elevador de tensión.

Dentro del sistema operativo Debian Buster estamos usando el lenguaje de programación Python en su versión 3.7.0. Aunque también están instaladas las versiones 2.7 y 3.4 de Python se opta por la versión más reciente de las tres por su compatibilidad con los paquetes necesarios para el uso de redes como son Keras y Tensorflow. También ha sido necesaria la instalación de bibliotecas útiles en el proyecto como por ejemplo networkx que sirve para el uso de gráfos y redes en 2D y 3D (en este proyecto se usa para construir el mapa de actuación del robot), numpy, math o el ya mencionado OpenCV.

Apéndice B. Implementación algorítmica.

B1. Encoder para la localización

Una vez se ha visto y explicado la idea principal del proyecto resulta fundamental profundizar en detalle cómo se han ido implementando las diferentes ideas desarrolladas en los capítulos previos. Para ello se diferencian cuatro apartados en el capítulo. Cada apartado se corresponde con cada uno de los temas del proyecto desarrollados en los capítulos anteriores, de esta manera el primer apartado se detalla cómo se ha logrado localizar al robot en base a los encoders, en el segundo la forma de entrenar y trabajar con las predicciones de las redes neuronales, en el tercero cómo se ha conseguido implementar el mapa y en el cuarto todo lo que no se ha desarrollado en ningún apartado. Como en otro anexo se incluye el código detallado en este capítulo se ilustran las explicaciones a través de pseudocódigo.

B1.1. Funcionamiento en condiciones normales

El primer paso que se dio fue contemplar cómo funcionaban los encoders y de qué manera se codifican en la raspberry. Cada uno de los dos encoders va alimentado a 3,3 voltios que se obtienen de los pines GPIO de la raspberry y tienen un contacto de toma de tierra a través de un módulo de salidas digitales. Luego cada encoder ocupa un pin de los ya mencionados GPIO configurados como entrada. De esta manera el encoder de la rueda derecha será el GPIO21 y el de la rueda izquierda será el GPIO20. Se entiende por rueda derecha la rueda que queda más a la derecha en una vista de planta del robot, es importante esta diferenciación ya que de ese convenció depende el signo del ángulo θ que es el que nos determina la orientación del robot. En la Fig. B2 se observa cómo se ilumina el LED del módulo de salidas digitales en función de si llega al receptor el haz de luz que sale del emisor en una primera prueba para ver cómo funciona la tecnología del encoder.

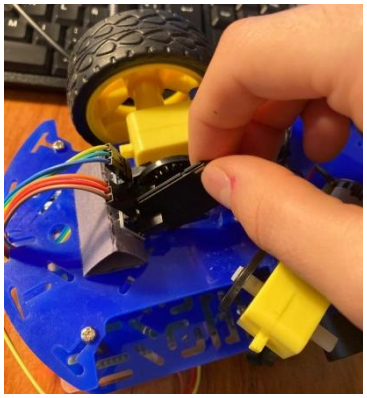


Figura B1

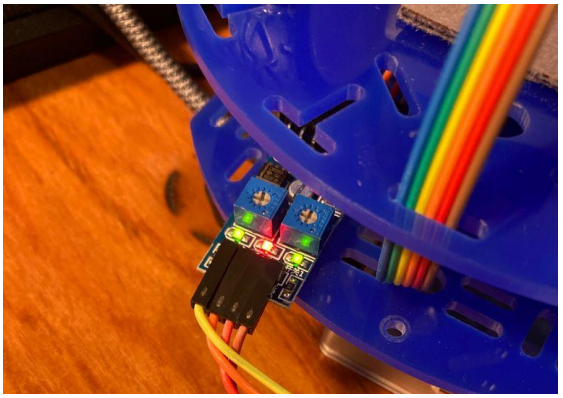


Figura B2

Una vez sabiendo qué entradas recibe la raspberry en función de cómo varía el encoder, se llevó una primera implementación de software. La base de funcionamiento del programa para detectar cambios es como se ilustra en la Fig. B3. En referencia a la figura B3 y habiendo ya configurado los pines 20 y 21 (según la configuración BCM) como entradas, lo que se hace es crear una variable que almacenará el valor anterior de dicho pin. Ahora se diferencian dos casos:

a) El valor actual del pin es igual que su valor anterior

b) El valor actual del pin es distinto que su valor anterior: esto significa que ha

habido un cambio en el receptor y por lo tanto la rueda del encoder ha recorrido una marca. Lo que se hace es aumentar en uno el contador de marcas de esa rueda y actualizar la variable que almacena el último valor del pin.

```
pul_20_ant=GPIO.input(20)
pul_21_ant=GPIO.input(21)
if GPIO.input(21)==True and pul_21_ant==False:
  marcas_d+=1
  pul_21_ant=GPIO.input(21)
else:
  marcas_d=marcas_d
  pul_21_ant=GPIO.input(21)
```

Figura B3

De esta manera ya se tiene un primer programa donde se cuentan las marcas que recorre cada rueda. El siguiente paso lógico es obtener Δs para conseguir, con las ecuaciones calculadas en el capítulo 2, los incrementos de las tres variables (*x*, *y* y *θ*), que modelan la localización del robot. El Δs se calcula multiplicando el número de marcas por la distancia a la que equivale cada marca mediante el factor de conversión de 0,9895 centímetros/marca pero teniendo en cuenta que el valor de las marcas no se reinicia en ningún momento, por lo tanto tendremos que almacenar el valor anterior del encoder. Para ello se usará una variable a la que se ha llamado *enc_d* o *enc_i* dependiendo la rueda. De esta manera:

$enc\_i = marcas\_d \cdot 0{,}9895$

$\Delta s = enc\_i - enc\_i\_ant$

$enc\_i\_ant = enc\_i$

El último factor a tener en cuenta para la configuración del encoder es el período de muestreo. El tiempo de actualización de la odometría tiene que ser más pequeño que el tiempo que tarda el encoder en cambiar de valor para no perder ninguna marca y alterar las medidas. El extremo se fija cuando el robot lleve velocidad máxima, es decir, suplir de la tensión máxima (5 V) a los motores. Esta velocidad es aproximando al alza 60cm/s que quiere decir que se recorren más de 60 marcas por segundo, o una marca cada 16,5 milisegundos. Por lo tanto se ha fijado un período de actualización de 5 milisegundos para asegurar el correcto funcionamiento. Otra problemática ligada al período de actualización es que los 5 milisegundos no fueran suficientes para ejecutar todo el código que se ejecuta periódicamente para actualizar la odometría. Se midió ese tiempo de ejecución y de ninguna manera se aproximaba a los 5 milisegundos que se han fijado de período, estando el tiempo medio de cómputo en 1 milisegundo.

B1.2. Casos excepcionales y problemática

Aquí se analiza cómo se han tenido en cuenta los posibles problemas o casos excepcionales a los que se ha tenido que dedicar un desarrollo de software específico. En el caso de la localización basada en encoder existen pocos casos excepcionales que haya que tener en cuenta. En este proyecto basta con tener en cuenta que tal y como mide el encoder no es capaz de distinguir si el robot se mueve con velocidad lineal negativa o positiva. Las ecuaciones de localización, que basadas en trigonometría, son capaces de aportar signo negativo o positivo para calcular los incrementos de *x* y de *y*. el incremento de θ no se calcula mediante trigonometría pero también distingue signo. Por lo tanto, para solucionar la actualización de los parámetros odométricos si el robot llevara velocidad

negativa, es decir que estuviera yendo marcha atrás, se ha creado una variable booleana que se llama *reverse* la cual se pondrá en "True" cuando al robot se le pase una velocidad negativa y en "False" cuando dicha velocidad sea positiva. De esta manera se ilustra en la Fig. B4 como se ha tenido esto en cuenta.

```
if self.reverse.value==False:
  self.x.value=self.x.value + x_inc
  self.y.value=self.y.value + y_inc
  self.th.value=self.th.value + th_inc
else:
  self.x.value=self.x.value - x_inc
  self.y.value=self.y.value - y_inc
  self.th.value=self.th.value + th_inc
```

Figura B4

Por último, se vuelve a destacar, que la problemática principal de medir la odometría con encoder es si el terreno fuera irregular o interfiriera en el contaje de las marcas del encoder. Por ello, en este proyecto donde el terreno no afectaba para nada en la medida del encoder se realiza con este sistema. La alternativa es basar la odometría en visión por cámara que, aunque sea más compleja y más cara computacionalmente, es un método más robusto.

B2. Clasificación de imágenes con redes neuronales

Este sub apartado se va a centrar en cómo se ha construido la red neuronal y cómo se han entrenado (todas han sido sometidas a un entrenamiento similar) las redes neuronales utilizadas. Cabe destacar que aunque la mayoría sean pre-entrenadas no quiere decir que no haya que entrenarlas con las imágenes propias sino que los pesos están mejor actualizados, el entrenamiento será menos en tiempo y tienen una arquitectura mejorada donde las capas se han añadido atendiendo a parámetros de optimización.

Para empezar, se ilustra en la Fig. B5 cómo se ha construido la red propia (que fue la primera en someterse al entrenamiento). Se ha hecho con la biblioteca keras y teniendo en

```
model = Sequential()
model.add(InputLayer(input_shape=(224, 224, 3)))
model.add(Conv2D(254, 3,3 ,activation= 'relu'))
model.add(Conv2D(254, 3,3 ,activation= 'relu'))
model.add(MaxPool2D(pool_size=(3,3)))
model.add(Conv2D(128, 2,2 ,activation= 'relu'))
model.add(Conv2D(128, 2,2 ,activation= 'relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Conv2D(32, 1,1 ,activation= 'relu'))
model.add(Conv2D(32, 1,1 ,activation= 'relu'))
model.add(MaxPool2D(pool_size=(1,1)))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(2,activation="softmax"))
```

Figura B5

cuenta los factores que fueron explicados el capítulo 3, es decir, capas convolucionales para la detección de las principales características y capas densamente conectadas para la capa de salida. Una vez construida mediante la siguiente orden se realizó un entrenamiento de 700 épocas:

```
history= model.fit(x_train, y_train, validation_data=(x_val,y_val), epochs=700, bat
ch_size=10)
```

La orden anterior utiliza el método *.fit* propia de la biblioteca keras que sirve para realizar el entrenamiento de una red neuronal. Al método es necesario pasarle los datos y las etiquetas de los elementos de entrenamiento y lo mismo para los datos de validación. Los *epochs (o épocas)* se corresponde a las veces que se va a repetir el entrenamiento donde se van ajustando los pesos de las neuronas, el *batch_size (o tamaño de lote)* es el número de datos de entrenamiento o validación que se cogen para cada *epoch*. Hay más datos que se utilizan con la orden *.fit* pero no son tan importantes como estos comentados.

Se ha comentado como en este apartado no existe una problemática específica para el proyecto más allá de las prestaciones de la red. Bien es cierto que ninguna red tiene una fiabilidad del 100% pero hay diferentes órdenes para asegurar una cierta robustez a la hora de clasificar las señales de tráfico (o imágenes en general).

Cuando se utiliza la orden `predictions = model.predict(image)` el resultado, al tener la última capa de salida función de activación softmax, sale con un peso ponderado (que como ya comentamos se suele confundir con la probabilidad). Por lo tanto, si la salida de la neurona no supera un cierto valor, no se da por válida esa predicción. En este proyecto esto quiere decir, que si el robot está situado en una intersección y trata de leer una señal pero el valor de ninguna neurona supera el 0,8 (ya se explicó que está acotada entre 0 y 1) el robot actuará como si no hubiera señal. Aunque no sea una solución muy precisa, se decide porque sino entorpecería mucho el experimento si se adoptaran soluciones más reales como por ejemplo preguntar al usuario, recalcular ruta o fiarse de la neurona con más peso. En base a los experimentos realizados no supone esta solución ninguna problemática ya que, cuando se trata de clasificar una señal, el peso de la neurona correspondiente a la clase de esa señal (por ejemplo, señal de obligatorio seguir recto) es 1 todas las veces. Si no hubiera señal en el cruce, todas las neuronas presentan un valor especialmente bajo (bastante inferiores a 0,1). Con esto se soluciona el único problema real que era la salida que genera la red a la hora de leer y clasificar una señal si no hubiera señal en esa intersección. Como ninguna neurona presenta un valor aceptable de fiabilidad, se asume que no hay señal. Luego hay otros problemas más pequeños que se resolverían con mejores herramientas y más tiempo como son por ejemplo la precisión de la red o la frecuencia de la raspberry.

B3. Carga del entorno de simulación

B3.1. Funcionamiento en condiciones normales

Al igual que en los apartados anteriores se va a explicar los comandos más importantes del código desarrollado para lograr una fácil comprensión del software del proyecto. Mientras que tanto para la localización del robot como para la construcción y entrenamiento se tenían nociones por haber sido trabajadas previamente, esta parte de código era de la que menos conocimientos se tenía. Después de varias opciones se ha decidido trabajar con las intersecciones de calles puesto que son los elementos que más información contienen del mapa.

Para ello es necesario que el usuario defina el escenario de actuación del robot y lo cargue dentro del software a través de un fichero *.txt*. Tal y como está configurado basta con incluir dicho fichero en la misma carpeta donde se encuentra el código desarrollado. Como ya se ha comentado, se procesa el fichero *.txt* para conseguir una matriz de nudos la cual servirá posteriormente para construir la ruta. Este procesamiento se hace usando la biblioteca *networkx* que es una biblioteca de python muy útil para la construcción y el estudio de grafos y redes.

El código para el procesamiento del mapa de nudos se ilustra a continuación en la Fig. B6 donde el objetivo es construir una matriz de nudos.

```
from matplotlib import pyplot as plt
import numpy as np
import networkx as nx

# lines to 2d array
with open('myfile.txt') as f:
    a = np.array([list(map(int,i.split())) for i in f.readlines()])

# define grid graph according to the shape of a
G = nx.grid_2d_graph(*a.shape)

# remove those nodes where the corresponding value is != 0
for val,node in zip(a.ravel(), sorted(G.nodes())):
    if val!=0:
        G.remove_node(node)

plt.figure(figsize=(7,7))
# coordinate rotation
pos = {(x,y):(y,-x) for x,y in G.nodes()}
nx.draw(G, pos=pos,
        node_color='blue',
        width = 4,
        node_size=400)
```

Figura B6

El fundamento de este programa es el siguiente:

Se abre el archivo *.txt* que debe estar en la misma carpeta como ya se ha comentado antes. Una vez abierto se leen todas las líneas de dicho archivo y se construye una lista (que se nombra como *a*) con esos elementos. Esa lista se transforma a un grafo de la misma dimensión que la lista creada con la orden `nx.grid_2d_graph(*a.shape)`. Ese grafo se recorre para eliminar todos los elementos que no sean 0. Esto se hace para encontrar una forma rápida de construir el mapa de nudos. Una vez eliminados todos los elementos distintos de 0 se representa gráficamente pudiendo variar el tamaño de la figura, el tamaño del nodo, el color del nodo, etc.

Después de esto y para facilidad posterior a la hora de calcular la ruta que va a seguir el robot se transforma ese grafo a matriz de nudos mediante la orden `mapa=np.array(list(G.nodes)).reshape(5,4,2)`. Las dimensiones (5,4) en este caso son por el mapa que se ha usado para la experimentación y el 2 representa las dos dimensiones que tiene cada elemento (es una matriz de puntos de la forma (*x,y*)). Una vez construida la matriz de nudos (donde cada nudo tendrá unas coordenadas de fila y columna) ya está el mapa implementado en la raspberry y lo siguiente sería analizar cómo se construye la ruta que ha de seguir el robot para llegar hasta el nudo definido por el usuario como nudo final.

B3.2. Casos excepcionales y problemática

Al contrario que en los otros dos apartados anteriores aquí sí que tiene lugar una problemática especial que requiere de un análisis detallado. Se explicará desde un ejemplo de un mapa de nudos de dimensiones 7x12. Ya se ha visto como se carga un mapa desde un archivo *.txt* y cómo llega a convertirse una matriz de nudos con sus coordenadas. Si el archivo *.txt* tiene la forma de la Fig. 47 a la izquierda el mapa de

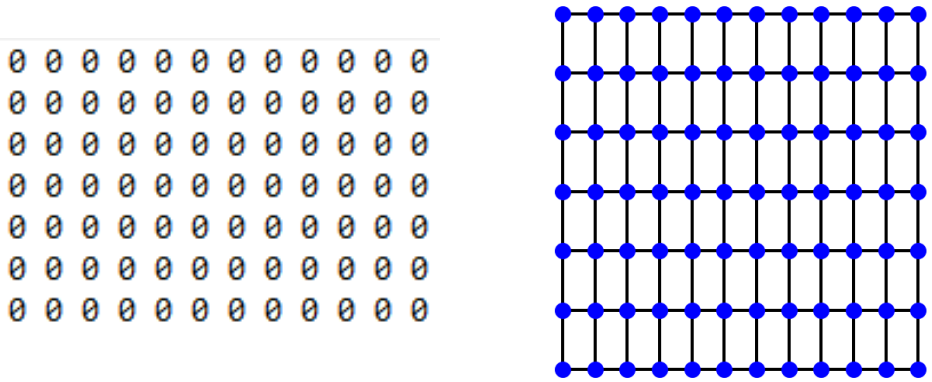


Figura B7

nudos quedará como en la Fig. B7 a la derecha y la orden `mapa=np.array(list(G.nodes)).reshape(7,12,2)` nos transformará el grafo a matriz de nudos. La matriz de nudos tendrá 168 elementos (7x12x2=168) porque el grafo tenía 84 nudos de dos dimensiones cada uno.

El problema viene cuando el archivo *.txt* contiene un elemento distinto de 0 como se ilustró en la Fig. 48 del capítulo 4 porque el grafo se construye sin ese nudo por lo tanto el total de nudos en ese caso es de 83 (por lo tanto 166 elementos) y no coincide con las dimensiones de la matriz. Ejecutando esto en python el error es el siguiente:

```
ValueError: cannot reshape array of size 166 into shape (7,12,2)
```

Figura B8

En conclusión, el código daría error y no se cargaría el mapa correctamente. Es sensato pensar que no es necesaria una matriz de nudos y que simplemente se puede construir la ruta trabajando con los elementos del grafo pero tal y como se construye la ruta, generaría otro error al intentar acceder a nudos que no existen.

La solución para este problema reside en señalar cuáles son los nudos que no existen pero no eliminarlos del grafo. Con estos nudos inexistentes se crea una lista que se tendrá en cuenta cuando el robot intente moverse a ese nudo. Es decir, dentro de código y por facilidad, se trabajará como si ese nudo o nudos existieran y sólo se tendrá en cuenta si ese nudo estuviera dentro de la ruta y el robot intentara moverse a este. Para ese caso se ha creado una variable booleana llamada *existeNudo* que se comprobará en cada intersección, es decir, cada vez que el robot se intente desplazar al nudo siguiente planificado en la ruta.

Por lo tanto, como se ve en el apartado siguiente, se recalculará la ruta planeada si el movimiento se ve restringido por alguna señal de tráfico o porque el nudo siguiente no existe en la realidad.

B4. Unión y finalización del software completo

Habiendo dedicado este capítulo a la explicación del código implementado y teniendo explicados los tres principales bloques del proyecto queda estudiar los detalles de cómo se ha reunido todo en un fichero principal que será el que se ejecutará en la raspberry. El proyecto se ha desarrollado con una programación orientada a objetos, de esta manera se tienen dos clases: una para todo lo relacionado con el robot como es la localización o el reconocimiento de imágenes a través de la cámara y otra clase para todo lo relacionado con el mapa como es la construcción de la matriz de nudos o la planificación de la ruta. A continuación se va a detallar el código en orden de llamada por el programa principal durante un ciclo completo de movimiento del robot.

B4.1. Construcción de la ruta

Este método va a pertenecer a la clase del mapa. Lo primero que va a realizar el robot, que estará detenido en algún nodo, va a ser calcular la ruta que tiene que seguir. Para ello necesita varios argumentos: el nudo en el que se encuentra, el nudo al que se quiere llegar y el ángulo con el que está orientado. Como el nudo en el que se encuentra y el ángulo con el que está orientado se pueden conocer con la odometría, el único parámetro que se le requiere al usuario es el nudo donde se quiere llegar.

Para ello, teniendo ya todos los parámetros, la ruta se puede calcular de muchas maneras. Como en este proyecto no se obtiene una exactitud milimétrica por la poca precisión del encoder se ha elegido el criterio del menor giro, de esta manera una ruta calculada en dos momentos diferentes, aunque tenga el mismo nudo inicial y final, no será la misma si la orientación del robot es diferente. Se ilustra en la Fig. B9 dos rutas posibles para los mismos requerimientos y con una orientación diferente.

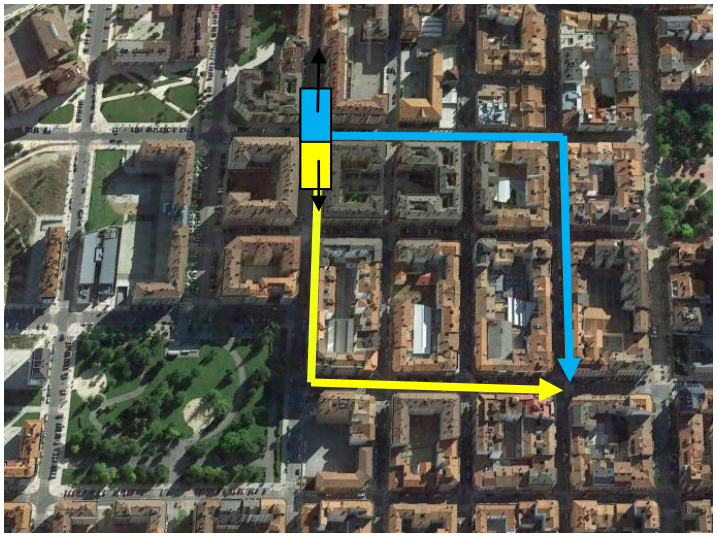


Figura B9

Cabe destacar que el robot posicionado en una intersección (o nudo) tiene cuatro orientaciones posibles y por lo tanto cuatro localizaciones diferentes en función de la orientación como se ve en la Fig. B9.

```
def planeaRuta(self):
      rutaCalculada=False
          if giro_vertical<giro_horizontal:
            while (avance_horizontal!=0):
              nudo_siguiente=nudo_actual+[0,1]
              self.ruta.append(nudo_siguiente)
              nudo_actual=nudo_siguiente
              avance_horizontal=-1
            while (avance_vertical!=0):
              nudo_siguiente=nudo_actual+[1,0]
              self.ruta.append(nudo_siguiente)
              nudo_actual=nudo_siguiente
              avance_vertical-1
          else:
            while (avance_vertical!=0):
              nudo_siguiente=nudo_actual+[1,0]
              self.ruta.append(nudo_siguiente)
              nudo_actual=nudo_siguiente
              avance_vertical=-1
            while (avance_horizontal!=0):
              nudo_siguiente=nudo_actual+[0,1]
              self.ruta.append(nudo_siguiente)
              nudo_actual=nudo_siguiente
              avance_horizontal=-1
      return self.ruta
```

Figura B10

Por lo tanto, el código va a calcular cuál es el menor giro que tiene que hacer y después irá añadiendo los nudos de la misma fila o columna hasta girar noventa grados y volver a avanzar por esa fila o columnas hasta llegar al destino. Teniendo esta arquitectura de mapa se llega a cualquier nudo con tan solo un giro (considerando que no tiene que girar para orientarse). A nivel de código y como se ve en la Fig. B10 se ha traducido en calcular cual es el menor giro y en base a eso ir añadiendo nudos a la ruta hasta que la variación horizontal o vertical sea cero, en ese momento se realiza un giro de noventa grados y se sigue avanzando hasta llegar al nudo de destino. Se muestra a continuación el pseudocódigo de cómo se calcula una ruta.

B4.2. Recorrer una calle

Una vez planificada la ruta el siguiente paso es seguirla. A tal efecto se le debe el

```
def go_to(self,nudo_act, nudo_sig):
        var_ver=nudo_sig[0]-nudo_act[0]
        var_hor=nudo_sig[1]-nudo_act[1]
        self.miroSignal()
        replantea=False
        if (var_hor==0):
            if (var_ver>0):
                if (th==math.pi/2):
                    if (self.puedoDcha==True):
                        self.giro_dcha()
                    else:
                        replantea=True
                elif (th==-math.pi/2):
                    if (self.puedoIzda==True):
                        self.giro_izq()
                    else:
                        replantea=True
                elif (th==0):
                    if (self.puedoRecto==True):
                        self.ve_recto()
                    else:
                        replantea=True
                elif (th==math.pi):
                    replantea=True
```

Figura B11

método que se va a desarrollar a continuación el cual va a gobernar el movimiento del robot de un nudo a otro, es decir, del nudo en el que se encuentra al siguiente nudo planificado en la ruta.

Lo primero que hace el robot cuando se ejecuta este método es, antes de suplir de potencia los motores, leer la señal de tráfico si la hubiere en esa intersección y, si la señal no lo restringe, avanzar hasta el próximo nudo para repetir este proceso hasta que se complete la ruta. A este método será necesario pasarle como argumento el nudo siguiente al que hay que ir, ya que con los parámetros odométricos propios del robot y haciendo uso de la cámara basta para calcular los parámetros necesarios y realizar el movimiento. Dependiendo de la orientación del robot será un movimiento de avance lineal o un giro a derechas o a izquierdas. A continuación se muestra una parte del pseudocódigo donde se puede observar la idea principal de cómo trabaja la función para recorrer una calle, lo que se conoce como *go_to_goal.*

B4.3. Reconstrucción de la ruta

Atendiendo a lo anterior, hay un escenario posible que no se ha tenido en cuenta a la hora de explicar el funcionamiento del robot, resulta trivial preguntarse qué pasaría si una señal de tráfico prohíbe el movimiento que iba a realizar el robot o incluso, como se ha mencionado anteriormente, el robot fuera a hacer un movimiento hacia un nudo que no existe en la realidad.

En ese caso, lo primero que se hace es mover el robot a otro nudo para volver a calcular una nueva ruta. El origen de este método viene porque es necesario calcular cuál es el nudo más cercano al nudo final de la ruta puesto que carecería de sentido mover el robot haciéndole alejarse de su origen. Es por ello que en términos de generación de software es el método que más ocupa de todo el proyecto.

Su funcionamiento se basa en:

1. Calcular los posibles nudos más cercanos
2. Elegir entre todos ellos el nudo más cercano al destino final.
3. Moverse a ese nudo
4. Recalcular la ruta manteniendo el mismo nudo de destino que había solicitado el usuario
5. Actualizar el nudo en el que se encuentra

Cabe destacar que solamente se llamará a este método cuando una señal de tráfico restringa el movimiento o el robot vaya a hacer el movimiento hacia un nudo que realmente no existe, es decir. Este movimiento realmente es una función llamada *go_nudocercano* (que irá en la clase robot) combinada con la función de calcular la ruta explicada anteriormente, es decir, si el robot en un nudo iba a girar a la derecha pero observa una señal de obligatorio seguir recto, seguirá recto y será cuando llegue al nudo el momento en el que recalculará la ruta de la misma manera que lo hizo al principio pero tomando como nudo de inicio el nudo en el que se encuentra realmente.

B4.4. Blindaje frente a los requerimientos de usuario

Existen escenarios y situaciones posibles que no han sido contempladas aún en la memoria, por ejemplo, ¿qué pasaría en la situación de la Fig. B12?

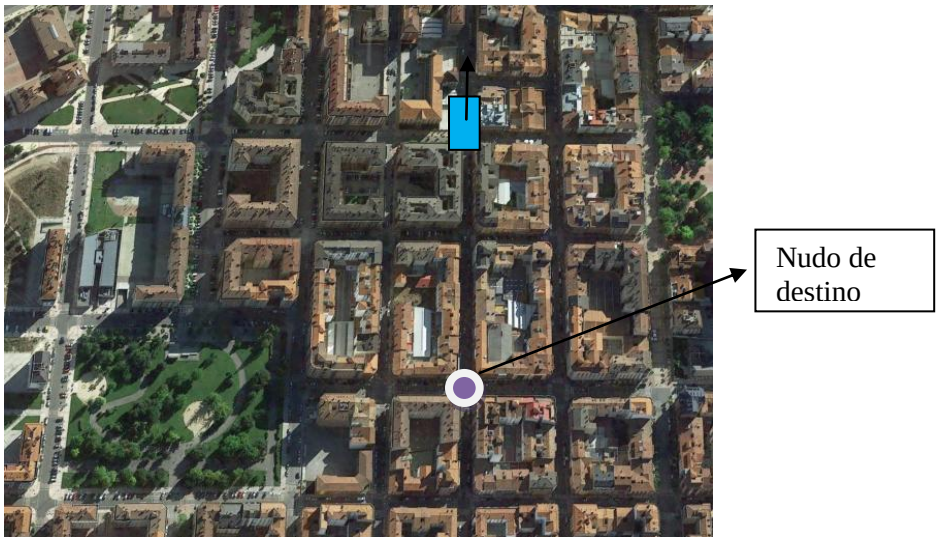


Figura B12

En este caso se ha planteado la alternativa de, como no se puede realizar un giro de 180 grados en las intersecciones en la vida real, mover al robot a un nudo cercano, bien sea el de la derecha o el de la izquierda y desde ese nudo calcular la ruta a seguir por el robot. También se planteó la alternativa de ir marcha atrás pero se descartó por el deseo de intentar simular de la forma más realista posible el movimiento de un vehículo por un entorno urbano.

9. Fuentes y referencias

- Figura 1: (a) https://www.irobot.es/roomba

  (b) https://automation.omron.com/en/us/products/family/ld

  (c) https://harmonicdrive.de/es/aplicaciones/entornos-especiales

  (d) https://www.spacerobotics.eu/robotica/

- Figura 2: http://www.sapiensman.com/tecnoficio/electricidad/velocidad_de_motores_electricos1.php

- Figura 3: https://dialnet.unirioja.es/descarga/articulo/4729008.pdf

- Figura 4: https://revistas.elpoli.edu.co/index.php/pol/article/view/1143/1482

- Figura 5: https://www.intel.es/content/www/es/es/artificial-intelligence/posts/difference-between-ai-machine-learning-deep-learning.html

- Figura 6: https://www.diegocalvo.es/clasificacion-de-redes-neuronales-artificiales/

- Figura 7: https://es.mathworks.com/help/signal/ug/denoise-speech-using-deep-learning-networks.html

- Figura 8: https://ichi.pro/es/redes-neuronales-convolucionales-para-principiantes-que-usan-keras-y-tensorflow-2-217123342499621

- Figura 9: https://www.juanbarrios.com/la-matriz-de-confusion-y-sus-metricas/

- Figura 10: https://torres.ai/deep-learning-inteligencia-artificial-keras/

- Figura 11: https://torres-ai.medium.com/deep-learning-aprendizaje-profundo-52084448ea0e

- Figura 14: https://programmerclick.com/article/76601378385/

- Figura 15: https://allenlu2007.wordpress.com/2019/05/17/convolution-and-variation/

- Figura 16: https://theaisummer.com/cnn-architectures/

- Figura 17: https://medium.com/datos-y-ciencia/modelos-cnn-en-la-clasificaci%C3%B3n-de-im%C3%A1genes-cl%C3%A1sicas-y-modernas-d072a6718689

- Figura 31: https://astelus.com/guia-turistica-de-barcelona/vista-aerea-del-distrito-del-ensanche-en-barcelona/

- Figura 32: https://earth.google.com/web/search/ciudad+rodrigo/@40.59696546,-6.53380493,656.0073523a,619.61960407d,35y,7.33794051h,39.50034567t,-0r/data=CigiJgokCRfK0FpZPkVAEe6Ut3c9wENAGSNAX8pB6_i_IbTuMTTA0yPA

- Figura 33: https://www.google.com/intl/es/earth/
- Figura 37: https://twitter.com/whoatte/status/1368328635892006924
- Figura 38: https://www.google.com/intl/es/earth/
- Figura A2: http://www.kramirez.net/Robotica/Material/Presentaciones/Odometria.pdf
- Figura A4: https://www.comohacer.eu/gpio-raspberry-pi/
- Figura B9: https://www.google.com/intl/es/earth/
- Figura B12: https://www.google.com/intl/es/earth/

Bibliografía

[1] Adrian Rosebrock - How to create a deep learning dataset using Google Images *https://www.pyimagesearch.com/2017/12/04/how-to-create-a-deep-learning-dataset-using-google-images/*

[2] Python Project – Traffic Signs Recognition, https://data-flair.training/blogs/python-project-traffic-signs-recognition/

[3] Vicente Rodríguez - Conceptos básicos sobre redes neuronales https://vincentblog.xyz/posts/conceptos-basicos-sobre-redes-neuronales

[4] Cuentos Cuánticos - Robótica: Estimación de posición por odometría https://cuentos-cuanticos.com/2011/12/15/robotica-estimacion-de-posicion-por-odometria/

[5] Gael Varoquaux https://scipy-lectures.org/packages/scikit-learn/index.html

[6] Chetan Kapoor https://installvirtual.com/install-python-3-7-on-raspberry-pi/

[7] Antonio José Toro Valderas - Implementación de redes neuronales en Raspberry Pi 3 con Movidius Neural Compute Stick http://bibing.us.es/proyectos/abreproy/71685/fichero/TFM-1685+TORO+VALDERAS%2C+ANTONIO+JOS%C3%89.pdf

[8] Adrián Fernández Vázquez y Noemi Delgado Santa Cruz - Construir un RC carautónomo con Raspberry Pi, NAVIO2 y TensorFlow/KERAS https://riull.ull.es/xmlui/bitstream/handle/915/16539/Construir%20un%20RC%20car%20autonomo%20con%20Raspberry%20Pi,%20NAVIO2%20y%20TensorFlowKERAS.pdf?sequence=1

[9] Adrián Fernández de la Torre - Portabilidad y optimización de una red neuronal para la detección rápida de daños en terremotos usando el toolkit OpenVINO https://eprints.ucm.es/id/eprint/62581/1/FERNANDEZ_DE_LA_TORRE_TFG_-_Portabilidad_y_optimizacion_de_una_red_neuronal_para_la_deteccion_rapida_de_danos_en_terremotos_usand_526057057.pdf

[10] Álvaro Casas Martínez - Reconocimiento de imágenes con Redes Convolucionales en C http://bibing.us.es/proyectos/abreproy/91350/fichero/TFG+Alvaro+Casas+Martinez.pdf

[11] Diego Calvo - Clasificación de redes neuronales artificiales https://www.diegocalvo.es/clasificacion-de-redes-neuronales-artificiales/

[12] Daniel Lerch - Deep Learning con redes pre-entrenadas en ImageNet https://medium.com/neuron4/redes-pre-entrenadas-en-imagenet-30d858c37b1f

[13] Mariano Rivera - Reeuso de Redes Preentrenadas http://personal.cimat.mx:8181/~mrivera/cursos/aprendizaje_profundo/preentrenadas/preentrenadas.html

[14] Keras Applications https://keras.io/api/applications/

[15] Valentyn Sichkar - Traffic Signs Preprocessed

https://www.kaggle.com/valentynsichkar/traffic-signs-preprocessed?select=label_names.csv

[16] Álvaro Artola Moreno - Clasificación de imágenes usando redes neuronales convolucionales en Python http://bibing.us.es/proyectos/abreproy/92402/fichero/TFG-2402-ARTOLA.pdf

[17] Configure Default Python version on your Pi https://raspberry-valley.azurewebsites.net/Python-Default-Version/

[18] Machine Learning: Seleccion Metricas de clasificacion https://sitiobigdata.com/2019/01/19/machine-learning-metrica-clasificacion-parte-3/#

[19] Juan Ignacio Barrios Arce - La matriz de confusión y sus métricas https://www.juanbarrios.com/la-matriz-de-confusion-y-sus-metricas/

[20] Inteligencia artificial fácil - Machine Learning y Deep Learning prácticos https://www.ediciones-eni.com/open/mediabook.aspx?idR=8dd2ca32769cb24b49648b15ef8e777e

[21] David Waster - Explicación alternativa para accuracy, precision, recall y f1-score https://steemit.com/spanish/@waster/explicacion-alternativa-para-accuracy-precision-recall-y-f1-score

[22] Jose Martinez Heras - Precision, Recall, F1, Accuracy en clasificación https://www.iartificial.net/precision-recall-f1-accuracy-en-clasificacion/

[23] Sam Westby - Install Tensorflow 2 on a Raspberry Pi 4 https://www.youtube.com/watch?v=GNRg2P8Vqqs&list=LL&index=6

[24] Wikipedia - ciudad planificada https://es.wikipedia.org/wiki/Ciudad_planificada

[25] Jordi Torres - DEEP LEARNING INTRODUCCIÓN PRÁCTICA CON KERAS https://torres.ai/deep-learning-inteligencia-artificial-keras/

[26] Luis Llamas - MACHINE LEARNING CON TENSORFLOW Y KERAS EN PYTHON https://www.luisllamas.es/machine-learning-con-tensorflow-y-keras-en-python/

[27] Rafael Marín - ¿Qué es OpenCV? Instalación en Python y ejemplos básicos https://revistadigital.inesem.es/informatica-y-tics/opencv/

[28] Na8 - Una sencilla Red Neuronal en Python con Keras y Tensorflow https://www.aprendemachinelearning.com/una-sencilla-red-neuronal-en-python-con-keras-y-tensorflow/

[29] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen - MobileNetV2: Inverted Residuals and Linear Bottlenecks https://arxiv.org/abs/1801.04381

[30] David Sarabia - AutoML, la máquina que crea otras máquinas inteligentes https://www.eldiario.es/tecnologia/automl-maquina-crea-maquinas-inteligentes_1_3091963.html

[31] EfficientNet: Repensar el escalado del modelo para redes neuronales

convolucionales (clasificación de imágenes) https://ichi.pro/es/efficientnet-repensar-el-escalado-del-modelo-para-redes-neuronales-convolucionales-clasificacion-de-imagenes-263228629273165

[32] Jonathan Quiza - Modelos CNN en la clasificación de imágenes clásicas y modernas https://medium.com/datos-y-ciencia/modelos-cnn-en-la-clasificaci%C3%B3n-de-im%C3%A1genes-cl%C3%A1sicas-y-modernas-d072a6718689

[33] Wikipedia – Python https://es.wikipedia.org/wiki/Python#:~:text=Python%20es%20un%20lenguaje%20de,en%20menor%20medida%2C%20programaci%C3%B3n%20funcional.