UNIVERSIDADE DA CORUÑA

Escola Politécnica Superior

Trabajo Fin de Grado

CURSO 2019/20

CARACTERIZACIÓN Y VALIDACIÓN DE ALGORITMOS DE VISIÓN ARTIFICIAL PARA LA RESOLUCIÓN EN TIEMPO REAL DE PROBLEMAS EN ROBÓTICA AUTÓNOMA MÓVIL

Grado en Ingeniería en Tecnologías Industriales

ALUMNO

Pablo Vázquez López

TUTORES

Francisco Javier Bellas Bouza
Alejandro Romero Montero

FECHA

JUNIO 2020

1 TÍTULO Y RESUMEN

1.1 Título

Caracterización y validación de algoritmos de visión artificial para la resolución en tiempo real de problemas en robótica autónoma móvil.

1.2 Resumen

Este trabajo fin de grado se enmarca dentro de la línea de desarrollo de robótica móvil del Grupo Integrado de Ingeniería (GII) de la UDC. Con el avance de los algoritmos de visión por computador, los robots móviles pueden hacer uso de información avanzada del entorno en tiempo real de cara a lograr resolver problemas más complejos. Pero este tipo de algoritmos requieren un uso elevado de recursos computacionales, que, en el caso de robots móviles de operación en tiempo real, deben ser optimizados. En este TFG se realiza la caracterización y validación de estos algoritmos optimizados, con el objetivo de proporcionar a los usuarios finales del robot móvil, un conjunto de indicaciones sobre sus propiedades de uso tales como rangos de funcionamiento, precisión en la detección en función de la posición o la iluminación, y otros. En concreto, se analizan tres algoritmos de visión ya implementados: detección de objetos, detección de marcadores avanzados (ArUco) y detección de carriles. Para ello, se definen pruebas de medición de la respuesta de los algoritmos y se desarrollan programas de control autónomo en el lenguaje Python que permiten realizar estas pruebas. Asimismo, se han documentado apropiadamente los resultados de los análisis realizados.

1.1 Título

Caracterización e validación de algoritmos de visión artificial para a resolución en tempo real de problemas en robótica autónoma móbil.

1.2 Resumo

Este traballo fin de grao enmárcase dentro da liña de desenvolvemento de robótica móbil do Grupo Integrado de Enxeñaría (GII) da UDC. Co avance dos algoritmos de visión por computador, os robots móbiles poden facer uso de información avanzada da contorna en tempo real para lograr resolver problemas máis complexos. Pero este tipo de algoritmos requiren un uso elevado de recursos computacionais, que, no caso de robots móbiles de operación en tempo real. Neste TFG realízase a caracterización e validación destes algoritmos optimizados, co obxectivo de proporcionar aos usuarios finais do robot móbil, un conxunto de indicacións sobre as súas propiedades de uso tales como rangos de funcionamento, precisión na detección en función da posición ou a iluminación, e outros. En concreto, analízanse tres algoritmos de visión xa implementados: detección de obxectos, detección de marcadores avanzados (ArUco) e detección de carrís. Para iso, defínense probas de medición da resposta dos algoritmos e desenvólvense programas de control autónomo na linguaxe Python que permiten realizar estas probas. Así mesmo, documentáronse apropiadamente os resultados das análises realizadas.

1.1 Title

Characterization and validation of artificial vision algorithms to solve real-time problems in autonomous mobile robotics.

1.2 Summary

This final degree project is part of the mobile robotics development line of the Integrated Group for Engineering Research (GII) of the UDC. With the advance of computer vision algorithms, mobile robots can make use of advanced information from the environment in real time in order to solve more complex problems. But this type of algorithms requires a high use of computational resources, which, in the case of mobile robots in real-time operation, must be optimized. This project performs the characterization and validation of these optimized algorithms, with the aim of providing the final users of the mobile robot, with a set of indications about their use properties such as operating ranges, accuracy in detection as a function of position or lighting, and others. In particular, three vision algorithms already implemented has been analyzed: object detection, advanced marker detection (ArUco) and lane detection. For this purpose, tests to measure the response of the algorithms have been defined and autonomous control programs have been developed in the Python language to perform these tests. Also, the results of the analyses performed have been properly documented.

UNIVERSIDADE DA CORUÑA

Escola Politécnica Superior

TRABAJO FIN DE GRADO

CURSO 2019/20

CARACTERIZACIÓN Y VALIDACIÓN DE ALGORITMOS DE VISIÓN ARTIFICIAL PARA LA RESOLUCIÓN EN TIEMPO REAL DE PROBLEMAS EN ROBÓTICA AUTÓNOMA MÓVIL

Grado en Ingeniería en Tecnologías Industriales

Documento

MEMORIA

Tabla de contenido de la memoria

Tabla de Figuras

Índice de Tablas

2 OBJETIVOS

Este Trabajo Fin de Grado tiene como objetivo principal *la caracterización y validación de tres algoritmos de visión artificial desarrollados para su uso en tiempo real en un robot móvil*.

Para lograr este objetivo principal, se han planteado los siguientes subobjetivos:

1. Conocer el robot móvil definido por el Grupo Integrado de Ingeniería de la UDC (GII) a nivel de características hardware, software, programación y campo de uso.

2. Establecer una metodología de pruebas adecuada al tipo de algoritmo a analizar.

3. Caracterizar el algoritmo de detección de objetos en tiempo real en el robot y documentar las conclusiones del análisis técnico.

4. Caracterizar el algoritmo de detección de carriles en tiempo real en el robot, implementar un sistema de control para que se mantenga dentro de los mismos, y documentar las conclusiones del análisis técnico.

5. Caracterizar el algoritmo de detección de marcadores ArUco en tiempo real en el robot y documentar las conclusiones del análisis técnico.

3 ESTADO DEL ARTE

3.1 Introducción

El presente TFG se enmarca en el campo de la robótica autónoma móvil, más concretamente, en el subcampo de la robótica educativa. La robótica educativa se centra en el diseño, análisis y aplicación de robots en el ámbito pedagógico, con el objetivo de fomentar y enseñar esta disciplina a todos los niveles educativos, desde infantil hasta posgrados. La robótica educativa [1] trata de despertar el interés de los estudiantes por todas aquellas asignaturas más tradicionales al crear un entorno propicio donde se promueva el desarrollo de habilidades y conocimientos basados en ciencia, tecnología, ingeniería y matemáticas, además de incentivar la cohesión de grupo y la capacidad de reflexión y resolución de problemas. Utiliza kits y materiales comerciales y se centra más en el uso de cibernética (sensores y motores) para promover competencias de las llamadas disciplinas STEM (*Science, Technology, Engineering, Mathematics*).

La robótica educativa tiene su origen en los años 90, cuando comenzaron a crearse ciertos dispositivos de forma local con fines educativos para ser usados en talleres para alumnado de educación primaria gracias en parte a la aparición del *IBM Personal Computer XT*, uno de los primeros ordenadores personales. Sin embargo, no es hasta el año 2000 cuando la robótica educativa surge como herramienta de formación de la mano del MIT (*Massachusetts Institute of Technology*), el cuál estableció un convenio con la compañía LEGO para que construyeran piezas que permitieran una integración con elementos de programación para que así los niños pudieran construir y programar dispositivos tecnológicos capaces de realizar ciertas acciones. Para ello, se trató de integrar las piezas de LEGO con el lenguaje de programación Logo, que había surgido a principios de la década de los 60 como uno de los primeros lenguajes en el ámbito educativo. Más adelante, en 2007, el MIT desarrolló Scratch, lenguaje de programación visual para enseñar a programar a niños mayores de ocho años. Actualmente Scratch se encuentra en su versión 3.0, desarrollada en 2018.

Actualmente vivimos en la época del Big Data. La enorme cantidad de datos en tiempo real de los que dispone sumado a los enormes avances de las últimas décadas en cuanto Hardware, permiten la ejecución de potentes algoritmos de visión por computador, aprendizaje automático o reconocimiento de habla. Los robots usados actualmente en la robótica educativa pecan por norma general de no tener una capacidad de cómputo suficiente para correr estos algoritmos, sumado al hecho de no contar con la mínima sensorización que permita al robot recibir los datos que estos algoritmos necesitan para trabajar (cámaras para visión artificial, micrófonos para reconocimiento de habla). Además de todo esto, es necesario adaptar su uso para que el alumno, del nivel educativo que sea, pueda comprender su funcionalidad sin necesidad de tener conocimientos técnicos avanzados.

Por dicho motivo, los robots que se utilizan en las aulas actualmente no permiten un acercamiento real de los alumnos a los usos y proyectos que se realizan en el mundo real. Se debería pues, tender hacía el uso de todas estas herramientas como forma de permitir a los estudiantes aprender con robots capaces de manejar estos modernos algoritmos y sin la necesidad de cursar titulaciones especializadas como ingeniería electrónica o informática.

Realizando un recorrido por los distintos robots educativos, nos encontramos como la opción más popular el *Lego Mindstorms* [2], **Figura 1**Figura 1. ***Lego Mindstorm EV3.***, un robot combinable con las piezas Lego y que en su última versión (EV3) cuenta con un procesador AMR9 con sistema operativo basado en Linux, 16 MB de memoria Flash y 64 MB de RAM, miniSDHD de 32 GB, altavoz de

alta calidad y pantalla de alta resolución de 178x178 pixeles, además de estar equipado con un giroscopio y un sensor infrarrojo. El precio medio de un kit es de 350-400 €. El problema con el EV3 viene por su poca escalabilidad por encima de la educación de Bachillerato, ya que, aunque existen opciones de programación más allá del lenguaje propio basado en bloques, como Robot-C, estas no son opciones 'oficiales' ni están integradas en un mismo IDE. *Lego Mindstorms* no cuenta con sensorización avanzada como cámaras o láser, y, además, no soporta algoritmia compleja como reconocimiento de objetos o habla al no contar con la capacidad de cómputo necesaria para ello.

Figura 1. *Lego Mindstorm EV3.*

Continuando en la educación secundaria, otro robot muy popular es el mBot [3],

Figura 2**,** un robot educativo ideal para iniciarse en la programación y la robótica, que está basado en un Arduino Uno y que utiliza conectores RJ25 por lo que no es necesario soldar o realizar cableado. Se puede programar usando el software mBlock basado en Scratch 2.0, o directamente en Arduino, en cuyo caso necesitaremos utilizar las librerías de Makeblock. Este robot utiliza un micro-controlador Atmega328 y posee conexiones para dos motores y 4 sensores. Utiliza sensores de luz, infrarrojos, led RGB y ultrasonido, y cuenta con conectividad Bluetooth y un peso del orden de los 400 gramos. El mbot es un robot que ronda los 80 € y no está pensado para soportar sensores avanzados, cámaras o ejecutar algoritmos complejos de reconocimiento de habla o visión por computador debido a la simplicidad de su Hardware.

Figura 2. *Robot mbot.*

En la línea de los anteriores robots, nos encontramos al Thymio [4], **Figura 3**, un robot que se han venido usando desde hace un tiempo tanto en la educación secundaria como en la universitaria. Se fundamenta en Aseba, un conjunto de herramientas que permiten a los usuarios sin experiencia programar robots de manera fácil y eficiente, permitiendo una programación visual, de texto o en bloque. De forma adicional, se está desarrollando un entorno de programación en Scratch para el Thymio-II. Por la parte del Hardware, cuenta con 9 sensores infrarrojos, 1 acelerómetro de tres ejes, 1 termómetro, 1 micrófono, 2 motores DC y un altavoz. A pesar de contar con un mejor Software que los modelos anteriores y un precio cercano a 120 €, su Hardware sigue sin permitir el uso de algoritmos o técnicas complejas por lo que su aplicación en proyectos avanzados de robótica móvil queda descartada.

Figura 3. *Thymio II.*

Como se puede observar, para poder ejecutar algoritmos avanzados como los ya nombrados anteriormente (visión por computador, reconocimiento del habla, aprendizaje automático) es necesario contar con no solo un buen entorno de programación adaptado a distintos niveles educativos, sino además una sensorización amplia y potente, y controladores con potencia de cómputo suficiente. Estos robots ya existen en el mercado, pero cuanto más nos acercamos a estas capacidades técnicas, más se encarece el precio. Por ejemplo, del orden de los 500-600 € tenemos dos robots móviles muy usados. El primero es el TurtleBot [5], que es de código abierto y permite la programación en ROS (*Robot Operating System*). Ha sido creado por *Open Robotics* basándose en el modelo *Turtle* de Logo, lenguaje de programación creado en 1967 y del que ya se habló anteriormente. Actualmente, el robot se encuentra en su tercera versión, TurtleBot3, que cuenta con un sensor de distancia de 360 grados (LIDAR) y utiliza como tecnología central SLAM (localización y mapeo simultáneos) para construir un mapa y conducir al robot por entornos de interior. Además, cuenta con la capacidad de control remoto desde portátil o *Smartphone* basado en Android. En la **Figura 4** podemos ver el modelo Burger, una de las tres versiones que ofrece esta nueva actualización. Su limitación viene de no poder usarse fuera de la enseñanza universitaria debido a la complejidad y conocimientos técnicos que se requiere, y a que su precio ronda los 500 € como ya se dijo anteriormente. Además, posee un pobre desempeño en tareas como reconocimiento de voz o interacción humano-robot al no tener sensores específicos para estas tareas.

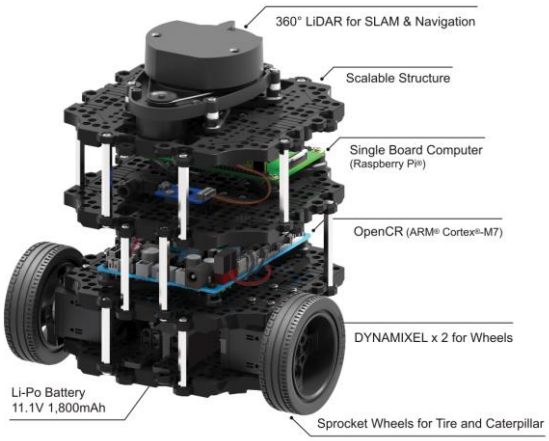

Figura 4. *TurtleBot 3, modelo Burger.*

Otro modelo de robot en este margen de precio es el e-puck [6], el cual permite la programación en distintos lenguajes como Python, C++ y una máquina virtual que corra ASEBA. Su última versión, el e-puck2, cuenta con WIFI y carga USB, un microcontrolador STM32F4 y una mayor cantidad de sensores, entre ellos: IR, sensor de distancia, cámara, 4 micrófonos digitales y almacenamiento microSD. Su procesador permite ejecutar algoritmos más potentes que el que utilizaba el e-puck original, pero sigue sin ser capaz de correr aquellos que son usados en la robótica del mundo real. En la **Figura 5,** se muestra el aspecto del robot.

Figura 5. *Robot e-puck.*

Llegamos finalmente a aquellos robots con capacidad para correr este tipo de algoritmos, donde el precio aumenta mucho más. Por ejemplo, el Khepera IV [7], **Figura 6**, cuenta con un entorno completo en Linux y se puede programar en C/C++. Cuenta con herramientas para realizar pruebas de navegación, inteligencia artificial, sistemas multi-agentes o programación en tiempo real. La nueva generación de Khepera cuenta con cámara a color, WIFI, conectividad Bluetooth, acelerómetro, giroscopio y sistema de odometría. Aunque su uso es muy popular en grupos de investigación y universidades, al igual que sucedía con el TurtleBot, su uso no es adecuado para secundaria y su precio ronda los 2.500 €.

Figura 6. *Vista superior e inferior del Kephera IV.*

Por último, en la gama alta encontramos al NAO [8], un robot humanoide de 58 centímetros desarrollado en 2008 por *Softbank Robotics* con la capacidad de interactuar de forma natural con todo tipo de público. NAO cuenta con dos cámaras, cuatro micrófonos, nueve sensores táctiles, dos de ultrasonido, ocho sensores de precisión, un acelerómetro y un giroscopio. Además, posee 53 LEDs RGB, un sintetizador de voz y dos altavoces. El robot también cuenta con suficiente poder de procesamiento para ejecutar los algoritmos más actuales al tener dos Intel ATOM de 1,6 Ghz en la cabeza y el torso. Además, mecánicamente, NAO posee 25 grados de libertad, lo que lo dota de un gran lenguaje corporal y humaniza más al robot. Por la parte software, posee un IDE llamado *Choreographe* compatible con Windows, MAC y Linux, y para usuarios avanzados se puede programar en C++, Python, JAVA.NET y MATLAB además de en ROS (*Robotic Operating System*). El problema de este robot es que su precio ronda los 6000€, lo que lo hace excesivamente caro para ser usados en centros escolares, y la mayoría de los centros educativos. En la **Figura 7** se muestra al robot NAO.

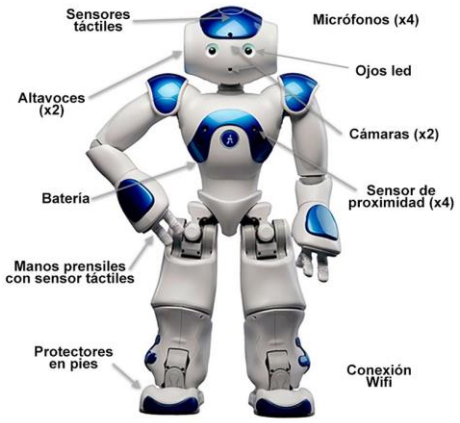

Figura 7. *Robot humanoide NAO con sus distintas partes.*

Para encontrar una forma de conseguir un robot móvil con suficiente capacidad de cómputo a un precio asequible, surgió la idea usar una base móvil a la que se acopla un Smartphone como elemento central de sensorización, comunicación y procesado, para ejecutar todos estos algoritmos de alto nivel. Una de las grandes ventajas de combinar una base móvil y un smartphone, es alargar la durabilidad del robot gracias a la actualización constante de Hardware y Software de los dispositivos móviles. Con esta idea en mente, el primer robot basado en smartphone fue el ROMO [9], que consistía en una plataforma móvil con ruedas compatible con IOS. Romo permitía una programación básica y estaba enfocado a la educación

primaria, lo que lo hacía no apto para ser usado en la investigación. Seguidamente se fabricaron otros robots de este tipo, como el WHEELPHONE [10], con un enfoque opuesto al anterior, centrado en su uso para nivel universitario e investigación, lo cual limitaba su uso en todas las etapas educativas.

Siguiendo por esta línea, *Shape Robotics* creó Fable, un sistema de construcción modular que los estudiantes pueden usar para crear su propio robot en solo unos minutos. El robot puede ser programado desde una programación visual y sencilla usando *Blocky*, similar a Scratch, para un nivel educativo de secundaria y primaria, y una programación en texto con Python para un nivel más avanzado. Fable puede ser adquirido en kit o comprar distintas piezas modulables de forma individual. Una de las ventajas de Fable es que es ampliable con piezas de LEGO o impresión 3D y su modularidad de piezas intercambiables permite convertir a tu Fable en un robot andante, una catapulta o un robot social. Fable no utiliza el procesador del Smartphone para el análisis de imagen, por lo que siempre depende de un ordenador externo. Su precio ronda los 600 € para el módulo de unión base, **Figura 8**, el cual cuenta con un dos fuertes servomotores, un puntero laser y un soporte para el *smartphone*. Si nos vamos a kits más completos nos movemos por los 1000 € para el kit medio y 5600 € para el llamado kit de clase.

Figura 8. *Fable Explorer.*

En resumen, tras realizar una revisión de aquellos robots más usuales en la robótica educativa, podemos observar que este es un campo de gran relevancia en la actualidad, pero no existe ninguna plataforma que sea utilizable desde secundaria hasta la universidad para la enseñanza de la robótica autónoma actual, es decir, que cuente con sensores potentes y variados y tenga una capacidad de cómputo capaz de correr los más modernos algoritmos. Es en este punto donde entra el robot *Robobo* que utilizaremos en este Trabajo de Fin de Grado.

3.2 Robobo

MINT S.L., una spin-off de la UDC, ubicada en A Coruña creó Robobo [11] en el año 2016, un robot educativo móvil que utiliza un Smartphone como centro de procesamiento. En la **Figura 9** se puede observar el aspecto general del Robobo, con una base móvil y un soporte para colocar el smartphone. Este robot será el empleado para la realización de este Trabajo de Fin de Grado, ya que soporta la ejecución de los algoritmos avanzados de visión por computador que se han venido comentando en el apartado anterior, y que se propone caracterizar y validar.

Para entender bien el funcionamiento del Robobo y en qué se diferencia del resto de plataformas móviles con Smartphone, vamos a proceder analizar su Hardware y su Software.

Figura 9. *Robobo.*

3.2.1 Hardware Robobo

El Hardware del robot está compuesto por el de la base y el del Smartphone, como se detallará a continuación.

3.2.1.1 Base

Toda la electrónica de la base Robobo [12] ha sido diseñada encima de una placa de circuito impreso (PCB) con la forma específica del robot, para acomodar encima todos los componentes electrónicos, sensores y actuadores, como veremos a continuación. Sus principales componentes electrónicos son un Micro-controlador basado en un *PIC32 microcontroller*. Posee una CPU que funciona a 80MHz, siendo suficiente para controlar los sensores, actuadores y llevar a cabo tareas diversas que el PIC debe realizar. Los cálculos complejos y de alto costo computacional son realizados por el teléfono inteligente mientras que la base se encarga de realizar cálculos y operaciones simples, como ajustes de motor-codificador o lecturas de batería.

La base posee la capacidad de moverse gracias a dos motores DC acoplados a dos ruedas alojadas en sus laterales, como se puede observar en la **Figura 9**. Además, cuenta con 4 sensores infrarrojos en la parte delantera y 3 en la trasera, leds RGB y un módulo Bluetooth para conectar base y smartphone. Además, también presenta otros dos motores DC, uno para girar el soporte del smartphone y otro para inclinarlo (PAN y TILT). Los 4 motores tienen un eje codificador

magnético de 6 polos para medir con precisión las revoluciones del motor. Además, están controlados por una señal PWM desde el Microcontrolador PIC.

En cuanto a su sistema de alimentación, utiliza una batería LiPo de 2500 mAh y 3.7 V montada sobre la base. El rango de voltaje de estas baterías es 4.2V-3.0V. Para poder alimentar al microcontrolador se utiliza un convertidor Buck-Boost para obtener los 3.3V que requiere. Por su parte, la batería necesita 7 V para funcionar, por ello se usa un convertidor elevador. Finalmente, hay un administrador de batería en la placa del circuito para cargarla a través de un puerto micro-USB. Todos estos sensores y Hardware se observan con detalle en la **Figura 10**.

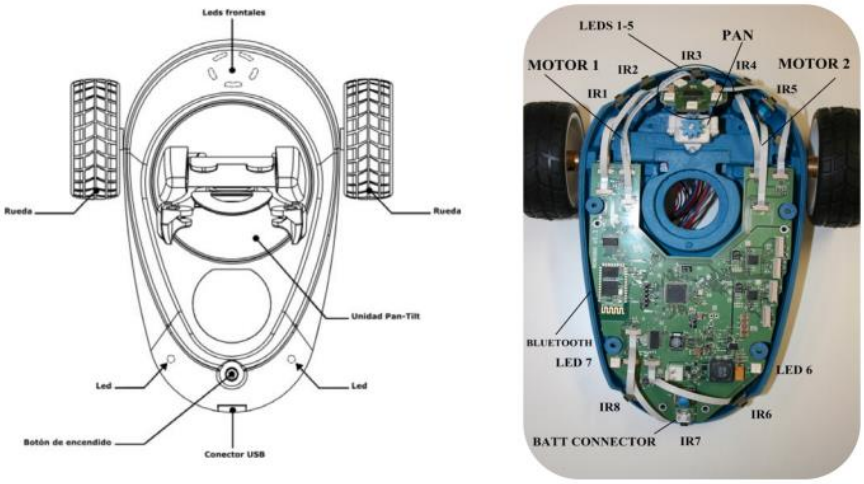

Figura 10. *Base del Robobo abierta mostrando elementos Hardware principales.*

3.2.1.2 Smartphone

Actualmente, Robobo es compatible únicamente con Smartphones basados en Android, con unas dimensiones máximas de 160 x 78 x 9,5 mm y mínimas de 134 x 67 x 6,5 mm. A nivel hardware, los Smartphones actuales presentan diferentes sensores, actuadores, y procesadores dependiendo del modelo. De forma general, podemos decir que aportan el siguiente hardware al robot:

- Dos cámaras de alta resolución.
- Una pantalla táctil LCD.
- Micrófono y altavoz.
- Giroscopio 3D, acelerómetro 3D y magnetómetro 3D.
- Sensores de luz, de proximidad y de temperatura.
- GPS.
- Conectividad WI-FI.
- Bluetooth.
- 3G/4G.

3.2.2 Software Robobo

Por la parte de Software, el robot se basa en el *Robobo framework*, el cual permite programarlo desde un ordenador con Windows, MAC o Linux. El *framework* está orientado en tres jerarquías dependiendo el nivel del programador: Scratch para principiantes, librerías en Python y JavaScript para usuarios intermedios, y librerías JAVA Android o ROS para avanzados. Todo es *Open Source*.

Para programar a Robobo es necesario contar con una base, un smartphone y un ordenador, todos conectados a la misma red wifi. Gracias a la App de Robobo (**Figura 11**) podemos conectar nuestro *smartphone* a la base mediante bluetooth y comenzar la programación sin depender de ningún tipo de cableado.

Figura 11. *Base del Robobo conectada mediante la App Robobo.*

El Robobo *framework* contiene métodos que permite realizar diferentes acciones sobre la base y el Smartphone, que podríamos clasificar como:

- **Funciones de movimiento:** Son aquellos métodos que permiten que el robot se desplace moviendo los motores alojados en la base, y también que gire la orientación del Smartphone moviendo los motores de la unidad PAN-TILT.

- **Funciones de interacción con el usuario:** Son aquellos métodos que permiten aplicaciones de interacción usuario-robot, como la interacción táctil, síntesis de voz, expresión de emociones o reconocimiento de sonidos.

- **Funciones de detección:** Son aquellas que permiten que el Robobo perciba información de su entorno, como los IR, o todos los métodos basados en el uso de la cámara, como la detección de colores, códigos QR o rostros.

Dentro de este último tipo de funciones de detección, MINT S.L. se ha propuesto el desarrollo de nuevos algoritmos de visión artificial. Como ya hemos dicho anteriormente, gracias al avance del Hardware y a las técnicas de aprendizaje automático, los robots móviles pueden hacer uso de una gran cantidad de información avanzada de su entorno en tiempo real. En este sentido, una de las principales ventajas de Robobo y que lo hacen único, se encuentra en el hecho de

poder ejecutar algoritmos estos algoritmos de visión en tiempo real en el propio procesador del Smartphone. Sin embargo, estos nuevos algoritmos son novedosos en el campo de aplicación de la robótica educativa, por lo que deberán ser caracterizados para conocer sus rangos de funcionamiento óptimo, y sus limitaciones.

Concretamente los algoritmos a analizar serán tres: detección de objetos en tiempo real, detección de marcadores avanzados (ArUco) y detección de carriles. Por lo tanto, el primer paso es conocer los fundamentos de la visión artificial y el aprendizaje automático.

3.3 Visión Artificial

Es un campo de la Inteligencia Artificial enfocado a que los ordenadores logren extraer información a partir de imágenes de forma automática [13], ofreciendo soluciones a problemas del mundo real. Esta disciplina utiliza como fuente de datos las variadas imágenes y videos extraídas a partir de los múltiples sensores que existen en la actualidad. Esto es, desde imágenes obtenidas por microscopios o satélites, pasando por imagen convencional, infrarrojas, rayos X, o cualquier señal que se pueda representar como una matriz de números. Se basa en distintas técnicas para preprocesar la imagen y eliminar ruido, posteriormente poder reconocer los objetos que en ella aparecen, reconstruir una escena de forma tridimensional o hacer detección de movimiento. Por ejemplo, en la **Figura 12**¡Error! No se encuentra el origen de la referencia. podemos observar cómo un algoritmo de visión artificial es capaz de realizar una segmentación de objetos en tiempo real, diferenciando y recuadrando de forma tridimensional distintos objetos de la escena como coches, señales de tráfico o personas.

La visión por ordenador puede ser abarcada desde distintas perceptivas. Desde la perspectiva de la ingeniería, busca automatizar aquellas tareas que puede realizar el sistema visual humano. Para lograrlo, es necesario que el sistema sea capaz de extraer de forma automática aquellos datos que posteriormente utilizará para analizar y generar contenido de importancia. Por otro lado, la vertiente científica de esta disciplina se centra más en analizar y estudiar todos aquellos principios teóricos que subyacen detrás de los sistemas artificiales que extraen información de las imágenes. Como comentábamos antes, los datos de una imagen pueden tomar múltiples formas, desde vídeos, vistas simultáneas de distintas cámaras, o datos multidimensionales de un escáner médico.

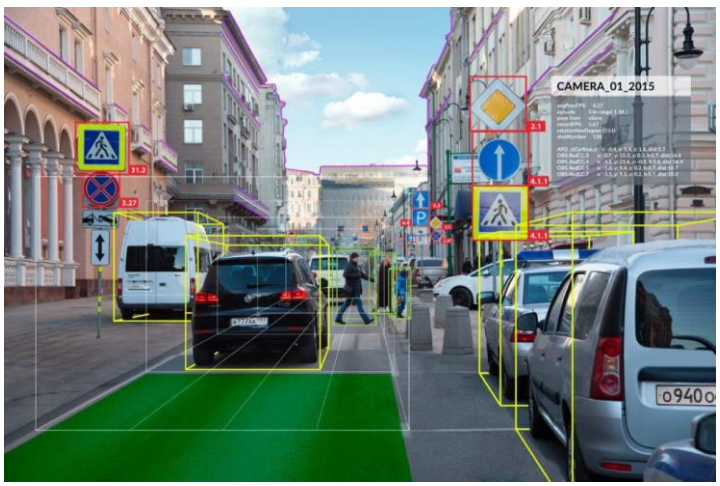

Figura 12. *Detección de objetos en una imagen con visión artificial.*

Sus usos y aplicaciones son múltiples [14]. Se utiliza habitualmente en el sector de la automoción, donde las inspecciones en fábrica, el ensamblaje y demás tareas de estampación, mecanizado o pintura dependen de la visión artificial. El sector de la electrónica es otro de los más beneficiados del uso de la visión artificial, ya que esta se ocupa de manipular e identificar los distintos componentes y comprobar de forma eficaz el correcto ensamblaje de piezas y la soldadura.

Las técnicas y algoritmos de visión artificial son muy amplias y variadas, por dicho motivo existen varias librerías que agrupan estás técnicas y simplifican su implementación y uso a unas pocas líneas de código. Una de ellas es OpenCV, que es considerada como la librería más popular de visión por computador. A continuación, hablaremos de ella por ser la utilizada en este TFG.

3.3.1 OpenCV

OpenCV (Open Source Computer Vision) [15] es una librería *open source* de visión por computador, análisis de imagen y aprendizaje automático. Fue originalmente desarrollada por Intel. Hoy en día, en 2020, se la sigue mencionando como la librería de visión artificial más popular del mundo. En el momento de redactar este Trabajo de Fin de Grado, OpenCV cuenta con la versión 4.3.0) y se encuentran desarrollando activamente interfaces CUDA y OpenCL. Muchos de los algoritmos que se validarán y caracterizaran en este Trabajo Fin de Grado utilizan esta librería, como es el caso del detector de carriles o los marcadores ArUco. Muchas de sus funcionalidades a la hora de representar imágenes son usadas para la elaboración de ilustraciones en este trabajo

La biblioteca tiene más de 2500 algoritmos optimizados, que permiten, entre otras, funciones de: detección de movimiento, identificar rostros, reconocer objetos y clasificarlos, reconstrucción 3D a partir de imágenes, y mucho más. Todo esto al alcance de unas pocas líneas de código, y con la versatilidad de ser una librería multiplataforma disponible para Windows, Mac, Linux y Android, y distribuida bajo licencia BSD, lo que permite que sea usada libremente para propósitos comerciales y de investigación. Puede programarse con C, C++, Python, Java y Matlab y además permite su ejecución en diversas arquitecturas de hardware como smartphones, Raspberry Pi o PC. Otra de sus ventajas está la extensa documentación que posee, explicada y actualizada, con ejemplos de su uso e incluso tutoriales públicos para aquellas personas no iniciadas en su uso.

```python
import cv2
faceClassif = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
image = cv2.imread('oficina.png')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
faces = faceClassif.detectMultiScale(gray,
  scaleFactor=1.1,
  minNeighbors=5,
  minSize=(30,30),
  maxSize=(200,200))
for (x,y,w,h) in faces:
  cv2.rectangle(image,(x,y),(x+w,y+h),(0,255,0),2)
cv2.imshow('image',image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Figura 13. *Script en Python usando OpenCV para la detección facial*

Muchas de las técnicas que se utilizan en visión artificial hoy en día utilizan redes neuronales basadas en *Deep Learning*, como es el caso de la imagen de la **Figura 13**, donde observamos un ejemplo de las líneas de código necesarias para

realizar una detección de rostro. ¿Pero qué es el *Deep Learning* y qué relación tiene con la visión artificial? Esas cuestiones las responderemos a continuación.

3.4 Machine Learning

El Aprendizaje Automático o *Machine Learning* [16] es un campo de estudio del área de las Ciencias de la Computación e Inteligencia Artificial. Su principal objetivo es automatizar una serie de operaciones con el fin de reducir la necesidad de que intervengan los seres humanos. Para ello, se utilizan algoritmos para procesar datos, aprender de ellos y luego ser capaces de hacer una predicción o sugerencia sobre algo. Esto puede suponer una gran ventaja a la hora de controlar una ingente cantidad de información de un modo mucho más efectivo. En la práctica, la máquina corre internamente un algoritmo que, en base a la constante entrada de datos externos en su estructura, modifica ciertos parámetros internos con el objetivo de lograr predecir escenarios futuros, o tomar acciones de manera automática según ciertas condiciones. Dado que estas acciones se realizan de manera autónoma, sin intervención humana, se dice que el aprendizaje es automático. Esta manera de resolver las cosas es muy distinta a la usada en la llamada informática clásica, donde para lograr que un sistema informático hiciera lo que el programador quería, se tenía que diseñar un algoritmo que definiera las acciones, el contexto y las reglas que se debían seguir para lograrlo.

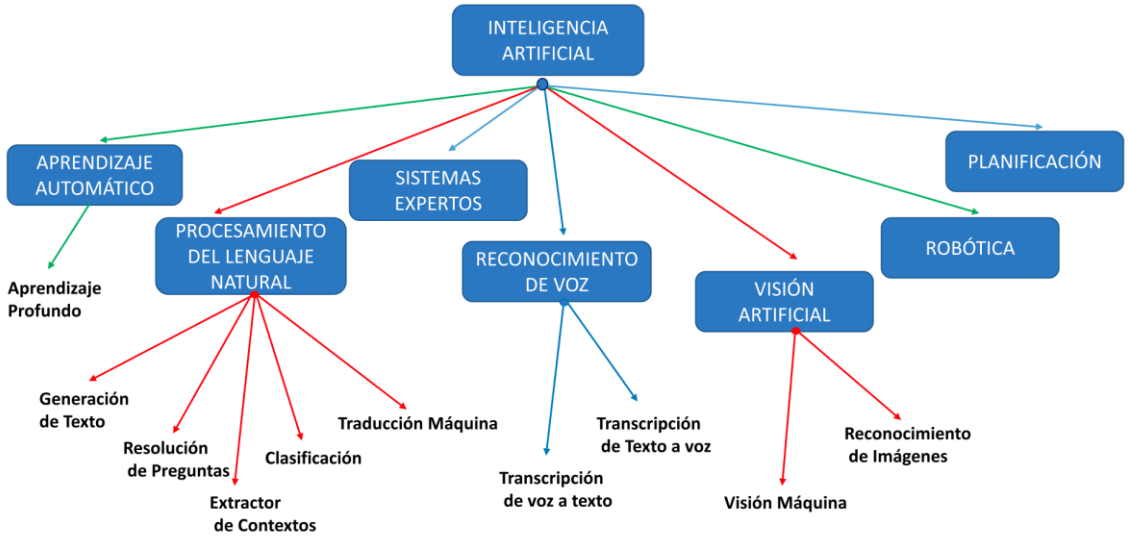

Figura 14. *Mapa conceptual de inteligencia artificial*

Como se puede observar en la imagen anterior, **Figura 14** el *Machine Learning* o aprendizaje automático es una de las ramas dentro de la inteligencia Artificial, al igual que el procesado de Lenguaje Natural (PLN) o la visión por ordenador de la cual hemos venido hablando hasta ahora. Por definición, se dice que aprender es lograr identificar una gran serie de patrones complejos a partir de una serie de parámetros de entrada, que en nuestro caso son la ingente cantidad de estímulos que recibimos gracias a nuestros sentidos. A mayor cantidad de datos, mejores serán las predicciones, y, por tanto, mejores la acciones que la máquina tome.

El *Machine Learning* posee tres principales métodos de aprendizaje: supervisado, no supervisado y por refuerzo. En la **Figura 15** podemos ver un esquema donde se muestran estas formas de aprendizaje y los principales campos de uso que se asocian con cada uno de ellos [17].

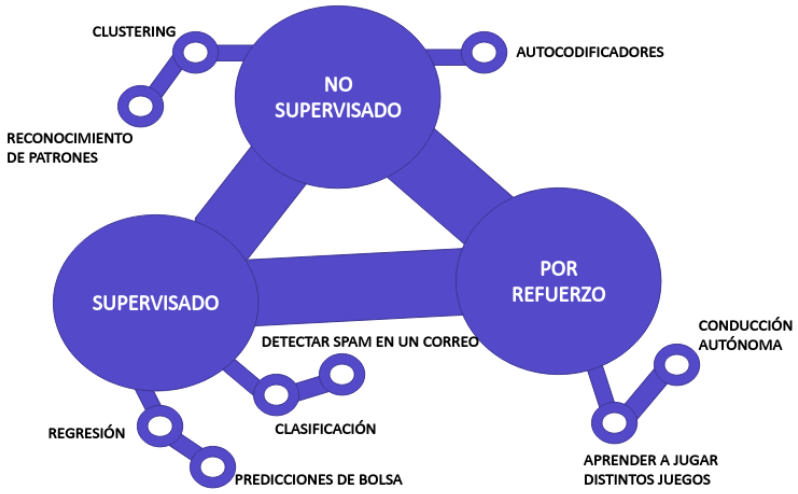

Figura 15. *Principales métodos de Machine Learning.*

- El aprendizaje *supervisado* son métodos guiados por un cierto conocimiento del problema, con los datos etiquetados por clases y donde el objetivo es predecir la clase de futuros datos.

- El aprendizaje *no supervisado* son métodos autoorganizativos donde no se tiene un etiquetado previo de los datos y su objetivo es determinar grupos/patrones no conocidos.

- El aprendizaje *por refuerzo* se basa en un sistema de recompensas donde el objetivo es aprender mediante la experiencia.

3.4.1 Deep Learning

El Deep Learning [18] es una rama dentro del *Machine Learning* que tiene su comienzo en 1943, cuando Warren McCUlloch y Walter Pitts crearon un modelo informático para redes neuronales llamado lógica umbral basado en matemáticas y algoritmos. Desde ese momento, las investigaciones se abrieron en dos frentes con su enfoque particular. El primero se centró en los procesos biológicos del cerebro y el otro en el uso de redes neuronales para la resolución de tareas en el área de la inteligencia artificial. En 1985, Frank Rossenblatt diseño el perceptrón, y para su construcción se fundamentó en el modelo de McCulloch Pitts, creando de esta manera la primera red neuronal artificial.

Pero ¿qué es una neurona artificial? Una neurona artificial es un modelo que trata de simular el comportamiento biológico de las neuronas reales. En la **Figura 16** se puede observar que una neurona puede ser tratada como una función matemática que recibe unos datos de entrada (inputs) con los cuales realiza una serie de operaciones y produce unos datos de salida (outputs). Internamente, como se ve en la imagen, se realiza una suma ponderada con los datos de entrada y unos valores llamados pesos, los cuales son unos parámetros autoajustables que validarán la importancia que debe tener cada dato de entrada en la suma ponderada. A esta suma se le añade un parámetro “b” que suele llamarse parámetro de sesgo (en inglés bias), ya que controla la predisposición de una neurona a dar su salida. La ecuación descrita anteriormente es $f(x1, x2) = b + 1\ x1 + w2x$.

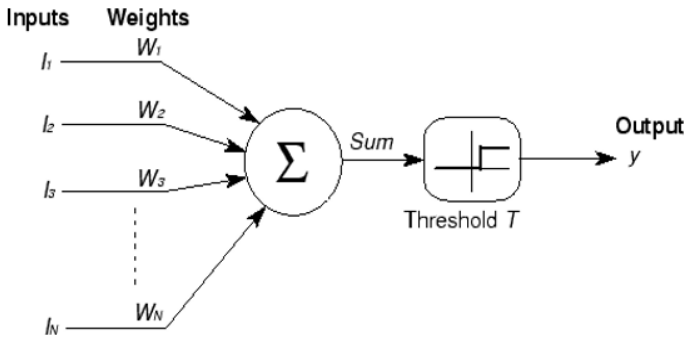

Figura 16. *Modelización matemática de una neurona biológica.*

Tras el sumatorio, el resultado atraviesa un bloque designado como *Threshold*. Esto es lo que se conoce como función de activación [19] y posee una gran importancia para el correcto funcionamiento de la neurona artificial. En las neuronas biológicas, si el estímulo supera un cierto valor umbral que viene predefinido, entonces la neurona se dispara, en caso negativo no. Las funciones de activación realizan esa misma tarea. Son funciones matemáticas como la *tanh*, la sigmoide o la *ReLU* que actúan a modo de filtro dejando solo pasar ciertos datos que se adecuen a su criterio e introduciendo una no linealidad a la salida. Por ejemplo, cómo se puede observar en la **Figura 17**, diferentes funciones de activación decidirán en qué medida se activa la función de salida. Como se ve en la imagen, La función tangente hiperbólica solo deja pasar los valores entre -1 y 1 y la sigmoide entre 0 y 1. Por su parte, la *ReLU*, que es una variación de una función lineal pero truncada por debajo de 0, dejará pasar todo valor positivo siguiendo ese comportamiento lineal y descartará todo valor menor que 0. Las activaciones no lineales permiten que las redes neuronales se combinen entre sí de maneras mucho más complejas y modelen sistemas que de otra manera no podrían ser modelados.

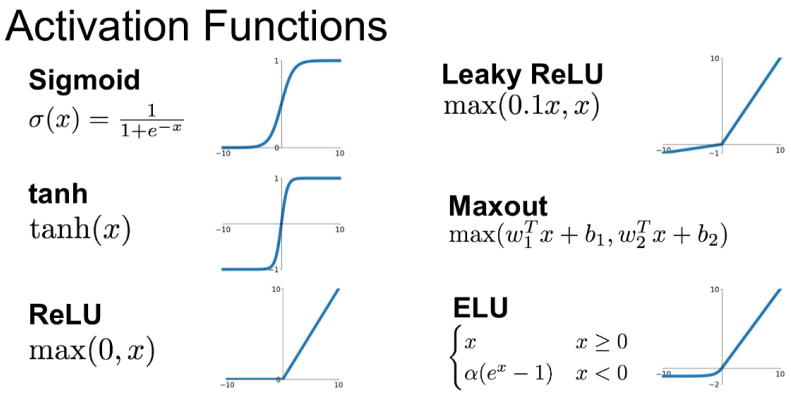

Figura 17. *Ecuación y forma de distintas funciones de activación.*

A pesar del logro de esta neurona artificial, Con el esquema de neurona que se acaba de explicar, su funcionalidad resulta muy reducida al menos que se combina con otras neuronas formando lo que se conoce como "red neuronal". Las redes neuronales [20] se dividen en 3 partes:

- *La capa de entrada*, que contiene la entrada de datos a nuestro sistema.
- *Las capas ocultas*, llamadas así por no ser accesibles a ellas más allá de la capa de entrada. Pueden contener cientos o miles de neuronas.
- *La capa de salida*. Esta capa es clave ya que es la que expende la información de salida de nuestra red neuronal. Su clave radica en el uso de las funciones de activación y de saber elegir correctamente las activaciones

que más se adecuan al problema enfrentado. Por poner un ejemplo, si estamos tratando de realizar una clasificación binaria, la función sigmoidal será la adecuada ya que da el valor de salida entre 0 y 1, significando el valor porcentual de confianza de la clasificación.

Originalmente, las redes clásicas, también llamadas perceptrones multicapa clásicos, contenían una única capa oculta lo que limitaba enormemente su desempeño y potencial. A causa del libro "Perceptrons", publicado por el MIT en 1969, la investigación en el área del *Machine Learning* estuvo parada cerca de 12 años, lo que se conoce como 'el invierno de la inteligencia artificial'. El libro argumentaba que el cálculo necesario para aplicar el Perceptrón a redes multicapa excedía la capacidad de computación de la época al requerir de un número desorbitado de operaciones. Sin embargo, gracias de nuevo al propio MIT, el que en 1985 publicó un trabajo acerca del mecanismo de '*backpropagation*' o retro-propagación, la posibilidad de utilizar más de una capa oculta se volvió una realidad. Con el paso del tiempo se ha ido buscando "más profundidad", es decir, mayor número de capas ocultas y de neuronas. Este fue el principio del *Deep Learning* (aprendizaje profundo).

El algoritmo de retro-propagación que hizo posible esto consistía en recorrer una primera vez la red con los pesos inicializados de forma aleatoria y comparar el resultado de salida con el real para calcular de esta manera el error producido. A continuación, la red era recorrida de forma inversa (retro-propagación) ajustando los valores de los pesos mediante el uso de algoritmos de optimización como el descenso del gradiente, el cual permite converger hacia el valor mínimo de una función mediante un proceso iterativo. Este proceso se repetía hasta que hubiera una convergencia y se produjera un margen de error aceptable entre los valores reales y predichos. Este proceso de propagación hacia delante y detrás ajustando los pesos hasta reducir el error del efecto esperado es lo que se conoce como *entrenamiento*. En la **Figura 18** podamos observar de forma gráfica la diferencia entre una red neuronal sencilla con una capa de entrada (roja), una sola capa oculta (amarillo) y una capa de salida (azul) y una red neuronal profunda, con 4 capas ocultas.

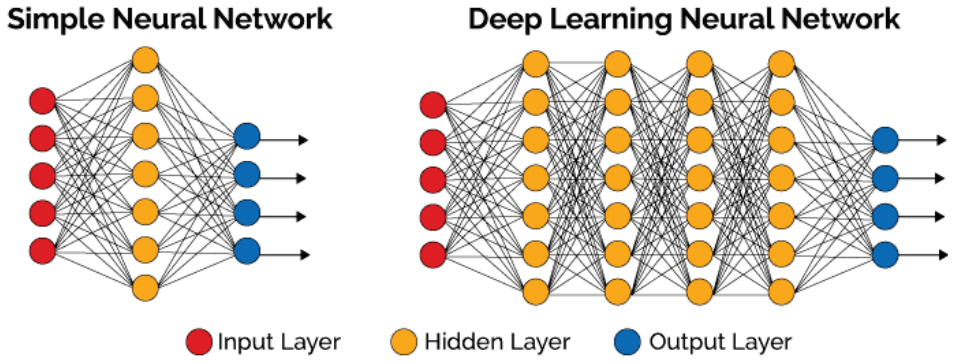

Figura 18. *Comparación de la Arquitectura de una red neuronal simple con una profunda.*

A pesar del increíble avance en esta tecnología, se produjo un segundo invierno entre finales de los 80 hasta entrados los 2000, momento donde el aumento de la potencia de cómputo y la gran abundancia de datos volvió a lanzar esta tecnología. El término Deep Learning fue acuñado en 2006 por Geoffrey Hinton y propone que las máquinas sean capaces de entender el mundo a través de una jerarquía de conceptos, de manera que cada concepto se defina a través de sus relaciones con conceptos más simples. Con esta enorme jerarquía, se intentan modelar

abstracciones de alto nivel en forma de datos usando arquitecturas computacionales que parten de datos expresados en forma matricial o tensorial.

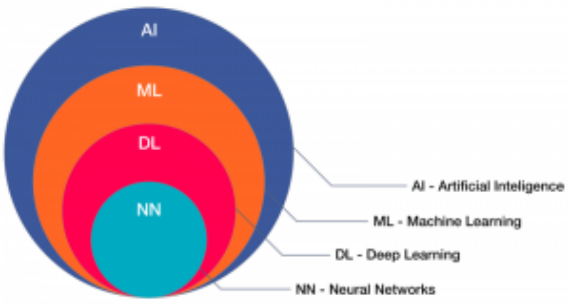

Figura 19. *Esquema de clasificación de la inteligencia artificial.*

Por resumir, en la **Figura 19** podemos observar cómo las redes neuronales actuales se encuentran englobadas dentro del Deep Learning y este a su vez dentro del *Machine Learning*, que junto con la rama de la visión artificial que ya hemos visto se engloban en la Inteligencia Artificial. Como acabamos de ver, el concepto del 'Deep Learning' está muy ligado a las Redes neuronales artificiales (RNA). Sin embargo, hoy en día no va de la mano de la perspectiva neurocientífica, sino que sigue un procedimiento que trata de entender la realidad mediante una composición de conceptos. En esencia, una red neuronal profunda se comporta como una enorme caja negra con salidas y entradas, donde sus parámetros se ajustan en base a un algoritmo de aprendizaje profundo que aprende relaciones entre dichas entradas y salidas. El descubrimiento y reconocimiento de patrones en el mundo que nos rodea es un factor fundamental en los progresos científicos y tecnológicos actuales.

En el presente documento se usará un tipo concreto de modelo Deep Learning, las redes convolucionales. Una red neuronal convolucional (CNN), es un tipo de Red Neuronal Artificial con aprendizaje supervisado que procesa sus capas imitando al córtex visual del ojo humano para identificar distintas características en las entradas, que hacen que pueda identificar objetos y clasificarlos. Para realizar esto, las CNN utilizan una jerarquía en la organización de sus capas ocultas. Las primeras capas son capaces de detectar cambios bruscos de contraste, líneas y conceptos generales de la imagen, las siguientes capas se especializan en detectar detalles más específicos u abstractos de la imagen, como una textura o formas complejas, y finalmente, la red es capaz de detectar un rostro o la silueta de un animal. Estas redes se dividen en dos partes bien diferenciadas:

1. **Extracción de características**: Esta es la parte inicial y está compuesta principalmente por neuronas convolucionales. Cuanto más se avanza a través del número de capas convolucionales menos reaccionan estas ante la variación de los datos de entrada y mayor es la abstracción alcanzada por las mismas para reconocer formas más complejas.

2. **Clasificación**: Se basan en la utilización de capas "densas" formadas por neuronas convencionales, similares a las utilizadas por los modelos de tipo "perceptrón".

Al igual que pasaba con la librería de OpenCV en la rama de la visión artificial, para poder trabajar con toda la matemática, técnicas y métodos de optimización que requieren el *Machine Learning* y el Deep Learning, se creó Tensorflow, del cual hablaremos a continuación.

3.4.2 Tensor Flow

Tensor Flow [21] es una biblioteca de código abierto diseñada para aprendizaje automático y construida por el equipo de Google, Google Brain para uso interno. Tensor Flow es la evolución de DistBeief, un sistema basado en redes de aprendizaje profundo desarrollado en 2011. Sin embargo, este sistema poseía ciertos problemas en cuanto rapidez y flexibilidad, problemas que su predecesor solucionó. Posteriormente fue liberada bajo licencia de código abierto Apache en 2015.

Como su nombre indica, Tensor Flow (flujo de tensores) nos permite el cálculo de flujos de datos y gracias a su interfaz, facilita la implementación de algoritmos de aprendizaje automático de forma sencilla y en dispositivos muy diversos, como smartphones, tabletas o robots. Entre sus ventajas están el hecho de poseer una gran estabilidad, al ser capaces no solo de correr en sistemas locales sino en la nube, además de poder ejecutarse tanto en una sola máquina como en un conjunto de plataformas, y funcionar tanto en CPU como en GPU, y recientemente en TPU. Al contar con una biblioteca tan flexible, se abre la puerta a la experimentación de forma dinámica y se simplifica enormemente el diseño rápido de sistemas de aprendizaje automático. Además, Tensor Flow permite la ejecución paralela de un modelo central usando una gran cantidad de dispositivos computacionales a su alrededor interactuando para actualizar estados compartidos. Por todas estas ventajas, la librería de Tensor Flow es la plataforma más utilizada en el campo del Deep Learning.

Tensor Flow cuenta con la siguiente Wiki donde explican con detalle cómo usar dicha tecnología: (https://www.tensorflow.org/tutorials?hl=es).

En este TFG, además de redes entrenadas con Tensor Flow, se utilizará *Tensor Flow Lite* [22], un conjunto de herramientas para ayudar a los desarrolladores a ejecutar modelos Tensor Flow en dispositivos móviles, integrados y de IoT. *Tensor Flow Lite* consta de dos componentes principales:

- El intérprete: Permite ejecutar modelos especialmente optimizados en muchos tipos de hardware diferentes, incluidos teléfonos móviles, dispositivos Linux integrados y microcontroladores.

- El convertidor: Permite convertir los modelos Tensor Flow en una forma eficiente para su uso en el intérprete, y puede introducir optimizaciones para mejorar su tamaño y rendimiento.

Tensor Flow Lite pose un soporte multiplataforma, desde dispositivos Android, iOS, Linux y varios microcontroladores y una API para múltiples lenguajes como Java, C++, Swift o Python. Cuanta además con herramientas de optimización de modelos como la cuantización, que puede reducir el tamaño y aumentar el rendimiento de modelos sin sacrificar precisión.

Actualmente Tensor Flow se encuentra en la versión 2.0 [23], que cuenta con nuevas funcionalidades o mejoras como la eliminación de APIs redundantes, la sustitución de las sesiones por funciones y la llamada ejecución ansiosa, la cual elimina la necesidad de crear primeramente un grafo computacional y luego una sesión para ejecutarlo. Ahora se puede ver directamente el resultado del código sin ejecutar dicha sesión.

4 METODOLOGÍA

El presente TFG es un trabajo científico-teórico dentro de un marco de investigación. Como tal, es necesario establecer la metodología de trabajo utilizada para la realización de las pruebas, la obtención de datos y la redacción de conclusiones. Los pasos seguidos para la realización del presente documento se muestran a continuación:

4.1 Procedimiento

En este TFG se validarán y caracterizaran tres algoritmos de visión artificial que MINT SL, spin-off de la UDC, ha creado con el objetivo de mejorar las funcionalidades del robot Robobo. Los tres algoritmos son:

- Una red neuronal para la detección de objetos en tiempo real realizada en Tensor Flow.
- Un detector de carriles basado en librerías de OpenCV.
- Un detector de ArUcos (marcador fiducial) basado en librerías de OpenCV.

Estos tres algoritmos han sido adaptados por MINT para ejecutarse en un S*martphone*. Los pasos a seguir para su caracterización son los siguientes:

1. **Conocer y analizar el algoritmo:** Antes de nada, se estudiará la documentación oficial de cada uno de estos algoritmos para poder entender su funcionamiento y como ejecutarlos correctamente.

2. **Establecer rangos de uso:** Para la correcta caracterización de los algoritmos se establecerán los rangos de uso en los que se pretende aplicar, teniendo en cuenta las indicaciones de los investigadores del GII.

3. **Definir las medidas a realizar:** Se establecerán los conjuntos de medidas que se llevarán a cabo para caracterizar cada algoritmo.

4. **Realizar de las medidas:** La recolección de datos se realizará mediante el uso de scripts en Python para imprimir y guardar en documentos de texto o archivos CSV los datos devueltos por cada método en cada una de las pruebas realizadas. Todos los *scripts* usados para las pruebas se encuentran disponibles y accesibles en el presente documento.

5. **Análisis de los resultados:** Una vez los datos han sido recopilados, se procederá a la realización de gráficas y tablas, con el objetivo de poder analizar de manera sencilla y clara los resultados obtenidos.

6. **Extracción de conclusiones:** Tras haber realizado un análisis de todos los datos obtenidos para cada prueba, se redactarán unas conclusiones para cada uno de los tres algoritmos sobre sus rangos óptimos de uso, limitaciones y fortalezas.

7. **Escribir la documentación:** Tras obtener unas conclusiones que caractericen su uso y funcionamiento, se realizará una documentación de algoritmo. En dichos informes figurará una explicada sencilla acerca de cómo llamar a dicho método haciendo uso de la librería en Python de Robobo, que nos devuelve el método y un ejemplo de su funcionamiento. Esta documentación se encuentra en los Anexos del presente documento.

4.2 Planificación del TFG

Al comienzo del cuatrimestre, se realizó un esquema general de las tareas a realizar para la realización del TFG en el tiempo disponible, de febrero a junio, de la siguiente manera:

Nº Tarea	Febrero semana 1	Febrero semana 2	Febrero semana 3	Febrero semana 4	Marzo	Abril semana 1	Abril semana 2	Abril semana 3	Abril semana 4	Mayo semana 1	Mayo semana 2	Mayo semana 3	Mayo semana 4	Junio semana 1	Junio semana 2
Tarea 1															
Tarea 2															
Tarea 3															
Tarea 4															
Tarea 5															
Tarea 6															
Tarea 7															
Tarea 8															
Tarea 9															

Tabla 1. *Distribución temporal de tareas.*

- **Tarea 1**. Toma de contacto con los algoritmos de detección de objetos en tiempo real, aprender cómo funcionan y a ejecutarlos correctamente.

- **Tarea 2**. Elección de dos redes de detección de objetos para comparar con la nuestra y definición de las pruebas a realizar para caracterizar el módulo de detección de objetos.

- **Tarea 3.** Realización de pruebas de detección de objetos. La gran mayoría de ellas realizaron en el laboratorio del GII, pero otras como las pruebas en movimiento, se realizaron en mi propio domicilio a causa del estado de alarma del Covid-19.

- **Tarea 4**. Familiarización con los algoritmos usados para la detección de carriles y la librería OpenCV (detector de bordes “Canny”, escala de grises y filtrado Gaussiano, etc…).

- **Tarea 5.** Realización de la primera versión del circuito (tramo recto) y pruebas con la primera versión del script creado para el control de movimiento del Robobo.

- **Tarea 6.** Familiarización con la librería ArUco y aprendizaje de las documentaciones oficiales de la librería ArUco en OpenCV, el método creado por MINT y la App de calibración.

- **Tarea 7**. Realización de las pruebas de ArUco: pruebas de luminosidad y de precisión en distancia y ángulo.

- **Tarea 8.** Redacción de la memoria. Anteriormente algunas partes de la memoria ya habían comenzado a escribirse.

- **Tarea 9.** Corrección de detalles en la memoria y puesta a punto.

5 RESULTADOS

Los diferentes entornos de pruebas en robótica educativa en los cuales se suele utilizar el Robobo se muestran en la **Figura 20**. Esta imagen ilustra el tipo de objetos que el robot debe detectar con la cámara, tales como señales, marcadores tipo QR, objetos de color, otros robots, etc., y sirve de marco para tener una idea del campo de aplicación en el que nos encontramos.

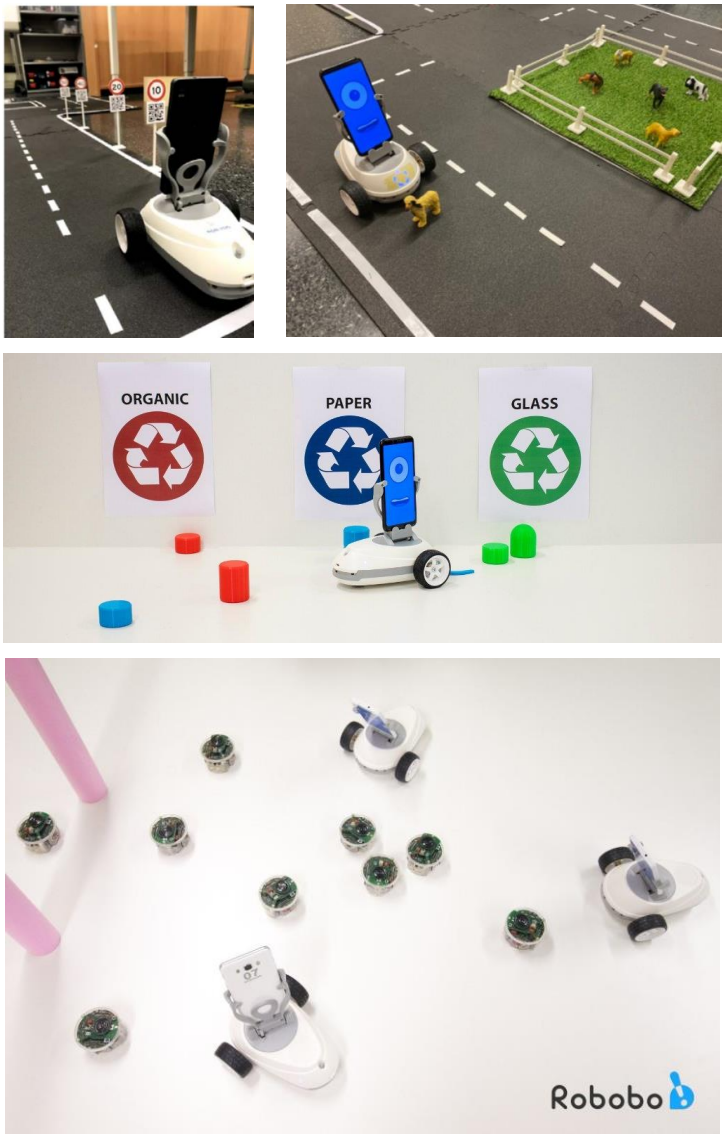

Figura 20. *Robobo realizando tareas en entornos de robótica educativa.*

Como se ha establecido anteriormente, haciendo uso de implementaciones ya existentes en Tensor Flow y OpenCV, MINT SL ha desarrollado tres algoritmos de visión artificial para ejecución nativa un *smartphone* Android. Los algoritmos son:

- Detección de objetos en tiempo real
- Detección de carriles
- Detección de Marcadores avanzados (ArUcos).

El objetivo del presente TFG ha sido validar el funcionamiento y caracterizar el uso de cada uno de los tres algoritmos antes nombrados, pero no implementarlos. Por lo tanto, en este capítulo, se mostrarán las pruebas llevadas a cabo para dicho propósito y los resultados obtenidos.

5.1 Detección de objetos en tiempo real

5.1.1 Introducción

La detección de objetos en tiempo real abre un mundo de posibilidades dentro del campo de la robótica. Permite que el robot pueda interactuar con su entorno de manera más avanzada y similar a como lo hacemos los humanos. La detección de objetos posibilita no solo conocer de qué objeto se trata sino su posición en la imagen. Poseer un módulo que permita realizar la detección de objetos es de gran importancia para cualquier robot y, por ende, para el Robobo, permitiéndole realizar tareas más avanzadas y sofisticadas.

El algoritmo para detección de objetos en tiempo real adaptado por MINT se basa en una red neuronal. Dicha red se llama MobileNet SSD V3, y es una red convolucional que, mediante *Tensor Flow Lite*, puede ejecutarse de forma nativa en un *smartphone*. La MobileNet SSD V3 es la versión más moderna hasta la fecha de la familia de redes neuronales convolucionales (CNN) MobileNet SSD.

El objetivo de las MobileNet [24] es permitir que el reconocimiento de imágenes en tiempo real pueda ejecutarse desde dispositivos de poca potencia, sin depender de un gran hardware para realizar los cálculos de forma rápida y en tiempo real. Son modelos pequeños, de baja latencia, parametrizados para cumplir con las limitaciones de recursos de una variedad de casos de uso. Las redes MobileNet se pueden construir para la clasificación, detección o segmentación de forma similar a cómo se utilizan otros modelos populares a gran escala, como *Incepcion* [25]. Por su parte, SSD son las siglas de "*Single Shoot Detection*" o detección en un solo disparo y han sido desarrollados por los equipos de investigación de Google para mantener el equilibrio entre los dos métodos de detección de objetos en tiempo real más usados, que son YOLO [26] y RCNN [27]. Hay, específicamente, dos modelos de SSD disponibles:

- SSD300: En este modelo, el tamaño de entrada se fija en 300 × 300. Se utiliza en imágenes de menor resolución, tiene mayor velocidad de procesamiento, pero es menos preciso que el SSD512. Los modelos de MobileNet SSD usados en este trabajo usan esta versión.

- SSD512: En este modelo, el tamaño de entrada se fija en 500 × 500. Se utiliza en imágenes de mayor resolución y es más preciso que otros modelos.

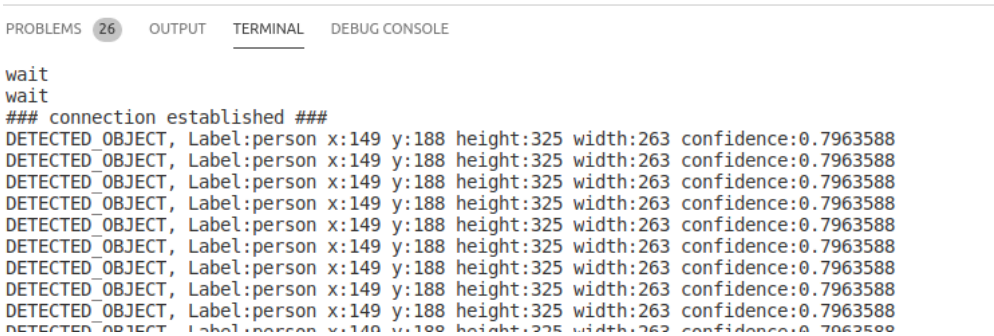

Figura 21. *Imprimiendo por pantalla lo que devuelve el método readDetectedObject*

De forma concreta, MINT ha implementado un método llamado *readDetectObject()*, (**Figura 21**) el cual devuelve un objeto que contiene la etiqueta del objeto detectado, su confianza, y la posición "x" e "y" del centro del rectángulo de detección, así como su ancho y alto. En la figura anterior, **Figura 21**, se puede observar lo que el método nos

devuelve al ser llamado. El método no requiere ningún dato como *input* y devuelve un objeto *DETECTED_OBJECT*, que contiene todos los valores antes nombrados (etiqueta, confianza, etc).

Este método de visión se encuentra disponible online con una wiki que explica su uso y funcionamiento. Para buscar dicha wiki se debe acceder al enlace que se deja a continuación y escribir en la barra de búsquedas a la derecha de la página: “readDetectedObject”. La primera entrada que arroja el buscador ya es la documentación del método:

https://mintforpeople.github.io/robobo.py/

El modelo MobileNet V3 usado no será el que se encuentra subido en el repositorio de GitHub de Tensor Flow, sino una versión reentrenada a la que se le han eliminado ciertas etiquetas y se le han añadido otras con objetos más comunes a nivel de robótica educativa, tales como señales de tráfico y otros Robobos. Para reentrenar la red, se realizó primeramente un descarte de etiquetas basado en una serie de criterios que se encuentran detallados a continuación.

5.1.1.1 Aprendizaje de una nueva red MobileNet SSD V3

La red usada (MobileNet V3) fue reentrenada eliminando ciertas etiquetas del *dataset* original, denominado COCO (http://cocodataset.org/), por no resultar interesantes desde el punto de vista de un entorno de robótica educativa donde el Robobo pudiera interactuar. Además, era importante que el Robobo pudiese detectar otros objetos más comunes a nivel de robótica educativa no existentes en COCO. Por ello, primeramente, se realizó un filtrado en base a unos criterios dados de las etiquetas originales. Estos criterios fueron:

1. Se descartarán los animales, a excepción de animales domésticos usuales como perro o gato.
2. Se descartarán aquellos objetos en general ajenos a un entorno donde el Robobo puede moverse como: "frisbee", "skis", "Tabla de Esquiar", "Balón deportivo", "cometa", "bate de beisbol", "guante de beisbol", "Tabla de skate", "Tabla de surf", "raqueta de tenis", "oso de peluche", "Secador pelo", "cepillo de dientes"
3. Al usar al Robobo en situaciones o entornos como la ciudad inteligente, es importante que conserve elementos de tráfico como: "Luz de tráfico" o “señal de stop” y ciertos medios de transporte usuales como coche, camión y bus. Algunas de estas etiquetas como los elementos de tráfico y ciertos vehículos como la motocicleta nos interesan para ser usadas en un entorno de simulación con el Robobo, pero no para la detección en un entorno real.
4. Por su entorno educativo es normal que lea objetos como ratón o teclado y otro tipo de material de una escena con alumnos, incluyendo a personas.
5. Aquellos objetos grandes como mochilas a pesar de pertenecer a un entorno educativo no nos interesan. Sin embargo, etiquetas como tazas, vasos y demás objetos de pequeño tamaño que el Robobo pueda mover, sí.
6. Podría ser interesante que detecte ciertos alimentos usuales en un aula como fruta, pero el resto de alimentos, como pizza, tarta o donut serán descartados.
7. Objetos como tostadoras, hornos, refrigeradores y de más aparatos de cocina no tienen relevancia.

Finalmente, teniendo en cuenta los criterios anteriormente mencionados, las etiquetas seleccionadas fueron las siguientes:

"Bicicleta, moto, tenedor, cuchillo, cuchara, bowl, portátil, reloj, tijeras, gato, perro, mando a distancia, teclado, libro, persona, coche, luz, señal de stop, botella, vaso de vino, taza, plátano, manzana, naranja, ratón, teléfono móvil, Florero".

A las etiquetas anteriores se les añadieron las siguientes, definidas por el equipo de investigadores de MINT:

"Robobo, semáforo, señal STOP, señal velocidad 50, señal velocidad 120, luz de tráfico verde, luz de tráfico amarilla, luz de tráfico roja".

Algunas etiquetas nuevas como luces de tráfico, moto o bicicleta han sido añadidas para ser usadas en un entorno de simulación de Robobo y fueron entrenadas con imágenes reales. Por tanto, no serán usadas para las pruebas de detección mostradas a continuación

En resume, se ha definido un conjunto total de 32 etiquetas, con el que se reentrenó una nueva red MobileNet SSD V3 más adaptada a nuestras necesidades.

5.1.1.2 Definición de las pruebas a realizar

Tras entrenar la nueva red, ya se disponía de la implementación final de la librería de reconocimiento de objetos a caracterizar. Se definieron dos grandes tipos de pruebas a realizar, con el robot *estático* y en *movimiento*. Las primeras se centraron en estudiar la precisión de la red con el robot parado, y las segundas en su velocidad de respuesta, algo de gran relevancia en robótica móvil. Además, se decidió realizar un modelado del comportamiento de la red en función de la distancia, de gran utilidad para su uso en robótica educativa.

En los apartados **5.1.2, 5.1.3** y **5.1.4** se detallan estas tres pruebas y los resultados que se han obtenido.

5.1.2. Pruebas en estático

El objetivo fundamental de estas pruebas es conocer la precisión en la detección de esta red dentro de sus rangos de funcionamiento. Como se ha utilizado una implementación basada en Tensor Flow Lite, es de gran interés para los investigadores de MINT, conocer su rendimiento en comparación con las implementaciones del estado del arte, que se ejecutan en Tensor Flow estándar.

Actualmente existe una amplia variedad de arquitecturas y modelos para la detección de objetos en tiempo real. Existen redes como la VGG19 o la ResNet que se encuentran actualmente en el estado del arte en lo que a precisión de detección se refiere. Sin embargo, para su uso en un robot se necesita que dicha red sea capaz de realizar la detección lo más rápido posible. Las redes antes nombradas ostentan la máxima precisión de detección en ImageNet, un set de datos de miles de imágenes usado para entrenar y comparar distintas arquitecturas de CNN. Sin embargo, estos modelos acostumbran a ser muy pesados y no son capaces de ejecutar una detección en tiempo real. Se busca, por lo tanto, una implementación capaz de detectar los objetos con una precisión aceptable y lo más cercano a tiempo real que se pueda conseguir. En este sentido, las dos aproximaciones más utilizadas son las redes YOLO y MobileNet. Por ello, se comparó la confianza obtenida por nuestra red con otras dos redes que se ejecutaban desde el ordenador. Estas dos redes fueron la MobileNet V2 y la YOLO V3.

La principal diferencia entre las versiones V2 y V3 es que MobileNet V3 es más rápido y más preciso que MobileNetV2 en tareas de clasificación. Esta última versión

trae mejoras en su arquitectura respecto a la V2 como el uso de *h-swish* en sustitución de la activación *swish* que usaba la V2 y que sirve para añadirle una no linealidad a la red. La *h-swish* es computacionalmente menos costosa. Esta nueva versión posee unos mAP (precisión media promedio) similares a la versión anterior, pero es de media un 25% más rápida.

Por otro lado, YOLO (You Only Look Once) [29] es un sistema para detección de objetos en tiempo real que hace uso de una red neuronal convolucional para detectar dichos objetos en imágenes. La detección de objetos en imágenes implica, no solamente identificar de qué tipo de objeto se trata, sino también localizarlo dentro de la imagen (obtener sus coordenadas en la fotografía).

Lo que distingue a YOLO de otro tipo de redes CNN usadas para detección de objetos es su velocidad, ya que su nombre son las siglas de *You Only Look Once*, lo que indica que solo necesita "ver" la imagen una sola vez para clasificarla. Esto permite realizar la detección de objetos hasta a 30 FPS (frames per second), por lo que es posible realizar la detección en tiempo real. Los sistemas de detección anteriores a YOLO reutilizan clasificadores para realizar la detección. Aplican el modelo a una imagen en múltiples ubicaciones y escalas y las regiones que logran una alta puntuación en la imagen se les considera detecciones válidas. YOLO, sin embargo, usa un enfoque totalmente distinto. Utiliza la imagen completa y la divide en cuadrículas y predice las probabilidades de cada cuadro delimitador. Mientras que redes como la R-CNN necesitan ejecutar miles de veces la evaluación de la imagen en la red para realizar una predicción, gracias a este enfoque, YOLO solo necesita realizarla una vez. Como ya se ha comentado, YOLO divide la imagen en pequeñas cuadrículas de SxS y en cada celda predice el nivel de confianza de N posibles cajas que contengan al objeto. La mayoría de las cajas producen unos niveles de confianza muy bajos. Toda caja con una probabilidad de acierto por debajo de un valor límite es descartada. A las cajas restantes se les aplica un filtrado que sirve para eliminar posibles objetos que fueron detectados por duplicado y así dejar únicamente el más exacto de ellos. Todo este proceso se puede observar esquematizado en la **Figura 22.**

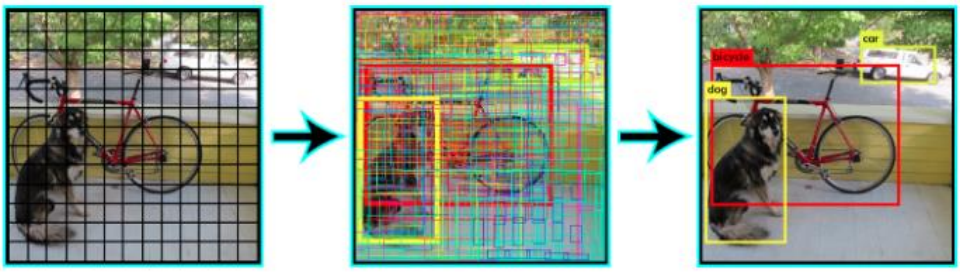

Figura 22. *Proceso de detección e identificación de objetos en YOLO.*

En este TFG se usará una red YOLO V3, la última y más moderna versión de esta red neuronal pre-entrenada con el set de datos de COCO. Este set de datos define 91 clases, pero los datos solo usan 80 de ellas. El único problema de YOLO es que sus predicciones a veces son poco fiables, ya que sacrifica precisión por velocidad.

Para caracterizar nuestra librería en estático se realizaron pruebas para la detección de todos los objetos de las etiquetas que contenía la nueva red MobileNet SSD V2, y se compararon los resultados con los obtenidos con la YOLO V3 y la MobileNet SSD V2, ambas ejecutándose en un ordenador con un procesador de octava generación Intel Core i5-8520U 60 Ghz. Por su parte, el modelo de

Smartphone usado para las pruebas en Robobo fue un HUAWEI P SMART 2019 con una CPU Hisilicon Kirin 710 y 3 GB de RAM.

Para la realización de estas pruebas se colocó al Robobo con una inclinación del soporte Tilt de 100 a excepción de la etiqueta persona, debido a su tamaño. El valor de confianza mínimo que se toma en este tipo de redes es del 30%, por lo tanto, todo objeto con un valor superior a este será tomado en cuenta en los datos a analizar.

El script usado para la obtención de los datos, pintado de imágenes y recolección de datos se encuentra disponible en el siguiente enlace de GitHub:

https://github.com/GII/robobo_vision/tree/master/Photo-Selector

5.1.2.1 Distancia fija del objeto.

Para esta prueba se colocó al *smartphone* en la base Robobo con una inclinación del Tilt de 100 y el objeto a una distancia fija de 40 cm. El objetivo de esta prueba es evaluar la precisión de detección de todo el nuevo dataset, para estudiar cómo responde nuestra red ante distintos objetos. La **Figura 23** muestra, en el lado izquierdo, la vista de forma externa, y la imagen de la derecha, el objeto visto desde el robot, con un recuadro de detección. A partir de la imagen obtenida (**Figura 23** parte derecha) se pasa a través de las tres redes obteniéndose los valores de confianza de detección para cada objeto del nuevo dataset.

Figura 23. *Ejemplo de vista externa (izquierda) y vista desde el Robobo (derecha) durante el proceso de detección de un objeto.*

Puesto que este tipo de redes de detección son muy sensibles a los cambios de orientación del objeto, se obtuvieron los valores de confianza para cada medida desde tres orientaciones distintas: de frente, de lado y desde atrás. Para una visión más clara de cómo se han tomado los objetos en las tres orientaciones, en la **Figura 24** se muestran unos ejemplos de imágenes obtenidas desde la cámara del *smartphone* en las tres orientaciones: vista frontal, lateral y trasera. Aquellos objetos simétricos como por ejemplo el “Bowl” solo serán analizados desde su única orientación.

Los resultados obtenidos se encuentran en la

Tabla 2, la cual se muestra a continuación. En dicha tabla, los objetos en color rojo son objetos que solo posee la nueva red que utiliza el Robobo, y por ello no pueden ser detectados por la MobileNet V2 y la YOLO V3. Aquellos objetos que no fueron detectados por una red son designados como “-“, mientras que si la etiqueta es errónea se coloca la etiqueta detectada en lugar de la confianza detectada.

Figura 24. *Vista frontal, lateral y trasera de los objetos Robobo (arriba) y coche (abajo).*

Etiqueta	Orientaciones	MobileNet SSD V3	MobileNet SSD V2	YOLO V3
Tenedor	De frente	0,5001	-	0,6244
	De lado	-	-	-
	Desde atrás	'florero'	'parkímetro'	'cuchara'
Cuchara	De frente	0,5085	-	0,4865
	De lado	0,3109	-	0,3416
	Desde atrás	'persona'	'parkímetro	0,3140
Bowl	De frente	0,5032	0,80416	-
Portátil	De frente	0,8286	0,884612	0,7503
	De lado	0,6286	-	-
	Desde atrás	0,5035	-	-
Reloj	De frente	0,8546	0,9587	0,9445
	De lado	-	-	-
	Desde atrás	-	'bowl'	-
Tijeras	De frente	0,5407	0,4783	"Tenedor"
	De lado	-	-	-
Gato	De frente	0,5052	"Pájaro"	-
	De lado	-	"Pájaro"	-
	Desde atrás	0,3214	"Pájaro"	-
Perro	De frente	0,5178	-	0,5930
	De lado	0,6130	"TV"	0,6532
	Desde atrás	"persona"	-	0,4819
Mando a distancia	De frente	0,7627	0,9428	0,8619
	De lado	0,5109	-	0,3304
	Desde atrás	0,5109	-	0,3304

Teclado	De frente	0,7262	0,5256	0,6045
	De lado	0,6158	0,5777	0,3341
	Desde atrás	0,5452	0,3417	0,3221
Libro	De frente	0,5496	0,343	-
	De lado	“Mando a distancia”	-	--
	Desde atrás	“Teléfono móvil”	“Teléfono móvil”	-
Persona	De frente	0,8575	0,9814	0,9124
	De lado	0,8220	0,9610	0,8914
	Desde atrás	0,8396	0,9412	0,9681
Coche	De frente	0,6592	0,6127	0,4517
	De lado	0,6104	0,5981	0,3578
	Desde atrás	0,3245	“Teléfono móvil”	“Camión”
Cuchillo	De frente	0,5457	-	-
	De lado	-	-	-
Señal de Stop	De frente	-	-	0,6261
	De lado	-	-	-
	Desde atrás	“Tijeras”	“Bate de Beisbol”	-
Botella	De frente	0,6798	0,9462	0,8675
	De lado	0,6474	0,8729	0,8815
	Desde atrás	0,6125	0,9146	0,8324
Copa de Vino	De frente	0,6088	-	0,7823
Taza	De frente	0,5087	0,4913	0,4652
	De lado	0,4402	0, 4028	0,4587
Plátano	De frente	0,6985	0,4998	0
	De lado	0,6521	0,4873	0,4778
	Desde atrás	0,7221		
Manzana	De frente	0,4302	0,4781	0,5418
Naranja	De frente	0,5585	0,38745	0,3994
Ratón	De frente	0,6820	-	0.5155
	De lado	0,5475	0,3258	0,4172
	Desde atrás	0,3147	0	0,3754
Teléfono Móvil	De frente	0,5332	0,5812	0,3910
	De lado	0,5038	“Libro”	-
	Desde atrás	-	-	-
Robobo	De frente	0,9084	-	-
	De lado	0,9040	-	-
	Desde atrás	0,8270	-	-

Tabla 2. *Valores de confianza para tres orientaciones en las tres redes.*

Para la obtención de los datos de la

Tabla 2, se realizó un script que captura la imagen haciendo uso de OpenCV y la pasa por las tres redes, devolviendo la imagen pintada y los resultados del reconocimiento (etiqueta, confianza y coordenadas “x”, “y”, ancho y alto del *bounding box*), Este script se encuentra disponible en el siguiente enlace en un repositorio de GitHub:

https://github.com/GII/robobo_vision/tree/master/Fixed-distance-precision-testing

En el repositorio también se encuentra las implementaciones usadas para usar las redes que se ejecutan desde el ordenador, YOLO V3 y MobileNet V2.

Analizando los resultados de la tabla es posible observar lo siguiente:

- Los utensilios de cocina como tenedor, cuchara o cuchillo presentan confianzas bajas y su detección suele ocurrir de forma poco frecuente. Cuando se varía la orientación de lado son detectadas peor por las tres redes, incluida la nuestra.
- Los objetos electrónicos como portátiles o *smartphones* empeoran considerablemente su precisión cuando no se ve la pantalla llegando a ser confundidos con libros u objetos similares al situarse de espaldas.
- Las etiquetas como el reloj dependen completamente de su vista frontal para la correcta detección, siendo confundidas por ejemplo con un “Bowl” en su vista desde atrás.
- Las etiquetas de animales usadas han sido de réplicas o juguetes, para poder ser usados en un entorno de robótica educativa, y posiblemente por ello las redes arrojen resultados de precisión inferiores al haber sido entrenadas con imágenes de animales reales. Las tomas obtenidas con objeto visto desde atrás arroja valores de confianza menores que si es visto de frente o de lado. Lo mismo le sucede al coche, consiguiendo aun así un valor de confianza alto tanto de frente como de lado.
- Aquellos objetos simétricos o prácticamente simétricos como la botella, la copa de vino o las frutas (manzana y naranja) obtienen valores de confianza independientemente de la orientación y muy similares para las tres redes.
- El objeto taza, por ejemplo, obtienen una precisión mayor si no se ve su asa (toma de frente), que si se ve (toma lateral).
- Por último, la precisión obtenida para la nueva etiqueta “Robobo”, consigue unos valores de confianza muy altos (cercanos al 90%), siendo algo peor la precisión de la detección si el Robobo se encuentra visto desde atrás. Como dato curioso durante las pruebas, se comprobó que aquellos objetos de color azul claro eran detectados por nuestra red (MobileNet SSD V3) como un Robobo en una gran mayoría de ocasiones. Esto es debido al color de la cara del mismo que se muestra en la pantalla del *smartphone*.

Tras realizar estas pruebas a una distancia fija (40 cm), era importante caracterizar su precisión en función de varias distancias y también comprobar su respuesta a distintas situaciones, como cuando un objeto ocluye parcialmente a otro. De los objetos de la

Tabla 2 se han escogido al coche, la persona, la botella, el ratón, la taza y el plátano para ser analizados en los siguientes apartados.

5.1.2.2 Variando la distancia al objeto

Como se ha dicho, en estas pruebas se busca ver en qué medida afecta la precisión de detección al variar la distancia del objeto en nuestra red y compararlo con la YOLO V3 y la MobileNet V2, ejecutándose ambas desde un ordenador. Para ello se realizarán cinco pruebas:

1. Pruebas en estático con objetos individuales
2. Pruebas en estático con objetos simultáneos
3. Comparativa de resultados en pruebas de objetos individuales y simultáneos
4. Pruebas con objetos ocluidos
5. Pruebas de detección de personas a distintas distancias e inclinación de Tilt.

5.1.2.2.1 Objetos individuales

La primera de las situaciones fue comprobar la respuesta de las redes con objetos individuales a diferentes distancias. Como ya se ha comentado anteriormente, se usó una inclinación de 100 en el *Tilt* y los objetos fueron colocados a una distancia mínima de 25 centímetros de la cámara, y a una distancia máxima de metro y medio. Se realizaron medidas con los objetos coche, botella y taza. La **Figura 25, Figura 27** y **Figura** 29 muestran, en la fila de arriba, las imágenes originales obtenidas, y en la de abajo, la imagen tras ser procesada por la red. Las imágenes pintadas usadas en la fila de abajo corresponden a los resultados dados por la MobileNet V2. Al tratarse de imágenes en estático, no es necesario repetir las medidas porque el resultado siempre es el mismo. Con estas medidas, se realizaron las gráficas de las **Figura 26**, **Figura** 28 y **Figura** 30 respectivamente, en donde se representa en el eje “x” la distancia en centímetros en el eje “y” los valores de confianza obtenidos en cada red.

Figura 25. *Objeto botella a distintas distancias.*

En la gráfica de la **Figura 26** observamos que las dos redes que se ejecutan en el ordenador (YOLO v3 y MobileNet SSD V2), detectan al objeto botella hasta los 110 y 120 centímetros respectivamente, mientras que la red que corre en el *smartphone* (MobileNet V3) detecta al objeto botella a todas las distancias de las que se disponen en la prueba, ofreciendo una *confidence* algo superior al 30% a la distancia máxima de

un metro y medio. Sin embargo, esta red ofrece de media valores de confianza menores que las otras dos.

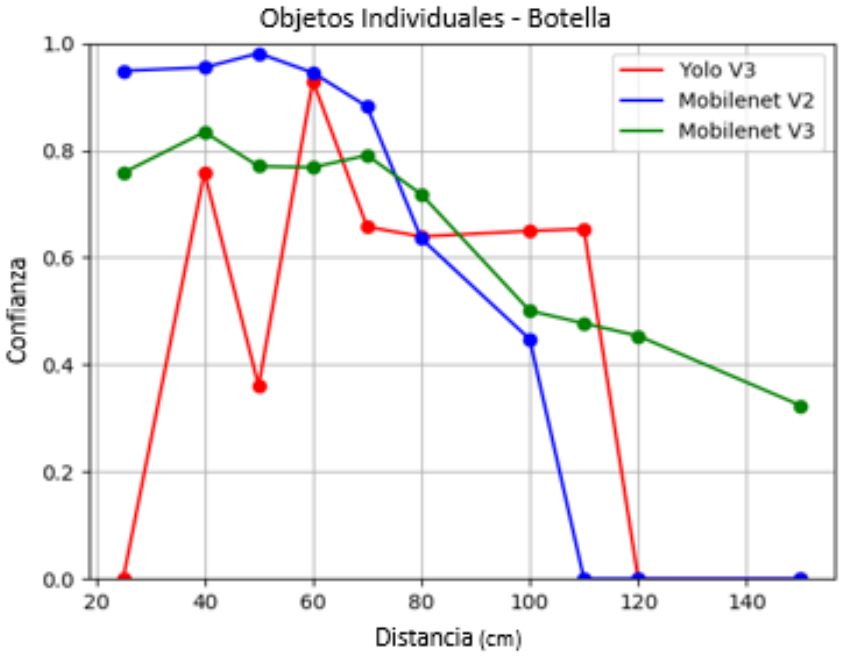

Figura 26. *Confianza del objeto botella a distintas distancias.*

Por su parte la red YOLO alcanza el pico de confianza a los sesenta centímetros y desde ahí mantiene una confianza constante algo superior al 60% hasta dejar de detectar al objeto botella a los 120 centímetros. La MobileNet V2 detecta con una mayor confianza cuanto más cerca esté el objeto, comportamiento que resulta más lógico, con un ligero pico a los 50 centímetros, y desde ahí el valor de su confianza va descendiendo paulatinamente hasta dejar de detectarlo a los 110 centímetros.

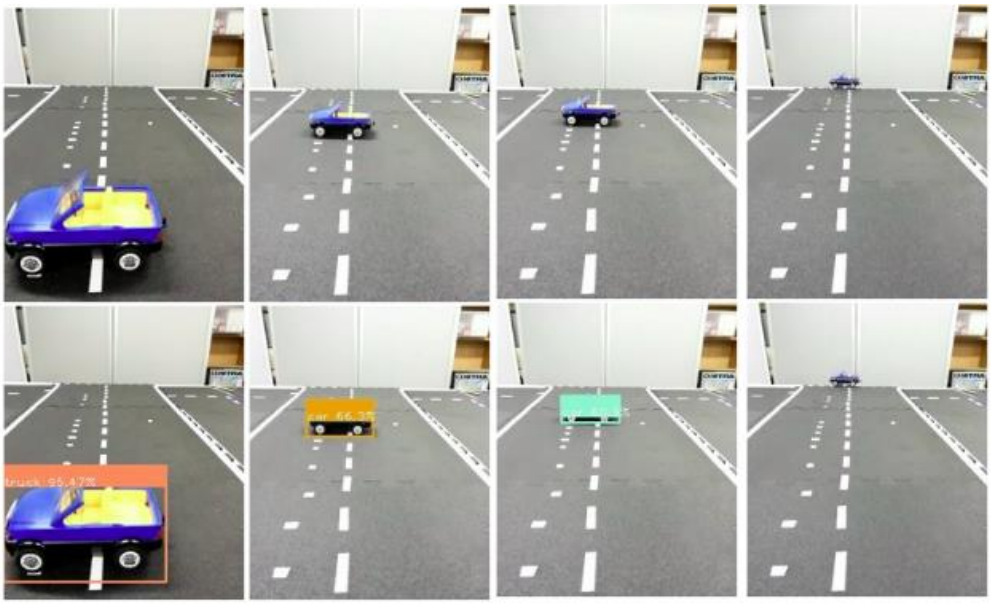

Figura 27. *Objeto coche a distintas distancias.*

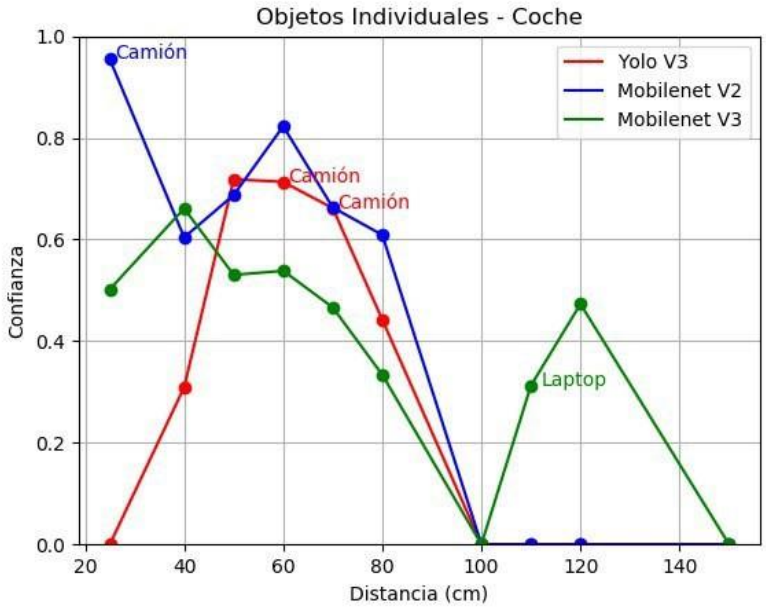

Figura 28. *Confianza del objeto coche a distintas distancias.*

En cuanto al objeto coche, vemos en la **Figura 28** como como las tres redes dejan de detectar el objeto coche al metro de distancia, sin embargo, la MobileNet V3, parece volver a detectarlo a los 120 centímetros. En la gráfica también se muestra el nombre de la etiqueta que la red detecta en caso de no tratarse de la correcta. Observamos, por ejemplo, como la confianza más alta de la red MobileNet V2 ha sido a los 25 centímetros, pero detectando al objeto coche como un camión. En este caso, ambos objetos son similares y más al tratarse de un coche de juguete como se observa en **Figura 27**

La red YOLO vuelve a mostrar el mismo comportamiento que en el caso del objeto anterior, registrando el mayor valor de *confidence* sobre los 50-60 centímetros. Por su parte, la MobileNet V3 (red que se ejecuta en *smartphone*) registra valores de *confidence* más bajos que las otras dos redes, pero logra una detección a mayor distancia. Este comportamiento también es similar al observado en el objeto botella.

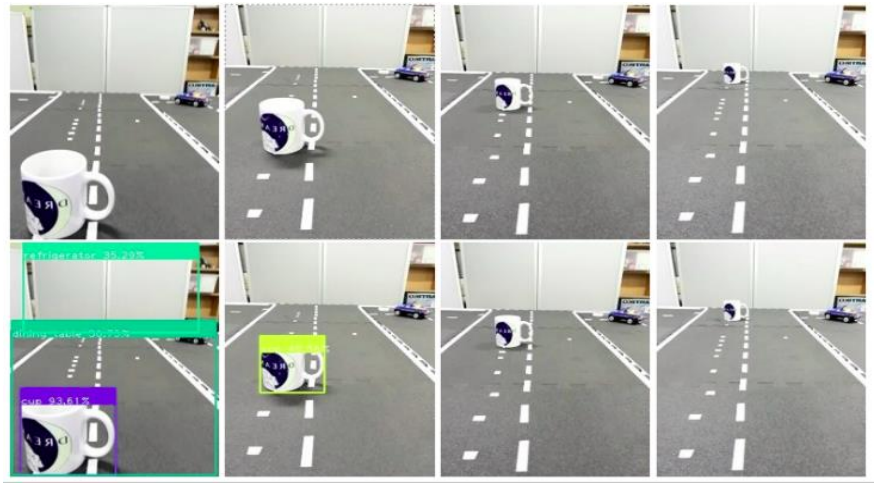

Figura 29. *Objeto taza a distintas distancias.*

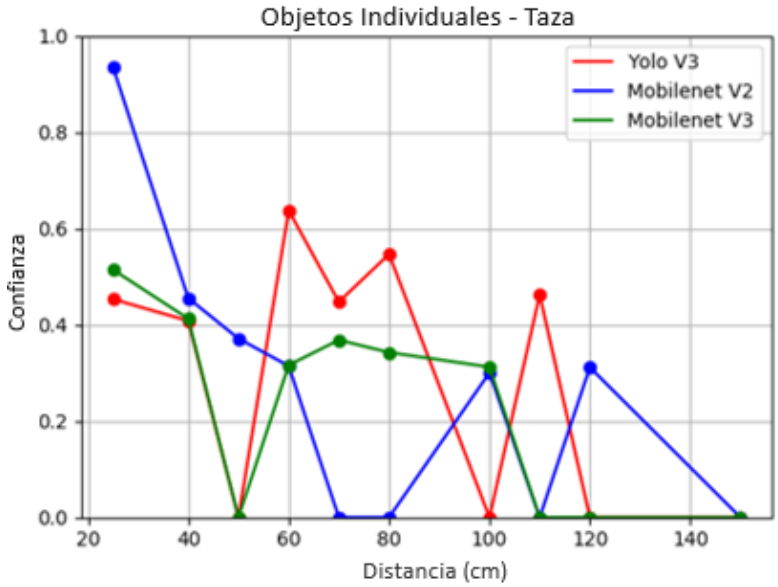

Figura 30. *Confianza del objeto taza a distintas distancias*

Por último, las medidas mostradas en la **Figura 29**, son graficadas en la imagen de la **Figura 30**, en la que vemos como el objeto taza muestra unos valores de confianza inferiores a los otros dos objetos usados. El valor más alto lo logra la MobileNet V2 a los 25 cm y desde ahí su confianza va bajando hasta que deja de detectarlo a los 70 cm, sin embargo, vuelve a detectarlo en la toma a 1 metro de distancia. Nuestra red, la MobileNet V3 detecta la taza hasta los 100 centímetros con confianzas entre el 25 y 45%. Por último, la red YOLO V3, no detecta la toma a 50 cm, igual que la red V3, pero alcanza valores de confianza más altos que las dos redes MobileNet. Todas las redes dejan de detectar a la taza sobre los 110-120 cm.

5.1.2.2.2 Objetos simultáneos

Una vez hemos observado cómo se comporta la red para la detección de estos objetos colocados en la escena de manera individual, queremos compararlo con su comportamiento cuando en la escena hay múltiples objetos del *dataset*. La distancia de la primera medida es a 50 centímetros debido al angular de la cámara del *smartphone*.

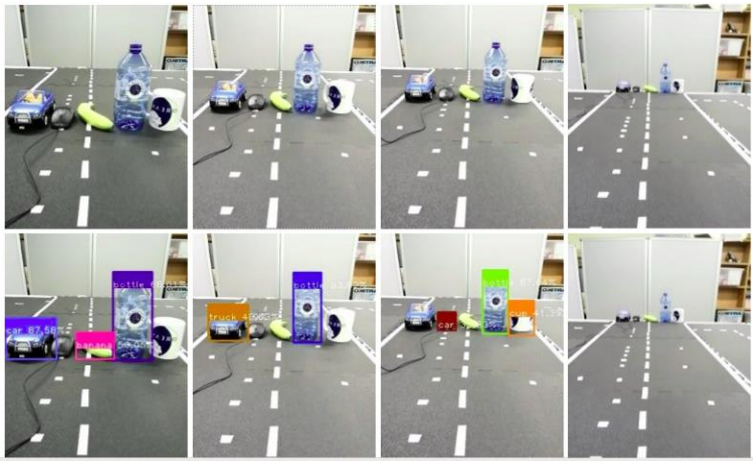

Figura 31. *Objetos de forma simultánea a distintas distancias.*

Como se puede ver en la Figura 30, se realizaron las medidas con cinco objetos: coche, ratón, plátano, botella y taza. En la gráfica de la **Figura 32**, se observa la confianza de los cinco objetos de la escena de la **Figura 31** al ser pasada por las redes que se ejecutaban desde el ordenador (YOLO V3 y MobileNet V2).

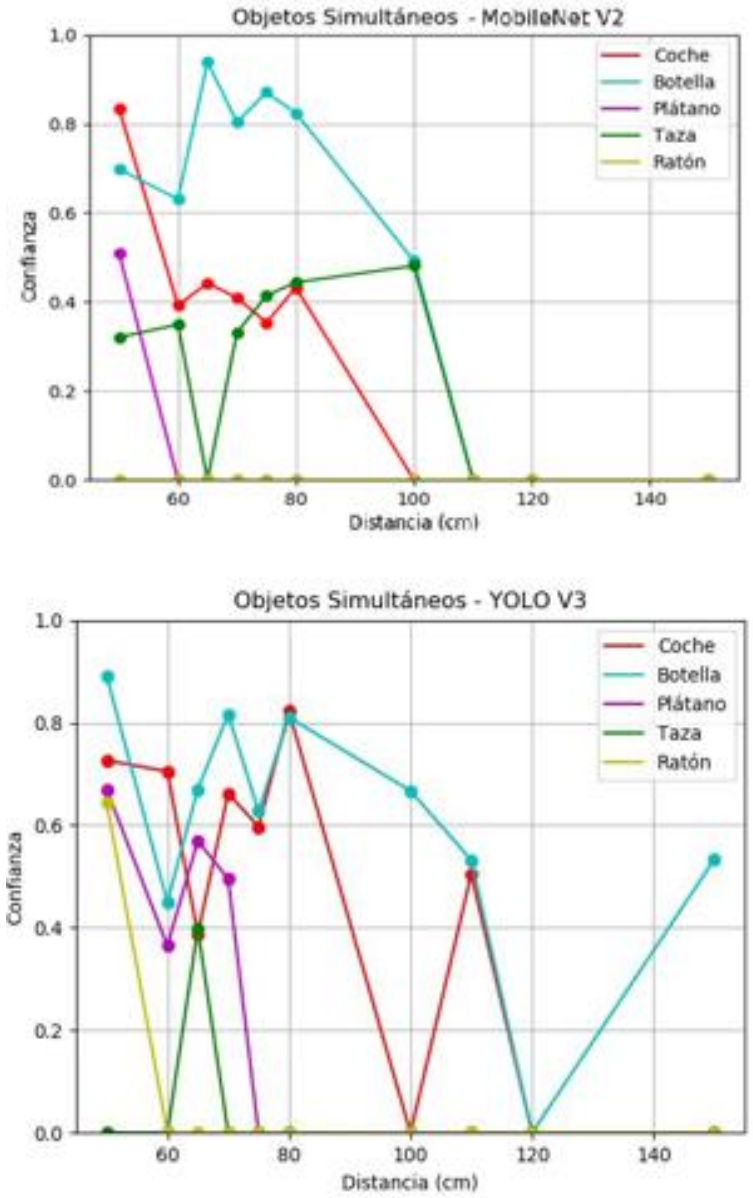

Figura 32. *Confianza de varios objetos en las redes YOLO V3 y MobileNet V3.*

En la Figura anterior (**Figura 32**) observamos que los objetos plátano y ratón ofrecen malos resultados de detección en la escena de la **Figura 31**. Por dicho motivo, para la comparación de las tres redes se descartaron estos dos objetos y se realizó únicamente con los objetos coche, botella y taza.

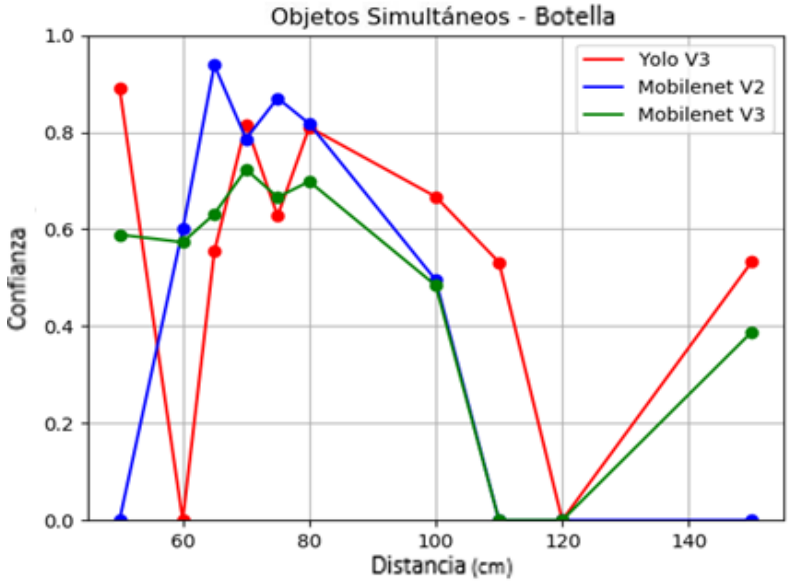

Figura 33. *Confianza del objeto botella con múltiples objetos en las tres redes.*

La **Figura 33** muestra los valores de confianza del objeto botella en función de la distancia, cuando se encuentra en una escena con múltiples objetos a la vez. Los puntos rojos son los valores ofrecidos al pasar las imágenes por la red YOLO V3, la azul por la MobileNet V2, y la verde por nuestra red reentrenada, la MobileNet V3. Observamos como el comportamiento de las tres redes para el objeto botella es muy similar al registrado cuando se encuentra de forma individual en la escena. La red YOLO logra una detección a más distancia que cuando el objeto estaba solo. Con esto comprobamos como el comportamiento de detección no varía de forma considerable al haber más objetos en la escena.

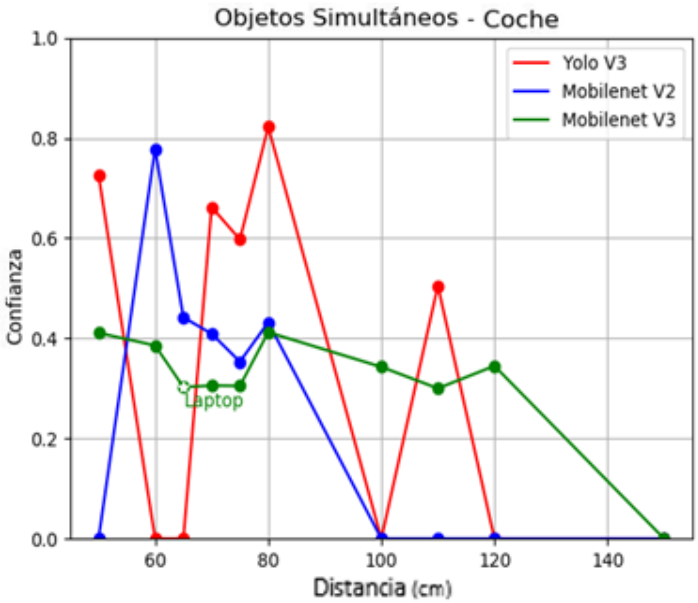

Figura 34. *Confianza del objeto coche con múltiples objetos en las tres redes.*

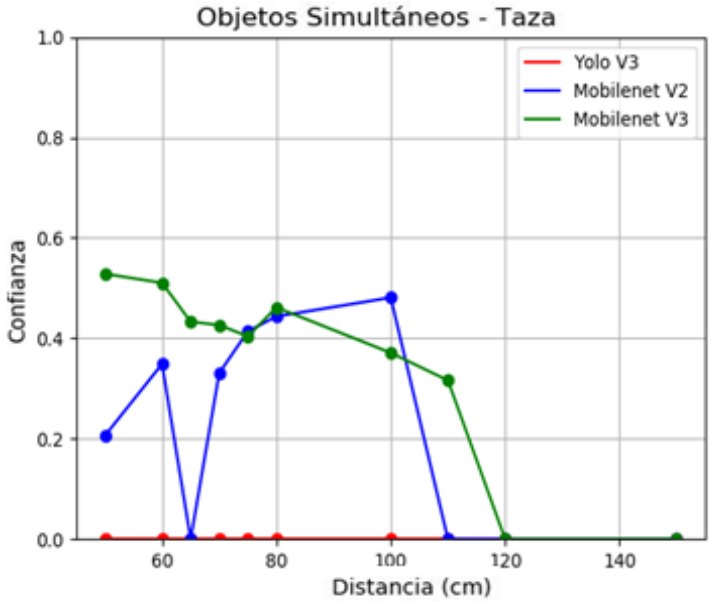

Figura 35. *Confianza del objeto taza con múltiples objetos en las tres redes*

Las gráficas de la **Figura 34** y **Figura 35** representan al objeto coche y taza, respectivamente, en función de la distancia, cuando se encuentran en una escena con múltiples objetos a la vez. En estas gráficas observamos que la confianza en la MobileNet V3 registra unos valores menores que las otras dos redes, pero por lo general logra alcanzar objetos a mayores distancias como es el caso del coche **(Figura 34)**, donde logra detectarlo hasta los 120 centímetros. Las detecciones para el objeto taza **(Figura 35**), son peores que para el objeto coche y el objeto botella, dando valores de confianza menores incluso cuando el objeto se encuentra relativamente cerca de la cámara. Sin embargo, continúan siendo superiores al valor estándar por defecto del 30 %. En esta escena con múltiples objetos, la red YOLO V3 no logra detectar a la taza a ninguna distancia, comportamiento inferior a cuando se encontraba sola en la escena.

Observamos que, para las tres redes, la distancia máxima de detección alcanza entre 100 y 120 centímetros, al igual que ocurría cuando los objetos se encontraban solos en la escena. Todos los objetos usados tienen un tamaño similar, ya que interesaba que pudieran ser objetos que el Robobo pudiera manipular o mover.

5.1.2.2.3 Objetos individuales vs simultáneos

Para hacer más visual y sencilla la comparación, se han graficado juntos los valores de confianza-distancia de un mismo objeto en simultaneo y en individual. Se ha realizado la comparación con los objetos botella y coche, que como se ha podido observar en las gráficas anteriores, son los que ofrecen mejores resultados de detección entre los objetos usados. La gráfica de la **Figura 36** muestra los valores de *confidence* de los objetos coche y botella situados de forma individual en las escenas de las **Figura 25** y **Figura 27**, respectivamente, comparados con ellos mismos en una escena con múltiples objetos (**Figura 31**). La comparativa se muestra primeramente para la MobileNet SSD V2, seguido de la YOLO V3 y la MobileNet SSD V3.

En la gráfica de la **Figura 36,** primera imagen, vemos como por parte de la red MobileNet V2, ambos objetos siguen un comportamiento muy parecido en la escena individual y en simultáneo, dejando de detectar el objeto a los 100-110 centímetros en ambos casos y dando unos valores de confianza más altos para el objeto botella. Este

comportamiento también se aplica en la YOLO, pero con el objeto coche **(Figura 36** segunda imagen).

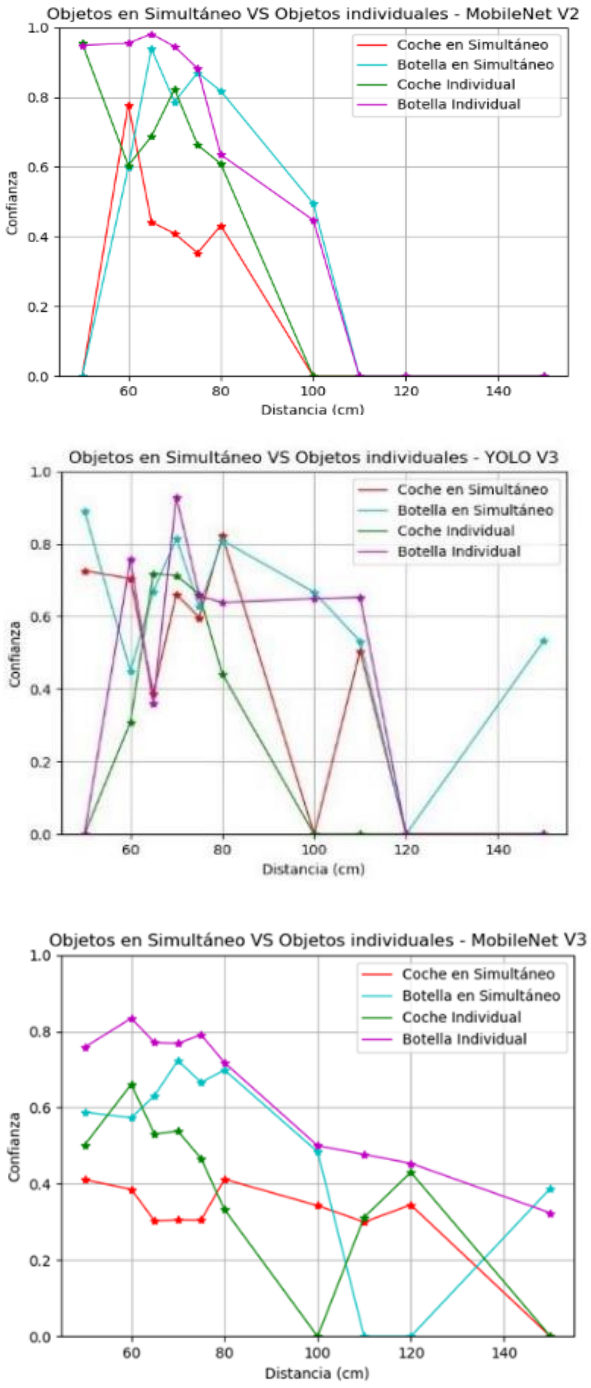

Figura 36. *Confianza de los objetos botella y coche en las tres redes***.**

Por último, nuestra red, la MobileNet V3 (**Figura 36,** tercera imagen**)**, mantiene un comportamiento de detección más estable, entendido como valores de confianza más altos cuanto más cerca está el objeto de la cámara y viceversa. También podemos confirmar al ver la comparativa de las 3 redes en perspectiva, que la MobileNet V3 ofrece resultados de *confidence* algo menores, pero también detecta a los objetos de forma correcta a mayores distancias, como se ha comentado anteriormente.

5.1.2.2.4 Objetos ocluidos

En el presente TFG también se estudió la respuesta de detección cuando los objetos se encontraban parcialmente ocluidos por otros. Esta es una situación habitual en robótica móvil que debemos analizar para acotar la aplicabilidad de esta librería. Utilizando los objetos coche y botella, se realizaron medidas a distintas distancias en dos posibles combinaciones: cuando el objeto coche ocluye parcialmente al objeto botella y viceversa. Para la comparación, se graficaron los datos obtenidos a partir de las imágenes de las y **Figura 39**, y las obtenidas con el objeto de forma individual (**Figura 25** y **Figura 27**, respectivamente). Los datos obtenidos se han realizado para las 3 redes. Primero se mostrarán los resultados obtenidos para la combinación del coche ocluyendo parcialmente a la botella () y luego de la botella ocluyendo al coche (**Figura 39**).

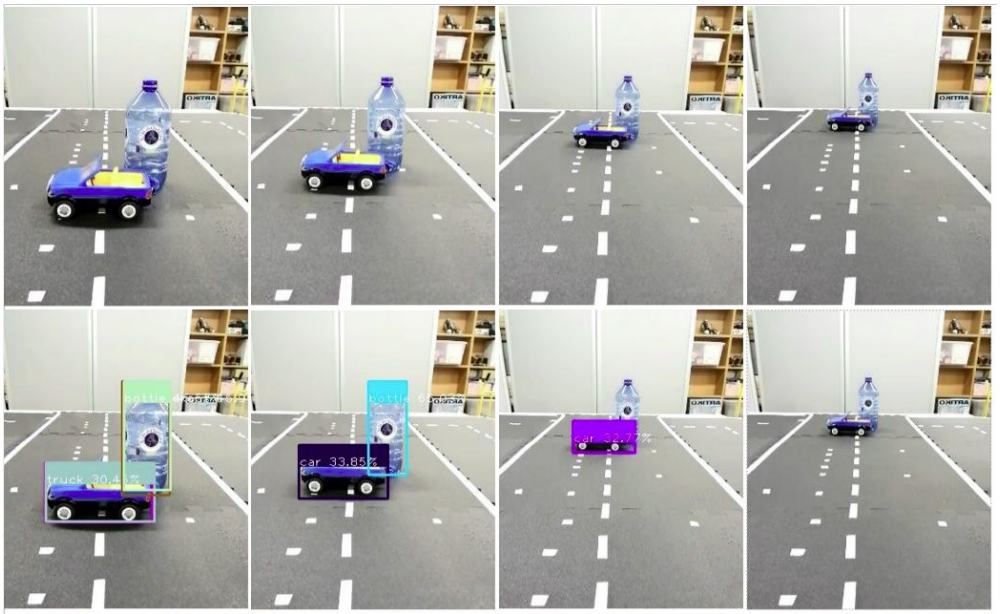

Figura 37. *Objeto coche ocluyendo parcialmente a botella*

En la **Figura 38** se muestra la confianza obtenida para el objeto coche y botella cuando el primero está ocluyendo parcialmente al segundo. En esta gráfica tenemos en la parte superior izquierda los resultados arrojados por la MobileNet V2, en la superior derecha la arrojada por la YOLO V3 y abajo, la arrojada por nuestra red, la MobileNet V3 reentrenada. Vemos como la red MobileNet V2 (**Figura 38**, lado izquierdo) ofrece resultados de confianza mayores cuando el objeto no resulta ocluido, como cabría esperar. Cuando se encuentra ocluido, sus valores de *confidence* se encuentran sobre el 40% para el coche y 60% para la botella. Por parte de la YOLO (**Figura 38**, lado derecho), las diferencias entre que el objeto esté o no ocluido no difieren tanto como en las MobileNet V2, y por su parte, la MobileNet V3 (**Figura 38**, abajo), también ofrece valores de confidence mayores para los objetos sin ocluir, pero la diferencia no es tan notable como con la V2.

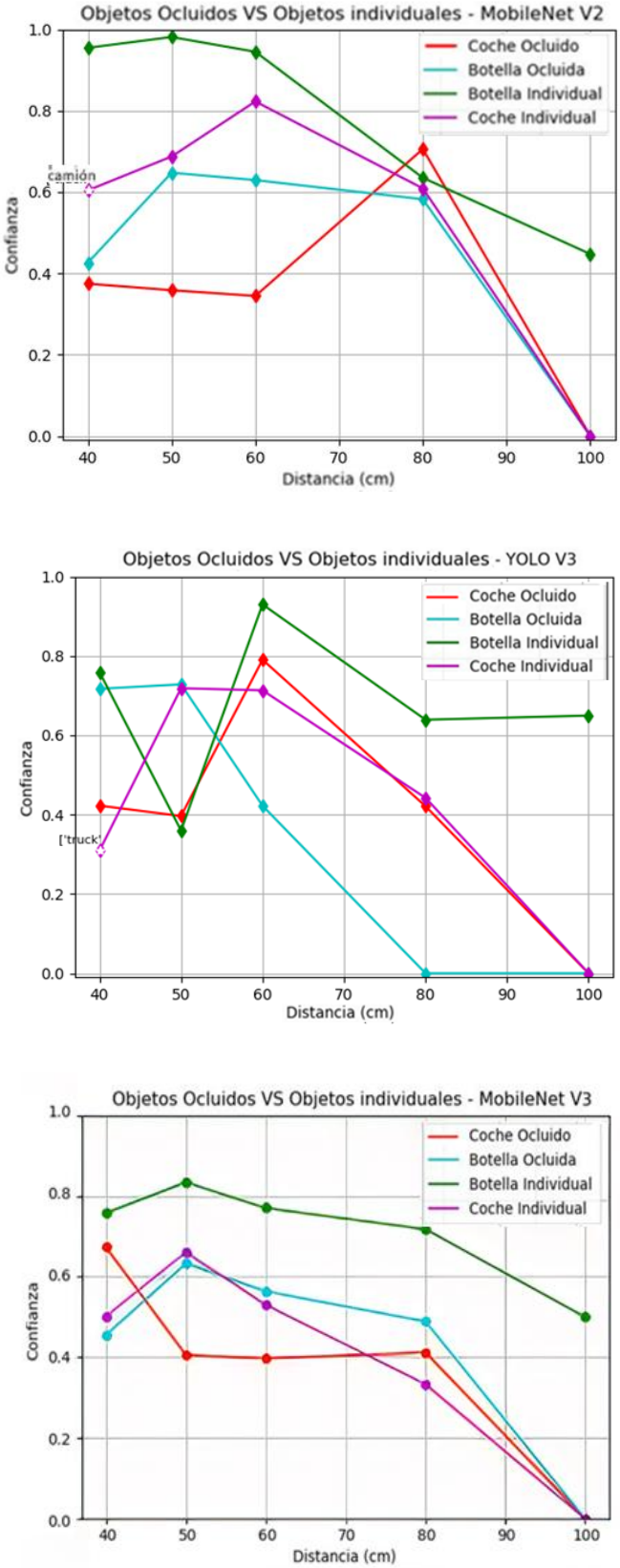

Figura 38. *Confianza del objeto coche ocluyendo a botella en las tres redes*

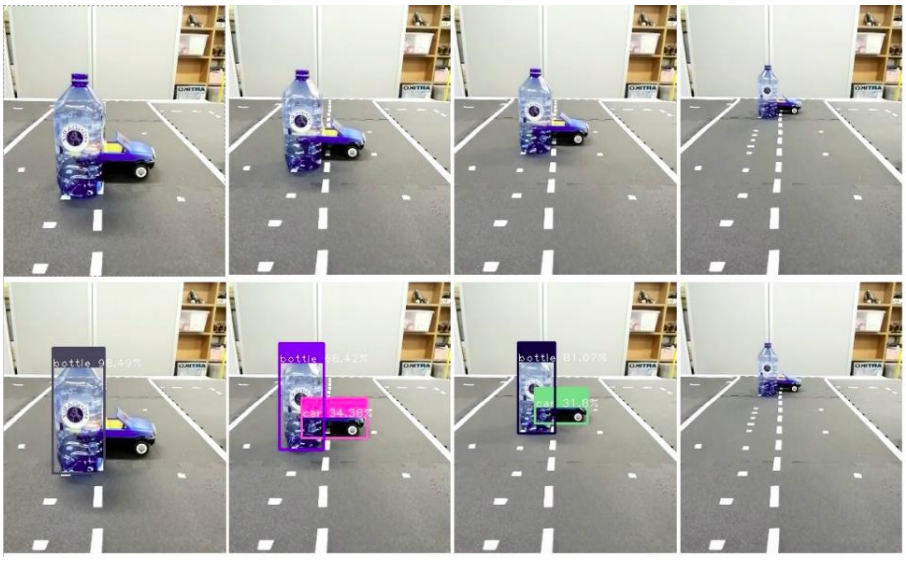

Figura 39. *Objeto botella ocluyendo parcialmente a coche.*

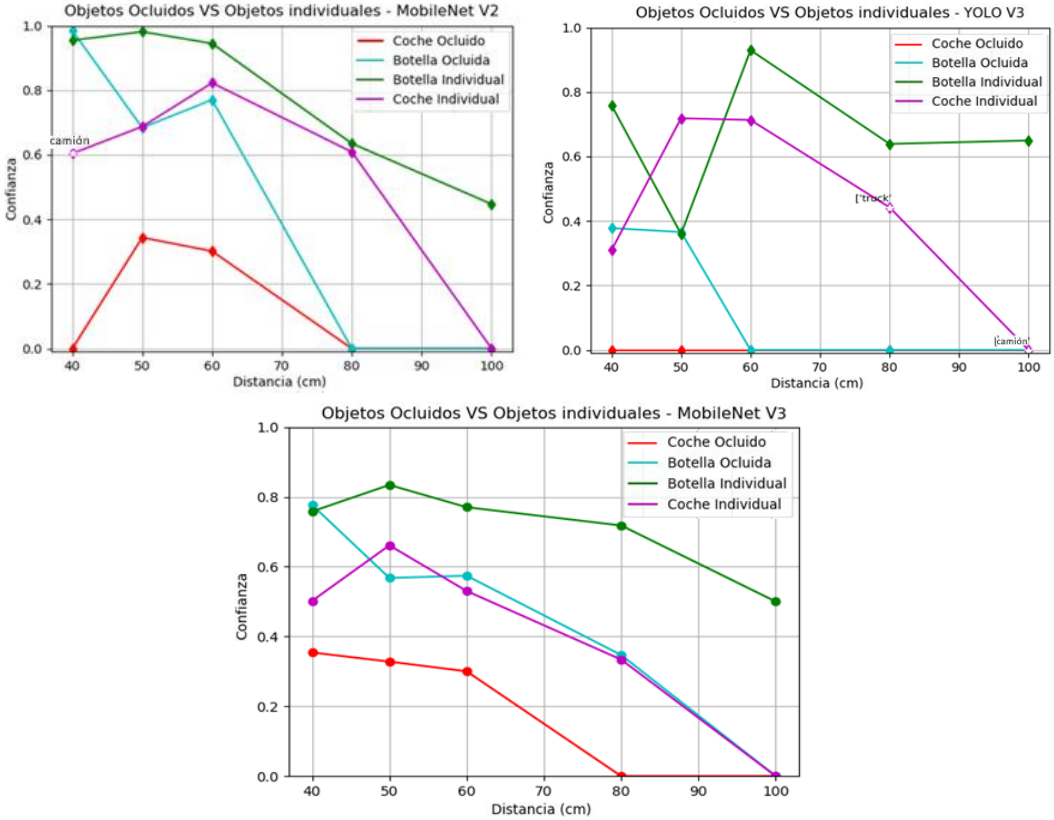

Figura 40. *Confianza del objeto botella ocluyendo a coche en las tres redes.*

Por otro lado, en la **Figura 39** se muestra la combinación opuesta, cuando es el objeto botella quien ocluye parcialmente al coche. En la **Figura 40**, se muestra lo que se obtiene al graficar los valores de confianza en función de la distancia de la **Figura 39** a través de las tres redes. Para esta combinación (**Figura 40**), obtenemos como comportamiento general en las tres redes que los objetos ocluidos son detectados de peor forma y con valores de confianza menores. Este comportamiento era el esperado en todo momento y solo interesaba comprobar como de “mala” era su detección en comparación. Al igual que en el caso anterior, el objeto botella es detectado con valores

más altos que el de coche al ser este último el que se encuentra ocluido. Ya vimos como para las redes MobileNet este objeto resaltaba frente al otro a pesar de ser el que estuviera ocluido. Por lo tanto, no es extraño comprobar como para esta combinación alcanza valores cercanos al coche en individual. De hecho, por lo general, el coche ocluido logra valores bajos de detección en las dos redes MobileNet y no llega ni a ser detectado en la YOLO V3.

5.1.2.2.5 Detección de personas a distintas distancias e inclinaciones de Tilt

Para no juntar a la etiqueta persona con el resto de los objetos analizados anteriormente, por poseer tamaños reales muy distintos, se analizó el comportamiento de esta etiqueta de forma independiente. Las medidas se realizaron con dos ángulos de *Tilt* distintos. El primero simula que el Robobo se encuentra en el suelo, con un *Tilt* con un valor de 60 para poder captar a la persona desde su perspectiva. El segundo simula la situación del Robobo encima de una mesa, una altura muy usual para un robot usado en un entorno educativo, este último con un *Tilt* prácticamente recto (80). Al tratarse de un objeto mucho más grande, las distancias usadas también fueron mayores, desde los 120 cm hasta los 480 cm en el caso de la toma desde el suelo, y de 160 a 520 cm en la toma desde la mesa. Las tomas están espaciadas 40 cm entre sí. La imagen de la **Figura 41** muestra las tomas obtenidas con la etiqueta persona cuando el Robobo se encuentra en el suelo con un Tilt de 60.

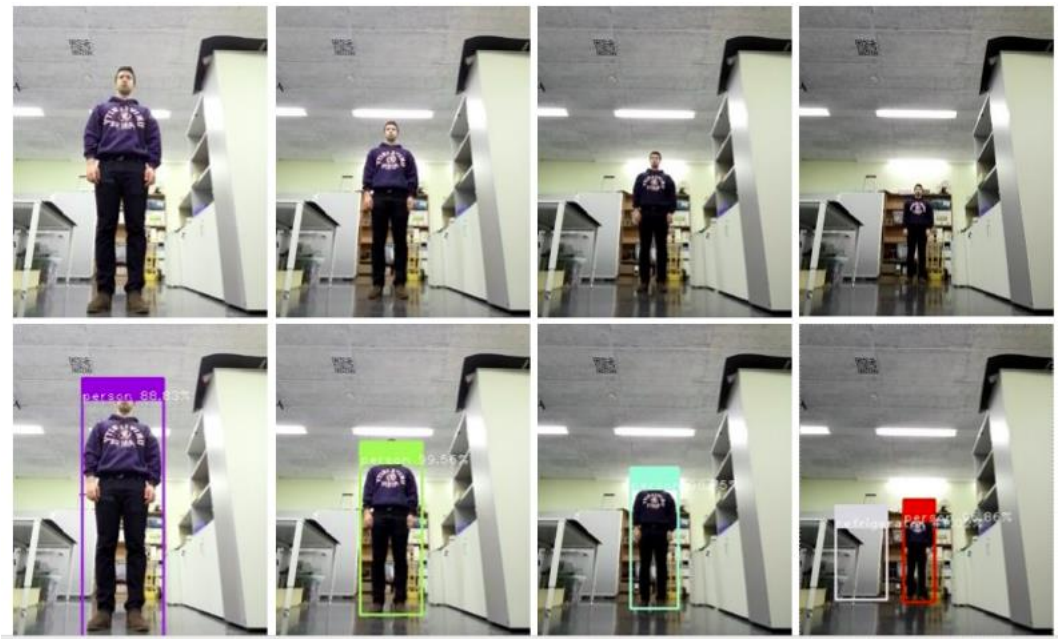

Figura 41. *Toma de la etiqueta persona con el Robobo en el suelo (Tilt 60).*

A partir de la toma de la **Figura 41**, se graficó la confianza obtenida en función de la distancia a través de las tres redes, tal como se observa en la **Figura 42.** En esta gráfica vemos como la etiqueta persona, al igual que ocurría con el objeto botella o coche obtiene resultados de detección superiores al de otros objetos como la taza, o el ratón. Sin embargo, en el caso de la etiqueta persona son mucho más sobresalientes, con una confianza muy cercana al 100 % en casi todo el rango de distancias. Observamos como la MobileNet SSD V3 obtiene valores de media menores, cercanos al 80 %. Este comportamiento es ya el mostrado hasta ahora

por esta red. De todas maneras, se observa en la gráfica como los valores de esta etiqueta son muy altos independientemente de la distancia, ya que, a casi cinco metros, la confianza de detección sigue manteniéndose entre el 80 y el 100% para las tres redes.

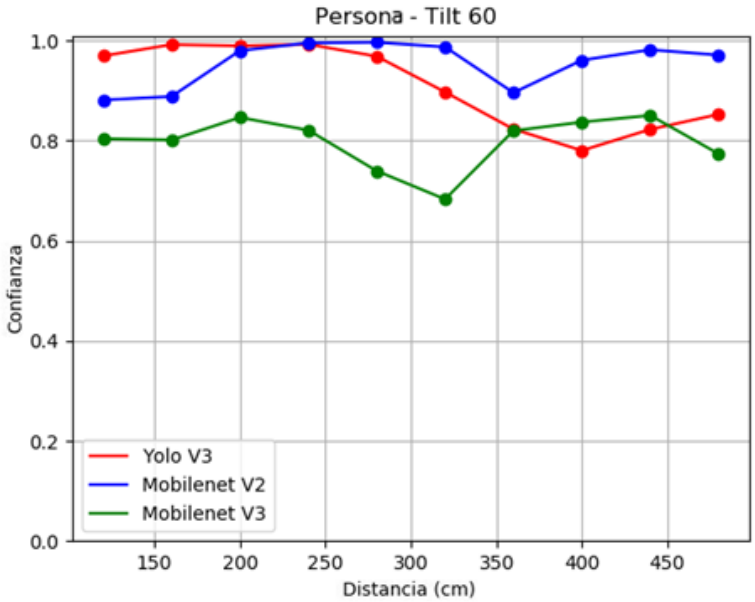

Figura 42. *Confianza de la etiqueta persona en las tres redes (Tilt 60).*

Por otro lado, en la **Figura 43**, vemos las tomas realizadas con el Robobo colocado encima de una mesa con un Tilt recto (Tilt 80).

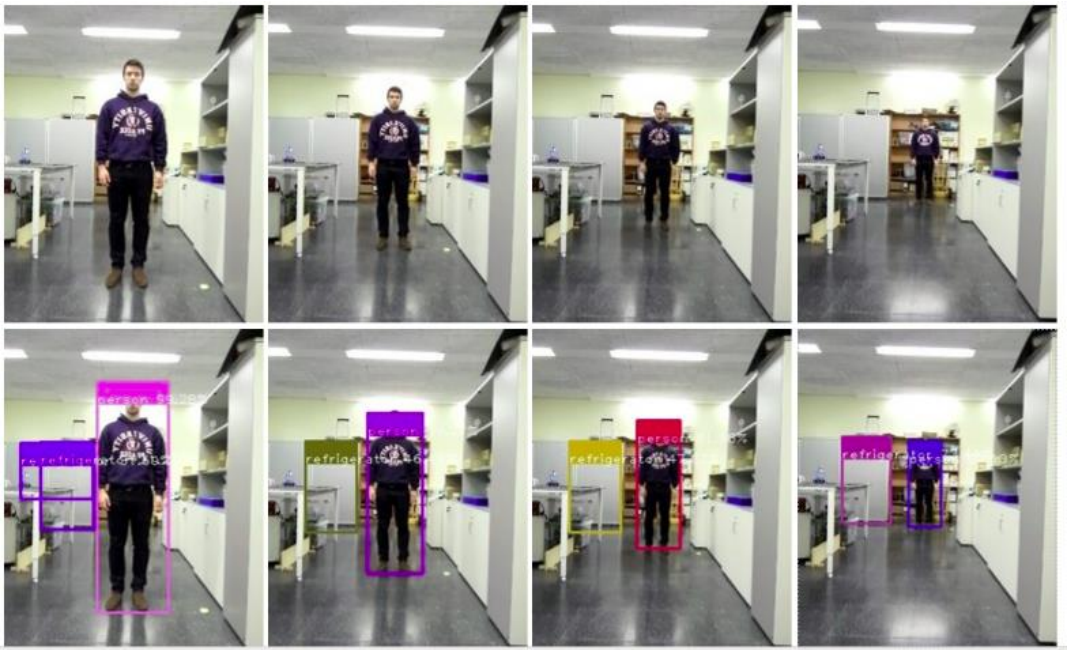

Figura 43. *Confianza de la etiqueta persona en las tres redes (Tilt 80).*

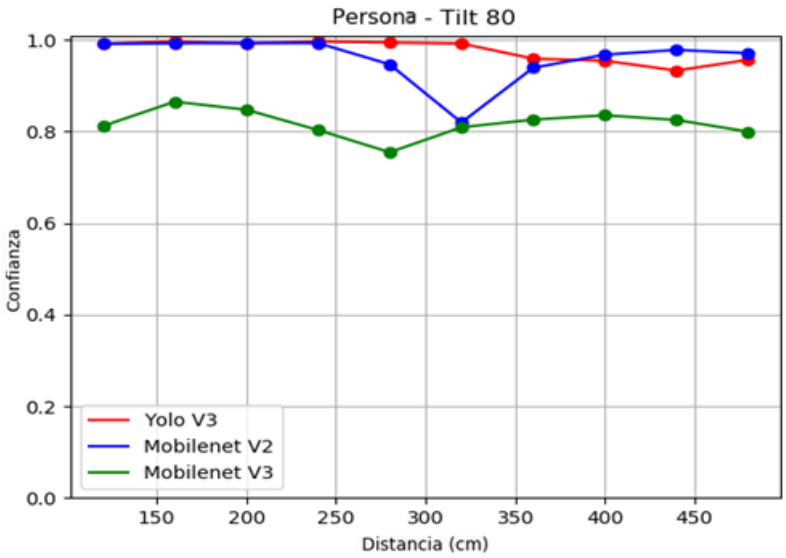

Figura 44. *Confianza de la etiqueta persona en las tres redes (Tilt 80).*

A partir de los datos obtenidos en la **Figura 43**, se obtuvo la gráfica de la **Figura 44**, donde se muestra sus valores de confianza en función de la distancia. Observamos en dicha gráfica como los valores de confianza continúan siendo muy altos, cercanos al 100% en todas las distancias, a excepción de la MobileNet V3, que registra valores algo menores (80-85%). Al encontrarse la etiqueta persona en una mejor perspectiva que la toma desde el suelo, los valores de confianza son algo superiores, tal como se esperaba.

Concluimos que la detección de personas posee los valores de confianza más altos de todas las etiquetas usadas para las pruebas en estático. Independientemente de la distancia, logra alcanzar valores muy cercanos al 100 % y su detección ocurre en todas las tomas, a diferencia de objetos que también poseen una precisión de detección alta como la botella o el coche. Comentar que, posiblemente, la persona sea la etiqueta de mayor tamaño que se use en el entorno educativo del Robobo, y en parte, su detección es tan alta debido a su gran tamaño comparado con los otros objetos del *dataset*.

5.1.3 Pruebas en movimiento

Las pruebas con el robot en movimiento se realizaron únicamente con el algoritmo MobileNet SSD V3 reentrenado, comparando por un lado su velocidad de detección cuando el algoritmo es ejecutado a bajo nivel (*Smartphone*) y a alto nivel (en un ordenador de sobremesa) y, por otro, tratando de conocer la velocidad límite a la cual la red ya no distingue los objetos.

En estas pruebas, el Robobo parte siempre de una posición fija a 60 centímetros del primer objeto, tal como se observa en la **Figura 45**. Desde esa posición, con el PAN girado 90 grados de tal manera que la cámara observe directamente al objeto, el Robobo comienza a moverse haciendo uso del método *moveWheels(),* el cual tiene un rango de potencias entre [-100 y 100]. El Robobo arranca inicialmente desde su posición fija con una potencia de giro en sus ruedas de 10. Desde ahí se repite el proceso aumentando de cada vez en diez la potencia de los motores hasta llegar al máximo (100).

Figura 45. *Esquema de las pruebas realizadas con el Robobo en movimiento.*

Para la realización de estas pruebas, en la red ejecutada desde el teléfono inteligente, se usó igualmente el método *ReadDetectedObject()* creado por MINT junto a la librería *Time* de Python. Cada vez que se detectaba un nuevo objeto, se guardaba en un archivo de texto: su etiqueta, su confianza y el instante de tiempo en que ocurrió la detección. Por su parte, para la detección local se usó el mismo procedimiento.

5.1.3.1 Pruebas con distancias fijas entre objetos

En primer lugar, se comprobó el rendimiento con una distancia fija entre los objetos de 40 centímetros. El objeto usado fue una botella, al tratarse de un objeto que la red detecta bien. Siguiendo el esquema de la **Figura 45**, se realizaron medidas a distintas velocidades del Robobo con nuestra red ejecutándose desde el smartphone y desde el ordenador, y se graficó el tiempo en el que ocurre la detección (eje "y") para cada potencia disponible (eje "x"). Los resultados se muestran en la **Figura 46**.

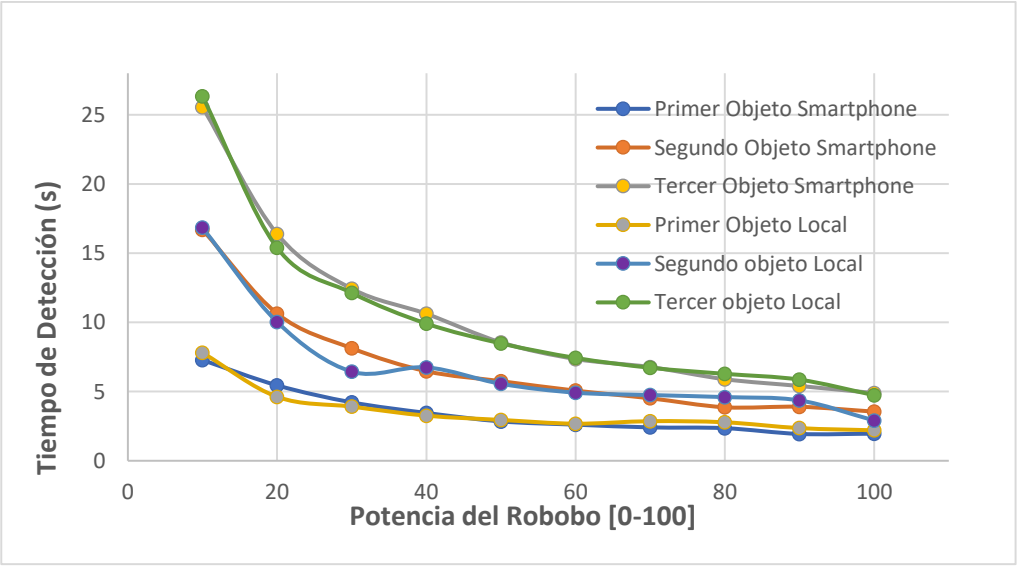

Figura 46. *Velocidad de detección de objetos de la red MobileNet V3 ejecutándose en un ordenador y en un smartphone.*

Analizando dicha figura, observamos que, con una distancia fija de 40 centímetros entre objetos, la red es capaz de distinguir a los tres objetos en todo el rango de

velocidades de los que dispone el Robobo. Vemos que también que cuanto mayor es la velocidad del robot, antes se detecta cada objeto, como resulta lógico, y además el instante de tiempo en el que ocurre cada detección es casi idéntico independientemente de que la red sea ejecutada en nuestro smartphone o en un ordenador portátil.

Pero surge ahora la duda de cómo se comportará al reducir la distancia entre dichos objetos, es decir, comprobar cuál es el límite de acercamiento donde la red ya no es capaz de distinguir entre un objeto u otro, o directamente no lo detecta.

5.1.3.2 Pruebas con distancias variables entre objetos

Para realizar esta prueba se colocaron tres objetos distintos del *dataset,* para poder así distinguir con más facilidad los límites de detección entre un objeto y otro. Del intervalo de velocidades, se escogió realizar las pruebas a potencia **20, 50 y 100**.

El primer objeto (*Florero*) se encuentra siempre a una distancia fija de 60 centímetros del punto de salida del Robobo. El segundo objeto (*Naranja*) y el tercero (*Botella*) se posicionan a 60 centímetros entre sí en la primera medición, y desde ahí se va reduciendo su distancia de 10 en 10 centímetros. La distancia entre ellos se toma desde el centro de un objeto al centro de otro, como se observa en la **Figura 47.**

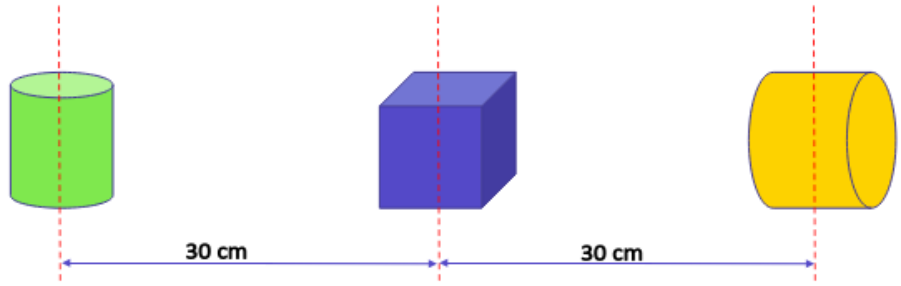

Figura 47. *Ejemplo de la forma de medir las distancias entre objetos.*

A continuación, se muestran en la **Figura 48** el tiempo en segundos en donde ocurre la detección para cada uno de los tres objetos en función de la distancia entre ellos. Esta detección se realiza tanto para cuando se ejecuta en local (ordenador) como desde el smartphone. **La figura 48**, arriba a la izquierda muestra su respuesta cuando el Robobo se movía con una potencia de 50. La imagen a la derecha de esta, con una potencia de 100 y finalmente en la imagen de abajo, se muestra para una potencia de 10.

En estas gráficas vemos como el comportamiento general de los tres objetos es similar, y como apenas hay diferencias en cuanto velocidad si el objeto es detectado desde un *smartphone* o desde un ordenador. Los tiempos de detección se reducen cuanto mayor es la velocidad, y los tres objetos parecen ser detectados sin problemas a excepción del objeto “naranja”, en la gráfica de la **Figura 48**, la imagen de arriba a la derecha. En dicha gráfica, si se observa, se puede ver que los puntos de color amarillo, que representan la velocidad de detección del objeto naranja ejecutado desde el smartphone no aparecen, y esto es debido a que la red MobileNet V3 no logró detectarlo a esa velocidad. Por su parte, la red ejecutándose desde el ordenador sí es capaz de detectarla (**Figura 48,** imagen de arriba a la derecha, línea gris).

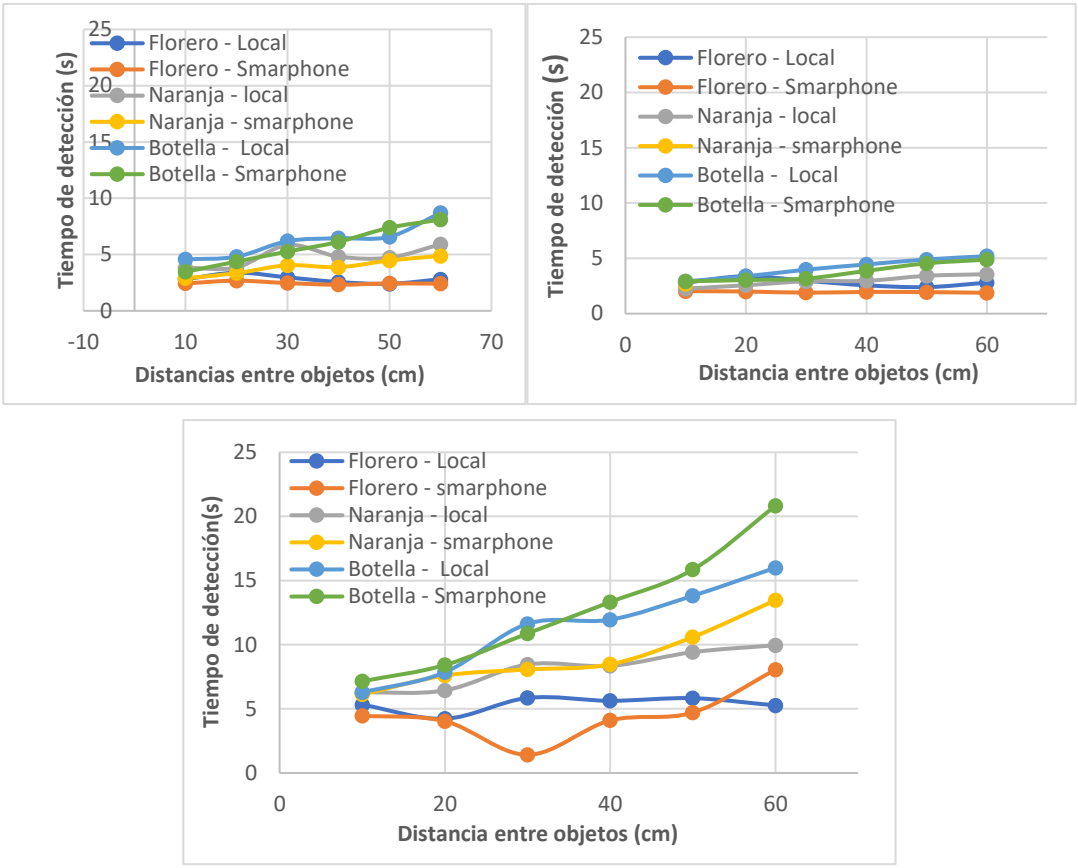

Figura 48. *Tiempo detección MobileNet V3 para potencias 10,50 y 100.*

Como hemos visto, ciertos objetos que poseen valores de confianza menores como es el caso de la *naranja*, son proclives a que su detección pueda fallar cuando la velocidad del Robobo es muy alta.

La conclusión general de este apartado es que la diferencia en cuanto detección y velocidad de respuesta del algoritmo MobileNet V3 es independiente de donde sea ejecutado, y es capaz de realizar las detecciones correctamente sin importar la velocidad del Robobo o la distancia entre objetos. Además, es importante destacar que, por lo general, las potencias usadas por el Robobo en un entorno real nunca alcanzarán valores tan altos como 80 o 100.

5.1.4 Modelo de detección en función de la distancia

Las redes utilizadas en la detección de objetos nos devuelven la etiqueta, la confianza y las coordenadas del objeto en la imagen. Sin embargo, no proporcionan la distancia real a la que se encuentra el objeto detectado. Esto resulta lógico, puesto que una imagen es una representación bidimensional de una escena tridimensional. Las redes analizadas devuelven entre otros datos, el alto y el ancho del rectángulo de detección que encierra al objeto (“Bounding Box”), con estos datos se puede obtener el área de detección multiplicándolos entre sí. Con esta área y midiendo distancia real del objeto, se puede crear una relación matemática entre ellos haciendo uso de un polinomio de interpolación. Usando las imágenes de las pruebas en estático, se puede saber la distancia real a la que se encuentra cada objeto pues estas fueron medidas previamente con una regla. Cuanto más lejos se encuentre el objeto en la escena, más pequeño será el valor de su área de detección. Por lo tanto, colocando las distancias

reales en centímetros en el eje 'Y' y sus áreas en el eje 'X', podemos ajustar los puntos a distintas funciones. Este modelado se ha realizado para los objetos coche, botella y persona, haciendo uso de nuestra red MobileNet V3.

Para evaluar como de bueno era el ajuste de los valores obtenidos con distintas funciones, se utilizó el coeficiente de determinación (R^2). Este coeficiente proveniente de la estadística, es usado para predecir futuros resultados o probar una hipótesis. El coeficiente determina la calidad del modelo para replicar dichos resultados.

En la gráfica de la **Figura 50**, observamos el resultado de representar área frente a distancia y ajustarlo a través de una función polinómica de segundo orden, otra de tercer orden, una potencial y una exponencial. El ajuste a una función cuadrática produce un R^2 de 0,9532 y el valor más alto lo alcanza ajustándose a una función potencial, con un R^2 de 0,9974.

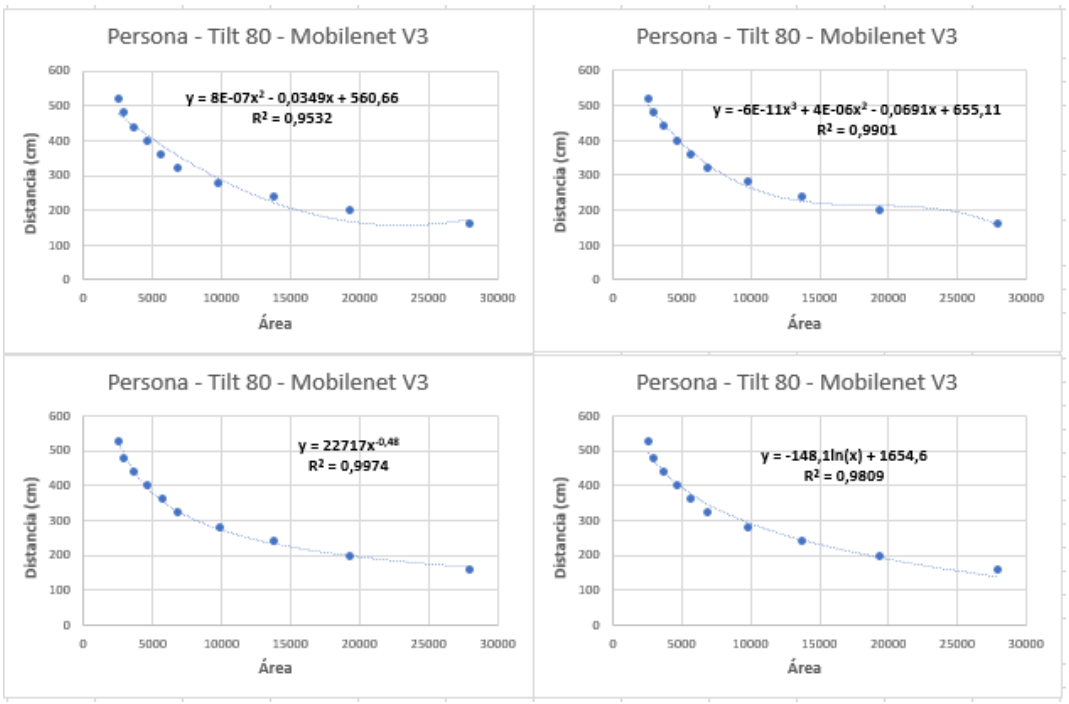

Figura 49. *Relación entre área y distancia para la etiqueta persona usando la red MobileNet V3.*

Con los resultados mostrados en la **Figura 49**, observamos como el área y la distancia parecen seguir una tendencia similar a una función cuadrática o una potencial. Para tratar de encontrar una relación más general entre los objetos usados, se juntaron los valores de área-distancia de los objetos coche y botella (**Figura 50**) en una misma gráfica. Los puntos de color naranja corresponden al objeto botella y los de color azul al objeto coche. En dicha gráfica se busca realizar el ajuste con dos de los objetos que mejor respuesta en cuanto detección mostraron, para tratar de encontrar una tendencia más general que pueda llegar a ser extrapolable al resto de objetos del *dataset*. A continuación, se muestra esta gráfica conjunta con los valores de ambos objetos.

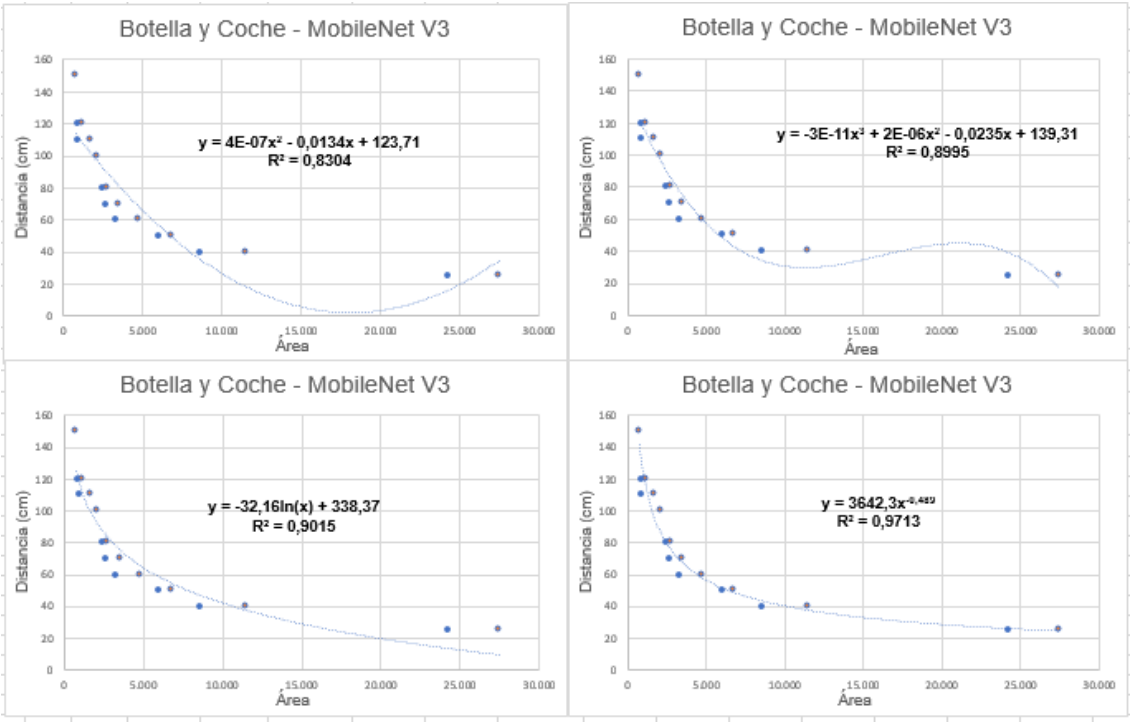

Figura 50. *Relación entre área y distancia para las etiquetas botella y coche usando la MobileNet V3.*

En la gráfica de la **Figura 50,** vemos como la tendencia se mantiene, pero la falta de medidas entre 25 y 40 cm, sumado al brusco cambio de área en esta zona conlleva a que se produzca un mal ajuste de los puntos. Las gráficas de la **Figura 49** y **Figura 50** se realizaron con las medidas obtenidas en el apartado **5.1.3** para analizar la precisión de los objetos en estático a distintas distancias. Estas medidas pueden resultar insuficientes por tratarse de solo diez. Para conseguir un ajuste más preciso se volvieron a tomas medidas con el objeto botella, pero esta vez a veinte distancias distintas, desde 25 cm hasta los 150, obteniendo más puntos intermedios. Con las medidas donde la detección fue exitosa se realizó la gráfica de la **Figura 51**. En esta gráfica vemos como el ajuste de los puntos vuelve a mostrar una fuerte tendencia hacía una función cuadrática, alcanzando el valor más alto de R^2 para una función potencial de la forma $y = 2910{,}6\,.\,x^{-0{,}475}$.

La función potencial de la **Figura 51** es similar a las producidas tanto por la etiqueta persona ($y = 22717\,x^{-0{,}48}$), como por la gráfica de la **Figura 50** ($y = 3642{,}3\,x^{-0{,}48}$)**.** Observamos como la función parece ser de la forma $y = n\,x^{-0{,}48}$, Siendo el valor de "n" mayor en función del área del objeto detectado. La conclusión que extraemos al graficar estos valores y tratar de ajustarlos a distintas funciones, es que dicha función dependerá del área de cada objeto, por lo tanto, cada objeto del dataset poseerá su propia función que relaciona área y distancia. Sin embargo, como se ha podido observar en las gráficas de la **Figura 49, Figura 50** y **Figura 51**, la relación entre estas dos varibles sigue una tendencia general que se asemeja al de una función cuadrática o una potencial de la forma $y = n\,x^{-0{,}48}$.

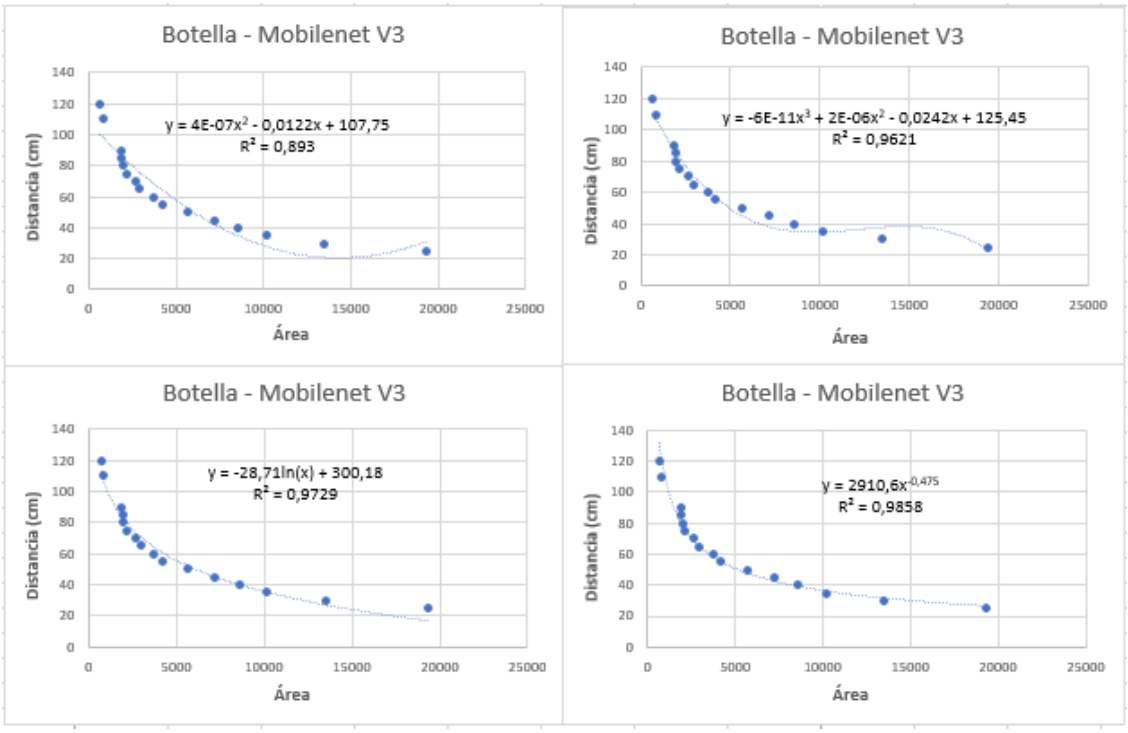

Figura 51. *Relación entre área y distancia para la etiqueta botella usando la red MobileNet V3 y ampliando el número de medidas realizadas a 20.*

5.1.5 Conclusiones de la caracterización de la librería de detección de objetos en tiempo real

Una vez analizado el comportamiento de la red MobileNet V3 implementada en la librería de detección de Robobo, tanto en pruebas con el robot parado como en movimiento, podemos sacar las siguientes conclusiones:

- La precisión en la detección de los objetos varía en función del tipo de objeto del que se trate. Como hemos comentado anteriormente, etiquetas del dataset como persona, coche o botella alcanzaban valores de detección en cuanto distancia y confianzas superiores a otros como ratón, plátano o taza. A pesar de ello, todas las confianzas dan valores superiores al 30%, y, por lo tanto, son consideradas como válidas. Como se observó en la
- Tabla 2, la rotación de los objetos afecta en gran medida a la precisión de la detección, llegando a no detectar o confundir la etiqueta en función de su orientación respecto a la cámara. Todos los objetos no simétricos obtuvieron valores de confianza más altos si eran vistos de frente o de lado, en vez de desde atrás. Los objetos usados para las pruebas de precisión a distintas distancias fueron colocados en la escena siempre de frente o de lateral por esa misma razón.

- La distancia máxima de detección ronda los 100 centímetros para la mayoría de los objetos que se usarán en el entorno educativo del Robobo.

- No existe una diferencia apreciable entre que un objeto se encuentre solo en la escena o acompañado de otros de cara a su detección, siempre y cuando no haya oclusiones.

- La oclusión parcial de objetos afecta a su detección y la empeora a razón de su distancia a la cámara. Los objetos ocluidos parcialmente son detectados peor que si no lo estuvieran y alcanzan de media distancias menores, en torno a 80 centímetros. El objeto que ocluye también empeora su detección. La oclusión no les afecta a todos los objetos del dataset por igual, hay objetos como la botella, con una detección superior a otros al encontrarse ocluidos.

- Las tres redes CNN arrojan unos valores similares en cuanto a detección, con la particularidad que la MobileNet V3 logra detectar objetos a mayores distancias, pero ofrece por lo general unos valores de confianza menores (en torno a un 10-15 % menos) en ciertas etiquetas como persona, coche o botella. Sin embargo, como se puede ver en la
- Tabla 2, esta red consigue unos valores de precisión similares a las otras redes, o incluso mejores en una gran cantidad de objetos como en el caso del teclado, el libro, la naranja o el ratón. Concluimos que, aunque la red MobileNet V3 implementada en la librería de detección de Robobo ofrece valores algo menores de confianza en ciertas etiquetas, es comparable en precisión a las otras dos redes que se ejecutan desde un ordenador.

- Utensilios de cocina como cuchara, tenedor o cuchillo no son detectados por lo general por la nueva red Mobilenet V3, por lo que se recomienda eliminarlos del conjunto de etiquetas final.

- No existe una diferencia apreciable entre la velocidad de detección si el algoritmo es ejecutado desde un ordenador o desde un Smartphone usando una versión ligera como es TensorFlow Lite.

- Las redes son capaces de detectar el objeto independientemente de la velocidad del robot y la distancia de separación entre objetos. Recordemos que, para este tipo de redes, por defecto se considera que una detección es válida por encima del 30% de confianza, y aquellos objetos que tengan valores de confianza en torno a este número es posible que puedan no ser detectados a la máxima potencia que posee el Robobo (100).

- Los distintos objetos del dataset tienen una relación distancia-área similar, que se asemeja a una expresión polinómica de segundo orden (cuadrática) o una potencial de la forma $y = n\,x^{-0,48}$, siendo “x” el área del objeto detectado, e “y” la distancia en centímetros a la que se encuentra. Como se observa, el valor de n es mayor en función del tamaño del objeto detectado.

5.2 Detección de carriles en tiempo real

5.2.1 Introducción

La conducción autónoma es un objetivo perseguido por una gran cantidad de empresas como Tesla, Renault, Uber o Toyota por su aplicación en la automoción. Siguiendo la escala propuesta por la SEA (*Society of Autonomous Engineers*), se clasifican las capacidades autónomas de un vehículo en 6 niveles [28]:

- El nivel 0 sería un coche sin automatización de ningún tipo.
- Un nivel 1, sería un vehículo con asistencia de carretera, como mantenimiento en carril o control adaptativo de la velocidad.
- En un nivel 2 el vehículo contaría con control longitudinal y lateral, aunque no cuenta con la capacidad de detección de objetos.
- Un grado de autonomía 3 si contaría con detección de objetos y el vehículo o el robot podría, por ejemplo, cambiar de carril o pararse quieto al ver una señal de Stop.
- En el nivel 4 ya tendríamos una conducción totalmente autónoma, pero en caso de fallo del sistema principal, el vehículo cuenta con respaldo para actuar.
- Por último, en un nivel 5, la figura del conductor no existiría. Nos subiríamos al vehículo y le indicaríamos el destino y este se pondría en marcha.

En este marco, el GII está desarrollando una línea de trabajo basada en el robot Robobo con el objetivo de estudiar la aplicabilidad real de estos modelos, utilizando para ello un entorno que simula un tramo de carretera a escala, tanto en simulación como en el mundo real (ver **Figura 52**). En este entorno se podrán estudiar problemas como la cooperación entre robots, la optimización de la autonomía, etc.

La conducción autónoma es un problema muy complejo que implica utilizar numerosos sensores en tiempo real, fusionar su información, y tomar decisiones que pueden poner en riesgo la vida de las personas. El primer problema a resolver para lograr una conducción autónoma robusta es detectar las líneas que delimitan el carril, de modo que el vehículo se pueda mover dentro del mismo. Para el caso de Robobo, MINT ha implementado una librería de alto nivel en el módulo de visión del robot que permite la detección de carriles en tiempo real.

Para ello, esta librería usa de base otra librería a bajo nivel que permite detectar todas las líneas presentes en la imagen. Partiendo de esta librería, se ha implementado un módulo capaz de realizar un filtrado de todas estas líneas horizontales para detectar exclusivamente la de los carriles.

En este TFG se pretende realizar la caracterización del módulo de alto nivel, y la creación de un control de movimiento capaz de mantener al robot dentro del carril en todo momento.

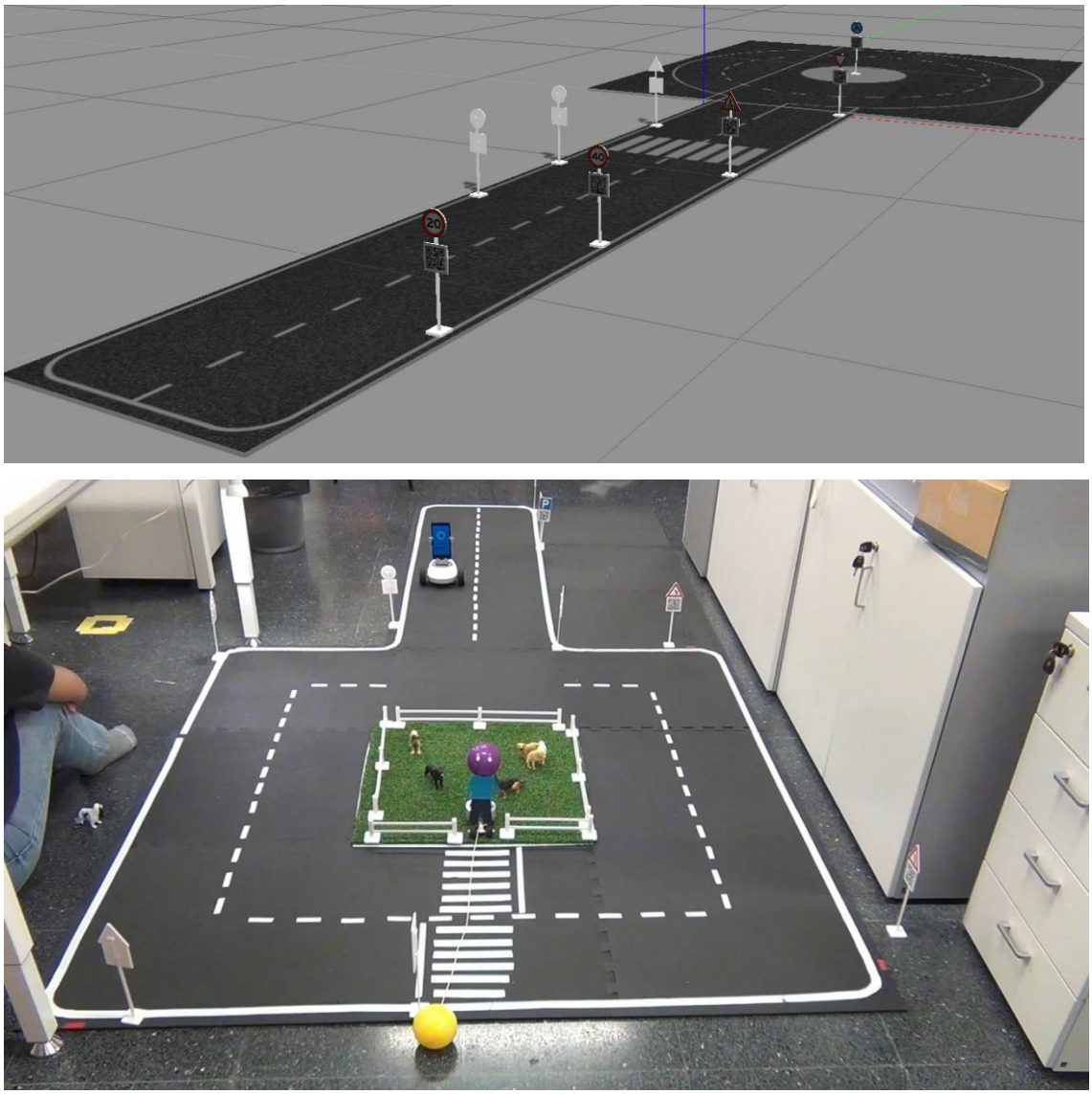

Figura 52. *Entorno de cuidad simulado (arriba) y real (abajo).*

5.2.1.1 Método para la detección de carriles

El módulo para la detección de carriles que MINT ha creado se puede llamar desde la librería en Python de Robobo usando el método *readLinePro()*, este método se encuentra disponible en el siguiente enlace con una wiki que explica su funcionamiento:

https://mintforpeople.github.io/robobo.py/

Para buscarlo se debe escribir “readLanepro” en la barra de búsquedas a la derecha de la página. El primer resultado se trata del método.

Este método devuelve 3 parámetros. Dos de ellos son diccionarios de Python y el otro es una matriz. Los diccionarios, llamados coeffs1 y coeffs2, contienen los parámetros de un polinomio de grado dos, es decir, los valores de “a”, “b” y “c” de la forma $ax^2 + bx + c$.

Los valores de coeffs1 corresponden al carril de la izquierda y los de coeffs2 a los de la derecha. Por su parte, la matriz se trata de una matriz de transformación, usada para transformar a la imagen en su forma original. Esto es así debido a que el método transforma la imagen entrante de la cámara en una perspectiva canónica (vista de

pájaro) para la extracción de carriles de la escena, y luego es necesario usarla si se quiere recuperar la perspectiva de la imagen original a la hora de pintar los carriles detectados sobre la imagen.

El método detecta los carriles de color blanco o amarillo sobre un fondo negro, pero también puede detectar el caso contrario, con un carril negro o amarillo sobre fondo blanco. Para lograr esto se utiliza el método *ToggleColorInversion()*, el cual recibe un booleano como entrada. Si es True, el método detectara líneas negras sobre blancas y si se encuentra a False, líneas blancas sobre negras.

Haciendo uso de OpenCV, y con los parámetros obtenidos, se pueden dibujar las ecuaciones de segundo grado que el método devuelve encima de la imagen original, para comprobar que la detección funciona correctamente. En la imagen de la **Figura 53** se observa al método en funcionamiento. En la imagen de arriba se observa a la imagen capturada por streaming con ambos carriles pintados gracias a los valores que el método devuelve (coeficientes de polinomio de segundo grado y matriz inversa de transformación), lo cuales se muestran en la imagen de abajo. Los valores de “coeffs1” son usados para pintar el carril de la izquierda (azul) y los de “coeffs2” el de la derecha (rojo).

PROBLEMS 18 OUTPUT TERMINAL ··· 2: Python

```
Lane, Id: <built-in function id>coeffs1:{'a': -0.002248810475757245, '
b': 1.1238394097568682, 'c': -128.3727435095285} coeffs2:{'a': -0.0031
57801722515817, 'b': 2.131639277795511, 'c': -65.1238254453947} minv:[
[0.4976359338061465, -0.20454545454545456, 72.0], [-0.0780141843971631
, 0.3114657210401891, 88.0], [-0.00022163120567375886, -0.001245836019
77219, 1.0]]
```

Figura 53. *Circuito pintado (arriba) gracias a los coeficientes devueltos por el método (abajo).*

Cuando el módulo de detección de carriles no detecta uno de los carriles devuelve un cero en todos los valores del diccionario “coeffs1” o “coeffs2”. Si, por ejemplo, es el carril izquierdo el que se deja de detectar, los tres coeficientes que devuelve “coeffs1” serán cero. El método realiza la detección en un área especificada de antemano (**Figura 54**). Se puede variar la forma de esta área de detección modificando sus coordenadas en el archivo *camera.properties* del *smartphone*, tal como se explica en la wiki del método.

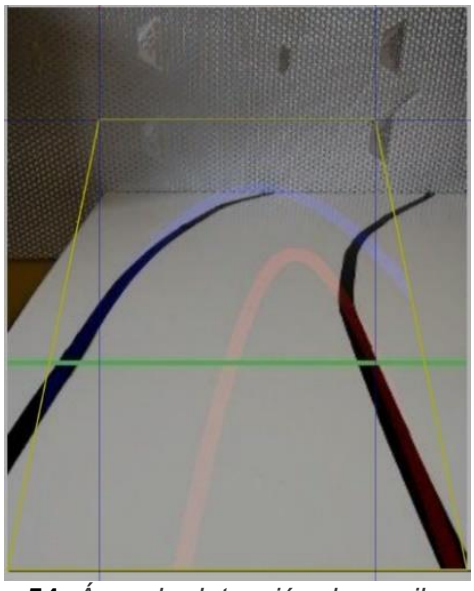

Figura 54. *Área de detección de carriles por defecto.*

5.2.1.2 Definición de las pruebas a realizar

Una vez explicado cómo funciona este método creado por MINT, analizaremos su funcionamiento para caracterizar los límites de detección del módulo. El procedimiento seguido para validar este algoritmo será analizar los coeficientes devueltos por el método y compararlos con la imagen pintada gracias a estos. Para ello, realizaremos unas pruebas que se pueden agrupar en:

- Análisis de los coeficientes devueltos con el robot parado
- Análisis de los coeficientes devueltos con el robot a distintas velocidades

En el primer tipo, realizaremos una prueba con el robot quieto, para analizar su comportamiento en un tramo recto, con una curvatura media y con una curvatura más pronunciada (rotonda). Con ello pretendemos conocer cómo responde ante distintos tipos de curva y analizar si los polinomios que el método devuelve se ajustan a la realidad.

Por otro lado, en las pruebas con el robot a distintas velocidades, escogeremos dos tipos de tramo:

- El tramo recto.
- El tramo con curva de la rotonda.

Para ello, lanzaremos al Robobo a distintas velocidades por cada uno de estos tramos y analizaremos su respuesta y comportamiento en cada situación. Con esta prueba queremos caracterizar su respuesta en función de la velocidad.

Antes de la realización de las pruebas, un primer punto a tener en cuenta es el valor del Tilt a usar. Se ha comprobado que el módulo no es capaz de detectar correctamente los carriles por encima de un Tilt de 90. Por dicho motivo, las pruebas se realizarán con un Tilt de 100. Este valor es con el que mejor responde la detección, ya que muestra el circuito entero, así como posibles objetos en la escena y se encuentra dentro del rango de 90-105 donde el método es capaz de efectuar correctamente la detección de carriles. A continuación, se muestran las pruebas realizadas:

5.2.2 Pruebas de caracterización de la librería de detección de carriles en tiempo real

5.2.2.1 Análisis de los coeficientes devueltos con el robot parado

Para la realización de dichas pruebas se colocó al Robobo en un cada uno de estos tramos y se analizó su detección. Para cada medida se guardó en un archivo TXT los datos devueltos por el método *readLinePro()* y se utilizó estos datos (coeficientes de una ecuación de segundo grado para cada carril y matriz de transformación inversa) para representar ambas curvas en la imagen de entrada.

Esta imagen pintada nos muestra de forma gráfica si el método es capaz de reconocer ambos carriles, uno o ninguno, en cada una de las situaciones planteadas. La **Tabla 3**, a continuación, muestra los coeficientes de la ecuación de segundo grado ("a", "b" y "c") para ambos carriles que el método *readLinePro()* nos devuelve.

Tipo de tramo	Coeficientes	Carril izquierdo	Carril derecho
Tramo recto	a	-0,0003	0,00659
	b	-0,13068	-3,16492
	c	74,62052	373,14513
Tramo con curva suave	a	0,00659	0,00172
	b	-3,16491	-0,82935
	c	373,14513	335,44032
Tramo con curva cerrada	a	-0,00224	-0,00315
	b	1,12383	2,13163
	c	-128.37273	-65,12382

Tabla 3. *Variación de la detección en función de la curvatura de la curva.*

En la figura a continuación (**Figura 55**) mostramos la representación de las ecuaciones de segundo grado de la **Tabla 3** pintadas encima de la imagen de entrada para visualizar la detección de forma más gráfica. En la imagen más a la izquierda vemos un tramo recto, con ambos carriles detectados sin problemas. En la imagen del medio es el tramo con una curva suave y el de la derecha, el tramo con curva de rotonda.

Figura 55. *Tramos con distinta curvatura con los carriles pintados.*

Observamos, analizando tanto los coeficientes de la **Tabla3**, como las imágenes de la **Figura 55**, que los polinomios devueltos por el método se adaptan sin problemas a la forma real del carril. En el tramo con una curva más abierta (**Figura 55**, imagen del medio),

5.2.2.2 Análisis de los coeficientes devueltos con el robot a distintas velocidades

Una vez analizado su respuesta ante distintos tipos de curva, se realizó un circuito con un tramo recto y el tramo curvo usado en la derecha de la **Figura 55**. Con estos dos tramos se varió la velocidad del Robobo y se analizó su comportamiento en cuanto detección. Para ello se usó el método de Robobo *moveWheels()*, el cual posee un rango de potencias entre -100 y 100. Los datos devueltos por el método en el tramo recto se mostrarán en la **Tabla 4** y los devueltos por el tramo curvo en la **Tabla 5**.

Potencia Robobo [-100, 100]	Coeficientes	Carril izquierdo	Carril derecho
10	a	-0,00025	0,00182
	b	-0,09464	-0,35717
	c	99,61821	229,01045
20	a	-0,00046	0,00062
	b	-0,12443	0,00994
	c	95,25327	221,93034
30	a	-0,00067	0,00032
	b	0,00519	0,21959
	c	93,99954	181,12094
40	a	-9,75E-05	0,00115
	b	-0,20689	-0,07992
	c	103,91006	218,52455
50	a	-0,00036	-0,00041
	b	-0,05887	0,55733
	c	114,05994	162,69328
60	a	0,00074	0,00090
	b	-0,00435	-0,16813
	c	77,78033	237,26366
70	a	-0,00065	0,00013
	b	-0,03542	0,29419
	c	85,56617	174,42363
80	a	-0,00135	-0,00022
	b	0,31929	0,32444
	c	82,46005	202,50999
90	a	-0,00065	0,0012818
	b	0,05087	-0,26857
	c	80,39137	232,643815
100	a	-0,00023	0,00150
	b	-0,18361	-0,38220
	c	98,73975	234,03333

Tabla 4. *Variación de la detección en función de la velocidad – tramo recto.*

A continuación, se muestra en la **Figura 56** la imagen pintada con los valores devueltos por el método en el tramo recto, para una potencia de 10, 50 y 100. Tal como se observa en la **Tabla 4**, independientemente de la velocidad del Robobo, el módulo detecta ambos carriles. Estos tienen un valor en el coeficiente "a" muy pequeño, indicando que la curvatura de las ecuaciones de segundo grado la hace verse como una recta (**Figura 56).** En los datos de la **Tabla 4**, también podemos ver como cuando el coeficiente "a" del polinomio es negativo, la curva gira hacía la izquierda y cuando es positivo hacia la derecha. Por ejemplo, tomando la medida realizada a una potencia de 50, en la **Tabla 4**, observamos como ambos coeficientes "a" son negativos y de un orden de 10^{-4}. Estos nos indica que el polinomio ajusta correctamente su forma al de una recta y ambas giran ligeramente hacía la izquierda, como se puede observar en la imagen del medio de la **Figura 56**.

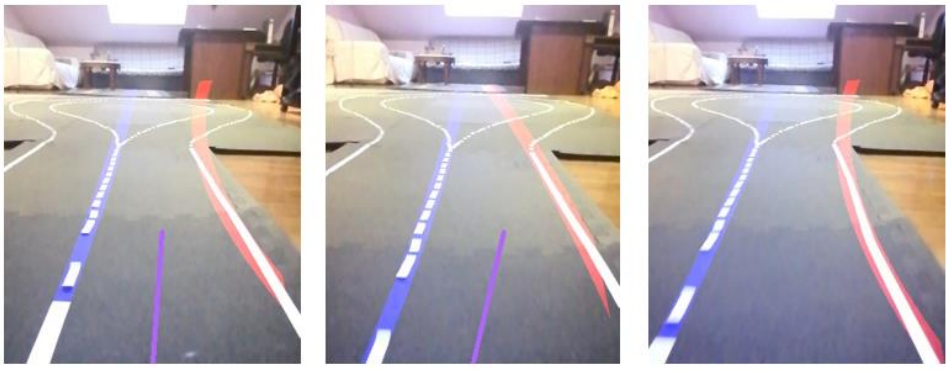

Figura 56. *Tramo recto a potencias 10,50 y 100 (de izquierda a derecha).*

Por su parte, en la medida realizada a una potencia de 100, la curva de la derecha gira hacia la derecha al poseer un valor del coeficiente "a" positivo y posee una curvatura más pronunciada que la otra curva del carril debido a que el valor de su coeficiente "a" es mayor (del orden de 10^{-3}). Como vemos, la velocidad del Robobo no afecta a la detección de ambos carriles si estos son vistos de frente. Pero, ¿y si es una curva? En la tabla a continuación (**Tabla 5**), se mostrarán los datos obtenidos por el método *readLanePro()* para distintos valores de potencia del motor. Para realizar estas medidas, el Robobo arrancó siempre desde un mismo punto y atravesó la curva obteniendo las mediciones en el mismo punto de esta.

Potencia Robobo [-100,100]	Coeficientes	Carril izquierdo	Carril derecho
10	a	0,00502	0,00042
	b	-3,13995	0,20513
	c	483,93066	245,93042
20	a	0,00343	-0,00078
	b	-2,57620	0,41216
	c	470,22546	221,92872
30	a	0,00458	0,00016
	b	-2,98044	-0,10115
	c	476,96021	273,46071

40	a	0,00380	-0,00028
	b	-2,73928	0,05253
	c	464,988724	264,48318
50	a	0,00345	0,00269
	b	-2,63602	-1,66550
	c	490,04459	530,67709
60	a	0,00482	0,00102
	b	-3,05128	-0,51971
	c	477,61479	318,57416
70	a	0,00393	-0,00020
	b	-2,56368	0,12257
	c	407,32328	250,32947
80	a	0,00360	0,00138
	b	-2,44999	-0,66004
	c	412,64149	337,96876
90	a	0,00455	0,00197
	b	-2,81838	-0,93268
	c	436,16242	355,8823
100	a	0,00316	0,00400
	b	-1,90650	-1,75274
	c	290,36685	431,87328

Tabla 5. *Variación de la detección en función de la velocidad – tramo curvo.*

Como podemos observar en la **Tabla 5**, a todas las velocidades a las que se lanzó el robot, este fue capaz de detectar ambas curvas. Por lo tanto, la velocidad del Robobo no afectan a la detección de los carriles curvos. Como se comentó anteriormente, cuando el método *readLanePro()* no detecta un carril devuelve ceros en sus coeficientes. Hecho que no sucede en la **Tabla 5**. En las imágenes a continuación (**Figura 57**), se observa su detección a una potencia de izquierda a derecha de 10,30, 60, 80 y 100. Como podemos observar en dicha figura, la luminosidad afecta en gran medida a su detección. Las curvas tienden hacia un punto brillante al fondo de la imagen, que se trata de un objeto de color gris. Observamos que aunque no detecte directamente objetos de color blanco u amarillo, el filtrado que el método *readLanePro()* realiza en ocasiones es muy sensible al brillo. Vemos también en la **Figura 57**, que la línea de la derecha (línea roja) no se adapta de forma perfecta a la curva y, de hecho, en la **Tabla 5**, su polinomio es cercano al de una recta, por ejemplo, las medidas realizadas a potencia 30 o 40.

Otro factor de gran importancia es el área de detección de la **Figura 53**. Si alguno de los carriles en la imagen se encuentra fuera o parcialmente fuera, el método devolverá cero en todos sus coeficientes. Por dicho motivo, era interesante caracterizar si la velocidad afectaba a la detección partiendo de que ambos carriles se encuentran siempre dentro de esta área.

Concluimos que el método es capaz de detectar a ambos carriles, adaptándose de forma correcta la forma del polinomio mientras su medición no se vea perturbado por ruido externo. Por otro lado, es muy sensible a este ruido, especialmente el debido al brillo en la escena.

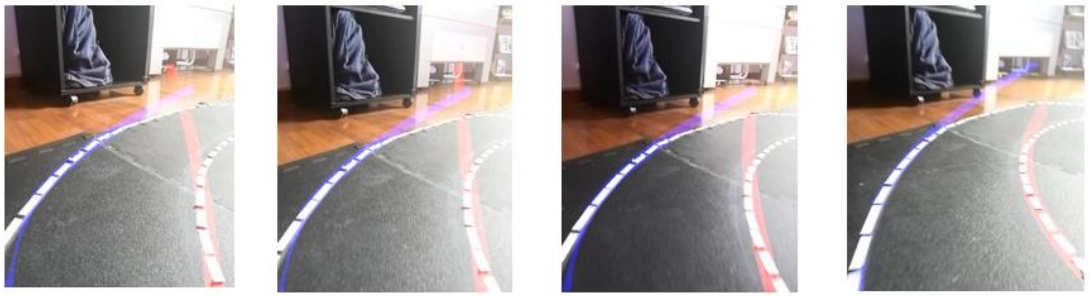

Figura 57. *Tramo curvo a potencias 20,30,50 y 80 (de izquierda a derecha).*

Ahora que hemos estudiado su respuesta tanto con el robot parado, como con el robot en movimiento a distintas velocidades. Podemos concluir que la detección ocurre sin problemas a cualquier velocidad independientemente de tratarse de un tramo recto o curvo. Sin embargo, observamos que el ruido en la escena afecta en gran manera a la detección errónea.

Tras este análisis del comportamiento del módulo, se quiso implementar un control de movimiento que permitiera que el Robobo recorriese el carril de forma autónoma, adaptándose al tipo de tramo o curva en función de los coeficientes devueltos por el módulo de detección de carriles que MINT ha implementado.

5.2.3 Control de movimiento en un carril

Para la realización del control de movimiento es necesario que el Robobo se mantenga dentro de ambos carriles en todo momento. Para lograrlo, lo ideal es que trate de orientarse hacia el centro del mismo.

Para realizar el control de movimiento necesitamos orientar a nuestro robot hacia el centro del carril, para lograrlo calcularemos el error como la diferencia entre el punto medio del carril en cada momento (punto variable) y el punto medio de la imagen (punto fijo). La imagen devuelta dependerá de la cámara que use nuestro smartphone, en nuestro caso, el HUAWEI P Smart 2019 que estamos usando para las pruebas, produce una imagen de 352 x 288 pixeles, por lo tanto, el punto medio de la imagen estará siempre a 144 pixeles. En la representación de la imagen de la cámara del smartphone, el origen del centro de coordenadas se encuentra en la esquina superior izquierda como se ve en la imagen de la

Figura 58**.**

En esta misma figura se puede observar una representación de la imagen que capta la cámara de nuestro smartphone. Como se comentaba anteriormente, el punto de color violeta es un punto fijo siempre con su coordenada “x” igual a la mitad del ancho de la imagen y su coordenada “y”, igual al largo de la imagen, en este caso 352. La razón de esta posición es porque la cámara frontal de nuestro smartphone se encuentra centrada en el centro del Robobo, y, por lo tanto, esa vertical indica la dirección de avance de nuestro robot cuando se mueve en línea recta, tal y como se ve en la **Figura 59**

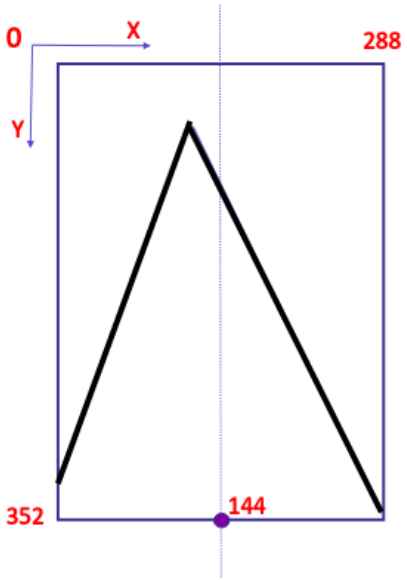

Figura 58. *Representación de los carriles en la imagen.*

Para encarar correctamente la dirección del Robobo y que no se salga del carril, tomamos la distancia media del carril a una cierta distancia en el eje “y” desde la parte de abajo de la imagen (y = 352). Al poseer la ecuación de ambas curvas podemos obtener de forma inmediata sus valores “x” dada una altura “y”. Por lo tanto, a una altura de (352 – h), obtenemos el valor (x1 y x2) de dichos puntos, representados en la imagen de la **Figura 60** como dos puntos de color naranja. Teniendo estos dos puntos, obtener el punto medio del carril (punto amarillo) es cuestión de restar el punto de corte del carril de la derecha al de la izquierda y dividirlo entre 2. Las dos líneas negras indican cómo ve un carril recto el robot, dichas líneas se van juntando hasta que se cortan en un punto.

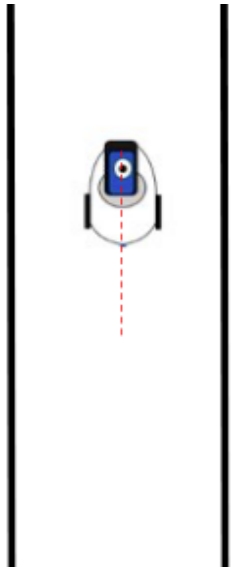

Figura 59. *Esquema de la dirección del Robobo.*

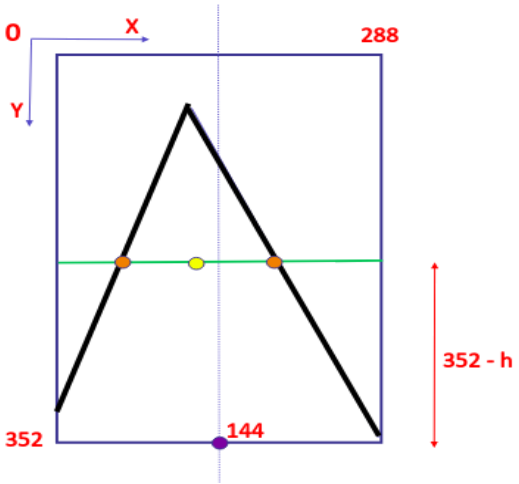

Figura 60. *Esquema del cálculo del punto medio del carril en la imagen.*

Como se observa (**Figura 60**), el punto amarillo está situado más a la derecha que el punto de color violeta, lo que indica que nuestro Robobo se encuentra desplazado un poco a la derecha del centro del carril. Con esta información en tiempo real es posible calcular nuestro error como $e = x - x'$.

Siendo *x* el punto medio del carril en cada instante de tiempo y *x'* la coordenada de la mitad de la imagen (144 pixeles). El control de movimiento tratará de ajustar la dirección para hacer coincidir los valores de *x* y *x'* en la medida de lo posible.

Sin embargo, se consideró que tomar como centro del carril únicamente un píxel podía dificultar de sobremanera un control de movimiento que se ajustase correctamente. Por ello se probó con un margen de +- 20 pixeles y con otro de +-10 desde el píxel considerado como centro del carril. A continuación, (**Figura 61**) se muestra en la ilustración de la izquierda como se vería un rango de +- 10 pixeles desde el centro de la imagen, y en la de la derecha un rango de +- 20 píxeles.

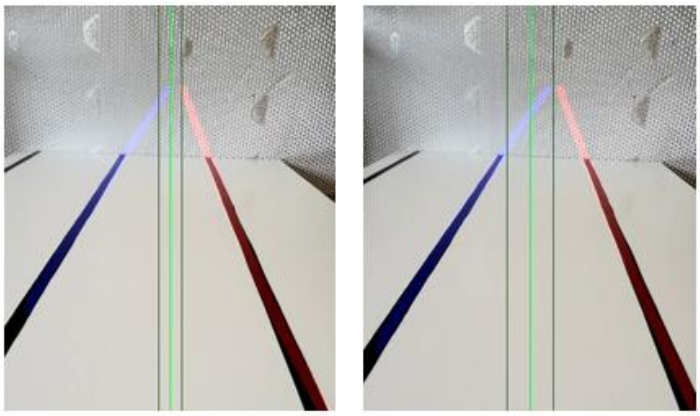

Figura 61. *Área del centro del carril para +-10 píxeles (izquierda) y para +-20 (derecha).*

En las pruebas realizadas, que se detallarán más adelante, se probó con ambos márgenes, siendo el margen de +- 20 peor al de 10, por dicho motivo se utilizó desde ese momento ese rango en el script que se ha creado. Sin embargo, se puede cambiar si se desea. Si el error es cero o cercano, nos encontramos en el rumbo correcto, si el error es positivo necesitamos girar para la izquierda para corregir el rumbo y si el error es negativo a la derecha.

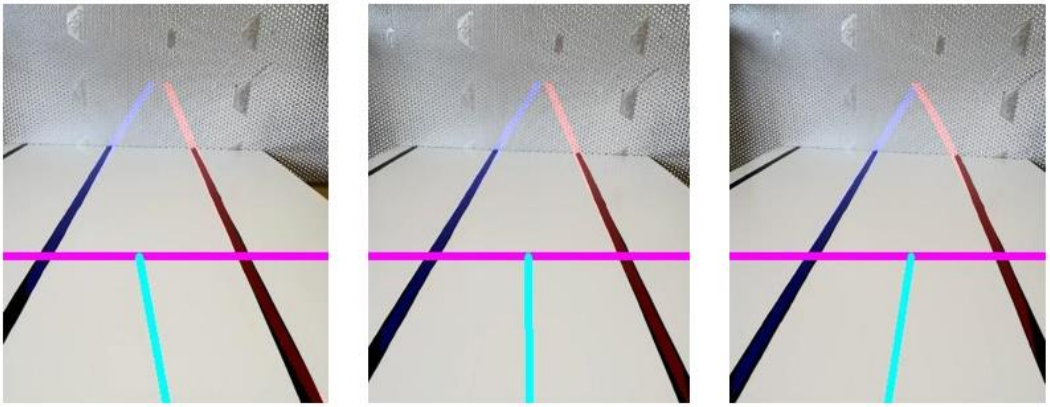

Figura 62. *Los tres casos de dirección en que puede orientarse el Robobo.*

En la imagen de arriba (**Figura 62**), se puede observar de forma gráfica como con los datos que devuelve el método *ReadLanePro()* podemos dibujar ambos carriles (el de la derecha en rojo y el de la izquierda en azul). A modo explicativo, la línea rosa representaría la altura a la cual se está considerando el centro del carril. Por último, la línea azul claro representa la recta que une el punto medio del carril con el punto medio de la imagen como ya hemos explicado anteriormente.

En la imagen de la izquierda se observa como la línea está inclinada hacia la izquierda, lo que indica que se debe girar a la derecha de forma proporcional a su distancia al centro del carril. En la imagen del centro, ambas líneas se encuentran en el mismo vertical o casi en ella, lo que le indica a nuestro script que debe mantener ese rumbo. Por último, la imagen de la derecha representa el caso en donde la línea de unión está inclinada hacia la derecha, por lo que se debe girar proporcionalmente a la izquierda. El Robobo cuenta con dos ruedas en los laterales que pueden moverse usando el método de la librería en Python de Robobo *moveWheels(),* que como ya hemos comentado anteriormente, permite un rango de potencias de -100 a 100. Este método recibe dos enteros, uno para la rueda de la derecha y otro para el de la izquierda. Si ambas ruedas reciben la misma potencia, el Robobo avanzará en línea recta. Pero si una de las ruedas recibe más potencia, el robot girada hacía ese lado. La pregunta que surge aquí es cuánto debe ser el aumento de velocidad en una rueda u otra para corregir de forma proporcional al error producido en cada iteración de tiempo.

Para solucionar esto se utiliza una estrategia basada en un controlador PID, por eso, antes de nada, explicaremos que es un PID y cómo funciona.

5.2.3.1 Controlador PID

El controlador PID [29] es uno de los más empleados en la industria para el control de sistemas realimentados. Algunas de sus fortalezas son su sencillez y que es capaz de dar un buen comportamiento en una gran variedad de situaciones sin necesidad de conocer con detalle la planta a controlar.

El algoritmo PID (proporcional, integral, derivado) está formado por la suma de tres componentes**,** Proporcional, Integral, y Derivativa. Matemáticamente, un controlador PID tiene la forma $Output(t) = Kp\,.\,e(t) + Ki\,.\,\int_0^t e(t)\,dt\, + Kd\,.\,\left(\frac{de}{dt}\right)$

Cada componente del PID es "independiente" de los demás, en el sentido de que cada uno calcula una salida de lo que "para el" deberías hacer para obtener la respuesta adecuada. Los tres componentes se suman para dar la salida del controlador. Cada uno cumple una cierta función y mejoran cierta parte de la respuesta. Y cuando los tres componentes trabajan juntos, en la proporción adecuada, consiguen un control satisfactorio (**Figura 63**).

Cada componente tiene un parámetro *Kp, Ki y Kd*, respectivamente. Estos parámetros indican la ponderación (o "la fuerza") que tiene en el resultado final. Que la respuesta del PID sea buena, reside en el ajuste correcto de estos tres parámetros.

En la respuesta global del controlador los tres componentes trabajan juntos e influyen uno sobre otros, por lo que no vale con ajustar cada uno de los parámetros de forma independiente. Existe una cierta "zona" dentro de los tres parámetros, donde el comportamiento es más o menos bueno. La dificultad de un PID es ajustar los parámetros *Kp, Ki y Kd*, para que el comportamiento sea bueno.

- El componente proporcional reacciona al presente
- El componente integral reacciona al pasado, y aporta "memoria" al controlador.
- El componente derivado reacciona al futuro, y aporta "predicción" al controlador

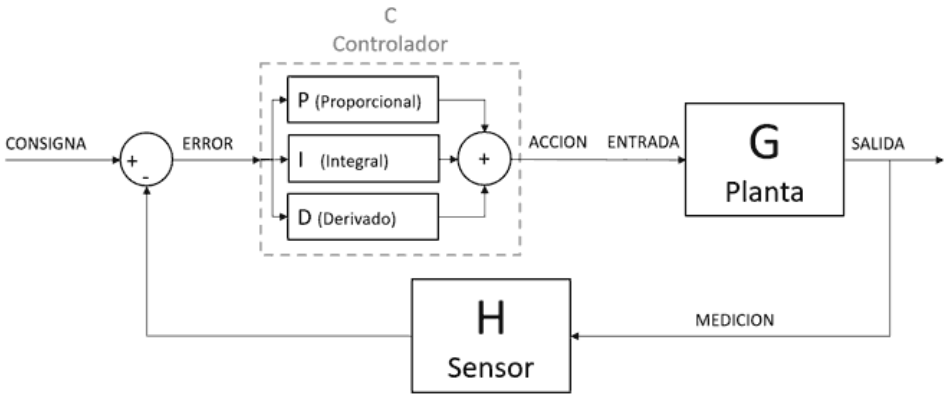

Figura 63. *Funcionamiento del PID.*

5.2.3.2 Implementación del script

Por lo tanto, lo primero que debemos hacer es implementar un algoritmo que simule el funcionamiento de un controlador PID para nuestro caso. El script está escrito en Python. Para las pruebas realizadas se probó en un primer momento con un control P, es decir, un control proporcional, que ajustara el valor de salida del incremento de la rueda en función del error existente. El algoritmo de un control proporcional sería el siguiente:

```
#cálculo del error
mitad = int(width/2)    #punto medio de la imagen

ep = x_medio - mitad

#Control proporcional
incr = ep * kp
```

Para realizar el movimiento se utiliza el método moveWheels(), el cual requiere de dos enteros, uno es la potencia de la rueda derecha y el segundo de la rueda izquierda. En función del valor del error se sabe si el Robobo esta desviado a la derecha, si está desviado a la izquierda o si debe mantener el rumbo. Los valores de potencia y el rango +- píxeles puede variarse si se desea.

```
#Parámetros
vel = 5
rang = 10
#control de velocidad
if ep > rang:
  rob.moveWheels(vel+incr, vel)
  print('\nTURN TO THE LEFT\n')
elif ep < -rang:
  rob.moveWheels(vel, vel + incr)
  print('\nTURN TO THE RIGHT')
else:
  rob.moveWheels(vel,vel)
  print('\nSTAY ON CURSE\n')
```

Si el valor del incremento es menor que 0, el valor se hace cero. Por otro lado, no nos interesa que el incremento pueda aumentar muy bruscamente la velocidad del Robobo o de una rueda en concreto, por ello hemos usado un limitador de velocidad que restringe el valor máximo de salida de esta variable. La velocidad máxima por encima de la actual se ha establecido en 15, pero puede variarse.

```
#Limitador del valor de salida
if incr > 0:
  incr= 0
elif incr > vel_max:
  incr = vel_max
```

Por otro lado, el método moveWheels recibe como parámetros dos enteros, el valor que se calcula del incremento de velocidad suele ser un float, por lo tanto, es necesario redondear a la unidad más cercana antes de introducirlo en el método moveWheels. Para ello se pasa este valor de incremento por una función que lo redondea a la unidad más cercana.

```
def round_unit(num):

    neg = False
    if num < 0:
      num = abs(num)
      neg = True
    elif (num - int(num)) > 0.5:
      num = int(num) +1
    else:
      num = int(num)
    if neg:
      num = -num

    return num
```

Para ejecutar el control de movimiento, el cálculo del error y el incremento deben realizarse dentro un bucle while, donde a cada iteración de tiempo se reciba un nuevo

valor del punto medio del carril. Lo primero de todo es obtener el valor del punto medio del carril y dibujar en la imagen de entrada ambos carriles. Para ello se llama al nuevo método que ha creado MINT, readLanePro, el cual nos devuelve los coeficientes de ambas curvas y una matriz de transformación.

```
def draw_line_pro(coeffs1, coeffs2, M, frame, h=130):

    # Obten la atura y el ancho de la imagen
    try:
        height, width, _ = frame.shape
        # Máscara para pintar las lineas
        mask = np.zeros_like(frame)

        # Cra un array con números del 0 al height -1
        plot_y = np.linspace(0, height - 1, height)

        # Arrays con evaluación de los coeficientes
        left_x = coeffs1['a'] * plot_y  2 \
                + coeffs1['b'] * plot_y \
                + coeffs1['c']
        right_x = coeffs2['a'] * plot_y  2 + \
                coeffs2['b'] * plot_y + \
                coeffs2['c']

        # Pinta las líneas (una roja, otra azul)
         cv2.polylines(mask, [np.int32(np.stack((left_x,plot_y), axis=1))], F
      alse, (255, 0, 0), 20)
        cv2.polylines(mask, [np.int32(np.stack((right_x, plot_y), axis=1))],
False, (0, 0, 255), 20)

        # Warpea la perspectiva
        mask = cv2.warpPerspective(mask, np.float32(M), (width, height))

        # Añade las líneas al frame original
        img = cv2.addWeighted(frame, 1., mask, 0.5, 0)
        x_r, x_l = right_x[height - h], left_x[height -h]

        #Calcula el punto medio
        x_medio = ((x_r - x_l)/2) + x_l
        #Línea de dirección
        cv2.line(img, (int(width/2), height- 1), (int(x_medio),height-
h), (255,102,178), 3)

    except ValueError:
        pass

    return img, x_medio
```

Para obtener la imagen del streaming, es necesario usar el módulo de streaming que MINT ha creado. Una vez creada una instancia de la clase *RoboboVideo*, ya

podemos llamar al método *getImage()*, el cual nos devuelve el frame captado por la cámara del smartphone. Este fotograma se manda como parámetro de la función *draw_line_pro* que acabamos de implementar. La imagen obtenida es mostrada usando el método *imshow* de OpenCV.

```
IP = '192.168.0.17'
rob = Robobo(IP)
rob.connect()
video = RoboboVideo(IP)
video.connect()
rob.setLaneColorInversion(False)

while True:
  frame = video.getImage()
  #obtengo los coeficientes de los carriles detectados y su matriz inversa
  obj = rob.readLanePro()

  frame, x_medio = draw_line_pro(obj.coeffs1, obj.coeffs2, obj.minv, frame)

  #Aqui se calcularía el error,
el incremento y se limita su valor y redondea.
Todo lo explicado anteriormente.

  cv2.imshow('smartphone camera', frame) #mostramos la imagen pintada

  #Si se pulsa la tecla 'q' finaliza el script
  if cv2.waitKey(1) & 0xFF == ord('q'):
    rob.stopMotors()
    video.disconnect()
    cv2.destroyAllWindows()
    break
```

Con todo este código ya se puede implementar un control proporcional donde únicamente es necesario ajustar el valor del parámetro kp de forma correcta. Sin embargo, es muy posible que, para un tramo con curvas, y especialmente si estas son pronunciadas, un control únicamente proporcional no valga. Sería necesaria implementar la parte integral, que tiene en cuenta el error pasado, como vimos en el apartado de teoría de PID. Para implementar un control proporcional-integral es necesario que guardemos los errores e incrementos pasados, por ejemplo, en una lista. En la primera iteración el error calculado será puramente proporcional, pero desde ahí se le restará a dicho valor, el error de la iteración anterior multiplicado por el coeficiente ki (parámetro control integral), y se le sumará el valor del incremento de la iteración anterior. Es necesario, además, crear un contador que haga que en la primera iteración el control sea solo proporcional, y a partir de ahí, sea proporcional-integral.

```
lista_ep = []
lista_incr = []
cont = 0
#cálculo del error
```

```
ep = x_medio - mitad
lista_ep.append(ep)
incr = abs(ep)*kp
lista_incr.append(incr)

if i >= 1:
  #control proporcional + integral
  incr = ep - lista_ep[-1] * ki + lista_incr[-1]
  lista_incr.append(incr)

cont+=1
```

Para evitar un desbordamiento de las listas, hacemos que cada vez que se acumulan 50 datos se borre el elemento más antiguo de dichas listas.

```
if cont > 50:
  lista_ep.pop(0)
  lista_incr.pop(0)
```

Para solventar el caso límite donde se pierde uno de los dos carriles de vista (bastante frecuente en curvas cerradas), el script mirará si los tres coeficientes de la curva son cero. En caso afirmativo se procederá a girar hacia el lado donde se debería encontrar el carril perdido hasta volver a dar con él.

```
#CONTROL DE VELOCIDAD

    if coeffs1['a'] == 0
        #acum = 0
        print(coeffs1)
        print('\ncarril izquierdo perdido\n')
        print('\nDebo girar a la derecha\n')
        #rob.moveWheels(vel , vel + 15)
        #x_medio =

    if coeffs2['a'] == 0
        #acum = 0
        print(coeffs2)
        print('\ncarril derecho perdido\n')
        print('\nDebo girar a la izquierda\n')
        #rob.moveWheels(vel + 15, vel  )

    if coeffs1['a'] != 0  and coeffs2['a'] != 0:
        print('Carriles detectados')
        #acum = 0
        #rob.moveWheels(vel,vel)

        if ep > rang:
            #rob.moveWheels(vel+incr,vel)
```

```
            print('TURN TO THE LEFT\n')
            print(f"\nRight wheel speed --> {text}\n")
            vel_right = vel +incr
        elif ep < -rang:
            #rob.moveWheels(vel,vel+incr)
            print('TURN TO THE RIGHT\n')
            print(f"\nLeft wheel speed --> {text}\n")
            vel_left = vel + incr
        else:
            #rob.moveWheels(vel,vel)
            print('\nSTAY ON COURSE\n')
```

Se encontró que el control mostraba un mejor comportamiento si el código anterior se encargaba de reorientar al Robobo en caso de perder un carril usando una velocidad de giro que varía en función de la velocidad a la que se mueve el robot. Una vez este volviese a ver ambos carriles, se aplicaría el control proporcional-integral mostrado en la página anterior.

El código entero y comentado se encuentra en el siguiente repositorio de GitHub:

https://github.com/GII/robobo_vision/tree/master/Control-Movimiento-PID

5.2.3.3 Pruebas de funcionamiento del controlador

En este apartado se realizará una comprobación del funcionamiento del control de movimiento dentro del carril, para lo cual se hace uso de la librería de detección de carriles. Por tanto, si el control funciona correctamente, estaremos logrando el objetivo de validar el funcionamiento de la librería en tiempo real.

Las pruebas se realizaron en un entorno como el que se muestra en la **Figura 64**. En esta figura se observa un circuito consistente en un tramo recto unido a rotonda que actúa como cambio de sentido. El circuito tiene tres partes bien diferenciadas. El tramo recto, el tramo curvo más suave a la entrada de la rotonda y la propia curva de la rotonda.

Tal como se comentaba en el apartado anterior (**5.2.3.2**), finalmente se usó un control proporcional-integral con unos valores de kp y ki de 0,15 y 0,97 respectivamente. En este circuito, lo más complicado de lograr era que el robot fuese capaz de adaptarse girando dentro del tramo de curva. Finalmente, con el control implementado se logró dicho cometido. Este circuito final, es el resultado de pruebas anteriores realizadas en circuitos más simples para ir ajustando su comportamiento debido a la sensibilidad que muestra la librería de detección de carriles ante la iluminación y el brillo.

Las siguientes imágenes a continuación muestran su comportamiento en un tramo recto (**Figura 65**) y un tramo curvo (**Figura 66**), en el circuito de la **Figura 64**. Las imágenes han sido extraídas de un vídeo con el robot grabado en funcionamiento. En dicho video se puede ver por un lado (parte derecha del video), una visión externa en donde se ve al Robobo moviéndose, y por el otro lado (parte izquierda del vídeo), lo que el robot está viendo con el streaming activado.

Figura 64. *Circuito con cambio de sentido completo.*

En el tramo recto (**Figura 65**), el control funciona sin ningún problema. Ya desde la primera versión de circuito que se utilizó (circuito recto), se había logrado su ajuste simplemente con un control proporcional. Al utilizar la pérdida de un carril para girar y reorientarse, ocasiona que el control en rectas sea en ocasiones ligeramente más brusco que el usado en la primera versión. Sin embargo, en conjunto, el control de movimiento actual es más robusto que aquella versión y permite adaptarse a curvas de distintas curvaturas.

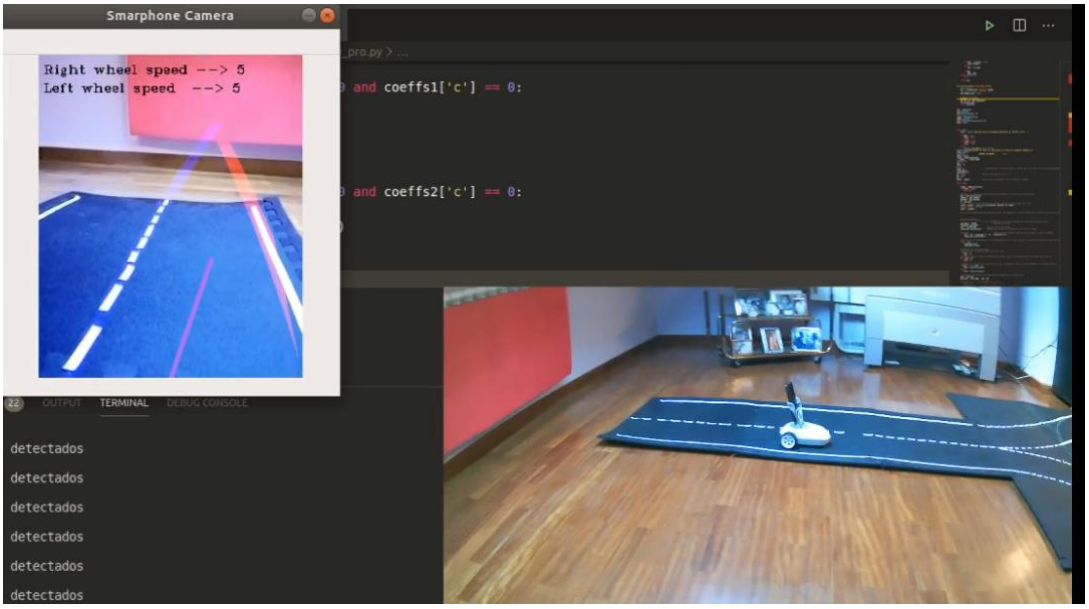

Figura 65. *Robobo moviéndose en un tramo recto.*

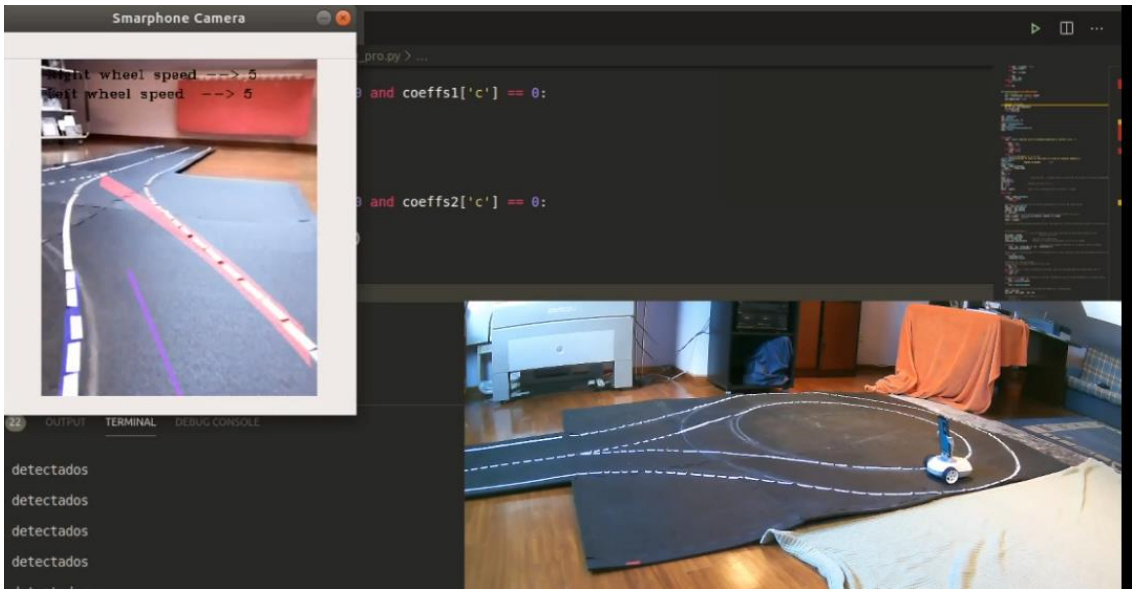

Figura 66. *Robobo moviéndose en un tramo curvo.*

Por su parte, en la **Figura 66** vemos como se traza correctamente la curva de la rotonda. Al ejecutar ese tramo de curva, el carril derecho había desparecido momentáneamente y el control reorientó al Robobo hasta que volvió a ver ambos carriles en la curva. Momento captado en la imagen de arriba (**Figura 66**). Mientras la iluminación de la escena no confunda al módulo de detección de carriles que se ejecuta por debajo, el control responde de forma correcta.

5.2.4 Conclusiones de la caracterización de la librería de detección de carriles en tiempo real

Una vez analizado el comportamiento de la librería tanto en pruebas de detección como en su uso práctico en el control del robot, podemos sacar las siguientes conclusiones:

- La detección ocurre sin problemas a cualquiera de las velocidades de las que dispone el Robobo. Al ajustar la curva a un polinomio de grado dos, la forma de esta curva no se podrá adaptar de forma perfecta a todo tipo de curvas, pero logra un buen comportamiento en los circuitos en los que se probó.

- La detección de carriles es muy sensible al ruido en la escena, los objetos de color blanco o amarillo, así como cualquier superficie que pueda brillar debido a la luz son altamente proclives para confundir al módulo. Por ello se recomienda encarecidamente que el entorno del carril se encuentre lo más aislado posible de estos colores y tenga una iluminación constante y uniforme que permita al robot ver en todo momento el circuito, pero no produzca que ciertas superficies saturen.

- El área de detección posee unos valores predeterminados que se encuentran en el archivo camera.properties, en la carpeta properties de la raíz del smartphone. Estos valores pueden cambiarse editando este archivo para lograr un área que se adapte mejor a nuestro circuito en particular.

- El control no solo adapta su orientación en el carril en base a como de lejos se encuentre del centro del mismo, sino que además en caso de perder un carril de

vista en algún momento (curva cerrada, por ejemplo), el Robobo se reorientará hasta volver a ver ambos carriles.

- El script cuenta con dos modos de ejecución: con streaming y sin él. El primero muestra lo que está viendo el Robobo, con los carriles detectados pintados en la imagen, así como la línea que muestra la dirección que toma el robot en todo momento. Además, se muestra la velocidad de ambas ruedas (izquierda y derecha). Por su parte, la ejecución sin streaming muestra las decisiones de orientación que el Robobo va tomando por pantalla, así como la velocidad de cada rueda en tiempo real. El problema de la ejecución con streaming es su lentitud. Por lo general, el control puede verse ralentizado en cuanto su toma de decisiones por culpa de la bajada de latencia al mostrar el streaming con todos estos elementos pintados. La ejecución sin streaming por su parte, es algo más rápida y fluida, pero no es posible visualizar que está viendo el Robot, lo que lo hace menos intuitiva a la hora de encontrar algún comportamiento anómalo.

5.3 Detección de marcadores avanzados (ArUcos markers)

5.3.1 Introducción

Dentro del marco de la visión por ordenador, es de vital importancia la estimación de la posición, desde realidad aumentada hasta navegación de robots en un entorno, normalmente, en interior. Sin embargo, este proceso es complicado de lograr en la vida real ya que requiere de encontrar correspondencias entre puntos en el entorno real y su proyección en una imagen 2D. De ahí la importancia de utilizar marcadores artificiales que sean fácilmente detectables con independencia de la orientación del robot. Uno de los tipos más habituales son los marcadores cuadrados binarios, ya que gracias a sus cuatro esquinas es posible estimar de forma correcta la posición de la cámara, y al encontrarse codificados de forma binaria, es más sencillo aplicar técnicas de detección de objetos y corrección de errores lo que los hace más robustos.

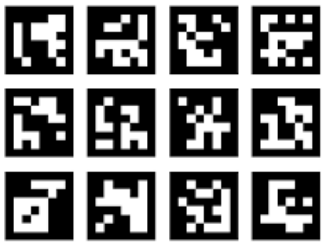

Figura 67. *Marcador ArUco.*

Con esta idea, Sergio Garrido y Rafal Muñoz crean en 2014 la biblioteca ArUco [30], una biblioteca popular para la detección de marcadores fiduciales cuadrados. Un marcador ArUco (**Figura 67**) es un marcador cuadrado compuesto por un borde negro ancho y una matriz binaria interna que determina su identificador (ID). Los marcadores ArUcos se pueden generar en distintas dimensiones. Si la detección no se realiza de forma exitosa, aumentando un poco su tamaño se soluciona este problema en la mayoría de los casos. La biblioteca ArUco cuenta con una wiki donde se explica detalladamente su funcionamiento:

https://docs.opencv.org/trunk/d5/dae/tutorial_ArUco_detection.html

La idea de uso de estos marcadores es imprimirlos y colocarlos en escenas en el mundo real, para que posteriormente sean detectados de forma única en la imagen.

Veamos algunos casos de uso. Por ejemplo, en la **Figura 68**, los marcadores han sido colocados en las cuatro esquinas de un marco de fotografía. Al identificar los ArUcos podemos sustituir el marco de la fotografía por otra imagen o video con la peculiaridad de que esta tendrá la perspectiva correcta gracias a las coordenadas que nos devuelve.

Su uso principal en robótica estriba en su capacidad de servir como sistema de localización del robot en un entorno. Si el robot pierde su orientación en algún momento, gracias a la detección de algún marcador ArUco puede saber fácilmente donde se encuentra y reorientarse.

La biblioteca ArUco cuenta con su librería en OpenCV, y los investigadores de MINT han implementado una variante que se ejecuta de forma nativa en el smartphone, para poder utilizar las funcionalidades de la detección de ArUcos en Robobo. Este algoritmo es el último de los tres algoritmos basado en visión artificial que se caracterizarán en este Trabajo Fin de Grado.

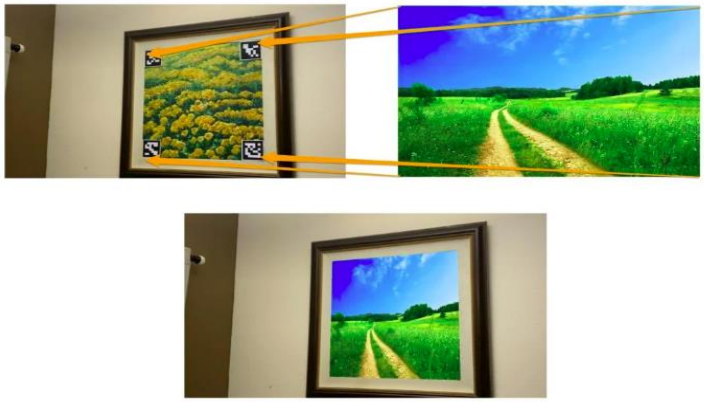

Figura 68. *Ejemplo de uso de los marcadores usando la técnica de homografía.*

Desde la librería en Python de Robobo, MINT ha implementado el método *readTag().* Este método, haciendo uso de la cámara de nuestro *smartphone*, devuelve un objeto *tag.* Por defecto, el método utiliza ArUcos de 4x4x1000 de 100 mm de lado. En el siguiente enlace se encuentra la wiki del método, con una explicación de su uso y su código:

https://mintforpeople.github.io/robobo.py/

Para busca el método, se debe escribir "readTag" en la barra de búsquedas a la derecha de la página. La primera búsqueda se trata del método.

Igual que los métodos analizados en los apartados anteriores, este método no requiere de ningún *input* y devuelve un objeto *tag que contiene* el ID del ArUco (identificador), las coordenadas de sus cuatro esquinas ("cor1", "cor2", "cor3", "cor4"), un vector de traslación ("tvecs") y uno de rotación ("rvecs"). En la siguiente imagen (**Figura 69**), vemos lo que nos devuelve el método cuando detecta a un ArUco. Tras crear una instancia de la clase Robobo, lo llamamos y mostramos por pantalla lo que nos devuelve. Las coordenadas "x" e "y" de las cuatro esquinas del marcador son devueltas en función del tamaño de la imagen de entrada y permiten identificar al marcador en la escena. Los vectores de rotación y traslación nos devuelven la posición relativa del marcador respecto a la cámara (pose). Con ellos podemos saber por ejemplo la distancia existente entre ArUco y Robobo.

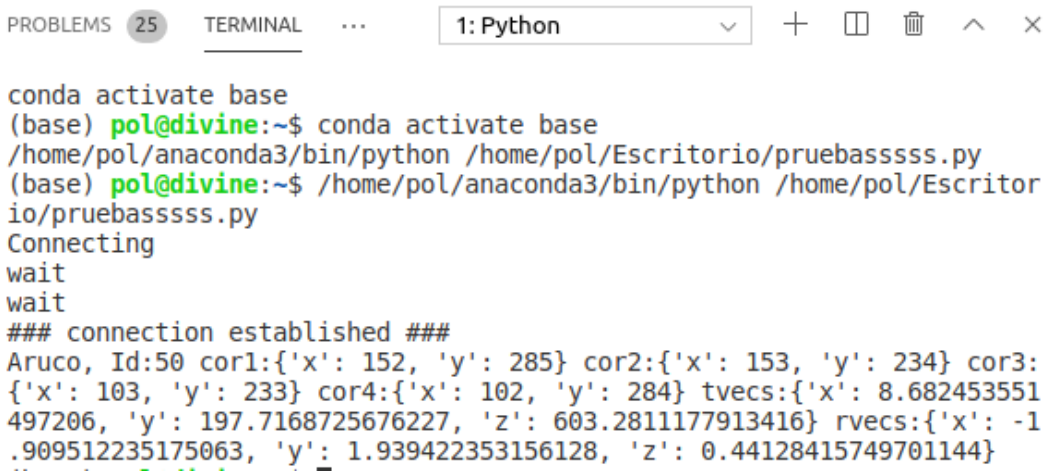

Figura 69. *Información devuelta por el método readTag()*

5.3.1.1 Calibración de la cámara del Smartphone

Antes de proceder a realizar medidas con la librería, es necesario calibrar nuestra cámara. Para ello se usa un ChArUco, un tablero de ajedrez combinado con ArUcos que intenta combinar los beneficios de ambos. Por un lado, en un tablero de ajedrez sus esquinas se pueden refinar con mayor precisión que en un ArUco, ya que cada esquina está rodeada por dos cuadrados negros. La parte del ArUco se utiliza para interpolar la posición de las esquinas del tablero de ajedrez, de modo que tenga la versatilidad de los marcadores, ya que permite oclusiones o vistas parciales. En la

Figura 70 se muestra un ChArUco.

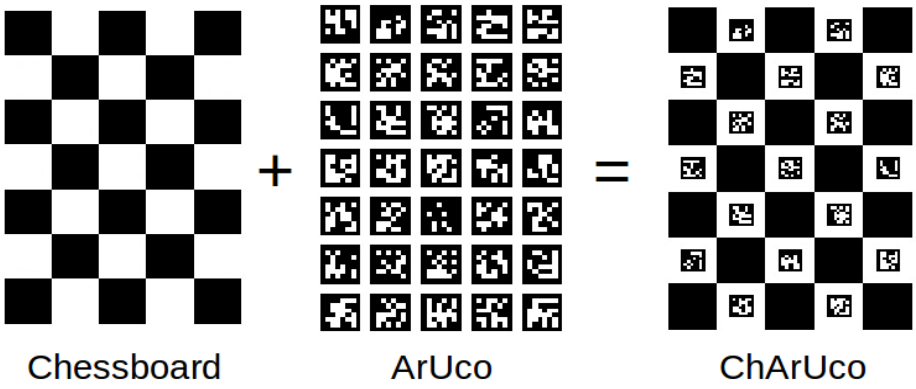

Figura 70. *ChArUco.*

Para realizar esta calibración, MINT ha creado una App para el smartphone que hemos usado en las pruebas. Es importante resaltar que si no se realiza esta calibración no se puede utilizar el método correctamente, puesto que los datos devueltos por el método serían erróneos. Por lo tanto, el alumno o la persona que vaya a usar este método debe realizar antes de nada la calibración con el teléfono que se vaya a usar.

Esta App calibra la cámara a partir de una fotografía de nuestro ChArUco y genera un archivo que contiene todos los parámetros de calibración llamado *camera.properties* dentro de la carpeta properties, en la raíz de nuestro teléfono. Esta App requiere que se le especifique en ajustes las medidas de nuestro tablero ChArUco (filas, columnas, tamaño cuadrado y tamaño marcador). Una vez calibrada la cámara de nuestro smartphone ya se puede utilizar correctamente el nuevo método

La **Figura 71** muestra el funcionamiento de la App. En el siguiente enlace se encuentra una explicación más detallada de su uso:

https://github.com/mintforpeople/robobo-hri-vision/blob/module_tensorflow/app/src/main/java/com/mytechia/robobo/framework/vision/TagCalibrationApp.md .

Para la calibración se imprimió un ChArUco de 8x11 de la siguiente web: https://calib.io/pages/camera-calibration-pattern-generator

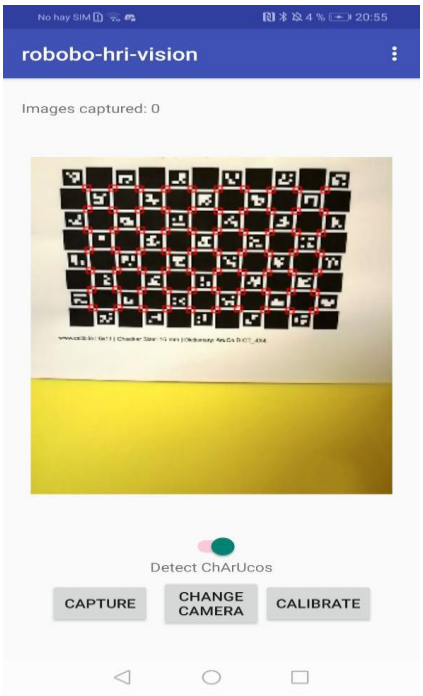

Figura 71. *App de calibración del módulo de detección de ArUcos.*

5.3.1.2 Script para el cálculo de la distancia y el ángulo

La biblioteca ArUco de OpenCV provee una manera de estimar la pose, es decir, la posición (x,y,z) y el ángulo que hay entre el marcador y la cámara. Para ello se necesita contar con los coeficientes de distorsión y la matriz de la cámara que estemos usando. Con estos datos se puede calcular los vectores de traslación y rotación y con ellos puedes conocer la posición de tu cámara respecto al marcador ArUco.

En el caso de nuestro método, la App de calibración se encarga de generar los coeficientes de distorsión y la matriz de la cámara de nuestro smartphone y con ello el módulo calcula los vectores de traslación (Tvec) y de rotación (Rvec). Ambos vectores son devueltos por el método al ser llamado. El método de la librería Robobo detecta marcadores ArUcos de 100 mm de lado.

A modo ilustrativo se ha implementado un script en Python para obtener la distancia del ArUco a nuestro Robobo y su ángulo en *degrees*. El valor del vector de traslación devuelve los valores en milímetros, es necesario pasarlo a centímetros. Por su parte, el vector de rotación debe ser transformado primero en matriz de rotación usando el método *Rodrigues()* de la librería de OpenCV y desde ahí pasar la matriz a ángulos de Euler. Este script será usado para convertir los valores devueltos por el método a una forma que podamos interpretar.

El código se encuentra disponible en GitHub en el siguiente enlace:

https://github.com/GII/robobo_vision/tree/master/Robobo-ArUco-Distance

En la **Figura 72** se muestra el script en funcionamiento, con el Robobo situado a 100 cm del ArUco y vemos que el código devuelve un valor de 99.62 cm.

Figura 72. *Script para detección de distancia y ángulo.*

5.3.1.3 Definición de las pruebas a realizar

Para la caracterización del funcionamiento de esta librería, se han establecido tres tipos de pruebas:

1- Caracterización de distancia y ángulo, donde se compararán varias distancias y ángulos reales con las medidas que el método devuelve.
2- Caracterización de PAN, TILT y YAW: A una distancia fija, se variarán estos tres valores para obtener las medidas de distancia que él método devuelve y sus rangos máximos de detección
3- Caracterización en base a la iluminación ambiente: Se variará la iluminación de la escena para analizar la respuesta de la detección del método ante distintos valores de luminosidad.

5.3.2 Pruebas de caracterización de distancia y ángulo

Para caracterizar las propiedades de detección del método, se imprimió un ArUco de 4x4x1000 de 100 mm de lado con un ID de valor 50, que como se ha comentado anteriormente, es el tamaño que usa el método por defecto. Este tag se colocó en una pared donde se trazó, desde la mitad del ArUco (en este caso 50 mm), una línea recta en el suelo. El robot se colocaba centrado en dicha línea y se iba variando su distancia desde la cámara al ArUco, como se ve en la **Figura 72**. Lo más importante en esta prueba es comparar la distancia real con la medida por el método en el caso de detección más favorable, con el tag en el centro de la imagen, para poder analizar el error de precisión existente, así como las distancias mínimas y máximas de detección.

Para esta prueba, no solo se comprobó a que distancia máxima era capaz de detectarlo, sino también cuanto tardaba en hacerlo (número de *frames*), para comprobar si a mucha distancia el algoritmo tarda más en discernir que se trata de un ArUco de ID 50 o incluso lo confunde dando otro valor. La prueba consistió en tomar medidas con el smartphone en la base Robobo con una inclinación en el *Tilt* de 80, e ir alejándolo cada

vez más del ArUco. Los datos obtenidos se encuentran en la **Tabla 6** mostrada a continuación.

Distancias reales (cm)	Distancias medias por el método (cm)	Error distancia absoluto	Error distancia relativo (%)	Detección	Detectado en el primer Frame
50	50,7552	0,7552	1,51	✓	✓
100	102,9791	2,9791	2,9	✓	✓
150	153,9814	3,9814	2,65	✓	✓
200	211,0923	11,0922	5,55	✓	✓
250	260,6690	10,6699	4,27	✓	✓
300	308,4118	8,4118	2,80	✓	✓
350	358,8124	8,8124	2,52	✓	✗
400	411,8600	11,8641	2,97	✓	✗
450	-	-	-	✗	-

Tabla 6. *Comportamiento de detección del ArUco a distintas distancias.*

En la **Tabla 6** observamos que la distancia máxima hasta donde la cámara logra detectar al ArUco son 4 metros y que error relativo en la distancia varía entre el 2 y el 5% en todo el rango que detecta. Parece que el error aumenta a medida que aumenta la distancia al marcador.

Figura 73. *A la izquierda, toma a 50 cm. A la derecha, toma a 300 cm.*

La imagen anterior (**Figura 73**) muestra a modo ilustrativo las medidas a 50 y 300 cm. El script usado para las pruebas hace uso de OpenCV para pintar el cuadrado verde a partir de los valores (x,y) de las cuatro esquinas, y pinta también en rojo el ID del ArUco detectado. Esto se hace para comprobar que la detección se ha realizado con éxito. La detección de la imagen anterior ocurrió en el primer frame. A los 3.5 metros de distancia, el método pintaba y detectaba correctamente el ArUco la mayoría de las veces, pero a diferencia de las distancias anteriores, alguna vez tardaba varios fotogramas en reconocer que se trataba de un ArUco de ID 50.

Las medidas de 4 y 4.5 m se realizaron desde otro lado de la habitación para poder alcanzar dichas distancias, comprobándose que esta última es la distancia límite para la que el método es capaz de lograr la detección (**Figura 74**). A la izquierda, imagen original y pintada a 400 cm. A la derecha, imagen original y pintada a 450 cm.

Figura 74. *A la izquierda, toma a 400 cm. A la derecha, toma a 450 cm.*

El otro gran factor a caracterizar, era el error en la distancia y en el ángulo medido en función del ángulo y la distancia real. Para esta prueba, se comenzó probando con colocar al Robobo orientado en distintos ángulos de cara al ArUco que se encontraba fijo en la pared, y con esta disposición ir realizando un barrido a distintos ángulos y distancias. Sin embargo, cuanto mayor era la distancia al ArUco, mayor era el tamaño del barrido que se tenía que realizar para poder abarcar desde los 0º (Robobo en la línea blanca, mirando de frente al marcador) hasta los 90º (ángulo límite donde el robot vería al ArUco completamente de lado).

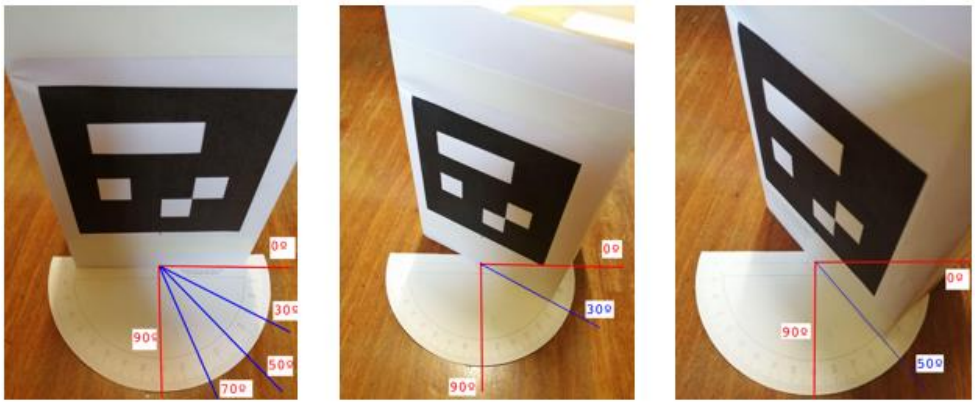

Figura 75. *Sistema usado para variar el ángulo del ArUco respecto al robot.*

Por ello, se optó por dejar al Robobo centrado en la línea blanca y rotar al ArUco desde el centro fijo de un transportador de ángulos, desde los 0 a los 90º como se ve en la **Figura 75** (se realizó la rotación solo de un lado, ya que desde el otro lado el resultado sería el mismo). Con esta nueva disposición se realizaron medidas para distintos ángulos con cada una de las distancias usadas (50, 100, 150, 200, 250 y 300 cm). Para cada distancia se tomaron datos cada 10 grados desde los 0º. Lo que se buscaba en esta prueba era caracterizar su precisión en función del ángulo existente entre el marcador y la cámara y, para cada distancia, conocer cuál es el ángulo máximo donde la detección deja de producirse. En la **Tabla 7** se muestran los datos obtenidos.

Distancias reales (cm)	Ángulo real (º)	Distancias medidas (cm)	Ángulo medido (º)	Error absoluto distancia	Error relativo distancia (%)	Error absoluto ángulo	Error relativo ángulo (%)
50	0	50,75	2,6617	0,7552	1,51	2,6617	-
	10	50,23	11,3031	0,2316	0,4	1,3031	13,031
	20	49,87	23,475	0,12316	0,25	3,475	17,375
	30	50,34866	30,295	0,34866	0,70	0,295	0,983

	40	50,1878	38,9395	0,1878	0,38	1,0605	2,651
	50	51,0692	51,4762	1,0692	2,14	1,4762	2,952
	60	51,7368	58,8777	1,7368	3,47	1,1223	1,870
	70	51,849	64,8777	1,849	3,70	5,1223	7,318
	80	51,5357	67,2749	1,5357	3,07	12,7251	15,906
100	0	102,7627	0,41	2,7627	2,76	0,41	-
	10	102,979	11,3212	2,979	2,98	1,3212	13,212
	20	102,09888	23,6666	2,09888	2,10	3,6666	18,333
	30	100,56416	32,3621	0,56416	0,56	2,3621	7,874
	40	102,2125	42,867	2,2125	2,21	2,867	7,167
	50	100,6037	51,8788	0,6037	0,60	1,8788	3,758
	60	102,463159	62,0102	2,463159	2,46	2,0102	3,350
	70	100,9085	66,0677	0,9085	0,91	3,9323	5,618
	75	100,9775	70,164	0,9775	0,98	4,836	6,448
150	0	153,9814	4,311	3,9814	2,65	4,311	-
	10	159,3614	13,911	9,3614	6,24	3,911	39,110
	20	161,882617	14,8809	11,882617	7,92	5,1191	25,596
	30	163,8387	28,7701	13,8387	9,23	1,2299	4,100
	40	158,9641	41,401	8,9641	5,98	1,401	3,503
	50	155,25552	49,8091	5,25552	3,50	0,1909	0,382
	60	155,2525	54,8091	5,2525	3,50	5,1909	8,652
	65	155,3434	56,6606	5,3434	3,56	8,3394	12,830
200	0	211,09228	0,6442	11,09228	5,546	0,6442	-
	10	212,86684	7,6425	12,86684	6,433	2,3575	23,575
	20	215,3305	15,5455	15,3305	7,665	4,4545	22,273
	30	219,7311	34,93	19,7311	9,866	4,93	16,433
	40	213,74166	44,35612	13,74166	6,871	4,35612	10,890
	50	213,83697	56,3281	13,83697	6,918	6,3281	12,656
250	0	260,669	4,4059	10,669	4,268	4,4059	-
	10	259,89122	8,3679	9,89122	3,956	1,6321	16,321
	20	272,42413	22,4571	22,42413	8,970	2,4571	12,286
	30	273,5999	33,7895	23,5999	9,440	3,7895	12,632
300	0	308,41185	7,9877	8,41185	2,804	7,9877	-
	10	325,63356	14,69154	25,63356	8,545	4,69154	46,915
	20	327,841	16,9154	27,841	9,280	3,0846	15,423
	25	331,0254	20,9158	31,0254	10,342	4,0842	16,337

Tabla 7. *Ángulo límite de detección del ArUco a distintas distancias.*

Como podemos observar en la **Tabla 7**, cuanto mayor es la distancia entre el ArUco y el robot, menor es el ángulo límite de detección. Dentro de cada distancia, cuanto mayor es el ángulo mayor es el error existente en cuanto distancia y ángulo. Suele rondar el 2-3% para las medidas realizadas a 50 y 100 cm, pero a partir de los 150 cm, el error aumenta al 7-9%. Para las medidas realizadas a 200, 250 y 300,

la tendencia de error se mantiene y según aumenta la distancia, estos valores se logran cada a ángulos más pequeños. En cuanto al error del ángulo, hay un error absoluto de entre 5 y 8º en cada medida de distancia tomada, que por lo general aumenta acorde el ángulo del ArUco es mayor.

Figura 76. *Marcador a 50 cm. Ángulos de izquierda a derecha: 70, 80 y 85º.*

Como se observa en la imagen de la **Figura 76**, a 50 centímetros de distancia del ArUco, el método no es capaz de distinguirlo si está a más de 80º. Por su parte, cuando la distancia sube a 100 cm el ángulo de detección baja 5 grados, dejando de detectarlo sobre los 75º. A los 150 centímetros, vemos cómo, aunque la detección ocurre a los 65º, detecta que se trata de una ArUco con un ID 705 en vez de 50, por lo tanto, el ángulo de detección máximo a esta distancia se encuentra entorno los 60º.

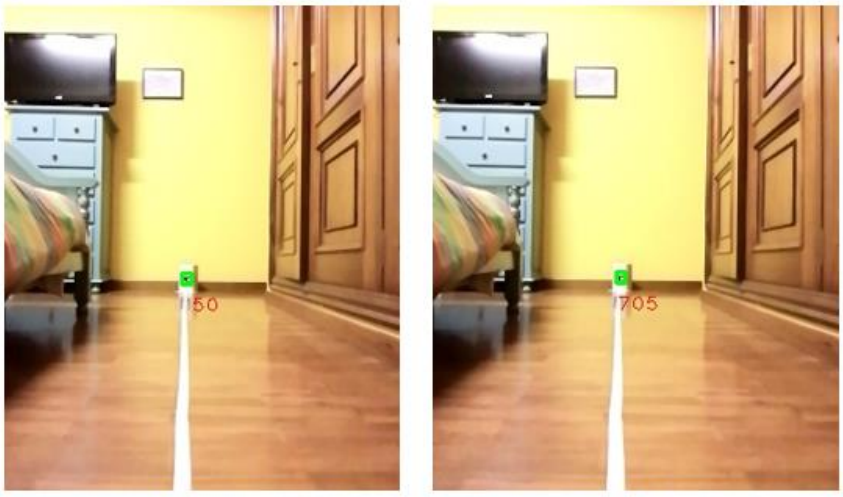

Figura 77. *Marcador a 300 cm. Ángulos de izquierda a derecha: 25 y 30º.*

A 200 centímetros de distancia del ArUco, el método no es capaz de distinguir al marcador si está girado a más de 50º. En la siguiente medida, a 250 cm, el ángulo de detección máximo baja a 30º. Por último, en la **Figura 77**, a 3 metros de distancia del ArUco, el método logra identificar y pintar al ArUco correctamente a 25º como se ve en la imagen izquierda, pero a 30 grados, aunque logra detectarlo, la identificación resulta errónea, con un ID de 705.

5.3.3 Pruebas para la caracterización de Pan, Tilt Y Yaw

En esta caso nos interesa conocer el comportamiento del método de detección de ArUcos cuando variamos los valores del Tilt y Pan del Robobo, haciendo uso de los métodos *moveTiltTo()* y *movePanTo()* de la librería en Python de Robobo.

Con el robot quieto a una distancia fija del ArUco (**Figura 78)**, variamos primeramente los valores del Pan, comparando la distancia fija usada con la distancia medida por el método y obteniendo sus rangos máximos de detección. Las medidas realizadas variando el Pan y el Tilt han sido llevadas a cabo para analizar su detección, simulando que el marcador se desplaza longitudinalmente en el "x" o "y".

Figura 78. *Robobo mirando de frente al marcador ArUco.*

A continuación, realizamos las mismas mediciones, pero variando los valores del Tilt. Por último, para conocer su respuesta frente al Yaw, es decir, el ángulo existente entre la cámara y el marcador rotando alrededor del eje z, hemos pegado el marcador ArUco al techo para poder variar fácilmente este ángulo al mover la dirección del Robobo. Para ello, primero se ha inclinado el Tilt de tal manera que la cámara quede completamente tumbada, apuntando hacia el techo (Tilt 5). Con la disposición de la **Figura 79** se ha variado la dirección del Robobo para 30, 60, 45, 90, 120, 150 y 180 grados.

Figura 79. *Robobo completamente tumbado, con Tilt 5.*

La distancia fija usada para las pruebas donde se varía el Tilt, Pan y Yawn ha sido de 100 cm.En la tabla a continuación (**Tabla 8**) se muestra la distancia medida para cada valor del Pan dentro del intervalo en donde el ArUco es detectado a esa distancia, de -25 a 25. Cuanto más cerca se efectúe la medida del marcador ArUco, mayor será el abanico de valores de Pan disponible.

Distancia real (cm)	Valores de Pan [-160,160]	Distancia medida (cm)	Ángulo medido (º)	Error distancia absoluto	Error distancia relativo (%)
100	-25	91,5238	18,1102	8,4762	8,476
	-20	98,9108	9,2106	1,0891	1,089
	-15	100,1671	5,7266	0,1671	0,167
	-10	99,9132	4,6517	0,0867	0,087
	-5	100,5875	9,7931	0,5875	0,588
	0	101,3829	8,6012	1,3829	1,383
	5	101,0170	1,1244	1,0176	1,017
	10	97,4534	12,3297	2,5466	2,547
	15	95,1190	2,7844	4,8811	4,881
	20	92,4490	10,6402	7,5509	7,551
	25	93,9243	16,7216	6,0756	6,076

Tabla 8. *Variación de la distancia medida a distintos valores del Pan.*

Como se observa en la Tabla 8, desde los 0º, el error en la distancia tiende a aumentar hacia ambos extremos (25 y -25). Con unos errores relativos en la distancia del 1% para Pans en el rango de -5 a 5. Desde esos valores aumenta hacía ambos lados hasta alcanzar un 6-7 % respectivamente. Vemos como su comportamiento frente a la distancia es simétrico. Según el valor de Pan aumente, el ArUco se trasladará a lo largo del eje “x” como se ve en la **Figura 80.**

Figura 80. *Marcador ArUco detectado con Pan 25, a 100 cm de distancia.*

Como hemos visto, al variar el Pan, el aruco no varía apenas su ángulo ya que únicamente se traslada en el eje “x” de la imagen. Por dicha razón, y para ilustrar esto, podemos ver en la **Tabla 8**, como el ángulo medido ronda entre el 1 y el 10-15º, pues esta sería la perspectiva de giro que alcanza el ArUco desde la cámara. Esto implica que si la detección del marcador Aruco se diera a un ángulo no mayor de 75º y en la esquina de la imagen, el método lograría detectarlo correctamente.

Distancia real (cm)	Valores de Tilt [5,105]	Distancia medida (cm)	Ángulo medido	Error distancia absoluto	Error distancia relativo (%)
100	65	100,0401	4,7704	0,0401	0,040
	70	99,4737	3,8029	0,5263	0,526
	75	100,4364	1,6723	0,4364	0,436
	80	100,9764	1,7510	0,9764	0,976
	85	101,0348	5.3293	1,0348	1,035
	90	97,9603	10,9205	2,0396	2,040
	95	94,8008	12,1124	5,1991	5,199
	100	92,3023	14,3246	7,697643	7,698
	105	88,7355	9,2744	11,2645	11,265

Tabla 9. *Variación de la distancia medida a distintos valores del Tilt.*

Por parte del Tilt, en la **Tabla 9**, vemos la variación de las distancias medidas a una distancia real fija de 100 cm de la cámara al ArUco en función del valor del Tilt. Al igual que sucedía con el Pan, al variar el Tilt, el marcador se desplaza longitudinalmente en el eje "y". Desde un valor de Tilt recto (Tilt de 80-85) vemos como el valor de la distancia medida varía en el rango de 65 a 105, que es el intervalo donde el marcador se encuentra en la imagen. El error en la distancia medida aumenta ya se incline más hacia el techo (Tilts de 80-85 a 65) o hacía el suelo (Tilts de 80-85 a 105). Sin embargo, el error parece ser mayor en los valores de Tilt donde el smartphone se encuentra más inclinado hacia el suelo. Observamos como su error aumenta conforme inclinamos más al smartphone, lo que equivaldría a desplazar en el eje "y" positivo al ArUco. Igual que con el Pan, el ángulo de cabeceo (giro alrededor del eje "x") apenas varía, pues el marcador se desplaza longitudinalmente sin apenas variar su ángulo respeto a la cámara. La conclusión es la misma que con la variación de Pan.

Si el ArUco se encuentra girado por debajo de su ángulo límite (**Tabla 7**), entonces la detección puede ocurrir independientemente de la posición (x,y) que ocupe el ArUco en la cámara.

En cuanto a la variación debida al Yaw, en la **Figura** 81 se muestra como lo ve la cámara del *smartphone* a 0º (primera toma) y 120º. Desde la cámara tumbada de la **Figura 79** hasta el techo había 340 centímetros. Por dicho motivo, para las medidas se colocó al Robobo en una zona elevada y se regulo su altura hasta que la distancia al ArUco fue de 100 cm, como en las tomas de Pan y Tilt. Para finalizar este apartado, en la **Tabla 10**, podemos ver los resultados obtenidos con la variación del Yaw:

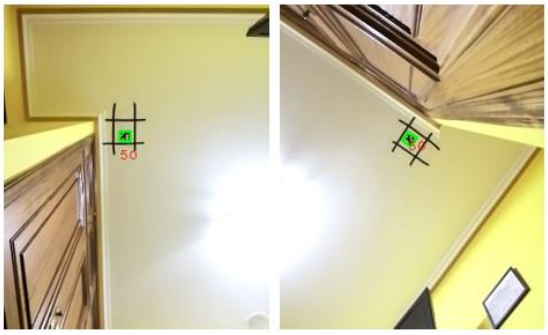

Figura 81. *Marcador ArUco detectado con Yaw a 0 y 120º.*

Distancia real (cm)	Valores de Yawn (º)	Distancia medida (cm)	Ángulo medido (º)	Error distancia absoluto	Error distancia relativo (%)	Error ángulo absoluto	Error ángulo relativo (%)
100	0	100,4069	2,5743	0,4069	0,407	2,574	-
	30	100,4959	30,3344	0,4959	0,496	0,33449	1,115
	45	101,5349	46,8855	1,5349	1,535	1,8855	4,19
	60	99,8687	64,5781	0,1313	0,131	4,5781	7,63
	90	99,1874	89,3083	0,8126	0,813	0,69165	0,769
	120	101,3449	119,1012	1,3449	1,345	0,8988	0,749
	150	101,9239	153,8473	1,9239	1,924	3,8473	2,565
	180	101,1022	178,529	1,1022	1,102	1,47092	0,817

Tabla 10. *Variación de la distancia medida a distintos valores del Yaw.*

Observamos en la **Tabla 10**, como su error en distancia varía entre el 1 y el 3% en el rango de 0 a 180º (media vuelta completa) de giro de Yaw. A diferencia de las medidas realizadas variando el Pan o el Tilt, el ArUco no se desplazaba longitudinalmente de ninguna manera en la imagen. Por lo tanto, las tomas a 0, 90 y 180 grados realmente son visto desde la cámara del *smartphone* como la "misma toma", donde únicamente varía el orden de sus cuatro esquinas. Para la toma a 0º, el método nos devolvió un valor de -91º, para la toma a 90º, un valor de 91º, indicando que el algoritmo tiene en cuenta el orden de la rotación en el ArUco, en función del orden de las esquinas. Para realizar la comparación hemos usado la coordenada de giro alrededor del eje "z" del vector de rotación. Como se observa, su error es por lo general muy bajo, con un error absoluto entre el 1 y el 4% y un error relativo que ronda el 1% en casi todos los ángulos probados. En cuanto a la distancia, su error también es bajo, sobre el 1%,

5.3.4 Prueba de caracterización en base a la iluminación ambiente

En las pruebas anteriores, el nivel de luminosidad siempre fue el mismo, con una luz artificial en el techo. Los ArUco no son muy sensibles al nivel de luz de la escena, dada su codificación binaria con un patrón de blanco y negro, pero debemos comprobarlo. Si, por ejemplo, un exceso de luz refleja en el marcador y distorsiona la medida de la cámara, es posible que no se identifique correctamente sus coordenadas o se detecte otro identificador de ArUco distinto. Por lo general, los marcadores responden mejor en condiciones de poca luz, pero es importante saber cuál es la cantidad mínima en la escena para que se produzca una identificación exitosa.

Para ello, se ha usado un foco con un control de luminosidad y se han realizado las medidas variando para cada distancia la luminosidad en función del espectro del que se dispone. Al no contar con un luxómetro profesional, se utilizó el sensor de luz de un *smartphone* y una App que simulaba al mismo. El *smartphone* se colocó a un metro y medio de la fuente de luz y se calibró previamente gracias al hecho de que un Lux es un Lumen por metro cuadrado. Conociendo el nivel de Lumens de la fuente de luz usada (Bombilla halógena G9), se halló previamente qué valores debería producir a distintas distancias del foco y con estos datos se calibró los valores de la App para hacerlos coincidir. Las medidas dadas por la App no deben tomarse como valores absolutos sino como una forma de cuantificar el incremento de luz en la escena. Las pruebas consistieron en ir posicionando al Robobo a distintas distancias del marcador ArUco (50,

100, 150, 200, 250 y 300 cm) e ir variando la cantidad de luz con el potenciómetro instalado en el foco. Con ello, se estimó el nivel de luz mínima necesaria para que el método logre identificar correctamente nuestro *tag*. Los resultados se encuentran en la **tabla 11.**

Distancias (cm)	Detección Correcta	Nivel de Luz Mínima para la detección (Lux)	ID
50	✓	5	50
100	✓	5	50
150	✓	5	50
200	✓	5	50
250	✓	5	50
300	✓	5	50

Tabla 11. *Nivel de Luminosidad mínima para la detección del ArUco a distintas distancias.*

Figura 82. *Marcador a 50 cm. Nivel de luz de izquierda a derecha: 3, 5 y 7 Lux.*

En la **Figura 82**, podemos observar cómo a 5 Lux, el método ya logra identificar nuestro ArUco y su ID correctamente. Observamos que no lo pinta tan bien como a 7 Lux debido posiblemente a la identificación incorrecta de las coordenadas de la esquina inferior derecha por la falta de luz. A 3 Lux, el método ya no logra identificar al *tag.*

Figura 83. *Marcador a 300 cm. Nivel de luz de izquierda a derecha: 3 y 5 Lux.*

Tras las medidas a más distancia (**Figura 83**), se ha comprobado que, con 5 Lux de luminosidad en la escena, el método es capaz de detectar los marcadores ArUco ya se encuentren a medio metro o a tres. Vemos como esta medida se repite en las mediciones independientemente de la distancia de la cámara al marcador. Como se ha comentado anteriormente, es posible que esa cantidad de luminosidad en la escena no sean exactamente 5 Lux, pero siendo el valor de 3 Lux, una escena prácticamente a oscuras, el valor mínimo de luz que requiere el método para detectar los *tags* es muy bajo y más todavía por el hecho de lograr la detección sin problema a esas distancias.

El script usado para todas las pruebas se encuentra disponible en el siguiente enlace en un repositorio de GitHub donde se explica su funcionamiento:

https://github.com/GII/robobo_vision/tree/master/ArUco-Measures

5.3.5 Conclusiones de la caracterización de la librería de detección de marcadores avanzados

Con el tamaño de ArUco de 100 mm de lado, que es el tamaño predeterminado por el método, se han llegado a las siguientes conclusiones:

- El módulo Tag de la librería Robobo es capaz de reconocer al ArUco hasta 4 metros aproximadamente. A partir de los 3 metros y medio, la detección ya no suele ocurrir en el primer fotograma, sino que necesita de algunos más para reconocer correctamente el identificador del marcador. Se observa que, a distancias más cercanas, el error existente entre la distancia real y la medida es menor, y aumenta cuanto más se aleja al Robobo del ArUco. Por lo general, alcanza un error relativo de entre 2 y el 5% para el rango de distancias detectado. La distancia menor de detección, es aquella donde el marcador quede total o parcialmente cortado en la imagen y por lo tanto ya no pueda ser reconocido. Con el ArUco de 100 mm de lado que utiliza el método por defecto, esa distancia mínima ocurre aproximadamente a 30 cm del ArUco, con un Tilt de 80.

- Variar el ángulo existente entre el marcador ArUco y la cámara del smartphone produce un mayor error en la distancia medida cuanto mayor sea dicho ángulo. Además, cuanto mayor sea la distancia al ArUco, mayor será este error. Dentro de cada distancia, cuanto mayor es el ángulo mayor es el error existente en cuanto distancia y ángulo. El error en cuanto la distancia ronda el 2-3% para las medidas más cercanas (50 y 100 cm) y a medida que aumenta la distancia, aumenta el error existente para un mismo ángulo. Por parte del ángulo, hay un error absoluto de entre 5 y 8º para las diferentes distancias tomadas, que por lo general aumenta acorde el ángulo del ArUco es mayor.

- A colación del punto anterior, se ha observado que el máximo ángulo de giro donde el método es capaz de reconocer al ArUco, mantiene una relación prácticamente lineal. A 50 cm detectaba correctamente al ArUco hasta los 80º, al metro hasta los 75, al metro y medio a los 65º, etc. Por lo tanto, a mayor distancia, menor tiene que ser dicho ángulo para lograr una detección exitosa. Por ejemplo, cuando se encuentra a tres metros de distancia, el ángulo no puede ser mayor de 25º.

- Al variar el Tilt, el marcador se desplaza longitudinalmente en el eje “y” hasta que desaparece parcial o completamente, momento donde la detección deja de producirse. Partiendo de un valor de Tilt recto (80-85) el error en la distancia medida va aumentando según se incline hacía el suelo (Tilts desde 80-85 a 105).

En el rango de 65 a 80-85 el error relativo es del 1%, pero a partir de un Tilt de 90, este aumenta de forma lineal hasta alcanzar un 11%.

- Al variar el Pan a una distancia fija, produce que el marcador se desplace longitudinalmente en el eje x de la imagen hasta que desaparece parcial o completamente, momento donde la detección deja de producirse. En el caso del Pan, partiendo de un Pan recto, con la cámara mirando de frente al marcador (0), las variaciones en ambos sentidos (de 0 a 160) o de (0 a -160) producen un mayor error en la distancia medida a razón de su valor. Produciendo un error que relativo en la distancia que ronda el 6-7% en los extremos. El ángulo de giro del ArUco varía al cual se detecta al ArUco varía en función del valor del Pan. Con un valor de 5 º, cuando ArUco y cámara se miran de, hasta los 12-15º en los extremos de detección.

- La conclusión extraída de variar el Pan y el Tilt, es que el desplazamiento del ArUco en la imagen (variación “x” e “y”) no afecta en gran medida a la correcta detección de su distancia al robot, con lo cual, si el Robobo detectara al marcador girado un cierto ángulo y en un extremo de la imagen, la detección ocurriría sin problemas, mientras el ArUco se encuentre girado por debajo de su ángulo límite (Tabla 7).

- Como ya se comentó anteriormente, si el Aruco es detectado parcialmente la detección no ocurre (el método devuelve ceros), Por lo tanto, la detección ocurrirá mientras el Aruco se mantenga con sus cuatro esquinas dentro de los límites de captación de la cámara.
- En cuanto al Yaw, el error en su distancia varía sobre el 1% y el de su ángulo entre el 2 y 4%. Dependiendo del orden de las esquinas del ArUco en la detección, el algoritmo distinguirá entre si este se encuentra por ejemplo a 90º o 180º.

- Por su parte, con la iluminación ambiente, el módulo es capaz de detectar e identificar correctamente al ArUco en todas las mediciones realizadas (50,100,150,200,250 y 300 cm) a partir de un valor de luz cercano a 5 Lux.

- Con estas pruebas se ha analizado casos representativos de variación de ángulo, distancias, desplazamientos longitudinales, etc. Las pruebas que se pueden realizar y las combinaciones entre ellas son casi infinitas, por ello lo que interesaba era ver su respuesta y errores en los casos más representativos para poder extrapolar su respuesta entre dos o más combinaciones de las pruebas llevadas a cabo.

6 DISCUSIONES

El presente TFG requería realizar una serie de experimentos para caracterizar el funcionamiento de tres nuevos algoritmos de visión por computador que MINT había implementado para su uso en el robot educativo Robobo. Estos tres algoritmos se adaptaron de librerías de código abierto para su ejecución nativa en un dispositivo móvil, y permiten al robot poder realizar tareas muy avanzadas para el marco de la robótica educativa. Así, el robot sería capaz de reconocer los objetos presentes en su entorno, de moverse por dentro de un carril de forma autónoma y de utilizar la gran precisión en la localización que ofrecen los marcadores ArUco.

Pero estos tres algoritmos no habían sido probados en un entorno real de robótica educativa. A MINT le interesaba saber, por ejemplo, si la red elegida era lo suficientemente precisa y rápida, o si el módulo de detección de carriles era capaz de detectar correctamente carriles rectos y curvos y se podría realizar con él un control de movimiento funcional en un entorno real. Para llevar a cabo este análisis técnico, tuve que dedicar mucho esfuerzo a formarme en materias que apenas había visto durante la carrera.

En concreto, al principio de este trabajo mis conocimientos sobre Python eran bastante básicos, y solo conocía vagamente el campo de la inteligencia artificial aplicada. El desarrollo del mismo me ha permitido conocer mucho más de cerca el campo del *Machine Learning* y las redes neuronales, al tener que implementar *scripts* para las pruebas realizadas donde tenía que ejecutar los modelos entrenados. También acerca de su funcionamiento en un entorno real al buscar caracterizar su uso con distintos objetos, distancias y velocidades. Aunque el objetivo del presente trabajo no fue implementar los algoritmos, trabajar con ellos, analizando su respuesta ante distintas situaciones me permitió tener un mejor entendimiento de los mismos.

El uso de librerías en Python para la recopilación y representación gráfica de datos, así como el continuo uso de OpenCV para diversas tareas como el pintado de imágenes con los datos devueltos por los módulos, me permitió ganar una mayor experiencia y comprensión en este lenguaje de programación tan usado a día de hoy. Considero que los datos recopilados, el análisis de los mismos, las conclusiones extraídas y la documentación presente en el anexo, son de gran ayuda para MINT a la hora de tener una documentación técnica fiable para justificar su funcionamiento en un entorno real.

Durante la realización del trabajo, ha habido una serie de detalles que considero que podrían ayudar a un usuario final, con pocos o escasos conocimientos de este tipo de algoritmos, a comprender mejor y usar estos tres nuevos métodos de la librería Robobo. Para la realización de las pruebas de rendimiento (velocidad de detección) no se utilizó el *streaming,* dado que ralentizaba mucho la ejecución del algoritmo. Su funcionamiento no es necesario para realizar tareas de detección de objetos, pero podría ser interesante de cara al usuario, incluir la opción de “ver” lo que el Robobo está viendo, con los objetos detectados pintados en la imagen.

En cuanto al detector de carriles, quizá es poco intuitivo la forma en la que te devuelve los carriles detectados (coeficientes de una ecuación de segundo grado y matriz inversa de transformación), pero, por otro lado, tiene la ventaja de que puedes ejecutar el control sin usar el streaming lo cual ralentiza la ejecución del mismo. En el proceso de implementación del control de movimiento observé, como ya comenté anteriormente, que el ruido de la escena debido principalmente al brillo afectaba a su detección, por eso se recomienda de cara al futuro filtrarlo en función del lugar donde se vaya a usar.

Por último, considero que el hecho de que el método para leer tags (ArUcos), devuelva las coordenadas de las cuatro esquinas, y los vectores de rotación y traslación

en forma de diccionario lo hace ver muy cómodo y ordenado, pero pensando en la usabilidad de cara al usuario final, devolverlo en la misma forma y dimensionalidad de Array que utiliza la librería ArUco de OpenCV en Python, podría facilitar al usuario el uso de métodos de dicha librería sin necesidad de transformar los datos de entrada. Por ejemplo, usar un booleano por defecto en *False* que permita decidir la forma en la que se devuelven los datos podría ser de gran ayuda. Esta mejora está pensada desde el punto de vista de un usuario que no conozca a profundidad el funcionamiento de estos módulos, para un uso más intuitivo.

Como se puede ver, mis recomendaciones simplemente se basan en mi experiencia al usar estos métodos durante la realización del presente trabajo, y con unas simples mejoras puede facilitar su uso y comprensión a alumnos no versados en estos temas.

De cara a un futuro podría ser interesante ampliar la gama de capacidades de visión artificial que el Robobo puede hacer. Por ejemplo, reconocimiento de gestos, en añadido al reconocimiento de rostros y de objetos. Con él se podría aumentar la interacción humano-máquina al permitir que el Robobo realice distintas acciones en función del gesto reconocido.

Al ser Robobo un robot educativo diseñado para ser usado por alumnos de diferentes edades y capacidades técnicas, no hay que olvidar aquellas tareas que podrían resultar de gran interés para los más pequeños, como, por ejemplo, funciones de realidad aumentada gracias a los marcadores ArUcos.

Además, se podrían utilizar otros modelos de redes neuronales, como el usado en este TFG, para tareas de reconocimiento de habla o para tareas de respuesta inteligente. Esto podría ser muy interesante, ya que permitiría al alumno hablar con el robot y darle ordenes sencillas, lo cual mejoraría todavía más la interacción con el usuario. En general, los tres algoritmos de visión artificial que se han añadido permiten una variedad casi ilimitada de funciones, más si se combinan entre sí.

Por último, en las pruebas realizadas con los marcadores ArUcos observamos como si este es detectado parcialmente, la detección no ocurre. Esto puede suponer un problema para su uso en la navegación del Robobo por un entorno, ya que sería necesario que el robot viera entero al marcador para poder orientarse. De cara a un futuro, sería importante solventar esta limitación del módulo de detección de *tags*.

7 CONCLUSIONES

En este capítulo se discutirán las conclusiones globales logradas acerca de los objetivos que se plantearon al principio del proyecto.

El objetivo global del presente trabajo era caracterizar y validar el uso de tres algoritmos de visión artificial que MINT había adaptado para su uso por el Robot Robobo. Para ello se subdividió el mismo en una serie de objetivos secundarios:

1. Conocer el robot móvil definido por el Grupo Integrado de Ingeniería de la UDC (GII) a nivel de características hardware, software, programación y campo de uso.

2. Establecer una metodología de pruebas adecuada al tipo de algoritmo a analizar.

3. Caracterizar el algoritmo de detección de objetos en tiempo real en el robot y documentar las conclusiones del análisis técnico.

4. Caracterizar el algoritmo de detección de carriles en tiempo real en el robot, implementar un sistema de control para que se mantenga dentro de los mismos, y documentar las conclusiones del análisis técnico.

5. Caracterizar el algoritmo de detección de marcadores ArUco en tiempo real en el robot y documentar las conclusiones del análisis técnico.

El primer y segundo subobjetivo se realizó sin problemas, pues antes de realizar las pruebas con un algoritmo se estudiaba su documentación y se aprendía cómo funcionaba y se ejecutaba. Lo mismo sucedió con el robot Robobo y sus características a nivel de hardware y software.

Antes de la realización de las pruebas de caracterización, se diseñaba la metodología de pruebas adecuada en función del tipo de algoritmo a analizar y los rangos de uso establecidos previamente con los investigadores del GII. Estas pruebas trataban de ser lo más rigurosas posibles y servir como base para la posterior extracción de conclusiones. Se considera que las pruebas realizadas para cada algoritmo han sido suficientes para caracterizar su funcionamiento y encontrar las limitaciones y bondades de cada método.

En cuanto al tercer subobjetivo, MINT había adaptado un algoritmo de detección de objetos en tiempo real haciendo uso de Tensor Flow Lite para ejecutarlo desde un smartphone. Dicho método utilizaba originalmente una red MobileNet V1.

Recientemente Tensor Flow había liberado la versión V3 en su GitHub. Esta nueva versión estaba entrenada con el dataset COCO. Al principio del proyecto, MINT quería reentrenar este modelo entre otras cosas para lograr que el Robobo detectara a otros Robobos, así como ciertos elementos de tráfico como semáforos. El objetivo de esta parte era analizar y caracterizar el funcionamiento de esta nueva red ejecutándose desde un smartphone para concluir si su velocidad de ejecución y precisión la hacían una elección válida para su uso en Robobo. Para lograrlo se realizaron una serie de pruebas para analizar dichos parámetros. Los resultados obtenidos arrojaron que la red que MINT quiere usar en el Robobo es completamente comparable con redes en el estado del arte como la YOLO V3 y que se ejecutan desde un ordenador. La nueva red logra de media unos valores de confianza muy parejos a las otras redes con las que se comparó y su velocidad de detección es comparable a si se ejecutara desde un ordenador. Además, en su caracterización se ha podido estudiar su comportamiento en diversas situaciones en función de la distancia.

Dentro de este subobjetivo también se encontraba el poder calcular la distancia a la que se encuentra un objeto detectado. Para lograrlo se buscó una relación común entre el área del objeto detectado (*input*) y la distancia real donde se encuentra (*output*). Se concluyó que, dado que cada objeto posee un área propia, cada objeto del *dataset* tendrá su propia función que relacione ambas variables. Sin embargo, se comprobó que todos los objetos usados se ajustaban a una tendencia cuadrática o una función potencial de la forma $y = n\,x^{-0,48}$. Donde en función del tamaño del objeto detectado, el valor de *n* varía.

El cuarto subobjetivo era la caracterización del módulo de detección de carriles en tiempo real. Igual que con el algoritmo anterior, se buscaba analizar su comportamiento ante distintas situaciones por medio de una serie de pruebas. Con ello se comprobó que el módulo de detección de objetos es capaz de adaptar de forma correcta el polinomio de segundo grado a los tramos rectos y curvos que se le presentaban y que además la velocidad a la que se mueve el Robobo no influía en su detección. La detección ocurre dentro de un área preestablecida, mientras ambos carriles o uno, se encuentren dentro de dicha área, la velocidad no afecta a su detección. También se comprobó que el método es bastante sensible al ruido producido por la luminosidad ambiental u objetos de color blanco o amarillo. Además, se quería crear un control de movimiento que permitiera al Robobo mantenerse dentro de ambos carriles. Finalmente, utilizando un control proporcional-integral y aislando ruido de ciertas partes del entorno que producían falsos positivos, se logró su correcto funcionamiento en un circuito consistente en un tramo recto con una rotonda. Con esto se demostraba que el método de detección de objetos en tiempo real creado por MINT permitía una “conducción autónoma” en un entorno real.

Por último, el quinto subobjetivo era la caracterización del método creado por MINT para la detección de marcadores ArUcos. Para lograrlo, se realizaron varias pruebas con el fin de analizar su funcionamiento en una variedad de casos de uso. Por un lado, se varió la distancia de la cámara al ArUco y por otro lado el ángulo del marcador respeto a la cámara en distintos ejes. También se analizó su respuesta frente a la luminosidad del ambiente. Con todas estas pruebas se logró una correcta caracterización de este módulo ante diversas situaciones, calculando además valores máximos de detección en cuanto distancia, ángulo y luz ambiental. Esto nos ha permitido calcular el error existente que posee el método en cuanto a distancia y ángulo.

En términos globales, las pruebas realizadas durante la realización de este trabajo han arrojado una mejor comprensión del funcionamiento de estos algoritmos de visión artificial al ejecutarse desde un smartphone y en un entorno real. Arrojando valores importantes para su caracterización y permitiendo encontrar posibles mejoras a realizar en futuras versiones. Por lo tanto, y a la vista de los resultados, se puede considerar que se ha cumplido con los objetivos planteados en el presente Trabajo Fin de Grado.

REFERENCIAS

[1] UNIR. (26 de noviembre de 2019). "Robótica educativa: ¿qué y cuáles son sus ventajas?" [Online]. Recuperado de:

https://www.unir.net/educacion/revista/noticias/robotica-educativa/549204689239/

[2] "Robotix.com" - Bloque inteligente EV3:

https://www.robotix.es/es/bloqueinteligente-ev3

[3] "Makeblocks | Robot Mbot." [Online]. Recuperado de:

https://www.programoergosum.com/cursos-online/robotica-educativa/249-robotica-educativa-con-mbot/que-es-mbot

[4] "Thymio | What is Thymio composed" [online]. Recuperado de:

http://wiki.thymio.org/en:thymiospecifications

[5] "Robotic | e-manual, Turtlebot3" [online]. Recuperado de:

http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

[6] "e-puck " [Online]. Recuperado de:

http://www.e-puck.org/index.php?option=com_content&view=article&id=18&Itemid=24

[7] "Khepera-IV | k-teams." [Online]. Recuperado de:

https://www.k-team.com/khepera-iv

[8] "NAO the humanoid robot | Aliverobots." [Online]. Recuperado de: https://aliverobots.com/nao/

[9] "ROMO | robotsaldetalle." [Online]. Recuperado de:

https://robotsaldetalle.es/pruebas/robot-romo/

[10] "WHELLPHONE." [Online]. Recuperado de: http://www.wellphone.com

[11] "ROBOBO | TheRoboboproject." [Online]. Recuperado de:

https://therobobоproject.com/que-es-robobo

[12] F. Bellas *et al., "Robobo: The Next Generation of Educational Robot*," Ferrol, A Coruña , Spain, 2015.

[13] "Computer vision | Hisour." [Online]. Recuperado de:

https://www.hisour.com/es/computer-vision-42799/

[14] "¿Qué es la vision artificial? | MC.AI.com." [Online]. Recuperado de:

https://mc.ai/que-es-computer-vision/ [Accedido: 21 de abril de 2020]

[15] "OpenCV.com | About OpenCV." [Online]. Recuperado de:

https://opencv.org/about/ [Accedido: 9 de abril de 2020]

[16] "Una breve historia del *Machine Learning* | Empresas.blogthinking." (18 de enero de 2019). [Online]. Recuperado de:

https://empresas.blogthinkbig.com/una-breve-historia-del-machine-learning/ [Accedido: 16 de abril de 2020]

[17] "Introducción al aprendizaje automática," Tema 1, Máster Universitario Ingeniería Industrial, UDC, Ferrol.

[18] “datahack.es” – Etapas del Deep Learning: https://www.datahack.es/historia-deep-learning-etapas/

[19] “*Undestanding the activations neural network* | Medium.” [Online]. Recuperado de: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

[20] “Capas de una red neuronal | *Deep Learning con Keras/Tensorflow en Python*,” Curso Udemi, David Fuentes Jiménez. [Online]. Recuperado de:

https://www.udemy.com/course/curso-de-deep-learning-con-kerastensorflow-en-python/

[21] “*Tensor Flow* | Home page.” [Online]. Recuperado de: https://www.tensorflow.org/

[22] “*Tensor Flow* | *Tensor Flow Lite*.” [Online]. Recuperado de:

https://www.tensorflow.org/lite

[23] *“Tensor Flow* 1.0 VS 2.0 summary of changes | datasciencentral.” [Online]. Recuperado de:

https://www.datasciencecentral.com/profiles/blogs/tensorflow-1-x-vs-2-x-summary-of-changes.

[24] A.G. Howard *et al., ”MobileNets : Efficient Convolutional Neural Network for mobile Vision Applications,”*.

[25] C. Szegedy *et al., ”Going deeper with convolution,”* 2013. [Online]. Recuperado de: https://arxiv.org/pdf/1409.4842.pdf

[26] Josepth Redmon, Ali Farhadi. *“YOLO V3: An incremental Improvement.”* [Online]. Recuperado de: https://pjreddie.com/media/files/papers/YOLOv3.pdf

[27] R. Girshick *et al.,”Rick feature hierarchies for accurate object detection andsemantic segementation,”* UC Berkeley, 2015. [Online]. Recuperado de:

https://arxiv.org/pdf/1311.2524.pdf

[28] “Los niveles de la conducción autónoma | CEA.” [Online]. Recuperado de:

https://www.cea-online.es/blog/213-los-niveles-de-la-conduccion-autonoma

[29] “Teoría del control PID | Luisllamas.” [Online]. Recuperado de:

https://www.luisllamas.es/teoria-de-control-en-arduino-el-controlador-pid/

[30] ”ArUco tutorial | OpenCv.” [Online]. Recuperado de:

https://docs.opencv.org/trunk/d5/dae/tutorial_ArUco_detection.html

Anexo I: Documentación de los nuevos métodos

De cara a la creación de una documentación de sencillo entendimiento, donde se explique de forma breve y concisa el funcionamiento del módulo se han realizado unos manuales de uso para cada nuevo módulo implementado por MINT. El manual explica de forma sencilla como ejecutar cada algoritmo haciendo uso de la librería Python de Robobo, que devuelve al usuario y cuenta con un ejemplo sencillo. Dichos manuales se encuentran disponibles en el siguiente enlace:

https://github.com/mintforpeople/robobo-programming/wiki/python-doc

En este enlace se encuentra la wiki en Python de la librería Robobo. La documentación de los tres métodos caracterizados en este TFG se encuentra subidos en dicha wiki.