Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN COMPUTACIÓN

Aplicación de técnicas de aprendizaje máquina para el control de un pequeño vehículo

Estudante: Rubén Covelo Bugallo
Dirección: Carlos Vázquez Regueiro, Xose Manuel Pardo López

A Coruña, xuño de 2021.

A mi madre, a mi padre y a mi hermano que siempre me han motivado y puesto a mi disposición los medios para desarrollar esta bonita etapa formativa.

Agradecimientos

Gracias a mis padres, hermano y familia por apoyarme y animarme en estos años, motivándome para que hiciese lo que me gusta y valorando mi trabajo, sin ellos esto no habría sido posible.

Gracias a mis amigos y compañeros de promoción por los buenos ratos que hemos pasado por las bibliotecas y por los momentos que hemos compartido.

Gracias a mis tutores por permitirme realizar este proyecto que ha supuesto un reto personal y la investigación de un campo que me apasiona.

Gracias a Carlos por su implicación, su motivación, por todo lo que me ha enseñado y por estar disponible cuando no tocaba.

Gracias a la Universidad y plantilla de la FIC por acogerme estos años, recibiendo siempre un trato cercano y afable.

Por último, gracias al esfuerzo y dedicación del personal docente de todas las etapas educativas, que me han formado en lo personal y profesional, permitiéndome llegar aquí.

Resumen

El principal objetivo de este trabajo, es desarrollar un sistema donde sea posible trabajar con algoritmos de aprendizaje máquina en tiempo real sobre un pequeño vehículo. La técnica escogida ha sido las redes de neuronas convolucionales. El sistema necesita recopilar el conjunto de datos de entrenamiento y test, es decir, las imágenes y acciones que se deben ejecutar para realizar una tarea. Para simplificar el proceso, sólo necesita que un operador controle directamente el robot. Hemos propuesto un mecanismo basado en flujo óptico para etiquetar automáticamente las imágenes capturadas y generar los comandos de control del robot.

Para el control del desarrollo, se ha seguido la metodología ágil. Las pruebas del proyecto fueron ejecutadas sobre una placa Raspberry Pi 3 Model B+ y el vehículo usado ha sido el robot Makebock Ranger equipado con una cámara Raspicam de bajo coste. Los experimentos se han realizado en un circuito real, alojado en un entorno controlado, variando el color y la forma de las superficies y sus líneas. Se han entrenado distintos tipos de redes (clasificación y regresión) para ejecutar una o varias tareas (seguir carril o seguir línea).

Abstract

The main objective of this work is to develop a system where it is possible to work with machine learning algorithms in real time on a small vehicle. The chosen technique has been convolutional neuron networks. The system needs to collect the training and test data set, that is, the images and actions that must be executed to perform a task. To simplify the process, you only need one operator to directly control the robot. We have proposed an optical flow-based mechanism to automatically tag captured images and generate robot control commands.

To control development, the agile methodology has been followed. The project tests were carried out on a Raspberry Pi 3 Model B + board and the vehicle used was the Makebock Ranger robot equipped with a low-cost Raspicam camera. The experiments have been carried out in a real circuit, housed in a controlled environment, varying the color and shape of the surfaces and their lines. Different types of networks (classification and regression) have been trained to carry out one or more tasks (following the lane or following the line).

Palabras clave:

- Aprendizaje automático
- CNN
- Raspberry Pi
- Camara
- Dataset
- Clasificación
- Regresión
- Sobreajuste

Keywords:

- Machine Learning
- CNN
- Raspberry Pi
- Camera
- Dataset
- Classification
- Regression
- Overfitting

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1
Introducción

En este primer capítulo introductorio de la memoria, se explicará la motivación de este proyecto, los objetivos que se quieren alcanzar y, por último, la estructura de este documento.

1.1 Motivación

Cada día se puede apreciar el aumento de la influencia del Aprendizaje Automático sobre la vida diaria, incorporándola en productos de uso común que nos facilitan las tareas y aportan mayor comodidad. Se pueden encontrar aplicaciones en diversos ámbitos como el financiero, donde se busca el análisis de transacciones para evitar el fraude, en la salud, con la ayuda en los diagnósticos por el análisis de imágenes o el sector automovilístico personalizando la experiencia de conducción.

El sector automovilístico esta incorporando esta tecnología en la experiencia de los usuarios con interfaces inteligentes que encontramos en los dispositivos de los vehículos. También se puede encontrar en el mercado que la mayoría de coches nuevos ya incorporan ayudas a la conducción, un primer paso para lograr alcanzar la conducción autónoma en los automóviles. Actualmente esta tecnología aún continua en desarrollo para poder conseguir la conducción sin necesidad de ningún tipo de intervención humana. Esto ocurre debido a que aún no están definidos muchos aspectos legislativos y la tecnología debe continuar evolucionando, aunque existen compañías como Tesla que ofrecen en sus vehículos las herramientas necesarias para poder circular de manera autónoma cuando se permita.

Estos proyectos cuentan a menudo con una gran financiación y están pensados para la ejecución sobre equipos de grandes características. La motivación de este proyecto, se basa en la búsqueda de un modelo de aprendizaje automático para que, un robot que simula ser un coche o un coche radiocontrol, por medio de una placa de bajo coste y ejecutando las técnicas de aprendizaje automático, sea capaz de realizar una tarea de forma exitosa.

1.2 Objetivos

El objetivo de este trabajo es lograr desarrollar una metodología para poder aplicar un algoritmo de aprendizaje automático sobre una placa de bajo recursos, con el fin de controlar un coche radiocontrol y que pueda realizar las tareas definidas por el usuario.

Las tareas definidas son:

- Realizar el seguimiento de un carril delimitado por líneas entre las que se posiciona el robot.
- Realizar el seguimiento de una línea sobre una superficie en la que es posicionado el robot.

Tras implementar las posibles soluciones definidas para conseguir realizar las tareas planteadas, se realizan una serie de pruebas en un circuito real para comprobar si se ejecutan exitosamente.

1.3 Estructura de la memoria

En esta sección se introducirá una explicación breve sobre los capítulos con los que nos vamos a encontrar en esta memoria:

- Capítulo 1: Es el capítulo actual, en él nos encontramos la motivación del proyecto, los objetivos propuestos y una descripción de la estructura de la memoria.
- Capítulo 2: Se introducen los fundamentos teóricos necesarios para la realización del proyecto.
- Capítulo 3: En este apartado se describen las tecnologías hardware y software necesarias para desarrollar el proyecto.
- Capítulo 4: En este capítulo se indican las metodologías de desarrollo empleadas, el análisis de requisitos, la planificación y el coste del desarrollo del proyecto.
- Capítulo 5: Se describen las arquitecturas y las implementaciones desarrolladas.
- Capítulo 6: En esta parte se explican las pruebas realizadas y se analizan los datos obtenidos.
- Capítulo 7: Es el último capítulo de la memoria donde se realizan las conclusiones obtenidas al finalizar el proyecto y las líneas futuras de desarrollo.

Capítulo 2

Fundamentos teóricos

En esta sección se describirán los fundamentos teóricos del aprendizaje máquina, sus tipos y se realizará un análisis de las técnicas de redes neuronales artificiales. También se explicarán algunos de los fundamentos básicos de visión por computador que se han manejado en este proyecto.

2.1 Aprendizaje máquina

El aprendizaje máquina o aprendizaje automático es un campo que pertenece a las ciencias de la computación, en concreto, se ubica dentro de la disciplina de la inteligencia artificial. Busca conseguir que las máquinas obtengan una inteligencia gracias a modelos matemáticos. Fue definida en 1956 en la Conferencia de Dartmouth por John McCarthy, donde argumenta que"para cualquier tipo de aprendizaje, puede realizarse una máquina para simularlo" [1] acuñando el término como la ciencia de lograr máquinas inteligentes.

El aprendizaje máquina busca el desarrollo de modelos, basándose en un conjunto de datos iniciales, sin necesidad de crear un programa explícitamente, consiguiendo así un modelo con el que seremos capaces de realizar nuevas predicciones sobre otros conjuntos de datos. En otras palabras, el aprendizaje máquina [2] busca que las máquinas sean capaces de aprender como lo haría una mente humana, a partir de la experiencia.

2.1.1 Tipos de aprendizaje máquina

Dependiendo de las necesidades del problema, el ambiente en el que se va a desarrollar, los factores que van a influir en la toma de decisiones, los datos y la tarea que deseamos realizar, podemos distinguir entre distintos algoritmos para lograr el aprendizaje. Entre estos tipos de aprendizaje destacan:

- Aprendizaje supervisado: en este modelo se establece una relación ente el conjunto de datos de entradas y salidas.

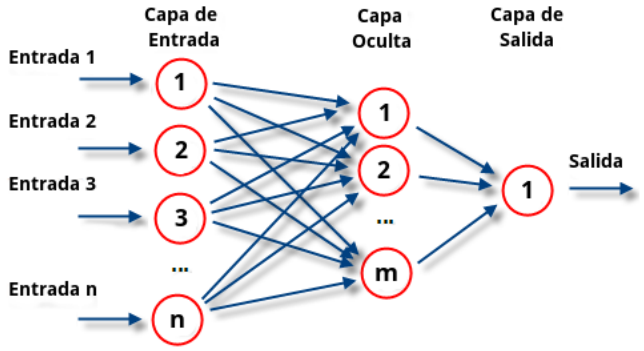

Figura 2.1: Estructura típica de una red neuronal artificial.

- Aprendizaje no supervisado: únicamente tenemos los datos de entrada por lo que el algoritmo busca reconocer estructuras similares y agrupar las entradas que se identifican con estos patrones.

- Aprendizaje semi-supervisado: debido a que en muchas ocasiones no se dispone de un conjunto de datos totalmente clasificados, este modelo combina los dos algoritmos anteriores, iniciándose con el aprendizaje supervisado sobre los datos de entrada y salida, aplicando posteriormente aprendizaje no supervisado sobre los datos que únicamente se dispone de la información de entrada sobre los modelos ya creados.

- Aprendizaje por refuerzo: Se basa en la recompensa de un agente, por la realización de acciones en un entorno de una máquina que debe de llegar a una meta. Esta recompensa o *feedback* puede ser positiva o negativa, ya que las acciones ejecutadas no siempre nos pueden acercar hacia la meta.

2.2 Redes de Neuronas Artificiales

Las redes de neuronas artificiales (RNA) son un modelo basado en el funcionamiento de las neuronas humanas. Se forma con las neuronas (nodos) y las conexiones entre ellas provocando una estructura de red. De este modo pueden intercambiar información con otras neuronas artificiales provocando que una información de entrada produzca un valor de salida. Cada neurona multiplica por un peso el valor de salida que envía a la siguiente neurona con la que está conectada. Dependiendo de este valor, se podrá aumentar o disminuir los estados de activación de las siguientes neuronas. Se debe indicar una función de activación, que es umbral para el valor de salida, que no se debe sobrepasar antes de propagarse a otra neurona.

Estas redes formadas por neuronas artificiales suelen construirse combinando diferentes tipos de capas como podemos observar en la figura 2.1. Las posibles capas que existen para la

construcción de las redes son:

- Capa de entrada: Estas neuronas serán las encargadas de recibir la información de entrada original procedente de nuestro conjunto de datos o del entorno y la introducen en la red.
- Capa oculta: Son las neuronas situadas en la parte intermedia de la estructura que no tienen, ni conexión directa con el entorno, ni salida directa con el mismo. En esta capa se procesa la información y trata de realizarse un modelo del entorno para poder predecir la información que reciben.
- Capa de salida: Estas neuronas reciben la información de entrada procesada y devuelven la respuesta de la red ante los datos de entrada.

Conociendo los elementos con los que podemos construir una red, existen dos tipos de red de neuronas artificiales atendiendo a su estructura, que son:

- Red monocapa: Es la red neuronal más sencilla y solo consta de capa de entrada y salida, estando directamente conectadas. Sirve para clasificar problemas linealmente separables por lo que a menudo no resultan interesantes por existir métodos estadísticos para resolver este tipo de problemas.
- Red multicapa: Este tipo de red tiene al menos las 3 capas descritas anteriormente donde, normalmente, cada capa es alimentada por la que le precede. Incorporan capas ocultas permitiendo que se consigan representar funciones no lineales.

Por último, dependiendo de como evoluciona el flujo de datos en la red se pueden distinguir entre dos tipos:

- Red unidireccional: La información circula únicamente en una dirección, es decir, la información es recibida por la capa de entrada y avanza en dirección a la capa de salida sin poder volver a capas ya visitadas.
- Red recurrente o realimentada: En este tipo de redes la información accede a la red por la capa de entrada y avanza hacia la capa de salida, pero en cualquier momento puede regresar a una capa anterior e incluso podría regresar de la capa de salida a la de entrada nuevamente.

Las redes neuronales aportan grandes mejorías en las operaciones, ya que pueden llevarse a cabo en tiempo real debido a que el procesado de la información se realiza en cada unidad, dependiendo de sus entradas y peso. Esto permite que las neuronas de una misma capa puedan trabajar en paralelo devolviendo una salida en el acto.

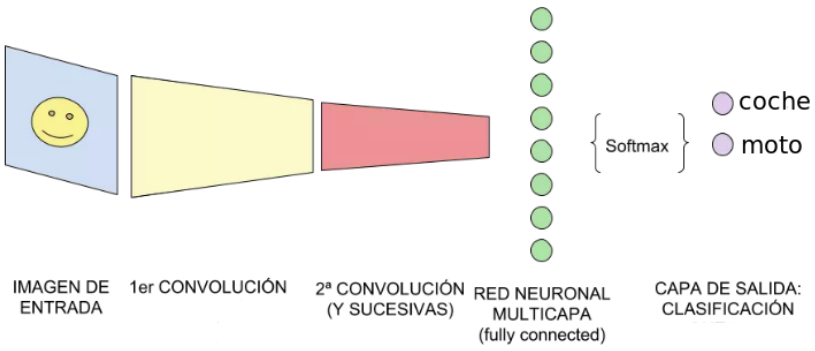

Figura 2.2: Arquitectura de una red neuronal convolucional (CNN).

La red de neuronas tolera fallos por lo que si parte de la red se corrompe y no trabaja podría seguir reteniendo estas capacidades y funcionando correctamente. Por último, una de las grandes ventajas que tiene es el aprendizaje adaptativo, proporcionando la característica de que partiendo de un modelo básico sea capaz de adaptarse a nueva información, buscando el reconocimiento de patrones semejantes a los ya aprendidos.

Hay que tener en consideración que pueden presentar una complejidad y tiempo de aprendizaje altos debido a que cuanto mayor es el número de tareas que necesitamos que aprenda, aumentará la complejidad, y si necesitamos identificar un gran número de patrones junto con mayor adaptación a nuevos patrones semejantes, necesitaremos más tiempo de entrenamiento. Esto requiere que para una tarea compleja, es necesario definir una gran cantidad de datos para el entrenamiento. En la red, se produce un efecto de caja negra en las capas de procesamiento no sabiendo la relación ni los pesos entre las variables y obteniendo como salida un valor que tiene que ser interpretado por el programador. [3, 4]

2.2.1 Redes neuronales convolucionales

Las redes neuronales convolucionales (del inglés CNN), son un tipo de aprendizaje automático derivado de las redes de neuronas artificiales. Los fundamentos de esta técnica fueron establecidos por Kunihiko Fukushima en 1980, en su obra *Neocognitron* [5]. Toman como inspiración el córtex visual de los animales para identificar características en las entradas que reciben, siendo diseñadas para trabajar con imágenes y tratando de otorgar la capacidad de ver al computador.

Las CNN contiene al menos una capa de convolución y están organizadas de manera jerárquica por lo que las primeras capas pueden detectar simples patrones en las imágenes, pero

a medida que descendemos y aumentamos el número de capas, se van especializando hasta llegar al reconocimiento de patrones complejos. En la figura 2.2 se puede ver la arquitectura de una red CNN. La estructura de esta red se basa en las siguientes operaciones:

- Entrada de la red: Cuando se indica la entrada a la red se realiza la transformación de la información que recibe (píxeles) a neuronas. En una imagen de entrada con las dimensiones 640x480x3 obtendríamos una capa de entrada con 92.160 neuronas. Para realizar esta operación debemos de cerciorarnos de que la entrada se encuentra normalizada, ya que se espera que los valores de entrada oscilen entre 0 y 1.

- Convolución: Es el proceso característico de este tipo de redes. Con un kernel definido, vamos recorriendo la imagen y realizando la operación del producto escalar entre los píxeles vecinos y el kernel definido. Este resultado será guardado en una matriz por lo que al final de este proceso iterativo conseguimos una nueva capa oculta. Si nos encontramos en una imagen de un canal obtenemos una matriz que será la capa oculta, pero en caso de trabajar con imágenes con 3 canales (RGB) se obtienen 3 matrices por cada canal que se combinarán para formar una única capa oculta.

  Al realizar la convolución se obtiene una matriz de menor tamaño por lo que puede ser necesario igualar las dimensiones originales. Para ello existe la operación padding que se encarga de agregar alrededor de la matriz original datos con valor cero. Gracias a esta operación también evitamos que si la información relevante se encuentra en las esquinas, al realizar la convolución serán más significativos los píxeles centrales, ya que se pasa más veces por ellos, por tanto añadiéndole esta información, se acerca hacia el centro de la matriz. En la figura 2.3 se puede ver el esquema del proceso de convolución.

  Por último, se aplica la función de activación sobre la matriz. Esta función se encarga de operar sobre la capa de salida, transformando los valores que obtuvimos en la convolución y propagando la salida hacia la siguiente capa. Es habitual usar la función RELU, que modifica los datos negativos dándoles un valor de 0 mientras que mantiene los positivos iguales, por lo que es una manera eficiente de desactivar las neuronas negativas. Su fórmula es la siguiente:

  $$f(x) = max(0, x)$$

- Reducción de muestreo: A medida que aplicamos la operación de convolución, se van generando nuevas neuronas. Con el proceso de reducción de muestreo conseguimos reducir el tamaño de la red, pero manteniendo las características más importantes que fueron encontradas en el paso anterior. Para llevar a cabo esta tarea suele aplicarse el algoritmo de Max-Pooling. Este se basa en la búsqueda del valor máximo en una ventana de muestra y pasa este valor como resumen de características sobre esta zona.

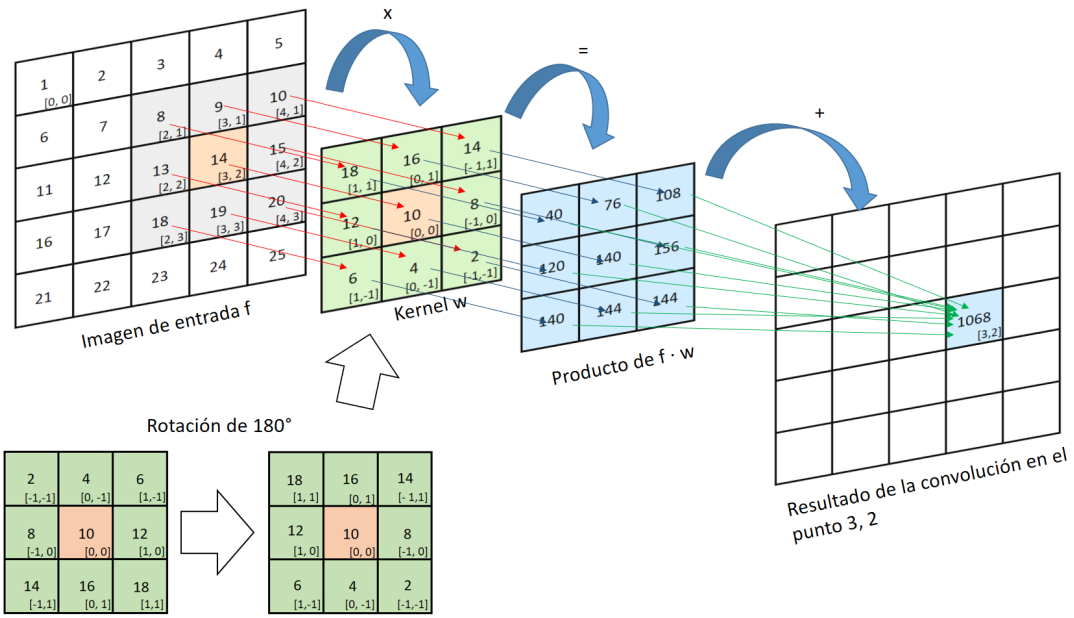

Figura 2.3: Proceso de convolución.

- Capas tradicionales: Una vez que realizamos la reducción de nuestra red debemos añadir las capas para realizar la clasificación. Antes es necesario realizar una transformación para eliminar la tridimensionalidad de la red que se conoce como aplanamiento. Con este aplanamiento aplicado podemos conectar una o más capas completamente conectadas hasta la última capa, que debe de tener el mismo número de neuronas que las clases que deseamos obtener. A esta última capa se le suele aplicar la función Softmax que distribuye la probabilidad de que la imagen sea etiquetada con cada clase. Por ejemplo, si deseamos obtener la clasificación de imágenes entre coches y motos deberíamos de terminar con una capa de 2 neuronas simples y Softmax nos indicaría la probabilidad de clasificar cada imagen como coche o moto.

En la actualidad existen diferentes redes neuronales convolucionales con gran reconocimiento debido a los buenos resultados de funcionamiento que se obtienen. Esta comparación se realiza en el desafío de reconocimiento visual a gran escala de ImageNet (ILSVRC) [6], que evalúa los algoritmos para la detección y clasificación de imágenes a gran escala. Gracias a esta competición se puede observar la evolución del progreso de la visión artificial. En la figura 2.4 vemos como descendió la tasa de error desde un 28% a un 2,3% entre el 2010 y el 2017.

Las redes neuronales convolucionales son la base de los modelos actuales para la visión por computador gracias a que son modelos mucho más flexibles y escalables que las redes tradicionales, donde se usan demasiados parámetros y no se pueden aplicar sobre problemas

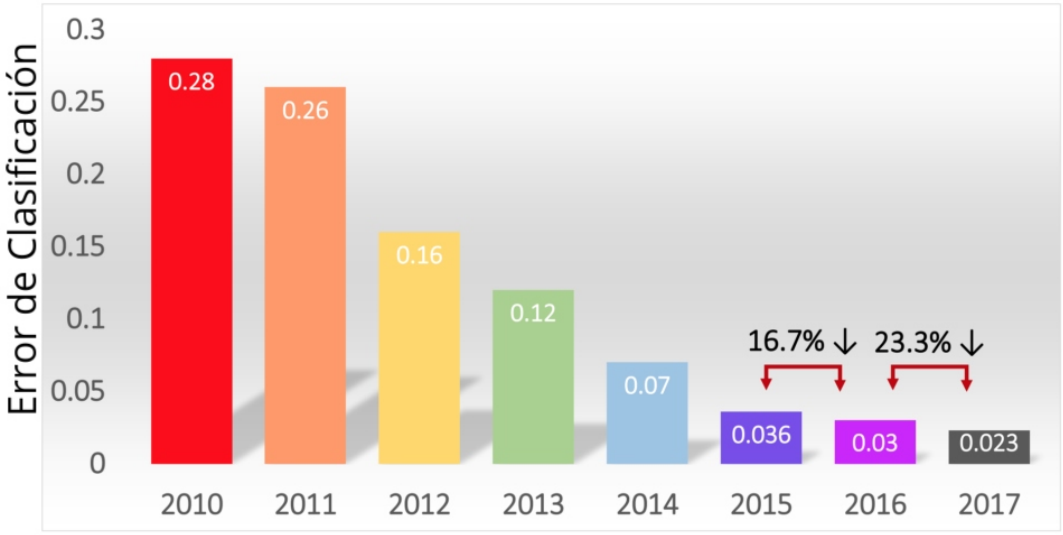

Figura 2.4: Tasa de error de clasificación en los 5 primeros algoritmos de la ILSVRC entre 210-2017.

reales.También son necesarias menos interconexiones, ya que no tenemos que tener todas las neuronas de una capa conectada con la siguiente, minimizando la computación comparando con la red tradicional y estando orientadas a obtener un buen rendimiento, debido a que se extraen características invariantes de la ubicación. Por todos estos beneficios fue el algoritmo elegido para llevar a cabo esta tarea. [5, 7]

2.3 Visión por computador

La visión por computador o visión artificial es una disciplina que nació como un campo de la inteligencia artificial que estudia como procesar y analizar las imágenes del mundo real con la finalidad de que puedan ser tratadas en una computadora mediante la transformación a información numérica. Para dotar de estas características al equipo es necesario la combinación de hardware y software adecuado que posibilite la captura de imágenes y datos sobre ellas.

Esta rama está presente en la tecnología que nos rodea como puede ser en la capacidad de otorgar visión 3D sobre un conjunto de imágenes 2D, la detección de objetos en sistemas de reconocimiento, por ejemplo, los sistemas de reconocimiento facial o en el análisis de vídeo.

2.3.1 Flujo óptico

El concepto de flujo óptico consiste en analizar el movimiento que realiza el observador en la escena y fue acuñado por El James J. Gibson en 1950 en su trabajo "The Perception of the Visual World" [8]. Por tanto el flujo óptico es el patrón de movimiento de los distintos objetos de la escena a través del espacio del tiempo, causado por el desplazamiento del ente y observador.

El flujo óptimo nos permite detectar el movimiento que se produce en la imagen, segmentar los objetos o, el caso usado en este proyecto, el control de la navegación gracias al rastreo de movimiento entre imágenes.

Capítulo 3

Fundamentos Tecnológicos

En este capítulo se indican las distintas tecnologías necesarias para el desarrollo del proyecto y se indica una descripción de lo que nos aportan al proyecto. Empezaremos con el equipamiento hardware y después se comentan las diferentes librerías y herramientas software. Para la elección de las distintas herramientas se buscaron aquellas que nos aportan un beneficio mayor que la competencia a la hora de desarrollar el proyecto. Todas estas tecnologías son de código abierto y gratuitas.

3.1 Hardware

En el hardware destacamos el uso del robot móvil, la placa de computación, la cámara y los circuitos.

3.1.1 Robot móvil

Se dispone de un robot Ranger Makeblock, robot educativo que permite la construcción de distintos tipos de vehículos como puede ser un coche o un tanque con una placa de control Me Auriga, basada en Arduino y compatible con sus IDLEs, que tiene la tecnología Bluetooth, lo que permite conectarse remotamente y ejecutar el código sin estar sujetos a una conexión alámbrica. Esta placa cuenta con puertos para controlar los distintos sensores y motores de los que dispone como son los giroscopios, sensor de sonido, buzzer pasivo o el sensor de temperatura. La conexión a estos puertos se realiza de mediante los conectores RJ25.

Para la simulación de un coche, los robots están construidos como se observa en la imagen 3.1 con los motores conectados por los puertos M1, M2 de la placa que podemos observar en la figura 3.2, y las dos ruedas con sus motores en la parte frontal del vehículo. También se dispone de un sensor para poder seguir las líneas en la parte inferior del robot y una rueda loca en la parte posterior que puede rodar hacia cualquier dirección. Por último, la placa Auriga

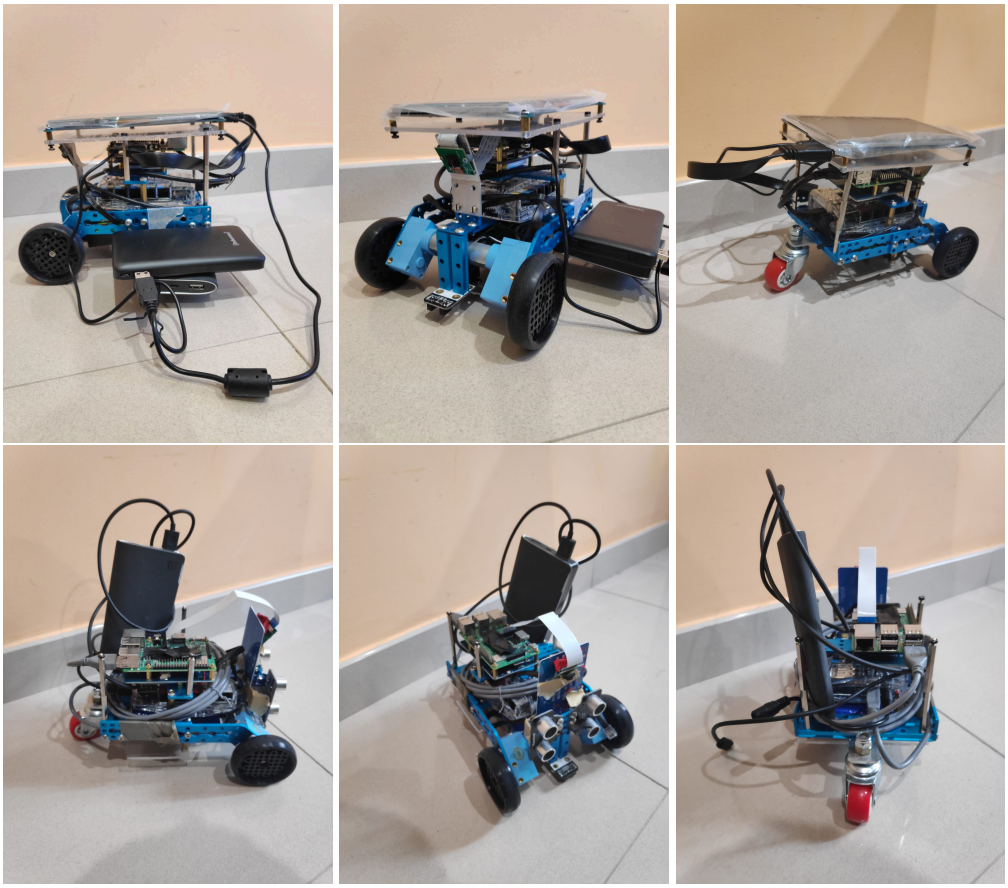

Figura 3.1: Los dos robots Makebock Ranger empleados en este proyecto.

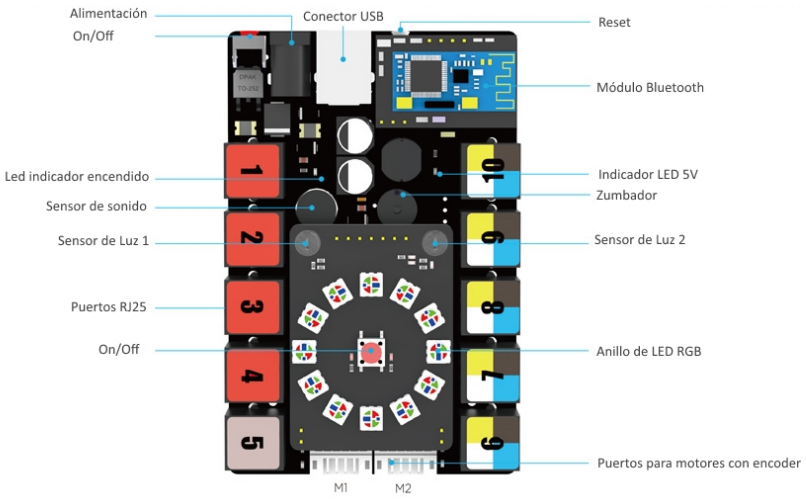

Figura 3.2: Placa Auriga que controla los robot Makeblock Ranger.

va alojada en la parte superior del robot. Esta placa es la encargada de la ejecución final de órdenes sobre los motores y será el medio por el que nos comunicaremos al robot.

Los motores de ambos robots deberían de aportar la misma potencia, sin embargo existe una diferencia entre la salida que aporta cada motor para un mismo valor provocando un leve giro aun cuando le indicamos que debe ir recto. La rueda loca ubicada en la parte posterior también influye en este giro, ya que si no está ubicada en el centro de los ejes de las ruedas refuerza este desplazamiento. Sobre el robot superior de la figura 3.1 fue posible desplazarlo sin realizar ninguna corrección. Sin embargo, el robot inferior necesitó un ajuste de 15 unidades de velocidad sobre el motor izquierdo para tratar de igualar a la potencia que entrega del motor derecho, ya que este motor funciona de manera anormal ofreciendo poca fuerza de arrastre.

Otro problema relacionado con los motores del robot fue su funcionamiento no lineal, ya que toda velocidad inferior a 20 unidades en un motor no es capaz de conseguir desplazar el robot, por lo que para poder realizar movimientos con una sola rueda, es necesario al menos un valor de 50 unidades de velocidad sobre el motor. Si ambos motores se encuentran funcionando, la velocidad mínima para desplazarse es de 30 unidades.

Como alimentación, para la placa Auriga y sus motores se usan 6 pilas de tipo AA de 1,2 V con una capacidad de 2500 mAh o una batería de Li-Ion. [9]

3.1.2 Raspberry Pi

Se cuenta con dos placas Raspberry versión 3 Model B+ que cuenta con una CPU Quad Core de 1,4 GHz Broadcom BCM2837 64 bit y con 1 GB RAM, que podemos ver en la figura 3.3. Cuenta con conexión Wi-Fi de 2.4 GHz y 5 GHz y Bluetooth 4.2. Esta característica permite el manejo de la placa a distancia. También cuenta con 4 puertos USB 2.0, conexión

Figura 3.3: Placa Raspberry Pi 3 Model B+ con módulo de la cámara conectado.

CSI para módulos externos y entrada HDMI. El puerto de alimentación es de tipo micro-USB, necesitando una alimentación de 5 V y 2.5 A aunque podría trabajar con menor amperaje.

La placa Raspberry se encuentra en la parte superior de ambos robots conectada a la placa de Auriga a través del puerto USB. Para la suministración de energía de la placa Raspberry se cuenta con el adaptador original así como baterías externas para su ejecución en remoto. [10]

3.1.3 Cámara

El módulo de la cámara que se ha usado para ambos robots es el modelo Raspberry Pi Camera Board v1.3. Este módulo cuenta con un sensor Omnivision 5647 que soporta la captura de imágenes a 5 MP con una resolución de 2592x1944 px. También soporta grabación en 1080p con una tasa de 30 fps, aunque podemos aumentar la tasa hasta 90 fps si descendemos la resolución a 640x480p.

El módulo se conecta a la Raspberry Pi por medio de un cable plano de 15 pines sobre el puerto CSI con el que cuenta la placa, diseñado especialmente para interconectarse con cámaras. En la figura 3.3 podemos ver la placa junto al módulo aquí descrito y conectado a través del puerto CSI.

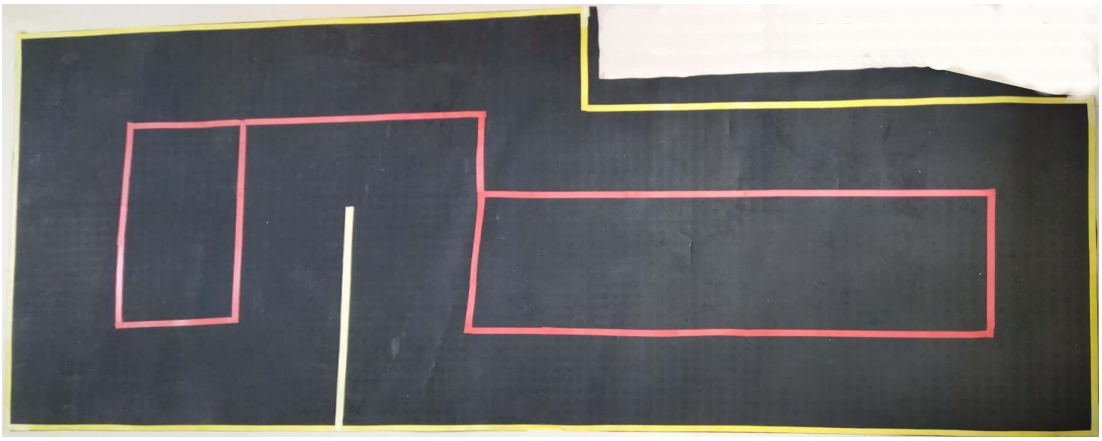

Figura 3.4: Uno de los circuitos usados en la experimentación.

3.1.4 Circuitos

En la figura 3.4 vemos un ejemplo de circuito creado en este proyecto. Los circuitos que se crearon contienen el fondo negro o azul. Los de color negro están elaborados sobre una goma dura y no desliza el robot. Sin embargo, sobre los circuitos azules ocurrieron problemas de adherencia, debido a que están elaborados de goma EVA por lo que, con el peso del robot, se hunde ligeramente y patina pudiendo provocar que el robot pierda el agarre.

3.2 Lenguajes de programación

Los lenguajes de programación usados fueron: Python, junto con Jupyter Notebook, y C++.

3.2.1 Python

Es un lenguaje de programación interpretado, orientado a objetos y de alto nivel, creado por Guido van Rossum a finales de los ochenta. Cuenta con estructuras integradas de alto nivel que, combinadas con el tipado dinámico, lo hacen muy atractivo para el desarrollo rápido de aplicaciones así como para su uso de *scripts*. La sintaxis simple de Python enfatiza en la legibilidad. Todas estas características permiten que se realice un código limpio con una estructura sencilla y fácil de comprender, sumado a las ventajas que aportan los lenguajes interpretados. [11]

Actualmente, es de los lenguajes más usados gracias a las ventajas que se describieron, empleado tanto por usuarios experto como por personas que quieren iniciarse en la programación. Además, Python incorpora librerías para diferentes campos de las ciencias siendo el lenguaje escogido por científicos e ingenieros de diversas ramas. Para implementar este có-

digo se hicieron uso de diferentes librerías que serán descritas en las siguientes secciones. La mayor parte de este proyecto está desarrollado en Python diferenciando dos secciones:

- El código para su ejecución en la Raspberry formado por ficheros .py.
- El código para la creación de los algoritmos de aprendizaje automático desarrollados a través de notebooks y con formato .ipynb.

3.2.2 C++

Es un lenguaje de programación que surge como extensión del lenguaje C. Fue creado por Bjarne Stroustrup en los años ochenta y su principal objetivo era otorgarle programación orientada a objetos al lenguaje C, que en aquel momento era de los lenguajes más demandados. Heredó toda la sintaxis de este lenguaje y contiene el paradigma de la programación estructurada nativa de C junto con la extensión para soportar el paradigma de programación orientada a objetos, por lo que es considerado un lenguaje multiparadigma. [12]

Al igual que Python, este lenguaje se encuentra en la clasificación de los lenguajes más usados. Sin embargo, a diferencia de Python, no resulta un lenguaje tan fácil de manipular por lo que es conveniente tener nociones de programación para su uso.

3.2.3 Jupyter Notebook

Jupyter Notebook es un entorno informático interactivo basado en la web para creación de documentos. Fue creado por Fernando Pérez dentro del Proyecto Jupyter. Se basa en una aplicación cliente-servidor que tiene que ser lanzada por el usuario y a la que se puede acceder mediante cualquier navegador una vez iniciado el servidor. Los documentos creados por Jupyter pueden ser exportados a otros formatos como PDF o HTML. La principal ventaja de este notebook se basa en la programación interactiva que nos facilita realizar modificaciones en bloques de código y ejecutarlo sin necesidad de ejecutar todo el fichero. Además, se puede obtener en directo las salidas, quedando registradas sin necesidad de almacenarlas en un fichero, por lo que facilita el trabajo de documentación y extracción de información para el análisis de los datos obtenidos tras la ejecución. [13]

La estructura interna del programa se basa en dos componentes:

- Los *kernels* o intérpretes: Forman los motores de ejecución y es posible tener más de uno si se trabaja con varios ficheros. En nuestro caso se hizo uso del kernel de IPython, pero existe soporte para los principales lenguajes de programación como C++, R, Java o PHP entre otros. En caso de finalizar la ejecución del motor de un fichero las variables que este almacenaba sobre nuestro código son borradas manteniendo la salida que se obtuvo en la ejecución, por lo que si posteriormente deseamos volver a ejecutar algún bloque del código con variables dependientes necesitamos cargarlas anteriormente.

- Panel de control: El panel de control o *Dashboard* es la interfaz web en la que se inicia el navegador una vez que lanzamos el servicio. Se muestran las carpetas que contiene el directorio desde el que fue lanzado el servidor y nos proporciona una interfaz para los trabajos relativos a la creación, modificación o eliminación de los archivos en este directorio. Otra de las opciones que nos facilita es realizar un control de los distintos kernels que se encuentran abiertos con nuestros archivos y su estado.

El proyecto Jupyter tiene otros productos innovadores y continúa desarrollando aplicaciones preparadas para los nuevos requisitos de los usuarios con productos como JupyterLab, manteniendo las funcionalidades de Jupyter Notebook y añadiendo flexibilidad a la interfaz del usuario o siendo extensible y modular, facilitando agregar nuevos componentes a los ya existentes. Por todas las facilidades que nos aporta Jupyter Notebook a la hora de ejecutar código de forma independiente, organizando en bloques, la facilidad para la documentación y visualización de los datos o incorporar funciones con el código ejecutado para realizar cálculos de rendimiento. Se decidió usar esta aplicación para el desarrollo del código en el ordenador donde se llevó a cabo, ya que, debido a la necesidad de obtener un rendimiento superior, en la placa Raspberry únicamente si hizo uso de las aplicaciones esenciales para ejecutar los algoritmos de aprendizaje máquina.

3.3 Manipulación de imágenes y aprendizaje automático

Para la manipulación de imágenes se ha usado la librería NumPy de Python, la extensión Pandas y la biblioteca OpenCV. Como librería para el aprendizaje automático hemos usado TensorFlow.

3.3.1 NumPy

NumPy es una biblioteca para Python que da soporte en la creación de vectores y matrices multidimensionales así como a las operaciones sobre estos tipos de datos gracias a la colección de funciones matemáticas que tiene disponible para aplicar.

Antes de la creación de NumPy, existían dos librerías para realizar operaciones similares llamadas Numeric y Numarray. En 2005 Travis Oliphant entendió que era necesaria la fusión de estas dos librerías, por lo que decidió extender Numeric con las funcionalidades de Numarray y creo NumPy, cuya primera versión salió en 2006. [14]

NumPy aporta grandes ventajas para la manipulación de grandes matrices de datos debido a que son más eficientes que estructuras de datos tradicionales como las listas o tuplas. También admite operaciones matemáticas avanzadas como el álgebra lineal u operaciones vectorizadas por lo que aporta las funcionalidades de Matlab al lenguaje Python.

3.3.2 Pandas

La librería Pandas nació como una extensión de la biblioteca NumPy, desarrollada en 2008 en AQR Capital Management por medio de Wes McKinney y publicada con libre acceso en 2009. Su objetivo es ser la herramienta fundamental de alto nivel para realizar la manipulación y el análisis de datos en Python. [15]

Pandas incorpora herramientas para la lectura y escritura de datos sobre los principales formatos como CSV, archivos de texto o bases de datos. También incorpora herramientas inteligentes para la alineación de los datos desordenados y el manejo de datos con elementos no válidos. Por último, se prestó gran atención sobre la importancia del rendimiento presentando una biblioteca altamente optimizada, por lo que las operaciones como uniones de grandes *datasets*, se realizan eficientemente con un gran rendimiento.

Todo esto le ha llevado a adquirir una presencia tanto en el ámbito académico como comercial siendo usado en diferentes disciplinas como las financias, la neurociencia o incluso la publicidad y el análisis web.

3.3.3 OpenCv

OpenCV es una biblioteca para desarrollar tareas de Visión Artificial. Fue creado por Intel y lanzada en una primera versión en 1999. Actualmente sigue siendo una de las librerías más populares de Visión Artificial, soportando distintos lenguajes de programación como Python, Java, Matlab y C++, que es el lenguaje en el que está desarrollada. También aporta una gran documentación y diversas explicaciones con una extensa lista de tutoriales que se mantiene activa y actualizada para los distintos lenguajes de programación que soporta. [16]

OpenCv proporciona un entorno de desarrollo fácil de manejar y muy eficiente, permitiendo aplicar gran cantidad de los principales algoritmos de visión artificial, con más de 2500 implementaciones para realizar tareas como son el reconocimiento de gestos o facial, la segmentación de las imágenes, o la aplicación de técnicas de transformaciones morfológicas sobre imágenes. Por otra parte, soporta también técnicas de aprendizaje automático como el soporte de máquinas vectoriales o los árboles de decisión.

Por último, OpenCV es multiplataforma soportando la placa de Raspberry Pi. El código está altamente optimizado y es robusto. Prueba de ello es que desde su creación no ha recibido un cúmulo de actualizaciones mayores encontrándose actualmente en la versión 4. Por estas ventajas fue una de las librerías más utilizadas tanto en el código del lenguaje C++ como en el lenguaje Python y en la plataforma de la Raspberry Pi.

3.3.4 Tensorflow

TensorFlow es una librería para el aprendizaje automático desarrollada por Google y lanzada en 2015 que proporciona diferentes técnicas de algoritmos de aprendizaje automático. Su objetivo principal es la creación de una librería para poder construir y entrenar sistemas de redes neuronales muy eficientes. Gracias a que su arquitectura flexible, permite implementar los cálculos en una o más CPU o GPU con una sola API. Está disponible en diferentes lenguajes como Python, C++ o Java entre otros, y soporta diferentes plataformas. [17]

Para mayor eficiencia, existe la versión ligera Tensorflow Lite [18], que proporciona un conjunto de herramientas para ayudar a ejecutar modelos de TensorFlow creados anteriormente en un equipo con mayores recursos, sobre dispositivos del IoT (Internet de las cosas) con una baja latencia y un tamaño de objeto binario pequeño. Tensorflow Lite se estructura en dos componentes:

- El conversor: Para poder ejecutar los modelos creados en Tensorflow es necesario aplicar el conversor de la versión Lite, que nos proporciona un nuevo modelo en un formato eficiente y que puede aplicar optimizaciones para la mejora del tamaño y rendimiento de los objetos.

- El intérprete: Este nuevo modelo es ejecutado gracias a este componente diseñado para ser ágil y rápido que se encarga de iniciar el modelo y realizar las predicciones de los datos que recibe como entrada.

Tensorflow incluye las funcionalidades de librería conocida como Keras, con la que colabora, para aportar técnicas de aprendizaje profundo. Por tanto, TensorFlow ofrece un ecosistema con una gran cantidad de herramientas de aprendizaje automático para poder desarrollar nuestro código y ejecutarlos en distintas plataformas, sumado a la extensa documentación y los diferentes tutoriales de los que dispone en su página web oficial.

3.4 Librerías de control del robot

En la Raspberry Pi hemos instalado las librerías necesarias para controlar el robot (AurigaPy) y gestionar la cámara (Picamera)

3.4.1 Aurigapy

Aurigapy es una librería creada por Fidel Aznar en 2018 con el objetivo de poder conectar el robot mRanger a una placa externa a través de esta librería desarrollada para Python [19].

Gracias a esta librería, podemos ejecutar diferentes acciones a través de nuestra placa Raspberry, ya que implementa las funcionalidades básicas de los movimientos de los motores así como la captura de datos de los principales sensores.

En este proyecto se ha utilizado la versión de publicación, debido a que es la única publicada desde el lanzamiento de la librería y fue necesaria su modificación para adaptar los comandos que se han usado en este proyecto.

3.4.2 Picamera

Picamera [20] es la librería que permite la ejecución de una cámara sobre una placa Raspberry Pi. Para su ejecución, debemos de obtener un módulo de cámara para esta placa y conectarlo al puerto CSI. Una vez conectada debemos de habilitar en la configuración la opción de la cámara y reiniciar la placa. Desde este momento, podemos hacer uso de distintas funcionalidades de esta librería como la captura de una o varias imágenes, grabar vídeo o realizar pequeñas modificaciones tales que capturar imágenes dimensionadas.

Picamera ofrece también operaciones avanzadas soportando trabajar con las librerías NumPy y OpenCV. Esto permite la captura de datos con los formatos que manejan estas librerías para una posterior manipulación, agilizando el procesamiento de las imágenes para trabajar con técnicas de aprendizaje máquina que describiremos en este proyecto.

3.5 Herramientas de edición y gestión de versiones

3.5.1 Matplotlib

Matplotlib es una biblioteca para la edición gráfica a partir de datos contenidos en listas o con los tipos de arrays con los que trabaja Numpy. Esta biblioteca contiene una colección de funciones que permite la simulación del comportamiento de las gráficas como las que podemos encontrarnos en MATLAB, pudiendo mostrar una gráfica con diferentes formas o tipos (Histograma, Diagrama de barras, Diagramas de puntos), o la manipulación y aplicación de pequeñas modificaciones sobre ella como el trazado de líneas en una área o la colocación de etiquetas. [21]

3.5.2 Overleaf

Overleaf [22] es un editor colaborativo de LaTeX, basado en la nube, que se utiliza para escribir, editar y publicar documentos científicos, lo que nos permite trabaja remotamente en paralelo. Fue creado en 2014 por John Hammersley y John Lees-Miller.

En este editor se ha documentado todo el desarrollo del proyecto formando esta memoria.

3.5.3 Git

Es un software de control de versiones diseñado por Linus Tovard en 2007 [23]. El control de versiones es la práctica de gestionar los cambios que se van desarrollando en el código. Estas herramientas nos permiten llevar un control sobre los distintos cambios que se van realizando y nos permiten retornar a una versión anterior para resolver errores.

3.5.4 Geany

Geany [24] es un editor de texto ligero que incluye características de un entorno de desarrollo. Fue creada en 2005 por Enrico Tröger. Soporta los lenguajes usados en este proyecto de las secciones 3.2.2 y 3.2.1.

Fue el editor de texto elegido debido a que viene preinstalado en todos nuestros equipos.

Capítulo 4

Gestión del Proyecto

PARA el desarrollo de este proyecto, fue necesario la definición de una metodología de trabajo, continuando con el análisis de los requisitos, la planificación, el seguimiento del desarrollo y la realización de una evaluación del coste del proyecto.

4.1 Metodologías de desarrollo

Existen diferentes tipos de metodología para el desarrollo de proyectos como pueden ser los métodos en cascada o en espiral. En este proyecto fue necesario la búsqueda de una técnica que nos aporte gran flexibilidad. Para ello, fue escogida la metodología ágil.

4.1.1 Metodologías Ágiles

Las metodologías ágiles son un enfoque de desarrollo donde los requisitos pueden ir evolucionando en cada etapa permitiéndonos adaptar el proyecto a medida que avanza. Las metodologías ágiles se basan en el manifiesto *agile* [25], elaborado en 2001 en Utah por 17 científicos. Este manifiesto recoge también los 12 principios de esta metodología [26]. También contiene los valores en los que se basa que son:

- Individuos e interacciones sobre procesos y herramientas.
- Software funcionando sobre documentación extensiva.
- Colaboración con el cliente sobre negociación contractual.
- Respuesta ante el cambio sobre seguir un plan.

Esta metodología puede resultar compleja en cuanto a la elaboración de un calendario con el tiempo necesario para completar el producto. Esto se debe a la posibilidad del aumento progresivo, sin un límite en el proyecto. Sin embargo, permite la fácil detección de las tareas

más relevantes en cada momento, basándonos en las necesidades del cliente. También permite que el cliente pueda ver resultados en cualquier punto, provocando una retroalimentación más rápida que las metodologías clásicas.

4.1.2 Aplicación a nuestro proyecto

En el desarrollo del proyecto se ha escogido la metodología ágil *Scrum*. De esta forma el equipo de desarrollo tiene unas tareas en unos periodos de tiempos determinados llamados *sprints*, que se corresponde con ciclos de tiempo cortos de una duración fija. Cada iteración debe de proporcionar una tarea completa hasta conseguir el producto final como combinación de estas pequeñas tareas, provocando que el producto final sea fácilmente entregado.

La base fundamental de esta técnica son las revisiones o reuniones, ya que plasman la transparencia del proceso y fomentan la comunicación para una mayor eficacia. Por ello, se realizó una primera planificación con el asesor externo, estableciendo los objetivos que se deben cumplir en cada sprint. Tras finalizar cada sprint se realizó una nueva reunión con el asesor externo, con la evaluación de los resultados, la actualización de los requisitos, correspondiente a la reunión de revisión, y se prosiguió elaborando una nueva planificación los nuevos objetivos.

4.2 Análisis de requisitos

Nuestro sistema debe de cumplir una serie de requisitos para poder considerarlo válido. Estos requisitos pueden ser de dos tipos: funcionales y no funcionales.

4.2.1 Requisitos funcionales

Los requisitos funcionales se definen como el comportamiento y la función que debe de tener el sistema. Los requisitos funcionales definidos son:

- El sistema no debe exceder los límites del circuito o en su defecto, puede invadir la parte externa del circuito siempre y cuando no lo abandone completamente y debe de volver a su punto de partida, a excepción de cuando se ordene lo contrario en la tarea seguir carril.

- El sistema debe de conseguir recorrer la línea volviendo al punto de partida en la tarea seguir línea.

- Deben de almacenarse las imágenes capturadas por la cámara a lo largo de todo el recorrido.

- Deben de almacenarse en un fichero .csv el listado del nombre de las imágenes y la predicción realizada junto con los tiempos de preprocesado, del algoritmo de aprendizaje, de ejecución del comando y del tiempo entre captura de imágenes.
- Cada versión del algoritmo debe de poder ser usado sobre diferentes circuitos.
- Cada versión del algoritmo debe de poder ser usado sobre diferentes robots.
- Los circuitos no puede encontrarse con tramos de carril cerrados ni obstáculos.
- La superficie del circuito no debe de producir brillo.
- El circuito puede tener uno o más carriles, pero cada carril debe de estar señalizado con cinta en ambos lados.
- Los carriles deben de tener al menos el ancho del robot, un mínimo de 17 cm.
- La superficie del circuito debe de ser plana.
- Debe de agilizarse el proceso para realizar una nueva tarea, debiendo facilitar la elaboración de un *dataset* y reentrenar la red.

4.2.2 Requisitos no funcionales

Este tipo de requisitos, también conocidos como atributos de calidad, indican restricciones funcionamiento que debe cumplir el sistema. En nuestro sistema, estos requisitos son:

- Las ejecuciones deben de realizarse en tiempo real, considerando que estas se realizan de este modo cuando el tiempo final medio es inferior a 100 ms.
- Debe realizarse el salvado de las imágenes con un nombre único que guarde la información del momento en el que fueron capturadas.
- Debe de poder ejecutarse el sistema remotamente.
- El sistema debe de poder ser capaz de ejecutar otro modelo de la forma más óptima posible.
- El sistema debe de soportar la ejecución en dispositivos que soporten la librería de la sección 3.3.4, TensorFlow Lite.

4.3 Planificación y seguimiento

Como se ha comentado en la sección 4.1.2, la metodología de desarrollo escogida fue *Scrum*. Por ello se llevaron a cabo una serie de *sprints*. A continuación vamos a comentar cada uno.

4.3.1 Sprint 0: Primeros pasos

Planificación

Fue la reunión inicial donde se establecieron los primeros pasos para el desarrollo del proyecto llevando a cabo las siguientes acciones:

- Definir el equipo de trabajo hardware.
- Definir los objetivos de las tareas a realizar.
- Definir la técnica de aprendizaje automático.
- Formación en el marco teórico e investigación.

Revisión

Tras finalizar el primer sprint se realizó la revisión inicial donde se modificaron los planes del proyecto porque, debido a las restricciones provocadas por la pandemia global, no fue posible obtener el hardware inmediatamente.

4.3.2 Sprint 1: Desarrollo de una red de clasificación

Planificación

En el siguiente sprint se establecieron las siguientes tareas:

- Búsqueda e instalación del software más adecuado.
- Creación de un *dataset* de imágenes.
- Diseño y desarrollo de un sistema de clasificación manual sobre una CNN.

Revisión

En esta revisión se justificó la elección de los diferentes software elegidos y aunque sin el hardware, se desarrolló el sistema de clasificación después de crear un *dataset* con unas imágenes de referencia enviadas por el asesor externo.

4.3.3 Sprint 2: Ejecución de modelos CNN en la Raspberry

Planificación

En la planificación de este sprint ya se contaba con todo el hardware por lo que se establecieron varias tareas nuevas:

- Instalación del software sobre la placa Raspberry.
- Diseño y desarrollo de un script para ejecutar el modelo sobre la Raspberry.

- Realización de pruebas sobre el modelo creado en la Raspberry.
- Automatización del proceso de creación del *dataset*.
- Nueva red de clasificación con este *dataset* automático.

Revisión

Acabado el segundo sprint, se analizaron los resultados obtenidos sobre modelos entrenados en la clasificación con *dataset* manual y con el *dataset* obtenido tras automatizar el proceso de etiquetado. Debido a los altos porcentajes de éxito obtenidos con el *dataset* manual y que no se tuvieron en cuenta todas las características visuales a la hora de realizar el *dataset* automático se decidió no continuar con su desarrollo.

4.3.4 Sprint 3: Desarrollo de una regresión y ejecución en la Raspberry

Planificación

En la planificación de este sprint se realizaron tareas de mejora y se incorporaron tareas nuevas:

- Optimización de los tiempos de ejecución sobre la Raspberry.
- Creación de un *dataset* con valores continuos de manera automática.
- Diseño y desarrollo de una red de regresión convolucional sobre el *dataset* continuo.
- Diseño y desarrollo del *script* para ejecutar el modelo sobre la Raspberry.
- Realización de pruebas de ejecución.

Revisión

Tras finalizar las tareas planificadas, se realizó una comparación de la red de regresión frente a la clasificación, comprobando que la regresión tenía un comportamiento más suave sobre las tareas debido a la continuidad de sus datos por lo que no se continuó con el desarrollo del modelo de clasificación.

4.3.5 Sprint 4 : Adaptación de la CNN a una nueva tarea

Planificación

En este sprint se realizaron las últimas tareas de desarrollo:

- Creación de un nuevo *dataset* con el objetivo de comprobar el funcionamiento de la red de regresión sobre una nueva tarea usando el "automatizador" desarrollado.
- Entrenamiento de la red para la nueva tarea Seguir Línea.

Tabla 4.1: Costes de los materiales necesarios para ejecutar el proyecto.

Material	Cantidad	Precio unitario (€)	Total (€)
Portátil Lenovo G-50-70 i7-4510U, 16 GB RAM	1	0	0
Placa Raspberry Pi 3 B+	2	38	76
Robot Makeblock Ranger	2	173	346
Raspberry Pi Camera Board v1.3	3	13	39
Suelo de circuito	3	17	51
Batería de litio 6000 mAh	1	10	10
Pilas duracell HR6 AA 2500 mAh pack 4 pilas	2	11	22
Batería externa 24000 mAh	1	26	52
Batería externa 2080 mAh	1	40	52
Total			**610**

- Ejecución sobre la Raspberry.
- Realización de pruebas sobre los modelos desarrollados.

Revisión

Tras comprobar el funcionamiento y la fiabilidad sobre una nueva tarea se da por finalizado el proceso de desarrollo de manera exitosa.

4.3.6 Sprint 5: Memoria

Planificación

Como último sprint se realizó la memoria del proyecto recopilando la información de las pruebas y vídeos demostrativos del sistema.

Revisión

En esta revisión se comprobó la memoria y se dio por concluido el proyecto.

4.4 Costes del proyecto

Se realizó un cálculo de los costes del proyecto teniendo en cuenta el coste humano y el coste material, donde no fue necesario la compra de ningún tipo de software al trabajar con herramientas de licencia libre.

4.4.1 Material

En el coste del material no hubo que emplear una gran suma de dinero debido a que el mayor coste en cuanto a material se refiere, correspondería con la adquisición del ordenador portátil, pero, en este caso, ha sido usado un equipo que se encuentra amortizado.

Tabla 4.2: Costes de personal del proyecto.

Tarea	**Analista (35€/h)**	**Programador (25€/h)**	**Asesor (50€/h)**	**Jefe proyecto (70€/h)**	**Total (h)**	**Coste (€)**
Sprint 0	26	20	7	5	**58**	**2.110**
Sprint 1	28	10	4	8	**50**	**1.990**
Sprint 2	19	100	4	4	**127**	**3.645**
Sprint 3	15	135	4	10	**164**	**4.800**
Sprint 4	4	44	6	6	**60**	**1.960**
Sprint 5	4	96	4	12	**116**	**3.580**
Total (h)	**96**	**405**	**29**	**45**	**575**	
Coste (€)	**3.360**	**10.125**	**1.450**	**3.150**		**18.085**

Tabla 4.3: Coste total del proyecto

Herramientas	**Total (€)**
Materiales	596
Humanos	21.175
Total	**21.771**

En la tabla 4.1 puede verse una lista de los materiales empleados junto a su precio. El mayor desembolso económico recae sobre la compra de los robots Makeblock Ranger. Fue necesaria la utilización de dos robots para poder comprobar la robustez del sistema desarrollado al aplicarlo sobre un robot con una posición de la cámara diferente. Debido a esta decisión fue necesario duplicar el número de placas Raspberry así como la adquisición de 3 cámaras para poder intercambiar entre las placas raspy. La alimentación del Raspberry se realiza con baterías externas mientras que para el Ranger utilizamos 6 pilas HR6 o una batería de litio.

4.4.2 Recursos humanos

El coste humano del proyecto engloba los roles de las personas que participaron el proyecto donde nos encontramos con el analista, encargado de estudiar el problema y diseñas la solución, con un coste de 35€/h. El programador, encargado de elaborar la solución con un coste 25€/h. El jefe de proyecto que lleva un control del proyecto y asiste a las reuniones de planificación y revisión de cada sprint, tiene un coste por hora de 70€/h. Por último, el asesor externo, experto en la materia, tiene un coste por hora de 50€/h. El total del desembolso humano desglosado puede consultarse en la tabla 4.2.

4.4.3 Coste total del proyecto

La suma de coste material junto con el humano nos da un coste total del proyecto de 9.281 € repartidos como se muestra en la tabla 4.3.

Capítulo 5

Desarrollo e Implementación

En este capítulo se abordará como se ha realizado el desarrollo del sistema para realizar las tareas, con un primer esquema que explica como usar el sistema ante una nueva tarea y la implementación de las distintas propuestas de las redes convolucionales junto con sus arquitecturas. La implementación de estas arquitecturas propuestas se puede consultar a través del repositorio del proyecto (Ver apéndice A).

5.1 Esquema general del sistema propuesto

Como esquema general del sistema, tenemos que realizar una serie de pasos básicos, independientemente del tipo de red y de la tarea que vamos a desarrollar:

- Primero debemos elaborar u obtener un *dataset* con las etiquetas adecuadas según lo requiera la tarea. En nuestro caso un comportamiento del robot.
- Tras esto debemos de elaborar nuestra red, a la que se le pasará el *dataset* dividido en los conjuntos de entrenamiento y validación sobre los que se realizará el proceso de entrenamiento para obtener un modelo de CNN con sus pesos que pueda realizar inferencias sobre nuevos datos.
- Una vez obtenemos el modelo generado con la librería de Tensorflow, debemos de convertirlo a un modelo optimizado *.tflite* para poder ejecutarlo en la Raspberry, y debemos de seleccionar el *script* de clasificación o regresión dependiendo de la red.
- Comienza la ejecución y el movimiento del robot, almacenando todas las imágenes y los tiempos junto con la predicción de la red.

El esquema completo se puede ver en la figura 5.1. Esta estructura nos permite reentrenar la red o crear una nueva para una nueva tarea de manera sencilla y con la premisa de que obtendremos tiempos mínimos que permitan la ejecución en tiempo real, debido a la optimización que se llevó a cabo en los *scripts* que se ejecutan en la Raspberry.

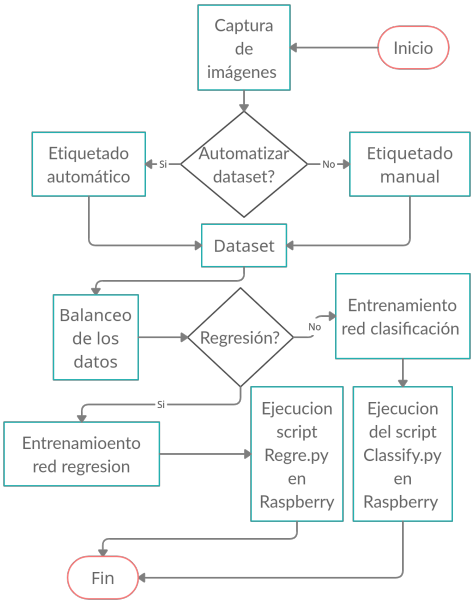

Figura 5.1: Esquema general del sistema propuesto.

5.2 Creación del *dataset*

El *dataset* contiene un conjunto de imágenes con las etiquetas que va a recibir nuestra red. Idealmente, la captura en directo de las imágenes junto con el comando que se ha realizado sería la situación perfecta para obtener un buen *dataset*. En este caso, no ha sido posible la captura de imágenes y sus comandos de desplazamiento por lo que fue necesario elaborar un sistema para poder crear el conjunto de datos.

5.2.1 Captura de imágenes

Para empezar con la elaboración del *dataset* fue necesaria la captura de un conjunto de imágenes sobre el robot a través de la cámara de la Raspberry. En total se obtuvieron más de 2.500 fotos a color con dos tipos de resoluciones, a 640x480 píxeles y 640x360 píxeles para cada una de las tareas a realizar. Cada imagen fue almacenada con un nombre único en una carpeta por cada uno de nuestros circuitos.

Para capturar estas imágenes la cámara del robot debe de posicionarse en la parte superior con una inclinación hacia el suelo con el que formará un ángulo aproximado de 110 grados. Se ha variado la posición entre circuitos ligeramente pero siempre manteniendo como mínimo un ángulo superior al recto, ya que en ese caso no se verían las líneas con suficiente claridad a través de la cámara.

La captura de imágenes se realizó por varios circuitos que se caracterizan por tener super-

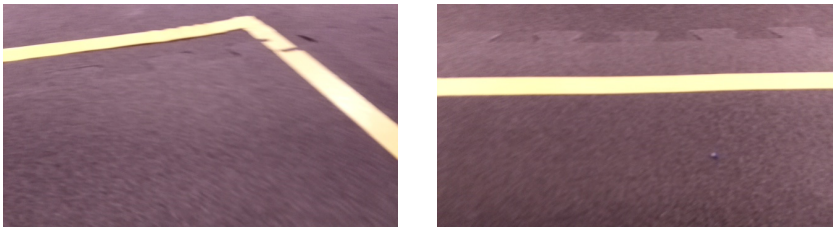

Figura 5.2: "Perceptual aliasing" en el *dataset* de Seguir Carril.

ficies de distinto color y en algunos de ellos se realiza una diferenciación de color en las líneas que limitan los carriles. Estos recorridos cuentan con distribuciones de uno a dos carriles. Los circuitos tienen curvas hacia la derecha e izquierda. En ambos casos se realiza una curva de 90º, a excepción de dos de ellos que cuentan con curvas circulares. Estas curvas pueden estar unidas formando un ángulo de 180º.

Para la captura de imágenes se ejecuta un *script* de Python sobre la Raspberry. Este código ordena salvar el contenido que lee la cámara a la vez que, mediante el control humano, se desplazó el robot como si estuviese recorriendo el circuito girando y acelerando manualmente, empujando el robot.

Con todas las imágenes recopiladas, se realizó un proceso de eliminación de imágenes duplicadas debido a momentos en los que el control humano no desplazó al robot y también se descartaron las imágenes que producían "Perceptual aliasing". Es decir, la eliminación de imágenes que no tienen una etiqueta clara y que perciben que para dos lugares diferentes se obtiene la misma etiqueta. Un ejemplo de este efecto en nuestros circuitos, se muestra en la figura 5.2, donde en la primera imagen del circuito, se realiza una curva a la izquierda y en la segunda se realiza un giro a la derecha.

5.2.2 Etiquetado manual

Una vez recopiladas las imágenes fue necesario clasificarlas siguiendo algún tipo de criterio. En este etiquetado, fue necesario ejercer como un experto, decidiendo como etiquetar las imágenes manualmente. Se definieron cuatro clases según el tipo de movimiento que debe ejecutarse en el robot, siguiendo los siguientes criterios:

- Recto rápido: En esta clase se recogen todas las imágenes en las que no se observa la línea horizontal que indica un cambio de dirección.
- Recto lento: En el momento que se detecta una línea horizontal debemos de disminuir la velocidad para poder manejar con mayor control el robot, ya que precede a un giro.
- Izquierda: Todas las imágenes en las que el robot comienza a girar hacia la izquierda hasta que acaba de realizar el movimiento de giro, fueron clasificadas en esta clase.

- Derecha: Por último, en esta clase se encuentran etiquetadas todas las imágenes en las que el robot gira hacia la derecha para realizar un giro debido a una curva.

El motivo de esta distribución se debe a la búsqueda de las clases básicas que permitiesen el desplazamiento por el circuito. Inicialmente se plantearon únicamente tres movimientos simples eliminando las clases recto rápido y lento siendo fusionadas en una clase avanzar recto. El problema radica en que al acercarse a una curva el descenso de velocidad se producía de manera brusca incurriendo en problemas futuros para el trazado de curvas, decidiendo la diferenciación en estas cuatro clases. Siguiendo estos patrones se creó la carpeta de clasificación manual con todas estas imágenes, guardando cada imagen bajo una carpeta a modo de etiqueta con el mismo nombre, para facilitar su lectura a la hora de realizar el entrenamiento de la red.

5.2.3 Etiquetado automático por flujo óptico

El proceso de etiquetado manual es un proceso tedioso y que depende de la percepción y la experiencia del experto clasificador. Por ello, se decidió implementar en el lenguaje C++ un clasificador automático para detectar las características de la imagen y poder clasificarlas de manera no manual, agilizando el tiempo y trabajo necesario si se desea elaborar un nuevo *dataset*, automatizando el proceso. [27]

Este clasificador automático funciona transformando un conjunto de imágenes en un vídeo. De este vídeo se extrae el flujo óptico de cada fotograma y se calcula las posiciones relativas de los puntos relevantes en la secuencia de imágenes. Por lo tanto, esto nos sirve para conocer el desplazamiento que se produjo entre una imagen y la siguiente, ya que vamos a conocer la posición del punto en la imagen previa a la actual.

Como podemos ver en la figura 5.3 se traza el vector que indica el desplazamiento que se está realizando entre las dos imágenes. Sobre este vector operaremos para obtener el ángulo. En la figura 5.4 vemos el flujo de otras imágenes para un *dataset* diferente donde se mantiene el mismo criterio. Localizar los vectores que representan el flujo óptico y operar con ellos.

Sobre cada vector se calculó el arco tangente para conocer el ángulo que forman con la horizontal. Para acabar, se realiza una media de los ángulos que se obtuvieron para conocer el movimiento que se produjo y el comando que se debe de ejecutar, obteniendo una distribución de los ángulos entre 0º y 180º.

Según este ángulo obtenido se etiquetaron las imágenes de la siguiente forma:

- Etiquetado de imágenes para la clasificación: Como trabajamos con datos discretos fue necesario la agrupación de estos ángulos en distintas clases definidas por la pertenencia a un rango:

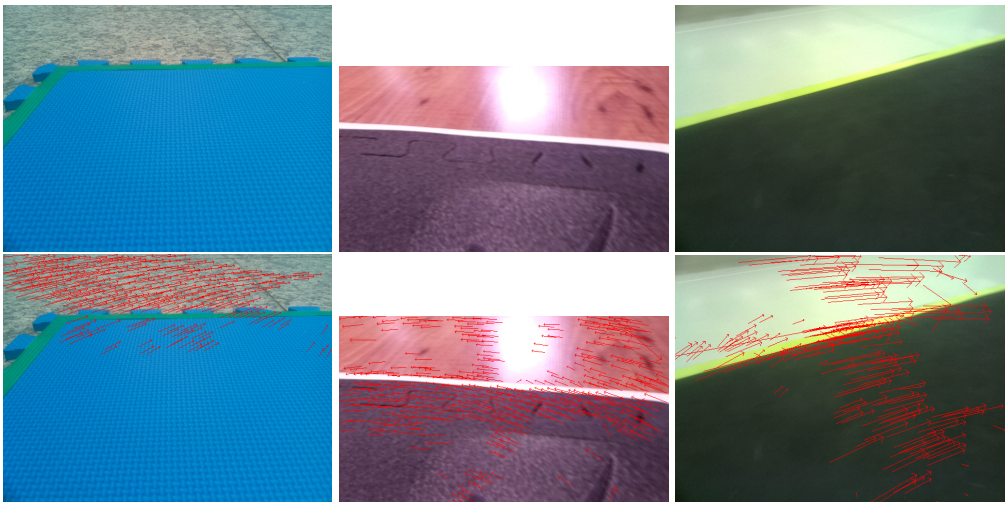

Figura 5.3: Flujo óptico de imágenes capturadas para el *dataset* de Seguir Carril.

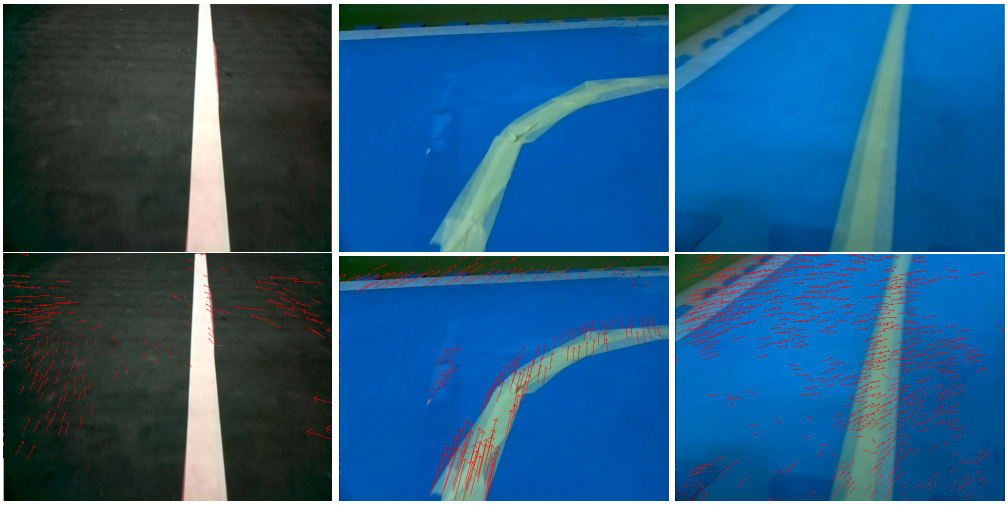

Figura 5.4: Flujo óptico de imágenes capturadas para el *dataset* de Seguir Línea.

Tabla 5.1: Distintas versiones de los *datasets* elaborados.

	Arquitectura	Método	Imágenes recopiladas	Imágenes tras incremento
Seguir Carril	Clasificación	Manual	2561	5122
		Automático	1245	2490
	Regresión	Automático	1915	3830
Seguir Línea	Regresión	Automático	2116	4232

  - Recto rápido: Se clasificaron con esta etiqueta todas las imágenes que se obtuvo un ángulo medio comprendido entre 80º y 100º.
  - Recto lento: Esta etiqueta agrupó las imágenes comprendidas entre ángulos de 60º y 80º y también las imágenes con ángulos entre 100º y 120º.
  - Izquierda: Todas las imágenes con ángulos menores a 60º fueron agrupadas como izquierda.
  - Derecha: Engloba a las imágenes con ángulos entre 120 º y 180º.

  Este *dataset* contiene un error en la clasificación de las etiquetas de recto porque, tratando de igualar los criterios seguidos para el etiquetado manual, no se tuvo en cuenta las líneas horizontales que delimitan el cambio de movimiento rápido a lento.

- Etiquetado de imágenes para la regresión: En esta opción trabajamos con datos continuos por lo que se guardó directamente para cada imagen la etiqueta obtenida en el proceso sobre un fichero .csv junto al nombre de la misma.

En el apéndice B se muestra el código en C++ de la versión del etiquetado automático para la regresión.

5.3 Arquitecturas de CNN para una tarea

5.3.1 Red de clasificación para la tarea Seguir Carril

5.3.1.1 Arquitectura

La red neuronal convolucional de clasificación para la tarea de Seguir Carril fue la primera red desarrollada. En este tipo de red esperamos recibir una imagen como entrada y obtener como salida la clase a la que pertenece. La estructura final de la CNN propuesta se puede ver en la figura 5.5, donde observamos que esta red convolucional únicamente tiene una capa de este tipo junto con las operaciones necesarias para ser conectada a una red neuronal clásica.

Esta estructura de la red nos genera una cantidad de parámetros elevada debido a que únicamente se aplica una reducción del tamaño de las imágenes. Esto puede ralentizar nuestra

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 80, 32)	896
leaky_re_lu (LeakyReLU)	(None, 64, 80, 32)	0
max_pooling2d (MaxPooling2D)	(None, 32, 40, 32)	0
dropout (Dropout)	(None, 32, 40, 32)	0
flatten (Flatten)	(None, 40960)	0
dense (Dense)	(None, 50)	2048050
leaky_re_lu_1 (LeakyReLU)	(None, 50)	0
dropout_1 (Dropout)	(None, 50)	0
dense_1 (Dense)	(None, 4)	204

Total params: 2,049,150
Trainable params: 2,049,150
Non-trainable params: 0

Figura 5.5: Estructura y parámetros de la red de clasificación propuesta.

clasificación de las imágenes, pero se consiguieron unos buenos resultados en el tiempo de ejecución como podemos comprobar en la siguiente sección 5.3.1.2.

En la figura 5.5 podemos observar los parámetros que se obtienen en la primera capa con un total de 896 parámetros obtenidos mediante el cálculo de:

$$Parametros = (Anchura_{Filtro} * Altura_{Filtro} * Numero_{FiltrosPrevios} + 1) * Numero_{Filtros}$$

La siguiente capa que nos genera parámetros se corresponde con la capa totalmente conectada que genera un total de 2.048.050 parámetros que se obtiene mediante la operación:

$$Parametros = Neuronas_{Previas} * Neuronas_{Capa} + 1 * Neuronas_{Capa}$$

Como última capa que genera parámetros nos encontramos la capa de salida con 4 neuronas correspondientes con cada una de nuestras salidas que genera un total de 204 parámetros siguiendo la formula superior.

La suma de estos parámetros nos da como resultado un total de 2.049.150 parámetros que debemos de entrenar usando imágenes de 80x64.

5.3.1.2 Entrenamiento

Con la arquitectura definida, se usó el *dataset* generado siguiendo las características la sección 5.2.2 y se cargaron las imágenes en el *script* de Python junto a sus etiquetas. En total se usaron 2561 imágenes 5.1. La primera tarea fue reducir el tamaño de la imagen a 80x64 mediante la operación de redimensión de la librería OpenCv para disminuir el tiempo de entrenamiento así como agilizar la captura de datos. Antes de dividir y comenzar a entrenar

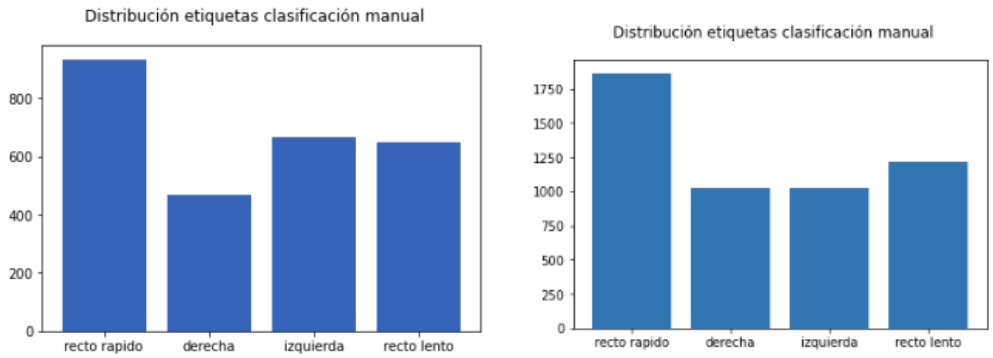

Figura 5.6: Distribución de las etiquetas antes y después de duplicar el *dataset* manual.

nuestra red, fue necesario transformar los datos categóricos a numéricos a través de la codificación One-Hot, basada en la creación de una referencia binaria por cada etiqueta de la que disponemos. En la distribución de las etiquetas que podemos observar en la parte izquierda de la figura 5.6, vemos como la clase derecha tiene menos cantidad de imágenes que el giro a la izquierda.

Debido a que se obtiene un conjunto de imágenes desbalanceadas, se decidió aumentar el conjunto de datos con la operación de volteo horizontal, duplicando el tamaño del *dataset* y consiguiendo una distribución uniforme en los giros como podemos ver en la gráfica derecha de la figura 5.6. Lo que nos dio un total de 5.122 imágenes para esta tarea 5.1.

Con los datos transformados y aplicando una normalización entre 0 y 1 a la imagen, se realizó la división de los datos asignando un 80% para entrenamiento y un 20% para test, seleccionados aleatoriamente de nuestro *dataset*, para evitar así el sesgo.

Con los conjuntos divididos, se realizó un proceso de entrenamiento en 15 etapas donde, para juzgar el rendimiento del modelo, se ha usado la precisión. Como función de perdida, que busca minimizar el cálculo de la perdida del modelo, se ha usado la clase *CategoricalCrossentropy*, que calcula la perdida de entropía cruzada entre las etiquetas de clasificación y las predicciones. Esta clase es óptima para su uso cuando tenemos más de dos etiquetas con codificación *One-Hot*. Por último, se ha usado el optimizador *RMSprop*, que busca la corrección de los efectos negativos debido a la acumulación en las cachés, manteniendo un promedio móvil del cuadrado del gradiente.

A la hora de ejecutar el entrenamiento, no necesariamente tenemos que obtener el mejor modelo en la última etapa. Para guardar el mejor modelo a lo largo del entrenamiento se introdujo un *callback* que guarda el modelo en el que, según la métrica de precisión, se obtuvo mayor éxito.

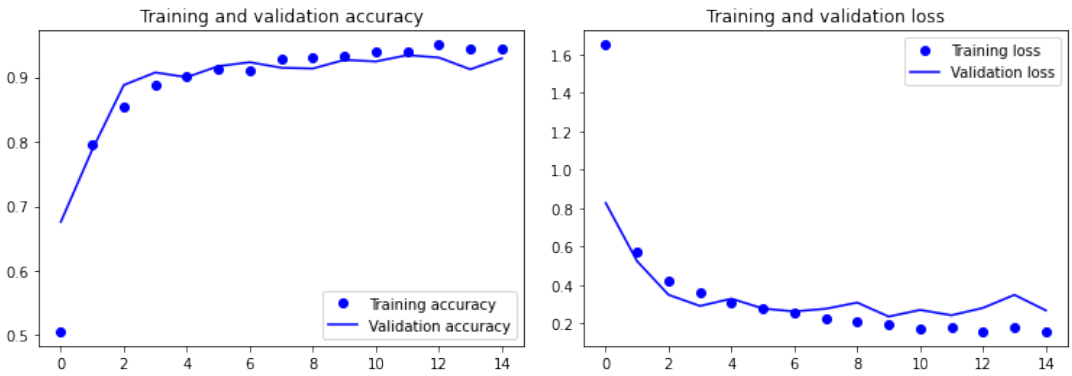

Figura 5.7: Gráficas de la red de clasificación: (izqda) precisión de validación y entrenamiento y (drcha) errores de validación y de entrenamiento.

5.3.1.3 Evaluación del modelo

Al trabajar con imágenes pequeñas y no tener un dataset excesivamente grande, el tiempo de entrenamiento es bajo, realizando la ejecución completa en menos de 4 minutos y logrando los mejores modelos entre la décima y duodécima etapa de entrenamiento.

En las figuras 5.7 se representan las dos gráficas encargadas de medir la precisión de entrenamiento y validación por un lado, y el error de entrenamiento y validación por el otro. Estas gráficas nos aportan conocimiento del estado del entrenamiento así como del ajuste del modelo.

Vemos como en el modelo que estamos entrenando se produce una variación de la evolución de las gráficas a partir de la duodécima etapa, cambiando el sentido. Esto nos indica que podemos estar sobreajustando, es decir, que el modelo aprende demasiado bien el conjunto de datos de entrenamiento, incluyendo el ruido estadístico y las fluctuaciones aleatorias. El problema con el sobreajuste implica que cuanto más especializado se vuelve el modelo para los datos de entrenamiento, menor es la capacidad de generalización de nuevos datos, lo que provoca un aumento en el error de generalización. En nuestro caso, puede provocar que no sea capaz de realizar el recorrido por un carril, ya que no corresponden con los datos con los que fue entrenado el modelo.

La gráfica 5.7 izquierda, indica la evolución progresiva del aprendizaje del modelo, viendo que a partir de la octava etapa de entrenamiento superamos el 90% de acierto en el conjunto de validación y test, lo que nos indica que el modelo es capaz de generalizar ante un nuevo conjunto de datos. La gráfica de la figura 5.7 derecha, muestra como la evolución del error de la validación desciende progresivamente hasta llegar a la etapa undécima que deja de decrecer para comenzar a crecer con un pico en la siguiente etapa. Esta es la característica de que nos encontramos ante un problema de sobreajuste por lo que, en este entrenamiento, los mejores modelos se encontrarán antes de la duodécima etapa de entrenamiento.

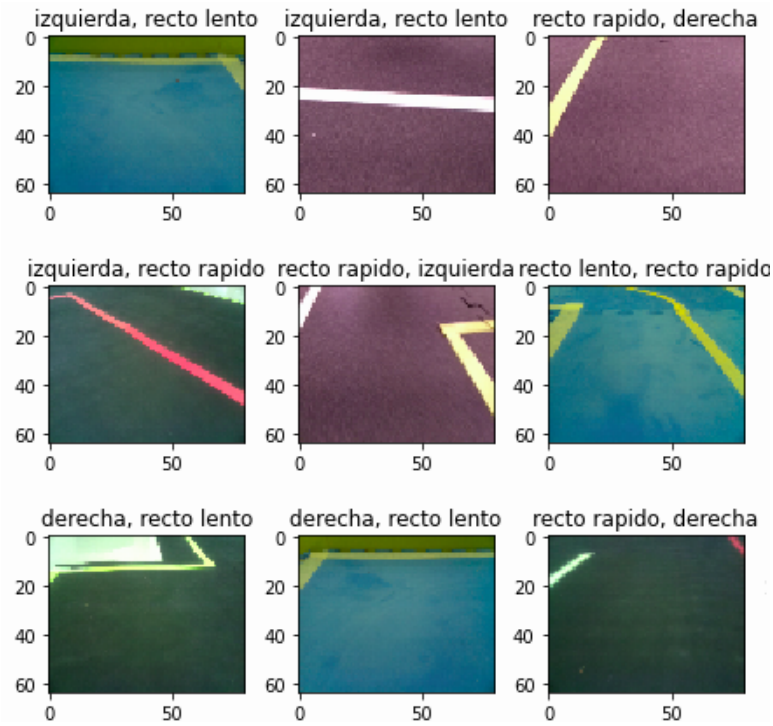

Figura 5.8: Ejemplos de las inferencias de la clasificación para la tarea Seguir Carril.

Tras comprobar la evolución del aprendizaje se realiza una impresión con la librería de Matplotlib de todas las imágenes que son incorrectamente clasificadas para realizar una nueva evaluación, localizando que la mayor parte de ellas se encontraban en el límite entre dos clases o que directamente el clasificador humano había fallado. En la figura 5.8 se puede ver dos imágenes donde la primera etiqueta corresponde a la red CNN mientras que la segunda corresponde al clasificador manual. Viendo que claramente se encuentran mal etiquetadas, por ejemplo, la imagen del circuito azul de la fila superior, se observa que se etiquetó esta imagen como recto lento, pero encajaría mejor ser clasificada como izquierda. Muchas de estas imágenes se encuentran el límite entre dos clases y debido al alto porcentaje de precisión que se obtuvo, no se modificó el conjunto de datos.

El mejor modelo entrenado se guarda como un fichero .h5 donde se almacena la red con la estructura y pesos creados que nos servirá para poder ejecutar la clasificación en distintos dispositivos.

Se han probado otras configuraciones para esta red sin obtener un mayor porcentaje de precisión que esta arquitectura, por lo que ha sido la seleccionada para ejecutar las pruebas.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 38, 24)	1824
conv2d_1 (Conv2D)	(None, 13, 17, 36)	21636
conv2d_2 (Conv2D)	(None, 5, 7, 48)	43248
conv2d_3 (Conv2D)	(None, 3, 5, 64)	27712
conv2d_4 (Conv2D)	(None, 1, 3, 64)	36928
dropout (Dropout)	(None, 1, 3, 64)	0
flatten (Flatten)	(None, 192)	0
dense (Dense)	(None, 100)	19300
dropout_1 (Dropout)	(None, 100)	0
dense_1 (Dense)	(None, 10)	1010
dense_2 (Dense)	(None, 1)	11

Total params: 151,669
Trainable params: 151,669
Non-trainable params: 0

Figura 5.9: Estructura y parámetros de la red de regresión propuesta para la tarea Seguir Carril.

5.3.2 Red de Regresión para la tarea Seguir Carril

5.3.2.1 Arquitectura

Para realizar esta tarea se intentó adaptar la red previamente diseñada, sin embargo, no fue posible obtener un buen modelo por lo que se diseñó una nueva estructura. Esta surge motivada por la investigación desarrollada por la compañía NVIDIA y que es explicada en su trabajo oficial "End to End Learning for Self-Driving Cars" para el manejo de un coche de manera autónoma, a través de varias cámaras [28]. En este estudio, trabajan sobre coches reales y con un mayor número de imágenes por estado, aprendiendo a detectar las carreteras y carriles. La arquitectura de la red de regresión definida se puede ver en la figura 5.9.

Mediante esta estructura de la red se generan un total de 151,669 parámetros, una disminución considerable de parámetros en comparación con la anterior red de clasificación para Seguir Carril, lograda gracias a las capas de convolución, que reducen sucesivamente el tamaño de las salidas.

Con las fórmulas que vimos en la sección anterior 5.3.1.1 calculamos los parámetros de cada capa obteniendo en la primera convolución, un total de 1.824 parámetros. En la segunda, se generan 21.636 variables nuevas. En la tercera convolución, aumenta la cifra hasta 43.248 nuevos datos. La cuarta capa genera un total de 27.712 parámetros y nuestra última operación, genera 36.928 nuevas variables. En total nuestras capas de convolución generan un total de 131.348 parámetros que debemos de entrenar.

A esta cifra debemos de sumarle los datos de las tres capas totalmente conectadas que ge-

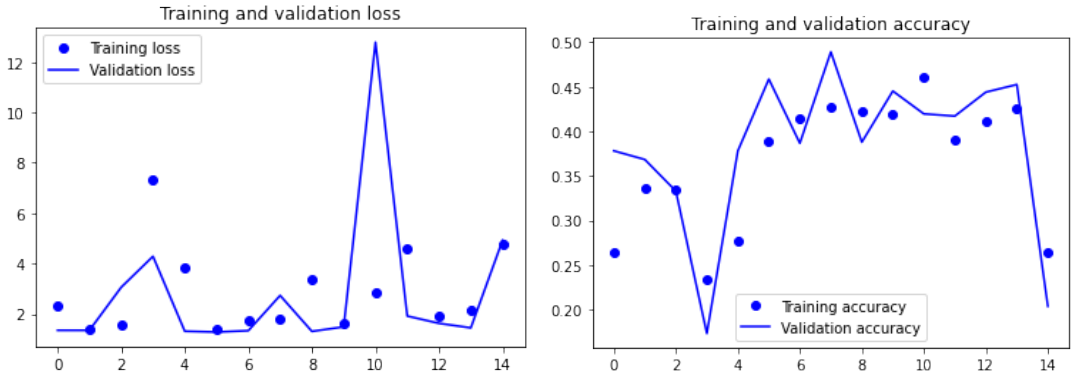

Figura 5.10: Error (izquierda) y precisión (derecha) de entrenamiento de la tarea de clasificación para Seguir Carril empleando la arquitectura de la regresión de la sección 5.3.2.1

neran 19,300;1010;11 variables nuevas con lo que obtenemos como resultado final los 151.669 parámetros entrenables.

Con esta red definida y viendo la diferencia de tamaños que nos aporta, se adaptó también esta arquitectura para la red de clasificación, comprobando si se podían mejorar los porcentajes de precisión para la tarea . En la figura 5.10 se muestran los resultados obtenidos, donde no ha sido posible conseguir adaptar esta estructura para realizar un aprendizaje correcto sobre el *dataset*, obteniendo unos resultados de precisión inferiores al 50%.

5.3.2.2 Entrenamiento

Para realizar el entrenamiento de esta red se usó el *dataset* generando según la sección 5.2.3 con valores continuos. Este contiene un total de 1915 imágenes con su respectivo nombre y el ángulo asociado. Por tanto lo primero que se hizo fue cargar en listas las imágenes, aplicándole una redimensión al tamaño fijo de 80x64, y su ángulo. El histograma de la figura 5.11 de la parte izquierda muestra la distribución de las imágenes de nuestro *dataset*. Se observa que tenemos una mayor distribución de ángulos en las zonas de los giros (zonas laterales) a partir del ángulo de 100º, que se corresponde con la zona de los giros a la izquierda.

Aplicando el aumento de imágenes sobre nuestro *dataset* conseguimos reducir la diferencia de ambas zonas laterales y también aumentar el número de los datos situados en los extremos e igualarlos para ambos giros. Se volvió a utilizar la operación de volteo horizontal de la librería OpenCV y se obtuvo la distribución de la figura 5.11, obteniendo un total de 3830 imágenes.

Debe de realizarse un proceso de normalización de nuestras listas, normalizando los valores de la imagen y del ángulo entre 0 y 1. Esta normalización es un paso esencial para lograr un buen entrenamiento, ya que de esta manera nos cercioramos de que tanto las imágenes como los ángulos tienen un mismo rango de valores controlado. De esta manera, al calcular

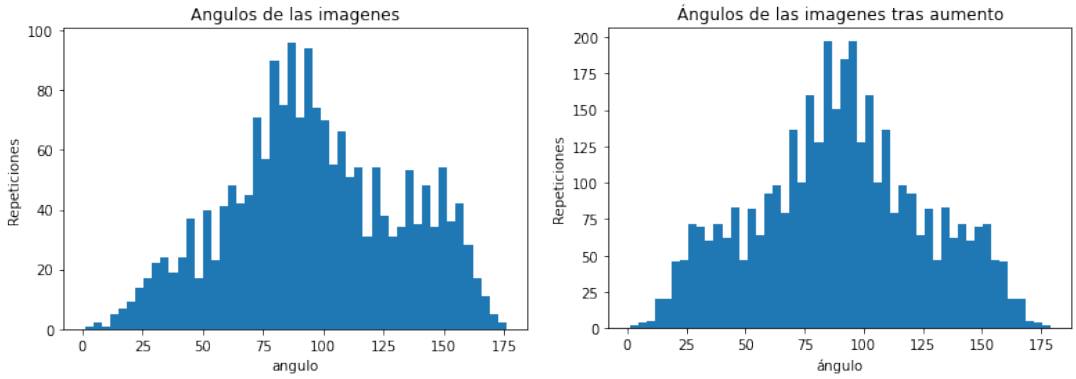

Figura 5.11: Distribución de las imágenes de la tarea Seguir Carril antes y después de aumentar el *dataset* original.

la tasa de aprendizaje, si no normalizásemos nuestros vectores de entrenamiento de entrada, los rangos de nuestras distribuciones de valores de características probablemente serían muy diferentes para cada característica produciendo continuas correcciones en la tasa de aprendizaje. Ello provocaría un estado oscilante en el que no somos capaces de fijar el máximo peso en el espacio de la red haciendo el entrenamiento excesivamente largo. Para continuar, se dividió nuestras listas de imágenes y ángulos en un 80% para entrenamiento, mientras que el 20% restante lo usaremos para los test, nuevamente de manera aleatoria para evitar el sesgo.

El siguiente paso es ejecutar las 15 etapas de entrenamiento utilizando como métrica para evaluación del modelo el Error Cuadrático Medio, que nos indica el error que se esté produciendo en el modelo en comparación con nuestra lista de ángulos. Este error está elevado al cuadrado para no tener la posibilidad de obtener valores negativos y que el error perfecto para el modelo sea 0. Como optimizador se hizo uso del algoritmo de Adam, que es una extensión del algoritmo de optimización de descenso de gradiente estocástico, donde se actualizan los pesos de la red de forma iterativa en función de los datos de entrenamiento.

5.3.2.3 Evaluación del modelo

Para esta red, el tamaño de las imágenes continúa fijado en 80x64 px y se redujeron los parámetros a, aproximadamente, 150.000, por lo que el tiempo de entrenamiento disminuyó, consiguiendo realizarse en menos de 2 minutos y logrando los mejores modelos en las etapas finales del entrenamiento.

Para evaluar un modelo de regresión no debemos basarnos en la precisión, sino en el error. La métrica más usual en problemas de regresión es el error cuadrático medio, métrica usada también en esta red. Con el entrenamiento del modelo se elabora la gráfica con el error de entrenamiento y validación correspondiente a la figura 5.12 donde puede verse como desciende el error del modelo a lo largo de cada etapa, viendo como llega a converger la gráfica

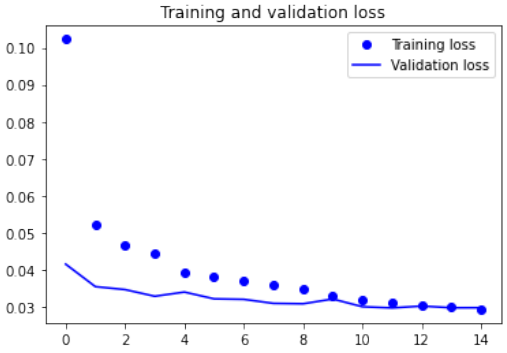

Figura 5.12: Evolución del error de validación y entrenamiento durante el entrenamiento de la regresión en Seguir Carril.

en las últimas iteraciones. Esto nos indica que el modelo aprendió correctamente y no es necesario seguir realizando etapas de entrenamiento porque, si se ejecutasen más iteraciones, podríamos incurrir en el problema del sobreajuste.

Debido a que estamos trabajando con salidas continuas no existen clases que nos indique el número de imágenes en las que se realizó una predicción errónea, sino que el modelo debe de predecir unos valores cercanos al etiquetado. Para comprobarlo, se realiza una impresión con la librería de Matplotlib del las imágenes de conjunto de test y su predicción que se muestran en la figura 5.13, donde podemos ver un par de valores continuos correspondientes a la predicción del modelo y la etiquetada automáticamente. Este par de valores discrepa en algunas imágenes, como las centrales, donde analizándolas en profundidad, la predicción encaja más en la salida esperada, pero realizando unas inferencias acordes al desplazamiento que se debería de predecir en las imágenes.

5.3.3 Red de regresión para Seguir Línea

En las anteriores secciones 5.3.2.1 y 5.3.2, se definieron dos modelos de redes para la creación de una red de clasificación y regresión para la tarea de Seguir carril. Debido al buen funcionamiento de la red de regresión en las pruebas experimentales que se explican en la sección 6.2 y que no fue posible adaptar la red de clasificación para seguir carril (5.3.2.1) para la versión de la tarea por regresión, se decidió comprobar la facilidad de la creación de un nuevo modelo basándonos en la arquitectura 5.3.2.

5.3.4 Arquitectura

Esta red fue diseña siguiendo la misma arquitectura que la explicada en la sección 5.3.2.1.

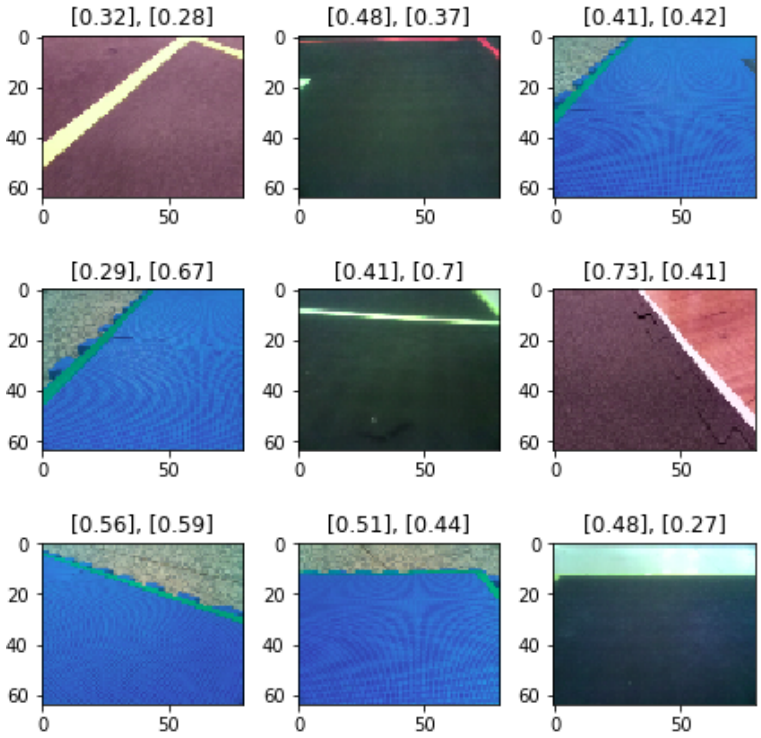

Figura 5.13: Ejemplos de las inferencias de la regresión para la tarea Seguir Carril.

5.3.5 Entrenamiento

Para realizar el entrenamiento de esta red, fue necesario la captura de nueva imágenes siguiendo las pautas definidas en la sección 5.2.1 y se creó el *dataset* haciendo uso del etiquetado automático de la sección 5.2.3 junto a la combinación que envía una salida continua. Se obtuvieron un total de 2116 imágenes. Los siguientes pasos son los mismos realizados en la sección 5.3.2.2, ya que se comprobó que en la distribución de las imágenes y sus etiquetas nos encontrábamos con una pequeña distribución mayor hacia los ángulos mayores de 125º. En la figura 5.14 podemos observar la distribución de los ángulos en el rango 0º hasta 180º en la figura izquierda, por lo que se hizo uso de la función de aumento del *dataset* con la operación de volteo de la librería de OpenCV, alcanzando la distribución de la figura 5.14 de la parte derecha con un total de de 4232 imágenes 5.1.

Con el conjunto de datos listo, se realizó el entrenamiento según se ha explicado en la sección 5.3.2.2,

5.3.6 Evaluación

Esta red tiene el mismo tamaño que las anteriores, 80x64 px, con un total de, aproximadamente, 150.000 parámetros, continuando con unos tiempos de entrenamiento inferiores a 2

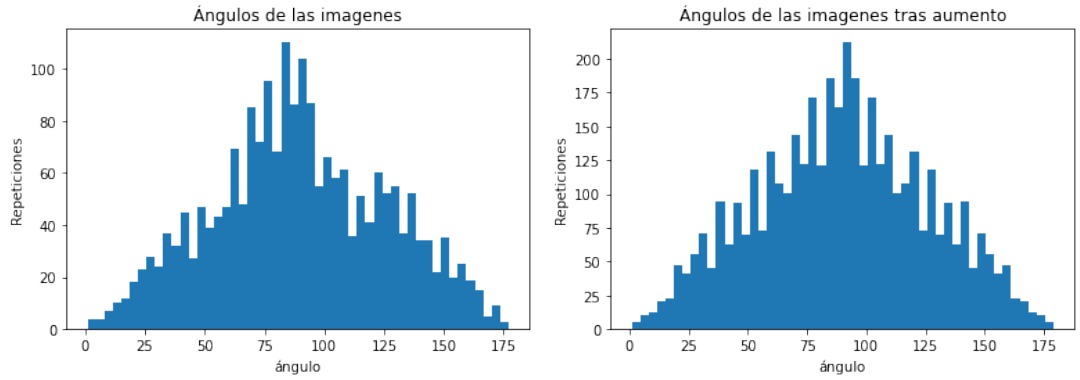

Figura 5.14: Distribución de las imágenes de la tarea Seguir Línea antes y después de aumentar el *dataset*.

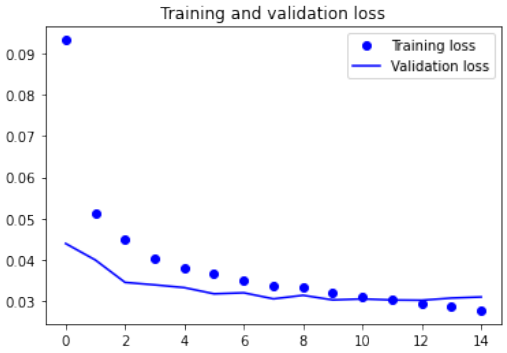

Figura 5.15: Evolución del error de validación y entrenamiento durante el entrenamiento de la regresión para Seguir Línea.

minutos.

Se realizó el entrenamiento en 15 etapas logrando los mejores modelos hasta la decimotercera etapa de entrenamiento. Basándonos en la gráfica de la figura 5.15, se produce un sobreentrenamiento en ese punto porque vemos que, una vez alcanzada la convergencia el modelo, aumenta la diferencia entre el error de entrenamiento y el de validación.

Para comprobar las inferencias que realiza el modelo, se realiza una impresión con la librería Matplotlib de las imágenes del conjunto de test y su predicción que se muestran en la figura 5.16, donde podemos ver un par de valores continuos correspondientes a la predicción del modelo y la etiqueta.

5.4 Ejecución de modelos CNN en la Raspberry

La Raspberry Pi cuenta con unos recursos limitados no siendo capaz de ejecutar las funcionalidades de la librería TensorFlow en tiempo real o incluso teniendo problemas con la

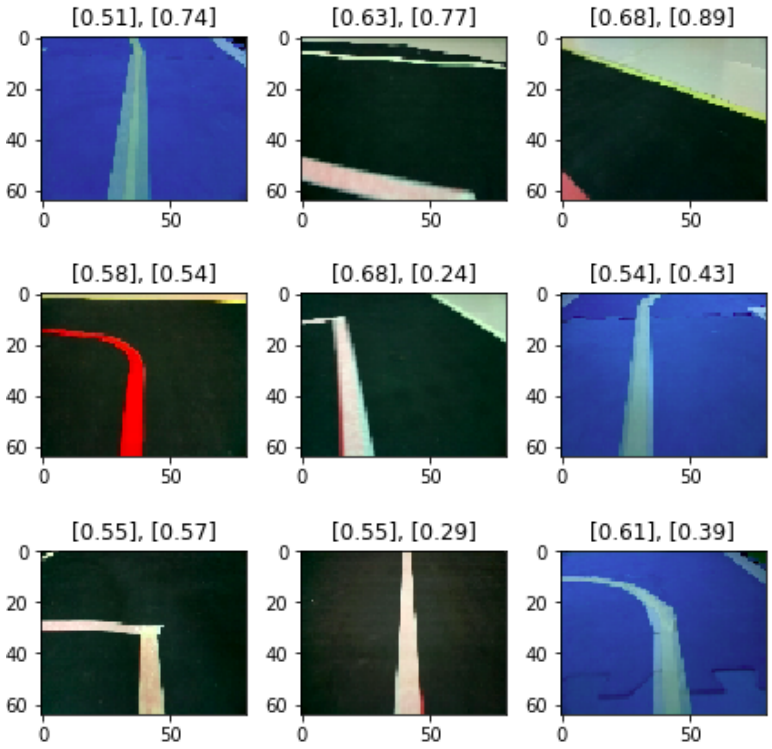

Figura 5.16: Ejemplos de inferencias de la regresión para la tarea Seguir Línea.

memoria RAM que necesita esta librería para trabajar. Por este motivo se hace necesario el entrenamiento en el equipo de desarrollo. Sin embargo, existe una versión reducida de Tensorflow, ya comentada en el capítulo 3, que nos permite ejecutar estos modelos en tiempos razonables: TensorflowLite. En el apéndice C hay una guía sobre los pasos necesarios para su instalación.

5.4.1 Adaptación del modelo de la CNN

Los modelos entrenados en sección 5.3 son almacenados en formato .h5. Este formato es el común a la hora de almacenar y trabajar con la librería de TensorFlow junto con Keras. Sin embargo los dispositivos IoT no soportan este formato. Por tanto, la librería Tensorflow tiene una funcionalidad para la conversión entre modelos .h5 y modelos .tflite soportado por estos dispositivos.

Para llevar a cabo esta transformación es suficiente tener instalada la librería de Tensorflow y realizar la conversión mediante alguno de estos pasos:

- El Api de Python: La librería de TensorFlow cuenta en su versión del API de Python con varias funciones que nos permiten realizar la conversión de los modelos como parte del desarrollo. Esta versión tiene la limitación de la necesidad de cargar los modelos .h5 si

```
tflite_convert --help

`--output_file`. Type: string. Full path
                        of the output file.
`--saved_model_dir`. Type: string. Full
            path to the SavedModel directory.
`--keras_model_file`. Type: string. Full
            path to the Keras H5 model file.
`--enable_v1_converter`. Type: bool.
        (default False) Enables the converter and
        flags used in TF 1.x instead of TF 2.x.
```

Código 5.1: Opciones del comando *tflite_convert*.

no se encuentra en la ejecución actual.

- Línea de comandos: El segundo método para la conversión de los modelos es a través de la línea de comandos, a través del comando *tflite_convert* (código 5.1).

Se ha decidido usar el método a través de la línea de comandos por aportar flexibilidad y facilidad para la conversión de diferentes modelos que se van almacenando en cada ejecución.

5.4.2 Medición del tiempo y salvado de datos

Inicialmente, las ejecuciones entre bucles se realizaban con tasas de varios segundos entre cada imagen. Esto imposibilitaba la ejecución de los modelos, ya que en ese espacio de tiempo el robot perdía completamente la ubicación en el circuito. Para realizar un análisis de los errores se decidió realizar la medición de los distintos tiempos de ejecución de los *scripts* de las Raspberry almacenando en un fichero .csv esta información:

- Tiempo de preprocesado: Donde se realizó una medición del tiempo necesario para procesar las imágenes
- Tiempo de inferencia del modelo: Midió la duración entre la entrada de la imagen y la obtención de la predicción del modelo.
- Tiempo de ejecución del Robot: Corresponde con el periodo necesario para realizar un movimiento en el robot.
- Tiempo total: Tiempo entre las iteraciones del bucle.

A mayores se almacenó el nombre de la imagen y la salida para evaluar offline las predicciones que se llevaron a cabo. Este proceso de recopilación de información acarrea consigo un tiempo necesario para la escritura de unos pocos milisegundos que no se tendrán en cuenta.

Tabla 5.2: Tiempos de ejecución de la clasificación: sin optimizar, modificando el *callback* de control del robot, sin redimensionar las imágenes y con dos hilos.

Optimización	Tiempo(ms)	Preprocesado	Predicción	Comando	Bucle
Sin optimizar	**medio**	336	138	72	**1590**
	máximo	903	840	350	**3090**
Callback modificado	**medio**	43	32	1	119
	máximo	94	46	1	**208**
Sin redimensión	**medio**	3	34	0	**79**
	máximo	5	49	1	**148**
Con dos hilos	**medio**	0	31	0	**31**
	máximo	0	35	1	**38**

5.4.3 Ejecución Secuencial

Inicialmente, se planteó una ejecución de los *scripts* de manera secuencial. El primer problema surge con los tiempos obtenidos tanto en la tarea de Seguir Carril como en las de Seguir Línea. Por lo que se decidió optimizar primero la tarea de clasificación para Seguir Carril y después adaptar las demás versiones. En la tabla 5.2 podemos ver un listado de los tiempos que se obtenían sobre esta tarea. El problema surge en que el tiempo medio entre cada iteración del bucle necesitaba 1,5 segundo con picos de hasta 3 segundos. El rendimiento que se observaba era pésimo en todos los campos.

El problema de esta ejecución incurre en la librería de Aurigapy. Esto se debe a que cada comando lanzado para su ejecución en el robot, llevan asociado un *callback*. La implementación de este *callback* produce conflictos con la captura de la cámara, bloqueándola. Esto produce una penalización representada en el tiempo entre bucles que recoge el periodo que tardo en realizar una nueva captura.

Podría ser una opción corregir esta librería para solucionar el problema de raíz, pero, dado que esta librería no fue creada en el proyecto y se deseaba modificar lo mínimo posible se optó por otra solución. Esta solución consistió en enviar un *callback* con una función que produzca un error, ya que de este modo nos encontramos con un error de ejecución en el *callback* que lo bloquea, pero no afecta a nuestro hilo, que continúa funcionando sin recibir interferencia a la ahora de comunicarse con el robot y no afectando en nuestros tiempos.

En la tabla 5.2 se muestran los tiempos conseguidos tras implementar esta solución donde el tiempo de ejecución del comando prácticamente se redujo al 100% oscilando ente 0 y 1 ms. De media se conseguían procesar sobre 8-9 imágenes por segundos, pero los picos podían descender esta captura de procesado a 5 imágenes por segundo. Este resultado era crítico para la ejecución en tiempo real por lo que se decidió intentar acelerar más el código.

5.4.4 Aceleraciones

Observando la tabla 5.2 se localiza que el mayor tiempo entre bucles correspondía al preprocesado de la imagen. Este preprocesado implicaba una disminución del tamaño de captura de la imagen de 640x480px a los esperados 64x80.

Sin embargo, el módulo de la cámara permite capturar las imágenes en la dimensión que se necesitaba fijando la captura de imágenes a 80x64. Esta mejora consigue acelerar la ejecución, reduciendo los tiempos según se indican en la tabla 5.2. Con estos tiempos se conseguían ejecuciones 14-13 fps, por lo que permitían ya un recorrido completo del circuito, pero, cuando en alguna situación crítica para el sistema, se producía un pico, el robot perdía el control del carril no consiguiendo realizar su tarea.

Analizando nuevamente la tabla 5.2 podemos ver que si sumamos el tiempo de preprocesado y predicción medio no obtenemos el tiempo entre bucles. Esto es debido a que la cámara necesita un tiempo para la captura de las imágenes que se ve reflejado en la diferencia entre el tiempo entre bucles y la suma del resto. El otro tiempo que más milisegundos consumía era la propia red de CNN, pero se mantiene estable entre los 34-49 ms mientras que el tiempo de captura se mantiene entre en los 40 ms pero alcanzando picos de 94 ms.

Se decidió acelerar el tiempo de captura de las imágenes debido a que el objetivo era crear un sistema que pueda ejecutar otras redes, pudiendo ser más complejas y con mayor consumo de tiempo, por lo que intentar optimizar la red y reducirla podría acarrearnos problemas futuros con tareas más complejas.

5.4.4.1 Ejecución en paralelo en dos hilos

El problema que se estaba produciendo en nuestra ejecución secuencial parte de que la ejecución de la cámara se bloqueaba para permitir la ejecución del procesado sobre la imagen. Una vez finalizado el procesado, la ejecución de la cámara necesita ser desbloquea y capturar nuevamente la imagen, consumiendo un tiempo que puede evitarse con la creación de un hilo independiente para la captura de imágenes, ya que no se producirá este bloqueo.

Para llevar a cabo esta aceleración, se implementó un *script* con una clase adicional que se encarga de crear un hilo dedicado a la ejecución del módulo de la cámara, aportando las funcionalidades básicas sobre la cámara de inicialización, actualización, lectura y finalización.

En la tabla 5.2 podemos observar los nuevos tiempos obtenidos. Estos nos permitían obtener una tasa de 30 fps lo que permitía una ejecución en tiempo real. La reducción del tiempo que se ha conseguido es uno de los pilares básicos que permite que, usando este sistema, se puedan realizar diversas tareas.

Hasta ahora no se ha comentado los resultados de la regresión para las tareas de Seguir Carril y Seguir Línea, ya que en un inicio iban a la par de los comentados en estas secciones y

Tabla 5.3: Tiempos de ejecución de la regresión para Seguir Carril y Seguir Línea con dos hilos.

Optimización	Tiempo(ms)	Preprocesado	Predicción	Comando	Bucle
Seguir Carril	**medio**	1	10	0	**11**
	máximo	3	25	1	**28**
Seguir Línea	**medio**	1	10	0	11
	máximo	2	20	2	**22**

el *script* de ejecución de la regresión no fue actualizado hasta finalizar con las optimizaciones. Sin embargo, una vez conseguido optimizar al máximo los elementos externos al modelo, se realizó una medición de los tiempos sobre estos modelos de regresión obteniendo los datos recopilados en la 5.3. Revisando la información de la sección 5.3.2.1, nuestra arquitectura de regresión tienen un mayor número de capas, pero esto reduce los parámetros a 150.000 frente a 2 millones de nuestra arquitectura para la clasificación de Seguir Carril (5.3.1.1). Esto permite que la regresión consigan funcionar a 11 ms, proporcionando un total de 90 fps.

Estas optimizaciones fueron realizadas con la fuente de alimentación original de la Raspberry. A la hora de realizar las ejecuciones sobre el robot en un circuito, no se puede hacer uso de esta fuente, ya que nos limitaría el desplazamiento que podríamos realizar a la distancia del cable de esta. Por ello, se hizo uso de las baterías externas. El problema que surge, es que una de ellas no es capaz de suministrar toda la alimentación que necesita la placa en la ejecución por lo que penaliza entre 10 y 15 ms. Aún con esta penalización, es posible la ejecución en tiempo real, cumpliendo con un requisito básico, por lo que se decidió no adquirí nuevas baterías externas de mayor potencia.

5.4.5 Adaptación de Aurigapy

No fue encontrada ninguna función en la librería de Aurigapy para poder ejecutar los movimientos de desplazamiento sobre el circuito. Para evitar modificar las funciones existentes, se crearon dos funciones encargadas de desplazar el robot para la red de clasificación y para la red de regresión.

Para la clasificación se creó la función de la figura 5.17. Esta función espera recibir el comando de desplazamiento indicándole la dirección en la que debe de avanzar y la velocidad a la que debe hacerlo. Admite el desplazamiento hacia delante, hacia la derecha, hacia la izquierda también hacia atrás aunque no se ha usado.

La función que se añadió para la regresión se puede ver en la figura 5.18, donde se espera recibir un ángulo, una velocidad y un offset que nos ayudara a realizar mayores giros. Para poder realizar los giros se ideó una operación que, según el ángulo, ralentiza el motor del lado al que se debe girar donde, si fijamos la velocidad a 50 y el offset a 20, las operaciones que se

```
def set_command_Clasi(self, command, speed, callback=None):
    # ff 55 07 00 02 05 <2short speedleft> <2short speedright>
    commands = ["forward", "backward", "right", "left"]
    assert command in commands, "Error, %r command not in %r" % (command, str(commands))

    if command == "forward":
        speed_left = -speed
        speed_right = speed
    elif command == "backward":
        speed_left = speed
        speed_right = -speed
    elif command == "left":
        speed_left = -speed
        speed_right = 0
    elif command == "right":
        speed_left = 0
        speed_right = speed
    else:
        assert True, "Error in set_command"

    if callback is None:
        rp = Response.generate_response_block(Frame.FRAME_TYPE_ACK, timeout=2)
    else:
        rp = Response.generate_response_async(callback, Frame.FRAME_TYPE_ACK)
    self.add_responder(rp)

    data = bytearray([0xff, 0x55, 0x07, 0x00, 0x02, 0x5] +
                     short2bytes(speed_left) +
                     short2bytes(speed_right))

    # print '[{}]'.format(', '.join(hex(x) for x in data))
    self._write(data)

    if callback is None:
        rp.wait_blocking()
```

Figura 5.17: Función de control del robot para la clasificación.

```
def set_command_Regre(self, angles,speed,offset, callback=None):
    # ff 55 07 00 02 05 <2short speedleft> <2short speedright>
    #motor derecho problemas de giro +10 compensar
    if angles==90:
        speed_left = -speed/2
        speed_right = speed/2
    elif angles<90:
        speed_left = -round(speed/(90/angles-offset))
        speed_right =speed
    elif angles>90:
        speed_left = -speed
        speed_right = round(speed/(angles+offset/90))

    if callback is None:
        rp = Response.generate_response_block(Frame.FRAME_TYPE_ACK, timeout=2)
    else:
        rp = Response.generate_response_async(callback, Frame.FRAME_TYPE_ACK)
    self.add_responder(rp)

    data = bytearray([0xff, 0x55, 0x07, 0x00, 0x02, 0x5] +
                     short2bytes(speed_left) +
                     short2bytes(speed_right))

    # print '[{}]'.format(', '.join(hex(x) for x in data))
    self._write(data)

    if callback is None:
        rp.wait_blocking()
```

Figura 5.18: Función de control del robot para la regresión.

deben realizar dependiendo del ángulo son las siguientes:

- Ángulo = 90º: Deben de girarse las ruedas a la misma velocidad por lo tanto se produce un movimiento en línea recta con la velocidad fijada a 50.
- Ángulo < 90º: Indica que el robot debe de girar a la derecha por lo que se establece una velocidad de 50 a la rueda izquierda y para la rueda derecha se realiza la operación de:

$$Rueda_d = 50/(90/(Angulo - 20))$$

- Ángulo > 90º: El robot tiene que girar hacia la izquierda por lo que se fija la rueda derecha a 50 y se disminuye la velocidad de la rueda izquierda. La operación que se lleva a cabo sobre esta es:

$$Rueda_i = 50/((Angulo - 20)/90))$$

Capítulo 6

Pruebas experimentales

En este capítulo se ejecutarán las pruebas experimentales de los modelos entrenados, evaluando los resultados obtenidos a la hora de realizar las tareas para la que fueron diseñados en un entorno controlado en el medio real.

6.1 Seguir Carril con red de clasificación

En esta sección se recopilan los datos obtenidos en la ejecución de la clasificación sobre la tarea de seguir un carril de una pista. El modelo que se ha utilizado, es el que se obtuvo en la sección 5.3.2.1. Para probar el correcto funcionamiento, se ejecutó sobre circuitos de colores negro o azul, con líneas de colores blanco, rojo y diferentes tonalidades de amarillo, variando las formas con respecto a los circuitos utilizados para la captura de imágenes.

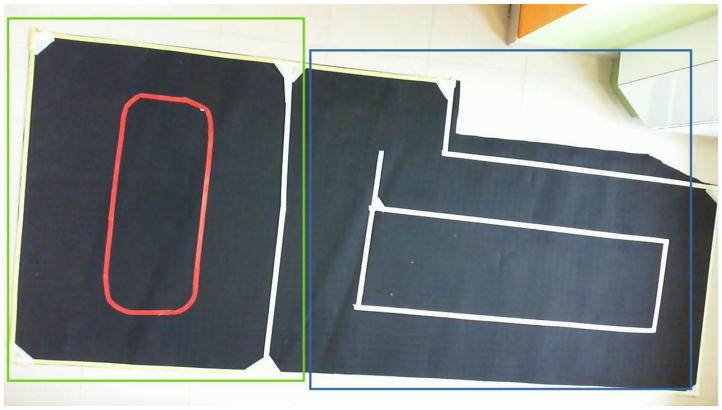

Figura 6.1: Circuito negro para las pruebas experimentales.

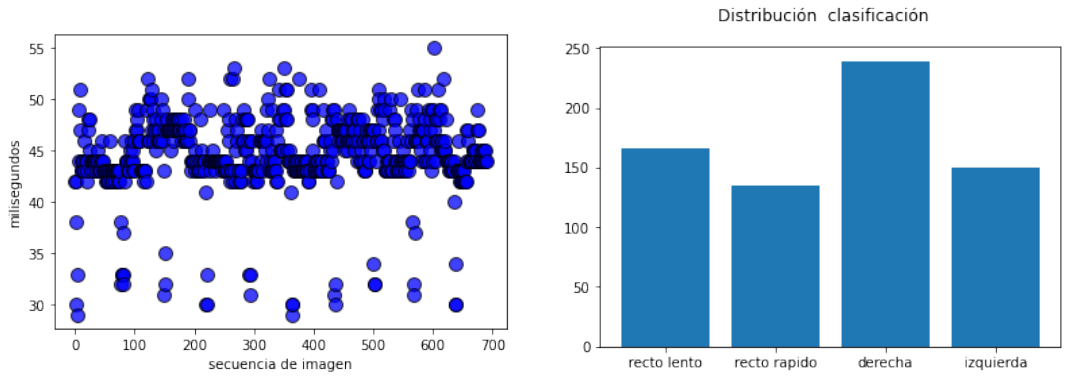

Figura 6.2: Seguir Carril con clasificación en el circuito negro: (izqda) tiempo entre bucles y (drcha) distribución de las etiquetas.

6.1.1 Prueba sobre circuito negro en sentido horario

Esta prueba fue realizada sobre el recorrido de la figura 6.1 marcado con un recuadro azul, modificando la forma y cambiando las líneas de color rojo a blanco, con respecto al circuito original. Sobre este circuito comenzó la ejecución grabándose los vídeos [29]. Se han fijado unas velocidades de desplazamiento de 55 unidades para la clase recto rápido, 40 unidades para recto lento y 70 unidades para los giros.

En la parte izquierda de la figura 6.2 podemos ver una gráfica de los tiempos necesarios para la realización de la clasificación entre imágenes, observando que la mayor parte de estos tiempos se alojan en el rango de los 40 a 50 milisegundos. Por tanto, el tiempo de ejecución cumple nuestros requisitos, siendo apto para poder realizar la clasificación en tiempo real, aunque, comparándolo con la duración obtenida en la sección 5.4, vemos como difieren en torno a los 10-15 ms de penalización por la fuente de alimentación.

En este recorrido, el sentido de giro implica que se deben realizar más desplazamientos hacia la derecha. En la parte derecha de la figura 6.2, podemos ver la distribución de las etiquetas viendo como predomina el giro hacia la derecha, acorde con lo que se esperaba. Sin embargo, el circuito contiene más rectas que curvas y no se ve demostrado en la distribución. Esto se debe a que a la hora del desplazamiento en la pista se producen movimientos bruscos, haciendo que el robot esté constantemente desviado, corrigiéndose y continuando su avance hasta la detección de un nuevo giro. Este es el principal problema de trabajar con datos discreto, ya que no existen puntos intermedios en el desplazamiento.

En la figura 6.3 se muestra la vista del robot sobre la clasificación de cada tipo de etiqueta. Debido a que el suelo sobre el que está alojado el circuito es blanco brillante, vemos como en la etiqueta derecha no se distingue la línea y el suelo, pero consigue clasificar correctamente la imagen. La última imagen corresponde con un giro a la izquierda, esto se debe a que tiene que corregir su trazado para no atravesar el límite.

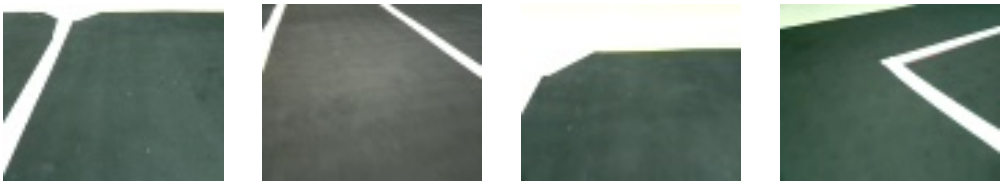

Figura 6.3: Vista de la cámara del robot en Seguir Carril con clasificación en circuito negro: (de izqda a derecha) etiquetas inferidas de recto lento, recto rápido, derecha e izquierda en el circuito negro.

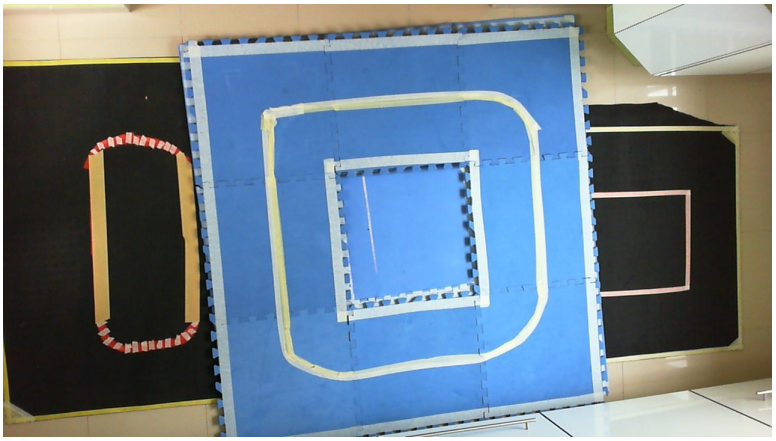

Figura 6.4: Circuito azul para las pruebas experimentales.

Aunque el tiempo de navegación entre imágenes sea elevado, este recorrido ha sido exitoso, ya que, se logró alcanzar el objetivo de regresar al punto de partida sin necesidad de la interferencia externa, logrando la correcta clasificación en curvas y rectas.

6.1.2 Prueba sobre circuito azul en sentido antihorario

Esta prueba fue realizada sobre el circuito azul 6.4, modificando el color de las líneas de amarillo por blanco y variando ligeramente las curvas en esquinas interiores. Se puede acceder al vídeo [30], donde se ve al robot realizando el recorrido en la parte interna y externa, levantándolo manualmente para el cambio de carril. Esta ejecución fue realizada con la misma fuente de alimentación que la de la sección 6.1.1. Las velocidades se modificaron a un desplazamiento de 55 unidades para la clase recto rápido, 40 unidades para recto lento y 90 unidades para los giros.

En la figura izquierda 6.5, podemos ver una gráfica de los tiempos necesarios para la realización de la clasificación entre imágenes viendo que se distribuyen en dos zonas diferenciadas entre los rangos 32 a 40 milisegundos y el otro rango entre los 45 y 55 milisegundos. En ambos casos, estos tiempos nos permiten realizar una navegación en tiempo real. Esta distribución irregular se debe a que, nuevamente, la fuente de alimentación no es capaz de suministrar

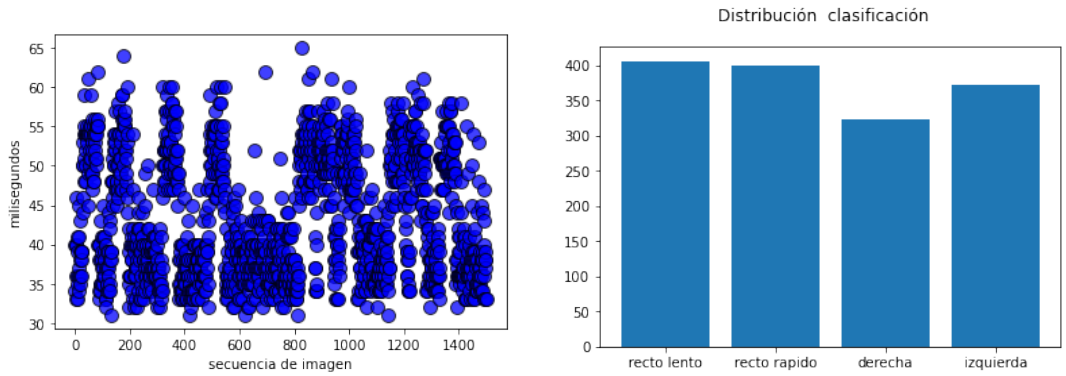

Figura 6.5: Seguir Carril con clasificación en circuito azul: (izqda) tiempo entre bucles y (drcha) distribución de las etiquetas.

Figura 6.6: Vista de la cámara del robot en Seguir Carril con clasificación en circuito azul: (de izqda a derecha) etiquetas recibidas de recto lento, recto rápido, derecha e izquierda.

toda la energía que debe, obteniendo un aviso de que se produce un bajo voltaje en momentos de la ejecución.

La distribución de las etiquetas en este circuito corresponde con la gráfica derecha de la figura 6.5. Aunque el sentido de giro sea hacia la izquierda, observamos que no hay notables diferencias entre el número de clasificaciones de cada tipo. Analizando el comportamiento, esto se debe a que al aumentar la velocidad de giro, cuando se realiza un movimiento de giro en una recta para compensar la desviación, es demasiado brusco, provocando que tenga que corregirse sucesivamente. El robot es cambiado al carril exterior manualmente, tras realizar una vuelta completa por el recorrido interior, sin embargo, no alcanza el objetivo de regresar a su posición.

En la figura 6.6 podemos ver algunas imágenes de la visión del robot y sus etiquetas, viendo que coinciden con lo esperado a excepción de la imagen que produce que el robot no sea capaz de regresar a su origen. Esta se corresponde con la primera imagen, donde nos encontramos una clasificación errónea de un giro a la izquierda como recto lento. Esto se debe a la situación analizada a la hora de realizar el *dataset* del "Perceptual aliasing", por lo que no fue posible evitarlo. Por tanto esta prueba no fue exitosa, confirmando los problemas de trabajar con este tipo de circuitos con curvas en esquinas.

6.1.3 Análisis del comportamiento

En estas pruebas analizadas se ha comprobado que la ejecución de la clasificación, es acorde al comportamiento esperado, pudiendo regresar a su origen en circuitos con configuraciones totalmente distintas a las usadas en su entrenamiento, pero para poder realizar los trazados de las curvas, dependerá de la inercia del desplazamiento, pudiendo provocarnos situaciones donde no sea posible resolver la clasificación exitosamente de la imagen. Este problema estaba previamente contemplado y podría tratar de solucionarse mediante una nueva clase en la que se obligase a retroceder y girar hacia un lado el robot, pero debido a los resultados que se estaban obteniendo en paralelo sobre la regresión para Seguir Carril, no se consideró mejorar el modelo, ya que, en la mayor parte de ejecuciones, consigue llegar a su objetivo.

Se realizaron una seria de pruebas más, resultando exitosas todas ellas, por lo que solo falló en una ocasión 6.1.2 ocurriendo un fallo que esperábamos. A estas comprobaciones a mayores, se puede acceder a ver su vídeo, con las siguientes características:

- Prueba sobre circuito negro en sentido horario [31].
- Prueba sobre circuito negro oval sentido antihorario [32] .
- Prueba sobre circuito negro oval sentido horario [33].

6.2 Seguir Carril con regresión CNN

En esta sección se realizan una serie de pruebas con el modelo creado en la sección 5.3.2. Para comprobar el correcto funcionamiento, se ejecutó sobre circuitos de colores negro o azul, con líneas de colores blanco, rojo, marrón y diferentes tonalidades de amarillo, variando las formas con respecto a los circuitos utilizados para la captura de imágenes.

6.2.1 Prueba sobre circuito negro en sentido antihorario

Para realizar esta prueba se utilizó el circuito 6.2, usado anteriormente en la sección 6.1.1, capturando los vídeos desde la posición aérea y la vista del robot [34]. Se realizó el desplazamiento en sentido antihorario con una velocidad fijada a 70 unidades con un offset de 10 unidades.

En la gráfica izquierda de la figura 6.7 podemos ver la distribución de los tiempos totales necesarios entre la captura de cada imagen a lo largo de la ejecución, donde se ha mantenido la fuente de alimentación de las anteriores pruebas. Los tiempos de la tabla 5.3, eran inferiores a los obtenidos, pero, con la penalización de nuestra fuente, vemos como aumentan ejecutándose en un rango estable entre los 15 y 22,5 milisegundos.

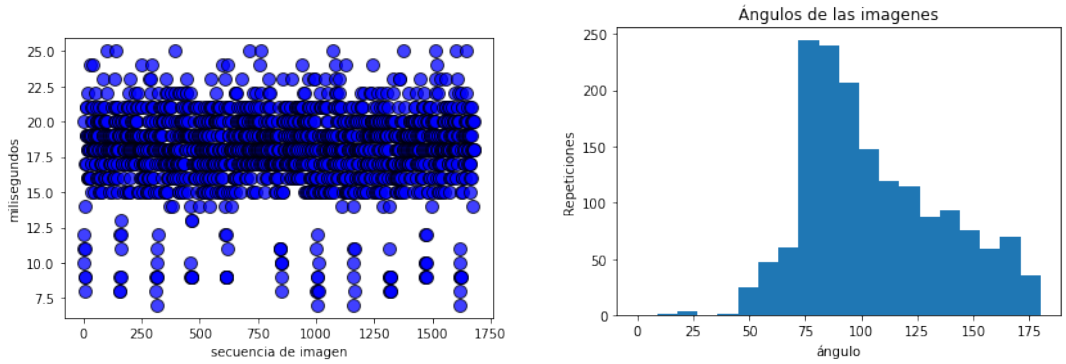

Figura 6.7: Seguir Carril con regresión en circuito negro (sentido antihorario): (izqda) tiempo entre bucles y (drcha) distribución de las etiquetas.

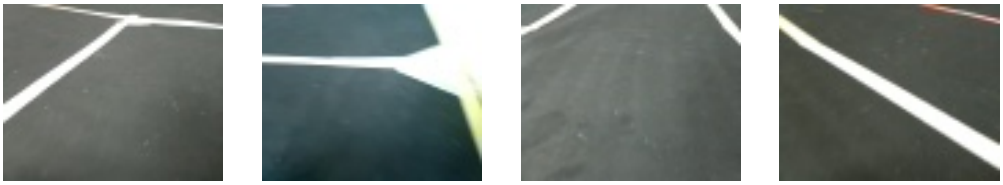

Figura 6.8: Vista de la cámara del robot en Seguir Carril con regresión en circuito negro: (de izqda a derecha) ángulos inferidos de 53, 123, 76 y 132º.

Al cambiar el espacio del problema a valores continuos, evitamos realizar giros bruscos cuando no son necesarios porque tenemos el abanico de valores entre clases del que no disponíamos en la clasificación. Un factor importante a tener en cuenta es el offset que indicamos y el funcionamiento no lineal de los motores, no soportando velocidades bajas. Las inferencias realizadas a lo largo de la ejecución corresponde con la figura derecha de 6.7, donde se concentra la distribución de los ángulos, mayoritariamente, a partir de los 90º. Los ángulos que indican que debe girar a la izquierda coinciden con la mayor parte de curvas del circuito, reduciendo los giros contrarios a los mínimos necesarios solo para corregir el trazado suavemente.

En la figura 6.8 se puede ver unos ejemplos de la visión de la cámara. La última imagen de esta figura obtiene una inferencia de 132º que indica un giro a la izquierda, pero el robot no es capaz de realizarlo a esa distancia de la línea por lo que se produce el error y comienza a producir un fallo en la predicción indicando que debe de girar a la derecha.

Esta ejecución no resultó exitosa en sus últimos pasos y se reintentó en distintos sentidos con nuevos fallos. Analizando el recorrido puede verse como el robot consigue trazar las curvas muy justo, saliendo incluso la rueda del circuito. Esto se debe a que el funcionamiento continuo del modelo sobre un circuito con curvas de 90º produce que el giro se realice gradualmente provocando una reacción tardía. Este problema sumado a que podemos vol-

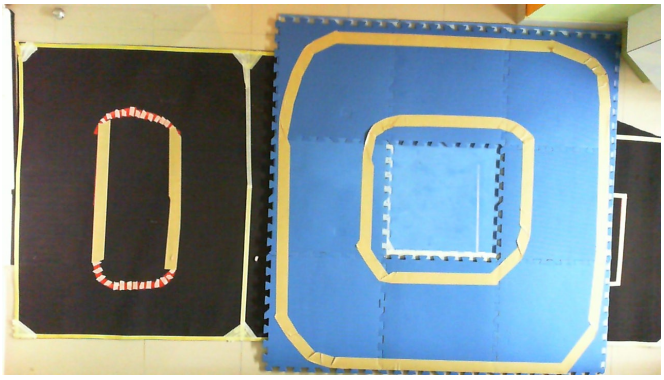

Figura 6.9: Circuitos de prueba con un único carril y esquinas redondeadas.

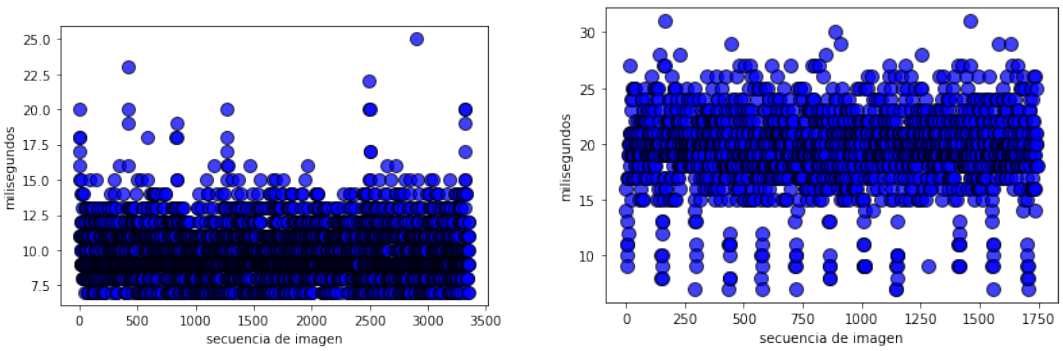

Figura 6.10: Seguir Carril con regresión en circuitos de carril único y esquinas redondeadas: tiempos entre imágenes del robots A (izqda) y B (drcha).

ver a encontrarnos con problemas de "Perceptual aliasing", hizo que se decidiese ejecutar la regresión sobre circuitos con curvas más circulares.

6.2.2 Prueba sobre circuitos de carril único y esquinas redondeadas con dos robots en paralelo

En esta prueba se ejecutaron ambos robots en paralelo en dos pistas separadas, capturando los vídeos de la vista desde los robots y la vista aérea [35]. En la imagen 6.9 se pueden ver los circuitos azul y negro que usaremos para la ejecución. El circuito azul es un circuito totalmente nuevo con un solo carril delimitado por una línea marrón tanto interior como exteriormente. El circuito negro es también nuevo, combinando en el interior líneas marrones, rojas y blancas, y en el exterior únicamente blancas. Sobre el circuito azul se ejecutará el código en el robot A mientras que en el circuito negro se ejecutará el robot B.

El robot A se ejecuta a una velocidad de 100 unidades y un offset de 20, mientras que en el robot B se descendió la velocidad a 60 unidades y el offset a 15.

En los tiempos de ejecución de la figura 6.10 se aprecia una gran diferencia entre los tiem-

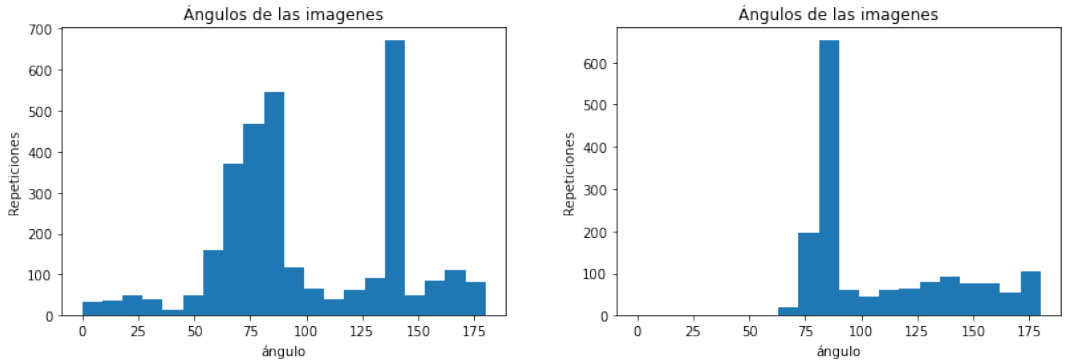

Figura 6.11: Seguir Carril con regresión en circuitos de carril único y esquinas redondeadas: distribución de las inferencias del robot A (izqda) y B (drcha).

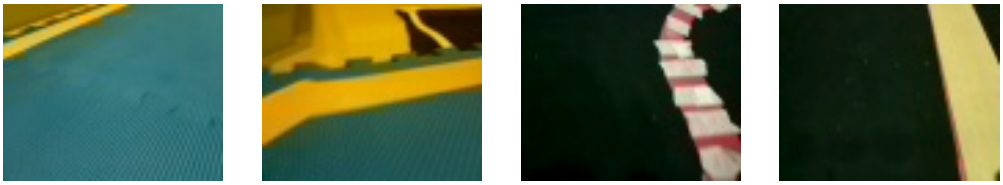

Figura 6.12: Seguir Carril con regresión en circuitos de carril único y esquinas redondeadas: vista de la cámara del robot A (dos primeras imágenes) y del robot B (dos últimas imágenes).

pos medidos para cada robot. Esto se debe a la diferencia de potencia entre las fuentes de alimentación. En el robot A, se usó la fuente de alimentación más potente de la que se dispone, obteniendo un rango de tiempos comprendido entre 7.5 y 12.5 milisegundos, con alguna anomalía que alcanzado los 25 ms. Sin embargo, en el robot B se mantiene la alimentación de la prueba 6.2.1, con un rango de tiempos que se encuentra entre los 15 y 25 milisegundos. Por tanto, las ejecuciones se producen en tiempo real.

Ambos robots giran en sentido horario, por lo que la mayor parte de los giros que realizan, deberían de ser hacia la derecha. Observando la figura 6.11 vemos que en el robot A predomina las inferencias en torno a los ángulos de 90º y 140º. Analizando sus características, se encuentra en lo esperado porque le hemos puesto una velocidad alta y un offset alto, por lo que estamos reforzando que se realicen giros bruscos sobre ángulos cercanos al recto. En el robot B vemos como los ángulos se encuentran más cerca del ángulo recto, viendo que el desplazamiento es mucho más suave.

En este circuito se comprobó el correcto funcionamiento modificando las velocidades y los offsets así como introduciendo líneas que están combinadas, consiguiendo realizar el recorrido sin necesidad de intervención externa. También se ejecutó otra prueba sobre estos dos circuitos y robots modificando nuevamente las velocidades y offsets y consiguiendo un recorrido exitoso a excepción de que la rueda trasera del robot A se queda enganchada, siendo

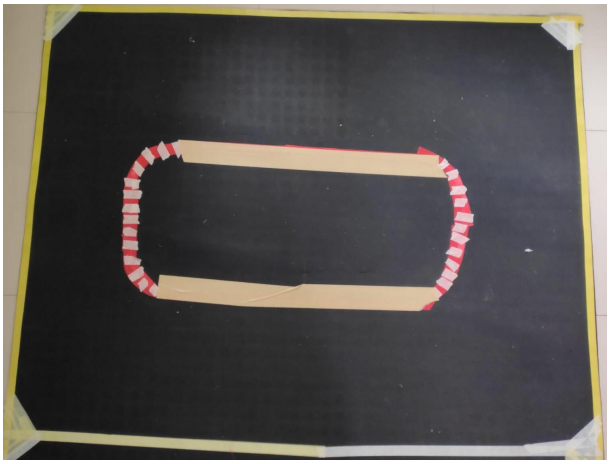

Figura 6.13: Circuito para la prueba de persecución.

necesaria la intervención externa para soltarla (vídeo [36]).

6.2.3 Prueba sobre circuito negro sentido horario: Persecución

Para la ejecución de esta prueba se utilizaron nuevamente ambos robots, siguiendo la misma distribución para el robot A y B que en la 6.2.2. El circuito usado es el de la figura 6.13, con líneas rojas, marrones, blancas y amarillas. Los vídeos capturados se corresponden con los de la vista desde la cámara de los robots y la vista aérea [37]. Sobre este circuito oval se ejecutaron ambos robot realizando una persecución en sentido horario con una velocidad fijada a 70 unidades y el offset en 10.

En la figura 6.14 podemos ver una gráfica de los tiempos consumidos por cada robot entre cada imagen. La primera gráfica corresponde con el robot A, mientras que la segunda con el robot B. Sobre el robot A, la gráfica nos indica que el rango de milisegundos que necesita se encuentra entre 7,5 y 12,5 milisegundos. Sin embargo, sobre el robot B obtenemos un rango superior de entre 15 y 25 milisegundos, debido a la diferencia entre ambas fuente de alimentación 5.4.

Las inferencias realizadas por ambos modelos, se representan en las gráficas de la figura 6.15, cuadrando con las salidas esperadas con una distribución notable en los ángulos menores a 90º por realizar giros a la derecha casi constantemente. La repetición de etiquetas a partir del ángulo recto, representan las pequeñas correcciones que se tuvo que realizar en el desplazamiento.

En la figura 6.16, se puede ver la comparación de la vista de la cámara del robot A y B sobre el mismo punto de la curva. La cámara del robot B está más enfocada hacia el suelo, proporcionándole la capacidad de capturar mejor el color de las líneas en las imágenes, ya que no le afecta tanto la luminosidad. Sin embargo, esto le hace perder campo de visión en

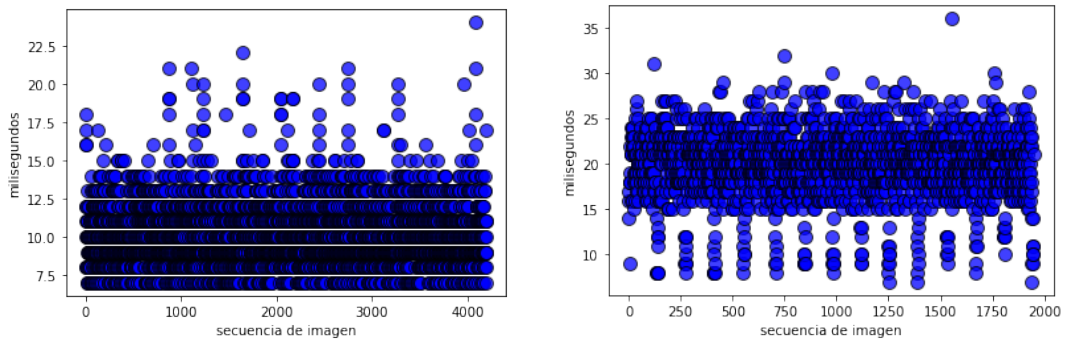

Figura 6.14: Prueba de persecución: tiempos entre bucles del robot A (izqda) y el B (drcha).

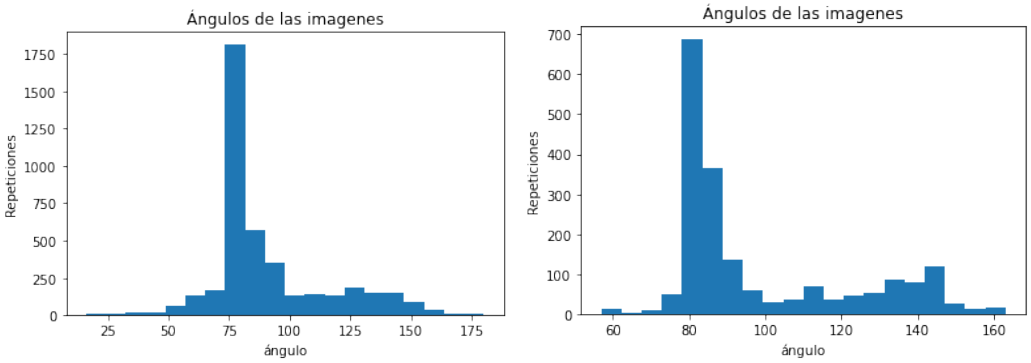

Figura 6.15: Prueba de persecución: distribución de las etiquetas realizadas por el robot A (izqda) y el B (drcha).

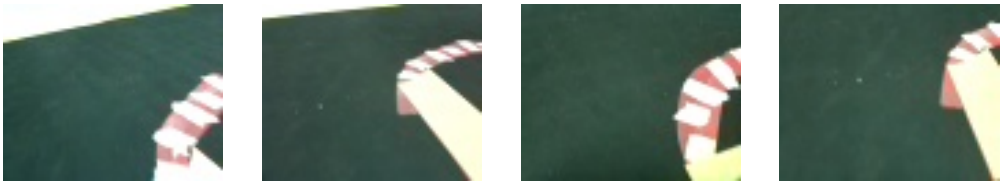

Figura 6.16: Prueba de persecución: vista de la cámara del robot A (dos primeras imágenes) y del robot B (dos últimas imágenes) en los mismos puntos del circuito.

comparación con las imágenes capturadas en el robot A. Aun así, ambos robots realizan el recorrido exitosamente alrededor del óvalo central, consiguiendo mantenerse en el carril en todo momento y conservando la distancia entre ambos robots.

6.2.4 Análisis del comportamiento

Al trabajar con la salida continua de la regresión, se introduce un comportamiento mucho más gradual con cambios acordes al estado actual. Si además añadimos un circuito con curvas circulares, el modelo ha sido capaz de funcionar sobre dos robots con diferente posición de las cámaras y motores ligeramente diferentes, sin encontrarse con los problemas de la clasificación, ya que se evita los cambios de dirección bruscos que puedan incurrir en una visión de una imagen problemática.

Para comprobar el funcionamiento sobre otros circuitos, se realizaron las siguientes pruebas, nuevamente todas ellas exitosas, de las que se enlaza los vídeos:

- Prueba sobre circuito negro oval sentido antihorario [38].
- Prueba sobre circuito negro oval sentido horario [39].
- Recorrido con colisión [40].
- Prueba sobre circuito negro y azul con 2 robots, variación de offset y velocidad [36].

6.3 Seguir Línea con regresión CNN

Sobre el modelo creado en la sección 5.3.3, se realizaron distintas pruebas variando las líneas y los colores de las mismas, así como las formas para configurar nuevos circuitos diferentes a los usados para capturar las imágenes de su *dataset*.

6.3.1 Prueba circuito negro en sentido antihorario

Para realizar esta prueba se utilizó el mismo circuito que el de la prueba 6.2. Se grabaron los vídeos de la vista aérea y desde el robot para comprobar el recorrido [41]. Se ejecutó en sentido antihorario, por lo que las curvas del circuito implican realizar mayoritariamente giros a la izquierda. La velocidad se fijó a 70 unidades y el offset en 20.

La gráfica izquierda de la figura 6.17 representa el tiempo total de inferencia entre las imágenes con un resultado en milisegundos en el rango de 15 a 22,5. Como en todas las pruebas, se cumple el requisito de que debe de ejecutarse en un tiempo medio menor a 100 ms.

En la ejecución se observa que se realizan pocos giros manteniendo un trazado recto cuando corresponde. La gráfica de la derecha en la figura 6.17 demuestra que la mayor parte de los ángulos obtenidos tras la ejecución del modelo, corresponde con el ángulo recto. Además,

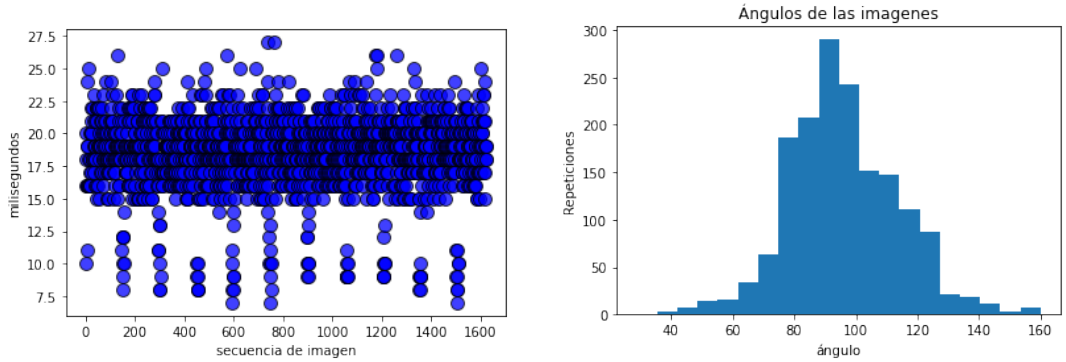

Figura 6.17: Seguir Línea con regresión en el circuito negro: (izqda) tiempo entre bucles y (drcha) distribución de las etiquetas de la clasificación.

Figura 6.18: Seguir Línea con regresión en circuito negro: vistas de la cámara del robot.

vemos que la distribución del resto de ángulos se acerca a este valor que, junto a nuestro sistema gradual, implica un desplazamiento de giro muy leve. Por último, vemos alguna inferencia cerca de los extremos. Estas ocurren debido a la necesidad de realizar un giro brusco para no perder la línea en las curvas al tener forma de ángulo recto.

En la secuencia de imágenes de la figura 6.18 vemos un resumen de los *frames* capturados por el robot, observando como pierde la línea en un ángulo de 90º y consigue recuperarse. Alcanzado igualmente el final de manera exitosa.

La ejecución sobre este circuito aportó más datos a cerca de los posibles problemas en los circuitos con ángulos rectos porque, aún habiendo sido una prueba exitosa, se pierde la referencia de la línea por unos instantes, ya que realizar estas curvas de manera perfecta no es posible debido a la implementación de la función de movimiento. Para observar la diferencia, se realizó una nueva prueba sobre un circuito con las curvas circulares.

6.3.2 Prueba sobre circuito azul en sentido horario

El circuito utilizado corresponde a la figura 6.19 capturando diferentes vídeos [42]. Este circuito está modificado con respecto al original con una línea con curvas circulares, eliminando las esquinas. Sobre este circuito se intentó modificar la velocidad aumentándola 10 unidades, pero, a la hora de traccionar para realizar un giro, se perdía el agarre con la superficie por lo que se mantuvo en 70 unidades y con un offset de 10. Se ejecutó en sentido horario.

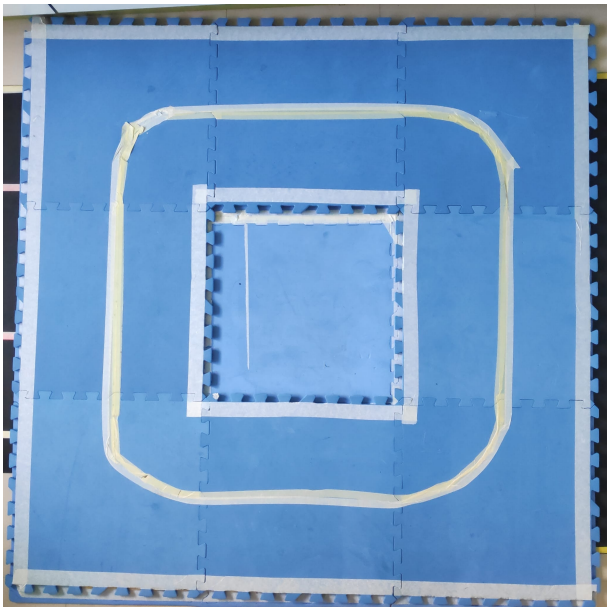

Figura 6.19: Circuito azul para la prueba de Seguir Línea.

En la gráfica izquierda de la figura 6.20, vemos como la distribución de los tiempos se mantienen en torno a los 15 y 25 milisegundos, mismo rango que el obtenido en la prueba 6.3.1. Sin embargo, vemos como se produce una anomalía, tras llevar 400 ejecuciones de la regresión, con una medición superior a 40 ms.

La figura 6.20 muestra la distribución de las inferencias realizadas. Observando esta gráfica vemos que se han realizado más inferencias de giro a la izquierda (ángulos entre 90 y 180) que de giros de las curvas del circuito hacia la derecha. Esto se debe a la poca fricción que la rueda interna tiene, produciendo el avance muy lentamente, ya que patina constantemente, y

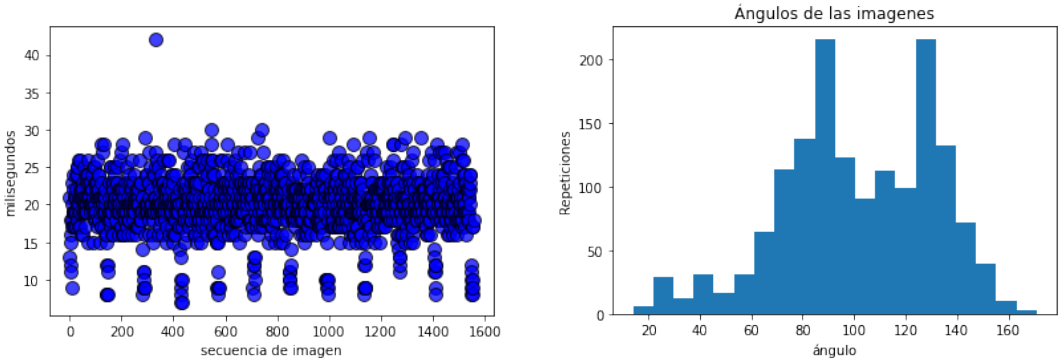

Figura 6.20: Seguir Línea con regresión en circuito azul: (izqda) tiempo entre bucles y (drcha) distribución de las etiquetas.

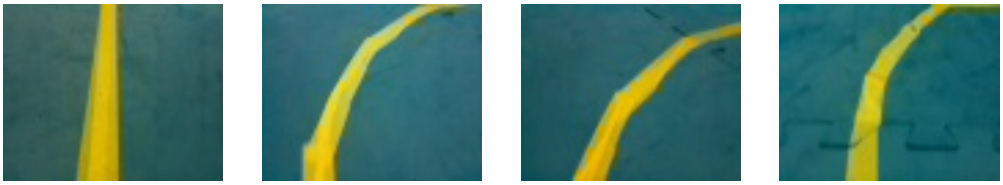

Figura 6.21: Seguir Línea con regresión en circuito azul: vistas de la cámara del robot.

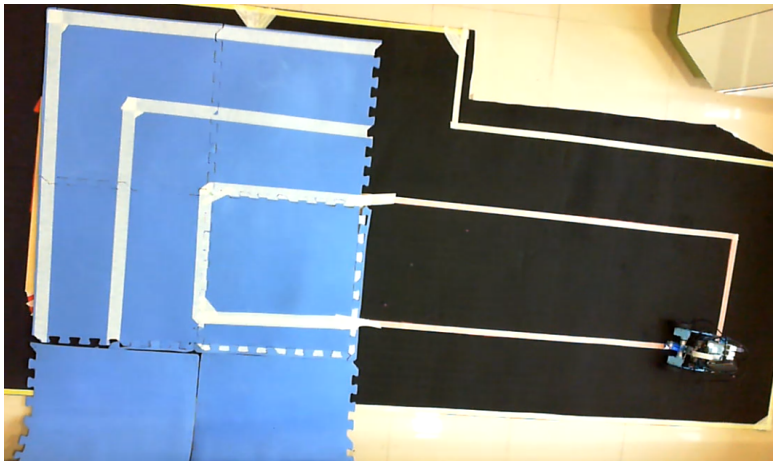

Figura 6.22: Circuito combinado para la prueba Seguir Línea.

tiene que ir aumentando la potencia hasta conseguir que se recupere la posición y continuar avanzando.

Este recorrido de curvas suaves permite la ejecución del modelo a la perfección sin ningún tipo de fallo permitiendo no perder de vista en ningún momento la línea que debe seguir, capturando a través de la cámara las imágenes de la figura 6.21.

6.3.3 Prueba circuito combinado

Como última prueba para el algoritmo de seguir la línea, se combinó el circuito azul y negro manteniendo la velocidad en 70 y el offset a 10 capturando los vídeos de su ejecución [43]. Estas configuraciones están a diferentes alturas, no suponiendo un problema para el robot, que fue capaz tanto de subir como bajar este pequeño escalón.

En los tiempos obtenidos mostrados en la gráfica izquierda de la figura 6.23, descendió ligeramente el rango superior con unos tiempos comprendidos entre 15 y 20 milisegundos.

En la gráfica derecha de la figura 6.23, se muestra la distribución de las repeticiones de las inferencias realizadas donde los ángulos mayores que 90º es superior a la de los ángulos menores. Esto es lo que se espera, ya que las curvas del circuito son hacia la izquierda por lo que los giros hacia la derecha únicamente se deberían realizar para compensar desviaciones.

Aun manteniendo los giros en ángulo recto, es capaz de seguir la línea y no perderla en

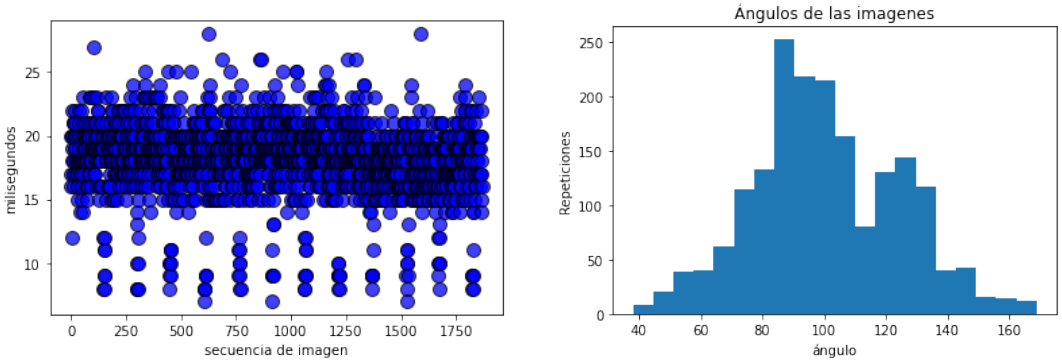

Figura 6.23: Seguir Línea con regresión en circuito combinado: (izqda) tiempo entre bucles y (drcha) distribución de las etiquetas.

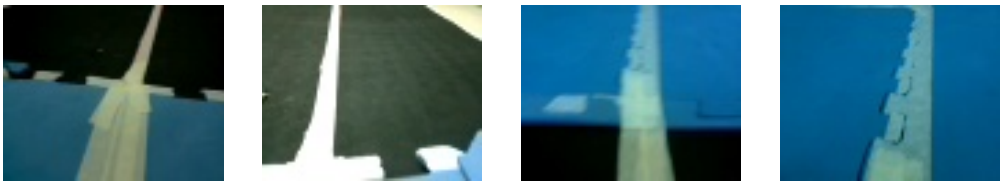

Figura 6.24: Seguir Línea con regresión en circuito combinado: vistas de la cámara del robot.

todo el recorrido. Pero esta prueba es importante para demostrar la robustez a la hora de seguir con la tarea en el punto crítico de intersección entre los dos circuitos azul y negro. En la figura 6.24 vemos la visión del robot en el paso de la superficie azul a negra y viceversa.

Esta ejecución fue exitosa comprobando que el modelo es capaz de generalizar sobre nuevas imágenes que no guarda más relación con las usadas originalmente, siendo capaz de seguir la línea en todo el recorrido.

6.3.4 Análisis del comportamiento

El comportamiento de la regresión fue exitoso en todas las pruebas aún teniendo problemas de desplazamiento por la tracción en superficies resbaladizas, pudiendo completar la acción sin necesidad de interferencia externa. También se ha demostrado que los modelos continuos nos aportan un mayor rango de control con un comportamiento más adaptado al entorno, que junto con circuitos con curvas progresivos, hacen que el sistema obtenga una salida exitosa en la mayor parte de las ejecuciones.

Los tiempos obtenidos en las pruebas también confirman la estabilidad del sistema a la hora de realizar diferentes tareas y su ejecución en tiempo real porque hemos obtenido unos tiempos similares en todas las pruebas haciendo uso de la batería externa menos potente, por lo que aún sería posible mejorar 10 ms hasta alcanzar los resultados de la tabla 5.3.

También se han realizado pruebas adicionales, todas ellas exitosas, que omitimos en esta

memoria pero cuyos vídeos están accesibles:

- Prueba sobre circuito negro oval sentido horario [44].
- Prueba combinada con sigue carril [45].

Capítulo 7

Conclusiones y trabajo futuro

En este capítulo de la memoria se presentan las conclusiones del trabajo realizado en este proyecto con un análisis de los resultados obtenidos así como el trabajo futuro, donde se abordarán aspectos que se podrían seguir expandiendo y mejorando.

7.1 Conclusiones

El objetivo de este proyecto era el desarrollo de un sistema para poder diseñar tareas de control (comportamientos) sobre un pequeño vehículo aplicando el aprendizaje automático. Se definieron dos tareas como punto de partida, fijando como objetivo para cada una de ellas conseguir realizar el seguimiento de un carril y el seguimiento de una línea. Como vehículo se han empleado dos robots móviles Makeblock Ranger, pequeños y de bajas prestaciones, pero suficientes para demostrar la viabilidad del sistema propuesto.

El primer paso del proyecto ha consistido en crear los diferentes datasets, es decir, conjuntos relevantes y equilibrados de imágenes etiquetadas con la salida deseada. Para facilitar este proceso en aquellos casos en los que no se puede o es difícil teleoperar el robot (como en nuestro caso), hemos diseñado un procedimiento de etiquetado automático de las imágenes capturadas. El etiquetado puede ser discreto (en clases) como contínuo (ángulo de giro). El proceso se basa en comparar parejas de imágenes consecutivas, calcular el flujo óptico y traducirlo a una etiqueta. De este modo, es posible crear fácilmente un dataset aunque haya que desplazar el robot manualmente o no se tenga acceso a los comandos de control enviados mientras se controla el robot. También se han balanceado y normalizado las diferentes clases para facilitar el entrenamiento.

El siguiente paso consistió en aplicar las técnicas de aprendizaje supervisado. Se han desarrollado dos arquitecturas de redes neuronales convoluciones (CNN) diferentes: una para clasificación y otra para regresión (para controlar el ángulo de giro del robot). La primera con cerca de dos millones de parámetros (pesos) y la segunda, mucho más eficiente, con algo más

de 150 mil parámetros (pesos). Sólo la CNN de regresión ha sido capaz de aprender a realizar las dos tareas propuestas (Seguir Carril y Seguir Pista). En ambos casos el entrenamiento se realizó en menos de dos minutos empleando la librería **tensorflow** en un portátil con una tarjeta Nvidia corriente.

El tercer elemento importante ha consistido en lograr la ejecución en tiempo real de los modelos desarrollados. Para ello se utilizó la librería *tensorflow lite* para convertir los dos modelos de CNN a un formato más eficiente para placas de procesamiento de bajas prestaciones. En nuestro caso, Raspberry Pi 3. Además, ha sido necesario modificar las librerías de control del robot Ranger, eliminar todas las fases de preprocesado y, por último, dividir la captura de imágenes en un hilo independiente. Como resultado final cabe destacar que hemos logrado unos tiempos medios de ejecución de 45 ms para la CNN clasificación y de 20 ms para la regresión. Con una buena fuente de alimentación, estos tiempos se reducen en 10 ms.

Se han desarrollado multitud de pruebas experimentales, tanto en los mismos circuitos en donde se extrajeron los diferentes datasets, como en circuitos modificados (cambiando la forma, el aspecto y el color de las líneas y carriles). En todas las ocasiones los modelos han sido capaces ejecutar correctamente cada comportamiento y de generalizar y operar en situaciones novedosas y no previstas.

Comparando el desempeño de los modelos de CNN empleados, podemos concluir que un sistema de clasificación en un problema con valores continuos pierde demasiada información entre cada clase. La regresión nos ha demostrado que con cambios progresivos es posible trabajar con toda la información que va recogiendo del medio, evitando los cambios bruscos y aportando un desplazamiento más fiable y robusto sobre los circuitos en los que fue probado.

7.2 Trabajo futuro

Aunque los sistemas de control desarrollados en este proyecto son totalmente funcionales, siempre es posible realizar mejoras para continuar aportando robustez o reducción de los tiempos de predicción y regresión con arquitecturas más pequeñas, o la incorporación de nuevos métodos de aprendizaje.

El primer aspecto podría ser comparar el proceso de etiquetado automático del dataset con otro obtenido de forma convencional, es decir, controlando el robot (bien de forma remota o bien con otro comportamiento) y anotar cada imagen capturada con el control correspondiente.

Otro aspecto a estudiar sería cómo afectaría el uso de una cámara gran angular. La ventaja es que se podría ver una parte más importante del entorno (en nuestro caso ambas líneas del carril, sobre todo en las esquinas cerradas). Pero como inconveniente estaría que habría información sin interés para el comportamiento, con el consecuente ruido.

En la misma línea se podrían estudiar otros comportamientos más complejos (robots móviles de investigación) o su adaptación a vehículos de mayores prestaciones y/o otro tipo de control (coches teledigiridos con dirección Ackerman). Así se podrían estudiar otras técnicas de *transfer learning*.

Por último, sería muy interesante aplicar técnicas de aprendizaje máquina capaces de aprender al mismo tiempo que se ejecuta la tarea. Bien entrenando desde cero (o a partir de otro tipo de conocimiento previo), bien ajustando y optimizando un modelo previamente entrenado.

Apéndices

Apéndice A

Repositorio del proyecto

En el repositorio del proyecto se encuentra el conjunto de *scripts* y el material necesario para la creación del proyecto [46].

En la carpeta raíz nos encontramos la siguiente distribución:

- Modelos: Contiene los mejores modelos en formato .tflite que se han conseguido del sistema.
- Raspberry: Contiene los *scripts* para la ejecución de los modelos en este dispositivo junto con la implementación del hilo paralelo para la cámara y la librería Aurigapy modificada.
- Seguir Carril: Contiene las arquitecturas implementadas para esta tarea.
- Seguir Línea: Contiene la arquitectura implementada para esta tarea.
- Vídeos de las pruebas ejecutadas.

Los *datasets* utilizados no comparten el mismo repositorio debido a la capacidad de la que se dispone es limitada, por lo que están disponibles a través del OneDrive plataforma. [47]

Apéndice B

Código del etiquetado para la regresión

En este capítulo se comentará el código empleado para etiquetas automáticamente las imágenes del dataset a partir del flujo óptico.

B.1 Cálculo del flujo óptico y etiquetado automático

El método propuesto se basa en comparan pares de imágenes consecutivas para determinar el movimiento de la cámara y, por tanto, del robot. El código se basa en una versión de internet[1].

En la sección del código B.1 se produce la inicialización de las imágenes sobre las que queremos averiguar el flujo óptico. Para ello, se realiza una transformación para trabajar con las imágenes en secuencia. También se crea y preparan las cabeceras de nuestro archivo de salida.

El siguiente paso que tenemos que hacer, es recorrer el total de nuestros ficheros. Dentro de este recorrido se realiza la extracción de las características de la imagen, determinadas por el flujo óptico, utilizando la detección de esquinas y aplicando el método Lucas-Kanade (sección B.2).

Se detectan los *inliers* gracias al método de RANSAC B.3, y se calcula las posiciones relativas en caso de que el número de inliers detectado sea mayor que un umbral que establecemos en 100.

El siguiente paso corresponde con nuestra operación final C.1, donde se recorren los puntos obtenidos en nuestra sección anterior y se comprueba si el punto pertenece al inlier. En caso de que esta comprobación sea válida obtendremos los datos del ángulo para ese punto.

[1] https://stackoverflow.com/questions/50210800/how-to-read-files-in-sequence-from-a-directory-in-opencv-and-use-it-for-processi

```
double f = 707.0912;
cv::Point2d c(601.8873, 183.1104);
bool use_5pt = true;
int min_inlier_num = 100;
FILE * camera_traj = fopen("vo_epipolar.xyz", "wt");
if (camera_traj == NULL) return -1;

std::string folder("/home/Escritorio/circuito/*_orig.jpg");
std::vector < cv::String > filenames;
cv::glob(folder, filenames, false);
printf("Folder: %s\n", folder.c_str());
printf("Number of files %ld\n", filenames.size());
std::ofstream myfile;
myfile.open("circuitoExterior.csv");
myfile << "nombre,angulo\n";

cv::VideoCapture video;
if (!video.open(filenames[0])) return -1;
cv::Mat gray_prev;
video >> gray_prev;
if (gray_prev.empty()) {
video.release();
return -1;
}
if (gray_prev.channels() > 1) cv::cvtColor(gray_prev, gray_prev,
    cv::COLOR_RGB2GRAY);
```

Código B.1: Apertura y de la secuencia de imágenes en el etiquetado para la regresión.

```
// Extract optical flow
std::vector < cv::Point2f > point_prev, point;
cv::goodFeaturesToTrack(gray_prev, point_prev, 2000, 0.01, 10);

std::vector < uchar > status;
cv::Mat err;
cv::calcOpticalFlowPyrLK(gray_prev, gray, point_prev, point, status, err);
gray_prev = gray;
```

Código B.2: Extracción del flujo óptico.

```
// Extract optical flow
std::vector < cv::Point2f > point_prev, point;
cv::goodFeaturesToTrack(gray_prev, point_prev, 2000, 0.01, 10);

std::vector < uchar > status;
cv::Mat err;
cv::calcOpticalFlowPyrLK(gray_prev, gray, point_prev, point,
    status, err);
gray_prev = gray;// Calculate relative pose
cv::Mat E, inlier_mask;

if (use_5pt) {
  E = cv::findEssentialMat(point_prev, point, f, c, cv::RANSAC,
    0.99, 1, inlier_mask);
} else {
  cv::Mat F = cv::findFundamentalMat(point_prev, point,
    cv::FM_RANSAC, 1, 0.99, inlier_mask);
  cv::Mat K = (cv::Mat_ < double > (3, 3) << f, 0, c.x, 0, f, c.y,
    0, 0, 1);
  E = K.t() * F * K;
}
cv::Mat R, t;
int inlier_num = cv::recoverPose(E, point_prev, point, R, t, f, c,
    inlier_mask);

// Accumulate relative pose if result is reliable
if (inlier_num > min_inlier_num) {
  cv::Mat T = cv::Mat::eye(4, 4, R.type());
  T(cv::Rect(0, 0, 3, 3)) = R * 1.0;
  T.col(3).rowRange(0, 3) = t * 1.0;
  camera_pose = camera_pose * T.inv();
}
```

Código B.3: Cálculo de *inliers* de la imagen.

```
if (inlier_mask.at < uchar > (i) > 0) {
    // cv::arrowedLine(image, point[i], point_prev[i], cv::Vec3b(0, 0,
    255)); //not print line of direction
    pend = (point[i].y - point_prev[i].y) / (point[i].x - point_prev[i].x);
    arctang = atan(pend) * 180 / PI;
    dist = sqrt(pow(point[i].y - point_prev[i].y, 2) + pow(point[i].x -
    point_prev[i].x, 2));
    // printf("\npunto\n");
    // std::cout << point << '\n';
    if (arctang < 0) {
      arctang = arctang + 180;
    }
    aux = aux + 1;
    aux2 = aux2 + arctang;
    distTotal = distTotal + dist;

  } else {
    // cv::line(image, point_prev[i], point[i], cv::Vec3b(0, 127, 0));
  }

  long ruedaDerecha = 0;
  long ruedaIzquierda = 0;

  if (i == point_prev.size() - 1) {
    angle = round(180 - aux2 / aux);
    if (0 <= angle and angle <= 180) {
      myfile << token;
      myfile << ",";
      myfile << angle;
      myfile << "\n";
    }
  }
}
```

Código B.4: Cálculo de los ángulos finales.

Se almacena el valor total para calcular la media de los ángulos obtenidos y se guarda en el fichero previamente iniciado para volver a realizar el proceso con una nueva imagen.

Apéndice C

Instalación de TensorFlow Lite en Raspberry.

La instalación de TensorFlow Lite, puede resultar compleja debido a la cantidad de paquetes que se deben instalar. Debido al complejo proceso que resultó conseguir instalar esta librería, se añade este apéndice con una guía de los comandos necesario para el funcionamiento (código C.1).

A mayores puede ser necesario la instalación de paquetes de las librerías comentadas en la sección 3.2, pero estas no ocasionaron problemas a la hora de su instalación. Se recomienda seguir este proceso de instalación de TensorFlow Lite porque otras instalaciones pueden incurrir en problemas con la ejecución del intérprete de la librería. Ha sido comprobado el día 16 de junio de 2021 y continúa siendo válido.

```
sudo apt-get install -y libhdf5-dev libc-ares-dev libeigen3-dev
python3 -m pip install keras_applications==1.0.8 --no-deps
python3 -m pip install keras_preprocessing==1.1.0 --no-deps
python3 -m pip install h5py==2.9.0
sudo apt-get install -y openmpi-bin libopenmpi-dev
sudo apt-get install -y libatlas-base-dev
python3 -m pip install -U six wheel mock
wget https://github.com/lhelontra/tensorflow-on-arm/relea
--ses/download/v2.0.0/tensorflow-2.0.0-cp37-none-linux_armv7l.whl
python3 -m pip install tensorflow-2.0.0-cp37-none-linux_armv7l.whl
```

Código C.1: Instalación de Tensorflow Lite en la Raspberry Pi 3.

Bibliografía

[1] J. McCarthy, M. L. Minsky, N. Rochester, and C. Shannon, "A proposal for the dartmouth summer research project on artificial intelligence," 1955, consultado el 23 de junio de 2021. [En línea]. Disponible en: https://arxiv.org/pdfhttp://www-formal.stanford.edu/jmc/history/dartmouth/dartmouth.html

[2] N. J. Nilsson, "Introduction to machine learning," 1998, consultado el 23 de junio de 2021. [En línea]. Disponible en: https://ai.stanford.edu/people/nilsson/MLBOOK.pdf

[3] P. Larranaga, I. Inza, and A. Moujahid, "Redes neuronales," consultado el 23 de junio de 2021. [En línea]. Disponible en: http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t8neuronales.pdf

[4] M. T. Hagan, *Neural Network Design*, 2nd ed. Martin Hagan, 1996.

[5] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," 1980, consultado el 23 de junio de 2021. [En línea]. Disponible en: https://www.rctn.org/bruno/public/papers/Fukushima1980.pdf

[6] "Imagenet large scale visual recognition challenge (ilsvrc)," 1980, consultado el 23 de junio de 2021. [En línea]. Disponible en: https://www.image-net.org/challenges/LSVRC/index.php

[7] S. Silva and E. Freire, "Intro a las redes neuronales convolucionales," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://bootcampai.medium.com/redes-neuronales-convolucionales-5e0ce960caf8

[8] J. J. Gibson, *The Perception of the Visual World*, 1st ed. HOUGHTON MIFFLIN COMPANY, 1950.

[9] Makebock, "Makebock ranger," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://www.makeblock.es/productos/mbot_ranger/

[10] R. P. Foundation, "Raspberry pi," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://www.raspberrypi.org/

[11] G. van Rossum, "Python," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://www.python.org/

[12] B. Stroustrup, "C++," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://www.cplusplus.com/

[13] F. Pérez, "Jupyter," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://jupyter.org/

[14] T. Oliphant, "Numpy," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://numpy.org/

[15] W. McKinney, "Pandas," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://pandas.pydata.org/

[16] I. Corporation, "Opencv," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://opencv.org/

[17] G. B. Team, "Tensorflow," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://www.tensorflow.org/?hl=es-419

[18] ——, "Tensorflow lite," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://www.tensorflow.org/lite/guide?hl=es-419

[19] F. A. Gregori, "Aurigapy," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://github.com/fidelaznar/aurigapy

[20] D. Jones, "Picamera," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://picamera.readthedocs.io/en/release-1.13/index.html

[21] J. D. Hunter, "Matplotlib," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://matplotlib.org/

[22] J. Hammersley and J. Lees-Miller, "Overleaf," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://www.overleaf.com/

[23] L. Tovard, "Git," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://git-scm.com/

[24] E. Tröger, "Geany," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://www.geany.org/

[25] "Manifiesto agile," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://agilemanifesto.org/

[26] "Principios manifiesto agile," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://agilemanifesto.org/principles.html

[27] S. Choi, "Visual odometry (monocular, epipolar version)," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://github.com/sunglok/3dv_tutorial/blob/master/src/vo_epipolar.cpp

[28] N. Developers, "End to end learning for self-driving cars," 2016, consultado el 23 de junio de 2021. [En línea]. Disponible en: https://arxiv.org/pdf/1604.07316v1.pdf

[29] R. C. Bugallo, "Prueba sobre circuito negro en sentido horario," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://github.com/rubicnhu/CNN-in-Raspy-fon-ranger-control/tree/main/Videos%20de%20las%20pruebas%20ejecutadas/clasificacion/Prueba%20sobre%20circuito%20negro%20en%20sentido%20horario

[30] ——, "Prueba sobre circuito azul en sentido antihorario," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://github.com/rubicnhu/CNN-in-Raspy-fon-ranger-control/tree/main/Videos%20de%20las%20pruebas%20ejecutadas/clasificacion/Prueba%20sobre%20icrcuito%20azul%20en%20sentido%20antihorario

[31] ——, "Prueba sobre circuito negro en sentido horario," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://github.com/rubicnhu/CNN-in-Raspy-fon-ranger-control/tree/main/Videos%20de%20las%20pruebas%20ejecutadas/clasificacion/Prueba%20sobre%20circuito%20negro%20en%20sentido%20antihorario

[32] ——, "Prueba sobre circuito negro oval sentido antihorario," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://github.com/rubicnhu/CNN-in-Raspy-fon-ranger-control/tree/main/Videos%20de%20las%20pruebas%20ejecutadas/clasificacion/Prueba%20sobre%20circuito%20negro%20oval%20sentido%20antihorario

[33] ——, "Prueba sobre circuito negro oval sentido horario," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://github.com/rubicnhu/CNN-in-Raspy-fon-ranger-control/tree/main/Videos%20de%20las%20pruebas%20ejecutadas/clasificacion/Prueba%20sobre%20circuito%20negro%20oval%20sentido%20horario

[34] ——, "Prueba sobre circuito negro en sentido antihorario," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://github.com/rubicnhu/CNN-in-Raspy-fon-ranger-control/tree/main/Videos%20de%20las%20pruebas%20ejecutadas/Seguir%20Carril/Prueba%20sobre%20circuito%20negro%20en%20sentido%20antihorario

[35] ——, "Prueba sobre circuito negro y azul con dos robots en paralelo," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://github.com/rubicnhu/CNN-in-Raspy-fon-ranger-control/tree/main/Videos%20de%20las%20pruebas%20ejecutadas/Seguir%20Carril/Prueba%20sobre%20circuito%20negro%20y%20azul%20con%20dos%20robots%20en%20paralelo

[36] ——, "Prueba sobre circuito negro y azul con variación de offset y velocidad," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://github.com/rubicnhu/CNN-in-Raspy-fon-ranger-control/tree/main/Videos%20de%20las%20pruebas%20ejecutadas/Seguir%20Carril/Prueba%20sobre%20circuito%20negro%20y%20azul%20con%20variacion%20de%20offset%20y%20velocidad

[37] ——, "Persecución," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://github.com/rubicnhu/CNN-in-Raspy-fon-ranger-control/tree/main/Videos%20de%20las%20pruebas%20ejecutadas/Seguir%20Carril/Persecucion

[38] ——, "Prueba sobre circuito negro oval sentido antihorario," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://github.com/rubicnhu/CNN-in-Raspy-fon-ranger-control/tree/main/Videos%20de%20las%20pruebas%20ejecutadas/Seguir%20Carril/Prueba%20sobre%20circuito%20negro%20oval%20sentido%20antihorario

[39] ——, "Prueba sobre circuito negro oval sentido horario," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://github.com/rubicnhu/CNN-in-Raspy-fon-ranger-control/tree/main/Videos%20de%20las%20pruebas%20ejecutadas/Seguir%20Carril/Prueba%20sobre%20circuito%20negro%20oval%20sentido%20horario

[40] ——, "Colision," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://github.com/rubicnhu/CNN-in-Raspy-fon-ranger-control/tree/main/Videos%20de%20las%20pruebas%20ejecutadas/Seguir%20Carril/Colision

[41] ——, "Prueba circuito negro en sentido antihorario," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://github.com/rubicnhu/CNN-in-Raspy-fon-ranger-control/tree/main/Videos%20de%20las%20pruebas%

20ejecutadas/Seguir%20Lineas/Prueba%20circuito%20negro%20en%20sentido%20antihorario

[42] ——, "Prueba sobre circuito azul en sentido horario," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://github.com/rubicnhu/CNN-in-Raspy-fon-ranger-control/tree/main/Videos%20de%20las%20pruebas%20ejecutadas/Seguir%20Lineas/Prueba%20sobre%20circuito%20azul%20y%20sentido%20horario

[43] ——, "Prueba circuito negro y azul," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://github.com/rubicnhu/CNN-in-Raspy-fon-ranger-control/tree/main/Videos%20de%20las%20pruebas%20ejecutadas/Seguir%20Lineas/Prueba%20circuito%20negro%20y%20azul

[44] ——, "Prueba sobre circuito negro oval sentido horario," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://github.com/rubicnhu/CNN-in-Raspy-fon-ranger-control/tree/main/Videos%20de%20las%20pruebas%20ejecutadas/Seguir%20Lineas/Prueba%20sobre%20circuito%20negro%20oval%20sentido%20horario

[45] ——, "Prueba combinada con sigue carril," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://github.com/rubicnhu/CNN-in-Raspy-fon-ranger-control/tree/main/Videos%20de%20las%20pruebas%20ejecutadas/Seguir%20lineas%20%2B%20Seguir%20Carril

[46] ——, "Repositorio del proyecto," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://github.com/rubicnhu/CNN-in-Raspy-fon-ranger-control

[47] ——, "Datasets del proyecto," consultado el 23 de junio de 2021. [En línea]. Disponible en: https://udcgal-my.sharepoint.com/:f:/g/personal/r_covelo_udc_es/EvDYf78EmBhFiRAe_2lrgNMBNls9LfNVNzjvzYP8Aqpq-w?e=F1ty5g