ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE INFORMÁTICA

MÁSTER UNIVERSITARIO EN VISIÓN ARTIFICIAL

TRABAJO FIN DE MÁSTER

Sigue persona con drone

Autor: Aitor Martínez Fernández

Tutor: José María Cañas Plaza

Cotutor: Julio Vega

Curso académico 2019/2020

Agradecimientos

En primer lugar, me gustaría dar las gracias a mi familia por el apoyo que me ha dado a lo largo de este tiempo. Todos ellos han mostrado interés por el trabajo y lo han hecho más llevadero. Sobre todo quiero mencionar a mis padres, a mi hermano y a Laura, los cuales me han animado, me han dado todo su cariño, y me han ayudado en los momentos más difíciles, pues sin ellos no hubiese sido posible conseguirlo.

Gracias a José María y a Julio, por su dedicación y apoyo en todos estos meses de desarrollo del trabajo, por lo ánimos y por sus ideas y ayuda.

Por ultimo, quiero dar las gracias a mis amigos que me han estado ahí tanto en buenos como en malos momentos: Pablo, Ione, Víctor, Fran, Nacho, Isa, Gero, ...

¡Muchas gracias a todos!

Resumen

Gracias al auge de la robótica en la actualidad, cada vez encontramos más productos robóticos a nuestro alrededor, por ejemplo robots industriales, robots en logística automatizada, aspiradoras, coches con funciones automáticas, etc. Además muchos de ellos están incluyendo técnicas de Visión Artificial (VA) para poder procesar el entorno en el que están de una manera más cercana a como lo hacen los humanos.

Es por ello que se necesitan más especialistas en este campo. Debido a esto, surgen plataformas dedicadas a la formación y aprendizaje de personas de todas las edades. Una de ellas es Kibotics. Esta plataforma pone al alcance de los más pequeños un entorno seguro de aprendizaje, tanto con robots simulados como reales. Gracias a Kibotics, los niños pueden aprender a manejar robots desde el navegador y sin ningún tipo de requerimiento previo. El entorno facilitado a los niños ofrece un conjunto de ejercicios educativos de programación de robots en dos lenguajes, *Python* y *Scratch*.

El objetivo principal de este Trabajo de Fin de Máster (TFM) es enriquecer la plataforma con procesamiento visual complejo y que a su vez sea sencilla de usar, permitiendo una detección visual robusta de objetos en imágenes utilizando técnicas de aprendizaje profundo. Esta arquitectura desarrollada sirve tanto para los robots reales como para los simulados.

Para comprobar el funcionamiento de ambas arquitecturas se han desarrollado dos comportamientos:

- **Comportamiento SiguePersona visual con drone real**. Programado en *Python*.
- **Comportamiento SiguePersona visual con drone simulado**. Programado en *JavaScript*.

Índice general

Índice de figuras

Índice de tablas

Acrónimos

API Application Programming Interface.

CNN Convolutional Neural Network.

COCO Common Objects in Context.

CSAIL MIT Computer Science & Artificial Intelligence Lab.

CUDA Compute Unified Device Architecture.

FPN Feature Pyramid Networks.

FPS Frames Per Second.

GLB binary graphic library transmission format.

GUI Graphical User Interface.

IA Inteligencia Artificial.

IOU Intersecion Over Union.

NMS Non-Maximum Suppression.

OpenCV Open Source Computer Vision Library.

R-CNN Region-based Convolutional Neural Network.

RNA Redes Neuronales Artificiales.

SDK software development kit.

SSD Single-Shot Multibox Detector.

STEM Science, Technology, Engineering and Mathematics.

SVM Support Vector Machine.

TFM Trabajo de Fin de Máster.

VA Visión Artificial.

VOC PASCAL Visual Object Classes Challenge.

YOLO You Only Look Once.

Capítulo 1

Introducción

En este capítulo se introducirá el contexto en el que se sitúa este proyecto y la motivación que ha llevado a su desarrollo. Para ello, es preciso comenzar con una explicación a grandes rasgos sobre la Visión Artificial (VA), aprendizaje profundo y la robótica educativa pues es la intersección entre esos campos donde se ubica este Trabajo de Fin de Máster (TFM).

1.1. Visión artificial y aprendizaje profundo

Desde la antigüedad el ser humano ha soñado con crear máquinas capaces de pensar. Cuando surgieron los primeros ordenadores programables, las personas se plantearon la idea de lograr que estos computadores adquirieran inteligencia, consiguiendo capacidades empleadas para realizar tareas propias de los humanos. Algunos ejemplos de estas tareas son entender el habla o las imágenes, y automatizar tareas rutinarias. El campo que desarrolla estas tareas se denomina Inteligencia Artificial (IA) [4] y cada vez tiene más presencia en temas de investigación.

En IA existen diversos desafíos muy interesantes; sin embargo, en la mayoría de ellos es extremadamente difícil alcanzar el rendimiento y la eficiencia del cerebro humano. Las máquinas nos superan en tareas como procesamiento de gran cantidad de datos, almacenamiento de información o tareas de razonamiento como el juego de ajedrez. Sin embargo, algunas habilidades que el ser humano realiza inconscientemente, como caminar o ver, son aún muy complejas para las máquinas.

La IA comprende diferentes campos: *Machine Learning*, incluyendo *Knowledge Engineering*, Redes Neuronales Artificiales (RNA) [5], procesamiento del lenguaje natural, Minería de datos, Visión Artificial (VA), etc. Este proyecto se enfoca en la VA, que trata de analizar y procesar imágenes de tal forma que un ordenador sea capaz de interpretar dichas imágenes. La VA intenta conseguir que una máquina realice el mismo proceso que el Sistema Visual Humano de tal forma que sea capaz de tomar decisiones y actuar en función de la situación en que se encuentre.

El aprendizaje de las máquinas es un punto de encuentro de diferentes disciplinas que engloba a la estadística, la programación y la optimización, entre otras. La VA intenta simular las capacidades del ojo y el cerebro humanos y en ella se utilizan comúnmente técnicas de aprendizaje máquina.

Uno de los problemas que se está estudiando ampliamente en VA en la última década es la conducción autónoma. Los humanos somos capaces de mirar a la carretera y saber al instante si el coche que conducimos está en una curva o una recta, si hay coches alrededor y cómo interactúan entre ellos. En función a la situación en la que nos encontramos sabemos qué acciones llevar a cabo para lograr una buena conducción. Sin embargo, este procedimiento es más complicado para los ordenadores. En la actualidad se está investigando ampliamente cómo emplear las Redes Neuronales Artificiales (RNA) para materializar comportamiento autónomo en vehículos.

Un claro ejemplo es la navegación en robótica (Figura 1.1), donde la visión constituye una capacidad sensorial más para la percepción del entorno que rodea al robot. Generalmente se recurre a técnicas de visión estereoscópica con el fin de reconstruir la escena 3D. En algunas ocasiones se añade algún módulo de reconocimiento con el fin de identificar la presencia de determinados objetos, hacia los que debe dirigirse o evitar. Cualquier información que pueda extraerse mediante VA supone una gran ayuda para el movimiento del robot.

Figura 1.1: Navegación en robótica mediante VA

La reducción de accidentes gracias a vehículos autónomos es una realidad gracias a la VA, ya que los sistemas de guiado que poseen estos vehículos están basados en esta visión. Algunos ejemplos de estos sistemas (Figura 1.2) son: los sistemas de aviso de cambio de carril, o de control de velocidad de crucero.

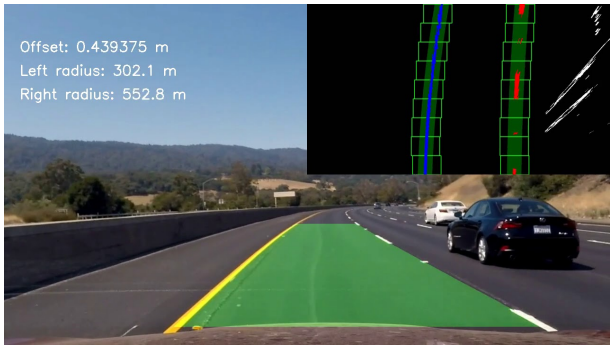


Figura 1.2: Conducción autónoma

Otro ejemplo en donde la VA supone un gran avance es la comunidad médica, donde permite diagnosticar con mayor rapidez y detalle enfermedades y lesiones. De esta forma es posible aplicar tratamientos personalizados y eficaces en menor tiempo. Un claro ejemplo de investigadores que emplean VA es el MIT Computer Science & Artificial Intelligence Lab (CSAIL) [6], donde el desarrollo de algoritmos que analizan mamografías de una forma novedosa permite ayudar a detectar el cáncer de mama (Figura 1.3) con hasta cinco años de anticipación.

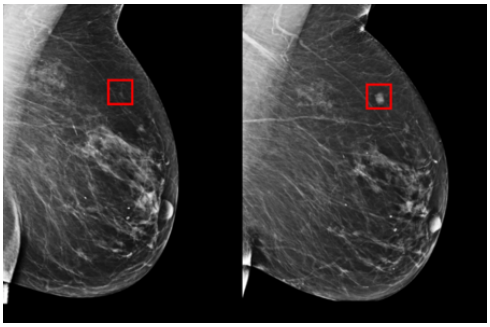


Figura 1.3: Detección de cáncer de mama

Otra aplicación es el mantenimiento e inventariado urbano. Es posible identificar problemas en instalaciones y mobiliario urbano (averías, mal estado de contenedores (Figura 1.4), socavones en la vía pública, etc) mediante cámaras ubicadas por ejemplo en autobuses. Los mantenimientos de infraestructuras de transporte, como vías y cables ferroviarios, pueden programarse automáticamente implantando sistemas de VA en los propios trenes.

Figura 1.4: Detección de contenedores

1.2. Robótica educativa

La robótica educativa es un campo en auge debido al gran desarrollo de la robótica y la necesidad creciente de especialistas y profesionales en todo el mundo. En 2015, la Comunidad de Madrid introdujo la asignatura “Tecnología, Programación y Robótica” en el plan docente de Enseñanza Secundaria[1]. En el año 2020- 2021 se prevé implantar la asignatura “Programación y Robótica” en el plan docente de Enseñanza Primaria[2].

Alineado con la robótica, en los últimos años, está creciendo el interés en la educación Science, Technology, Engineering and Mathematics (STEM). Esta nueva forma de enseñanza promueve el pensamiento científico y la adquisición de conocimientos

tecnológicos aplicables a situaciones reales permitiendo el desarrollo de competencias en la resolución de problemas.

Para impartir este tipo de asignaturas es necesaria la infraestructura adecuada. Las plataformas como la de *LEGO* o la de *Arduino* permiten una enseñanza y aprendizaje sencillos de la robótica resultando muy gratificante para el alumno por lo vistoso de los resultados y la obtención de una aplicación real inmediata.

Es importante acercar este conocimiento tan complejo de manera adecuada a edades cada vez más tempranas para facilitar un mayor desarrollo en el conocimiento de esta tecnología. Es por ello que cada vez hay más plataformas y entornos STEM que desarrollan software orientado a niños. Algunos ejemplos de plataformas educativas son:

- Scratch[7]. Es un proyecto utilizado en docencia y liderado por el *MIT*[1] para programar animaciones, interacciones y juegos de manera sencilla por su interfaz visual. Toda la funcionalidad de un lenguaje de programación está embebida en bloques gráficos que agrupan funcionalidades específicas.

- LEGO[8]. Se trata de una plataforma que dispone de multitud de robots programables también con su sistema gráfico basado en *LabView*[2].

- Kodu[9]. Es un sistema de programación visual para el desarrollo de videojuegos desarrollado por Microsoft[3]

- Snap![10]. Se trata de un lenguaje visual y una plataforma inspirada en Scratch pero con funcionalidad destinada a edades más avanzadas que permiten el desarrollo de funcionalidad más compleja.

- OpenRoberta[4] [7]. Es una plataforma educativa en la nube desarrollada por el instituto alemán Fraunhofer IAIS para que los niños puedan programar robots utilizando Lego Mindstorms o robots programables(Figura 1.5)

[1] `https://www.mit.edu`

[2] *LabVIEW* es una plataforma y entorno de desarrollo para diseñar sistemas, con un lenguaje de programación visual gráfico

[3] `https://www.microsoft.com/es-es`

[4] `https://lab.open-roberta.org/`

Figura 1.5: Plataforma Open Roberta

- Vex[5]. Plataforma orientada a la disciplina STEM de enseñanza que ofrece una gran cantidad de robots programables, así como tutoriales para aprender a montarlos y manejarlos.

- AppInventor[6]. Es un entorno de desarrollo software creado por Google Labs y actualmente mantenido por el *MIT* que ofrece toda la infraestructura necesaria para crear aplicaciones en Android. De esta manera, y visualmente, el usuario puede desarrollar una aplicación con bloques de código visuales, parecidos a Scratch.

- Arduino Web Editor[7]. Arduino Web Editor es una plataforma que proporciona al usuario todo lo necesario para poder programar su código en línea. De esta manera, no es necesario que el usuario instale ningún tipo de software en su sistema. Esto facilita mucho el aprendizaje dado que es inmediato (Figura 1.6)

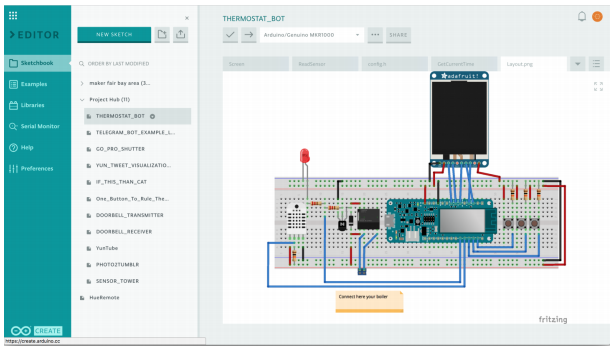


Figura 1.6: *Arduino* Web Editor

[5] `https://www.vexrobotics.com/`
[6] `https://appinventor.mit.edu/`
[7] `https://create.arduino.cc/`

Otra plataforma de robótica educativa, en la que se ha desarrollado este trabajo, es *Kibotics* (Figuras 1.7 y 1.8), que ofrece distintos entornos simulados y con robots reales a los alumnos. Con esto, los alumnos pueden programar la inteligencia de robots autónomos para que realicen distintas pruebas sin ningún riesgo. De esta manera es posible aprender robótica en casi cualquier edad de manera sencilla y con la única necesidad de acceso a internet.

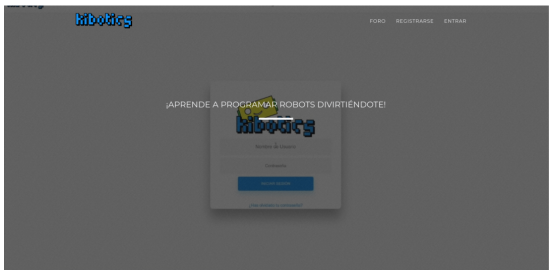


Figura 1.7: Plataforma *Kibotics*

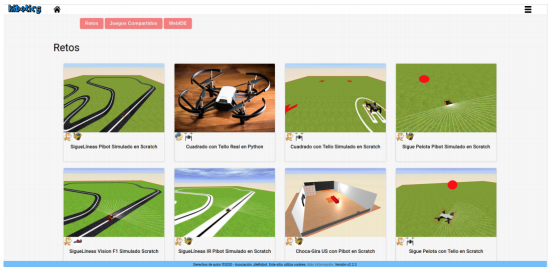


Figura 1.8: Parrilla de ejercicios en *Kibotics*

Esta plataforma permite el aprendizaje de la robótica con un simulador en un entorno web. De esta manera, los usuarios sólo necesitan conexión a internet. Soporta distintos modelos de robots como el dron Tello o el Mbot, entre otros, tanto de manera simulada como en los robots reales. Además ofrece dos lenguajes de programación, Scratch (Figura 1.9) y Python (Figura 1.10) para llegar a usuarios de distintas edades.

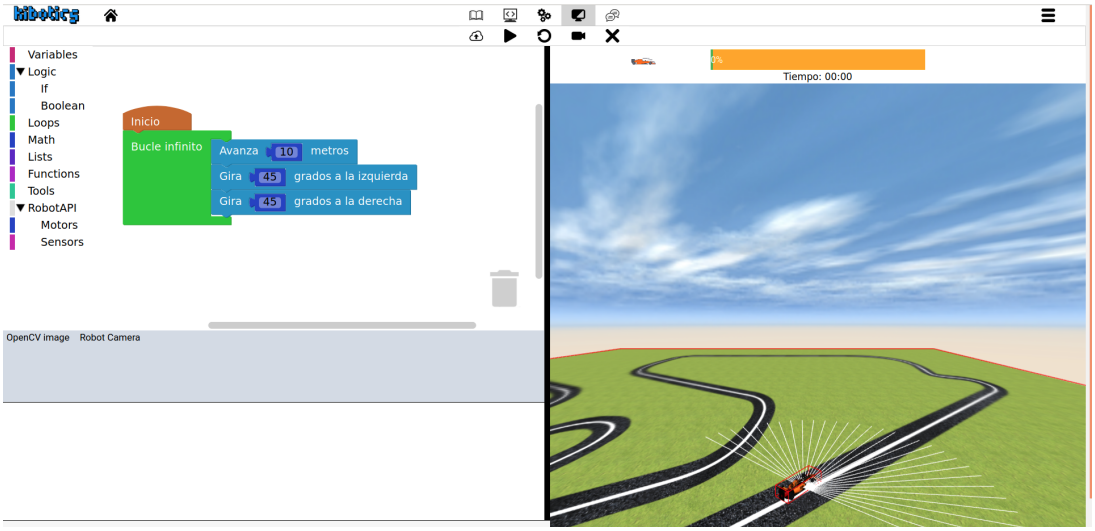


Figura 1.9: Editor y simulador de un ejercicio *Scratch*

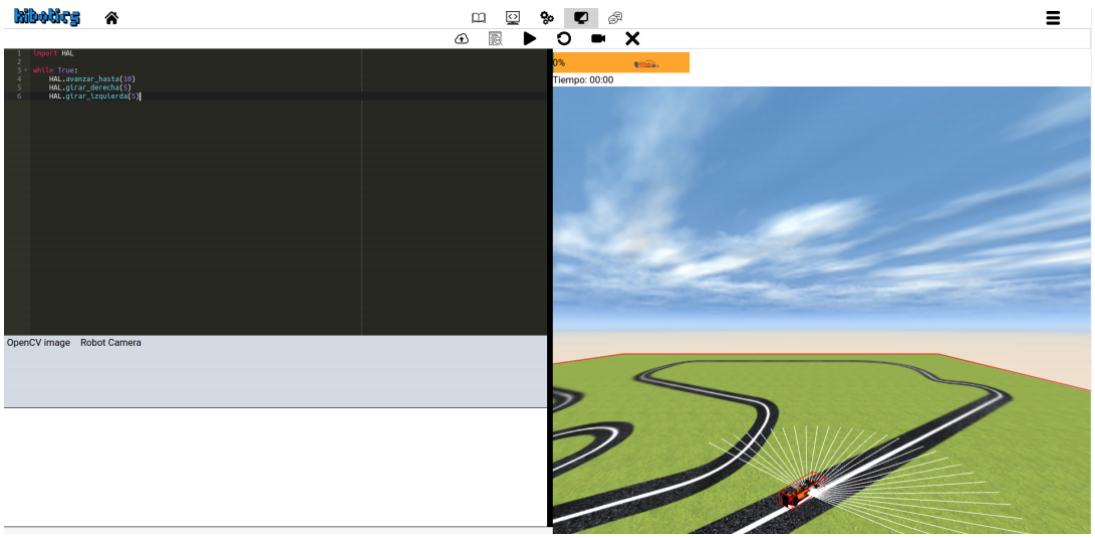


Figura 1.10: Editor y simulador de un ejercicio *Python*

1.3. Objetivos

La meta principal de este TFM ha sido la creación de la arquitectura necesaria que permite añadir el uso de procesamiento visual complejo de manera sencilla en la plataforma *Kibotics* en particular funciones de detección visual robusta de objetos en imágenes utilizando técnicas de aprendizaje profundo. Esto permitirá a los niños usuarios de *Kibotics* programar comportamientos interesantes que manejen visión.. Este objetivo se ha dividido a su vez en otros dos subobjetivos que con una aplicación robótica concreta que use esa detección visual robusta, tanto con un robot real como con un robot simulado. Ambos validarán experimentalmente la infraestructura desarrollada.

- **Comportamiento SiguePersona visual con drone real** que permite tener la infraestructura preparada para poder usarlo en los cursos que se desarrollan en la plataforma. El estímulo a identificar en las imágenes son las personas en el entorno del drone.
- **Comportamiento SiguePersona visual con drone simulado** para tener la infraestructura preparada también en el entorno simulado. Esto conlleva la introducción de personas simuladas en los escenarios de *Kibotics*.

Requisitos

Las soluciones a desarrollar para alcanzar los objetivos descritos deben cumplir además, los siguientes requisitos:

- El desarrollo de la versión con el robot real debe hacerse en *Python 3*.
- Para el desarrollo del código en la versión simulada se debe usar *HTML-5*, *CSS-3* y *Javascript*.
- Debe funcionar dentro de *Kibotics v2.0* o superior.
- La infraestructura de *DeepLearning* tiene que funcionar ágilmente, en tiempo real, para que el comportamiento de seguimiento sea fluido y satisfactorio.

Capítulo 2

Estado del arte

Una vez descrito el contexto general de este TFM, en este capítulo se profundiza en redes neuronales para detección visual de objetos en imágenes, puesto que serán la base principal del procesamiento visual necesario en el comportamiento sigue-persona. Estas redes neuronales se entrenan típicamente con grandes bases de datos con objetos etiquetados. Se presentan en este capítulo las más relevantes.

2.1. Redes neuronales para detección visual de objetos

Rigoberto Vizcay [1] se basa en redes *Convolutional Neural Network (CNN)* para la detección de vehículos y peatones. Una red neuronal convolucional es una red que presenta una o varias capas convolucionales. La red que implementó constaba de dos capas de convolución y dos de agrupamiento, además de la capa de salida completamente conectada. En la Figura 2.1 se puede ver la estructura de la red.

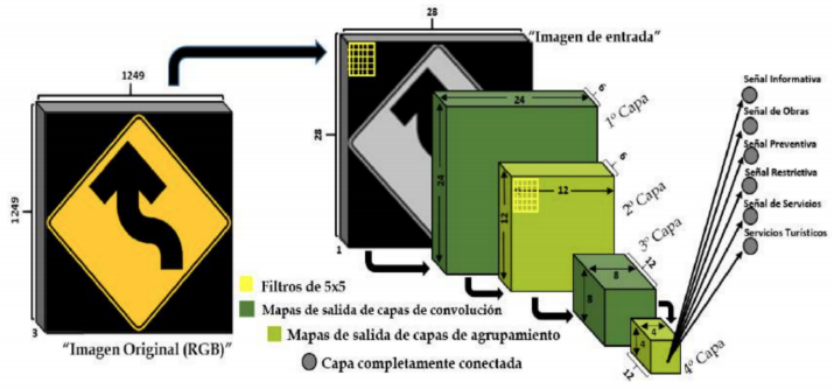


Figura 2.1: CNN Rigoberto Vizcay [1]

Y. Abdullah, G. Mehmet, A. Iman and B. Erkan [11] plantean el uso de *Region-based Convolutional Neural Network (R-CNN)* y Faster R-CNN para la detección de vehículos. Por un lado, R-CNN para realizar las detecciones realiza tres fases que se pueden ver en la Figura 2.2:

1. Se emplea el algoritmo *Selective Search*, el cual solo realiza las pruebas de detección a las regiones candidatas a tener una posible detección. Con ello se extraen aproximadamente 2000 regiones de la imagen (*Region Proposal*).

2. Se implementa una red neuronal *Convolutional Neural Network (CNN)* en la parte superior de cada región.

3. Se extrapola la salida de cada CNN y se ingresa en una *Support Vector Machine (SVM)* para clasificar la región. Además se realiza una regresión lineal para restringir el cuadro de la detección.

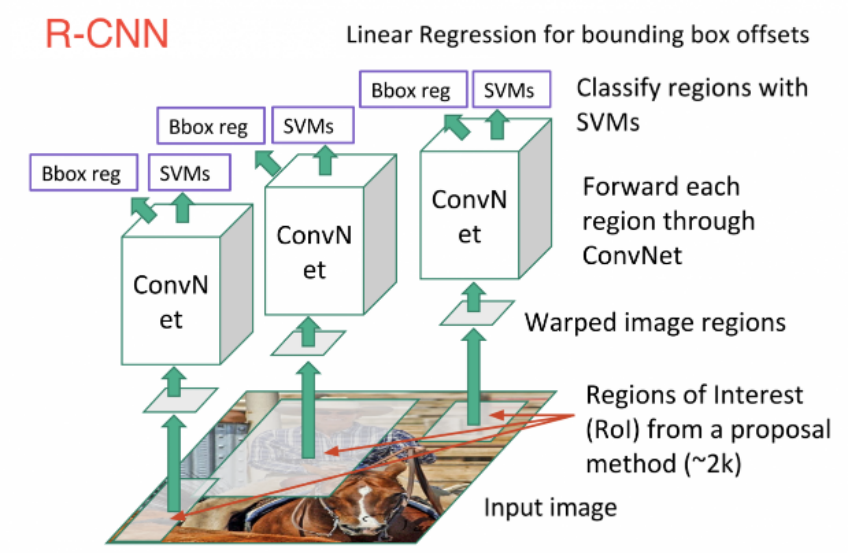


Figura 2.2: Fases de R-CNN

Por otro lado, Faster R-CNN (Figura 2.3) es una versión más rápida que R-CNN, en la que se modifican algunos aspectos. En este método se incluye la técnica *region proposal*, la cual determina qué regiones de la imagen tienen mayor probabilidad de contener objetos y por tanto cuáles son las regiones que se introducirán en el clasificador. Esto optimiza mucho el trabajo pues evita introducir al clasificador regiones que no sean de interés.

Ignacio Arriola [12] hace también uso de Faster R-CNN. En este trabajo se han entrenado y comparado tres redes Faster R-CNN para la detección de peatones partiendo de diferentes parámetros iniciales. Con ello se pretendía estudiar la transferencia del aprendizaje, experimentando con dos redes pre-entrenadas y una inicializada con parámetros aleatorios. Las redes pre-entrenadas se trataban de una Faster-R-CNN [13] y una red convolucional Resnet 101 [14] como extractor de características. Cada modelo fue entrenado con una base de datos diferente (uno con *Common Objects in Context (COCO)*[1] y otro con KITTI [2]).

[1] http://cocodataset.org/#home
[2] http://www.cvlibs.net/datasets/kitti/

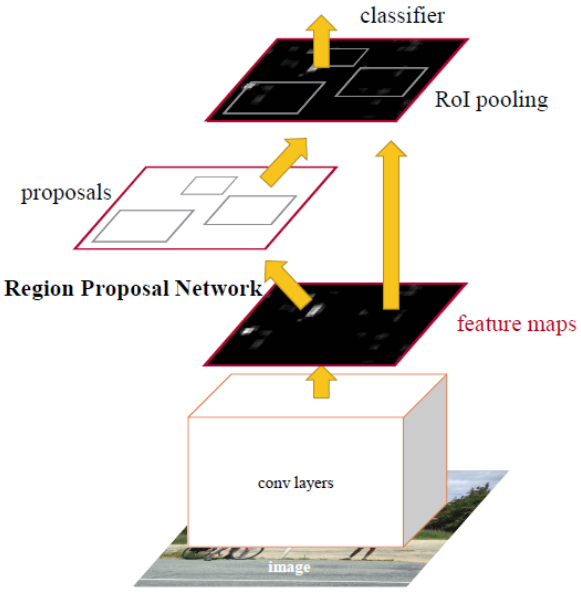


Figura 2.3: Fases de Faster R-CNN

Single-Shot Multibox Detector (SSD)

Otra arquitectura sobresaliente de detección de objetos es el Single-Shot Multibox Detector (SSD) [2]. El principal beneficio de esta arquitectura es el hecho de que integra todos los cálculos necesarios en una sola red neuronal, reduciendo la complejidad en comparación con otras enfoques que requieren propuestas de regiones externas. Esto reduce enormemente el tiempo de cálculo cuando la red tiene que procesar una imagen. La arquitectura puede verse en la Figura 2.9, y puede dividirse en varias etapas [15], a saber:

- ***Reshape***. la primera tarea que debe abordar la red es remodelar la(s) imagen(es) de entrada a un tamaño fijo en el que trabajen el resto de las capas. En el caso de un detector de SSD, esta forma es n $\times$ 300 $\times$ 300 $\times$ 3 (siendo n el tamaño del lote de entrada, ya que n imágenes pueden ser evaluadas simultáneamente en la red neuronal). Se pueden utilizar otros tamaños de imagen, pero éste ofrece un buen equilibrio entre el rendimiento y la carga de cálculo.

- **red base**: este primer grupo de capas se reutilizan de un modelo típico de clasificación de imágenes, como el VGG-16 [16]. Las primeras capas de esta arquitectura se utilizan en este diseño, truncadas antes de la primera capa de

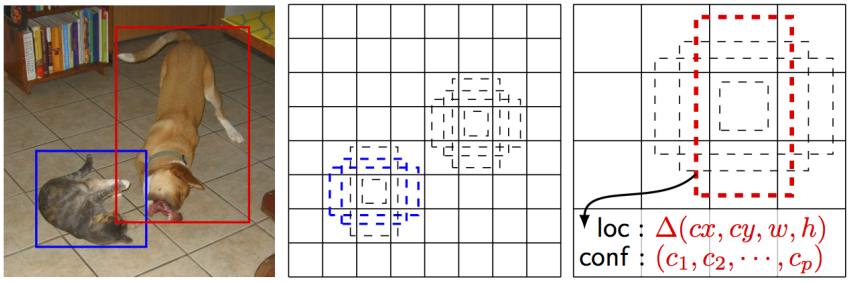
clasificación. De esta manera, la red puede aprovechar los mapas de características de la red de clasificación, para encontrar objetos dentro de la imagen de entrada. Después de la primera parte de la red, se añaden varias capas convolutivas, disminuyendo su tamaño. Esto tiene el objetivo de predecir las detecciones a múltiples escalas. Una cosa que hay que mencionar en este punto es que la red base puede ser una diferente en lugar de la VGG-16, como una *MobileNet* [17], que está altamente optimizada para funcionar en dispositivos de gama baja.

- **Box predictors**: para cada capa de la red de base, se realiza una convolución de imagen, generando un pequeño conjunto (típicamente 3 o 4) de anclas de tamaño fijo, con diferentes relaciones de aspecto para cada celda de una cuadrícula sobre el mapa de activación (Figura 2.4). Como estos mapas tienen diferentes tamaños, el sistema es capaz de detectar objetos en diferentes escalas. Las anclas se conviven entonces con pequeños filtros (uno por canal de profundidad), que producen resultados de confianza para cada clase conocida, y compensaciones para el cuadro delimitador generado. Estas puntuaciones se pasan a través de una operación de *softmax*, que los comprime en un vector de probabilidad. Así, para cada objeto detectado (en esa escala), la red calcula la puntuación de cada clase y su posición estimada dentro del mapa de características (por lo tanto, en la imagen también).

- **Postprocesamiento**: Dado que pueden producirse varias detecciones en la misma zona para diferentes clases y escalas, se realiza una operación de no máxima supresión [22] a la salida de la red para retener las mejores cajas, bajo un criterio combinado de puntuación de detección y puntuación IoU (Intersección sobre Unión), que mide la calidad de superposición entre dos cajas delimitadoras, como puede verse en la figura2.5.

You Only Look Once (YOLO)

Otro enfoque interesante es el sistema You Only Look Once (YOLO) [18]. Su principal ventaja es su velocidad de inferencia, debido al hecho de que realiza un único análisis de toda la imagen, dividiéndola en una cuadrícula de células. Cada celda predice hasta 5 casillas, que contienen una puntuación de objeto (el IOU predicho de la propuesta con un objeto, independientemente de su clase), las coordenadas de la casilla delimitadora, y una probabilidad para el objeto perteneciente a cada clase. Este diseño funciona más rápido



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Figura 2.4: Un grupo de cuadrados generados en cada punto de cada mapa de características [2].

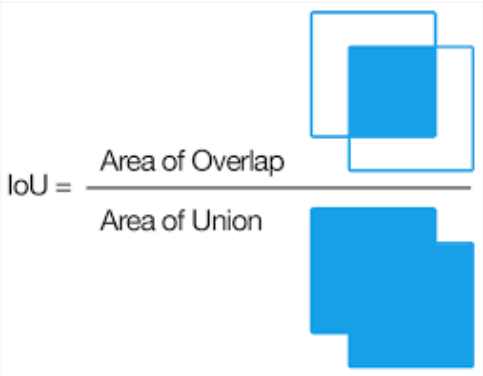


Figura 2.5: Representación gráfica de una IoU entre dos bounding boxes.

que otros métodos [18], sin embargo presenta un rendimiento pobre al detectar objetos pequeños.

Este diseño fue revisado en *YOLO9000* [3], introduciendo varias mejoras como la normalización de los lotes a la entrada de las capas convolucionales, o el concepto de cajas de anclaje: las propuestas de cajas siguen un conjunto fijo de relaciones de aspecto, elegidas previamente utilizando la agrupación en un conjunto de entrenamiento. Como puede verse en la figura 2.6, la limitación de las formas de la propuesta a 5 tamaños fijos mejora el rendimiento, manteniendo al mismo tiempo un alto *IuO*. Una inspección visual muestra que las anclas seleccionadas parecen tener una forma razonable para la mayoría de los objetos que la red pretende detectar. Además, el número de capas profundas fue aumentó de 26 capas a 30, y se realiza un modelado semántico en las etiquetas a través de diferentes conjuntos de datos, lo que permite a la red capacitarse en diferentes conjuntos de datos bajo una estructura semántica llamada *WordTree* (Figura 2.7).

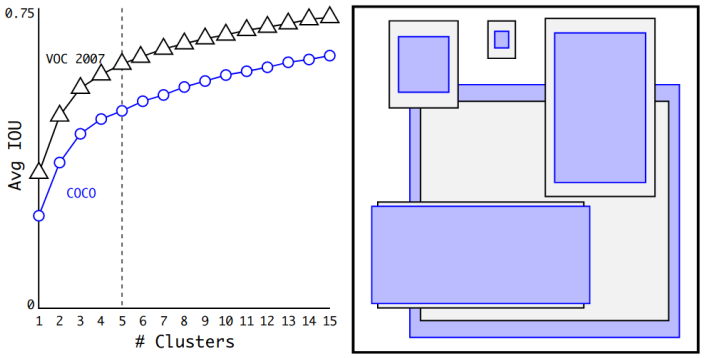


Figura 2.6: El resultado del ancla k significa agrupación en VOC y COCO para YOLO9000. Usar tamaños de ancla de $k = 5$ en la derecha produce un buen equilibrio entre la simplicidad y la mejora en el IuO obtenido con respecto a usar clusters de $k - 1$ (source: [3]).

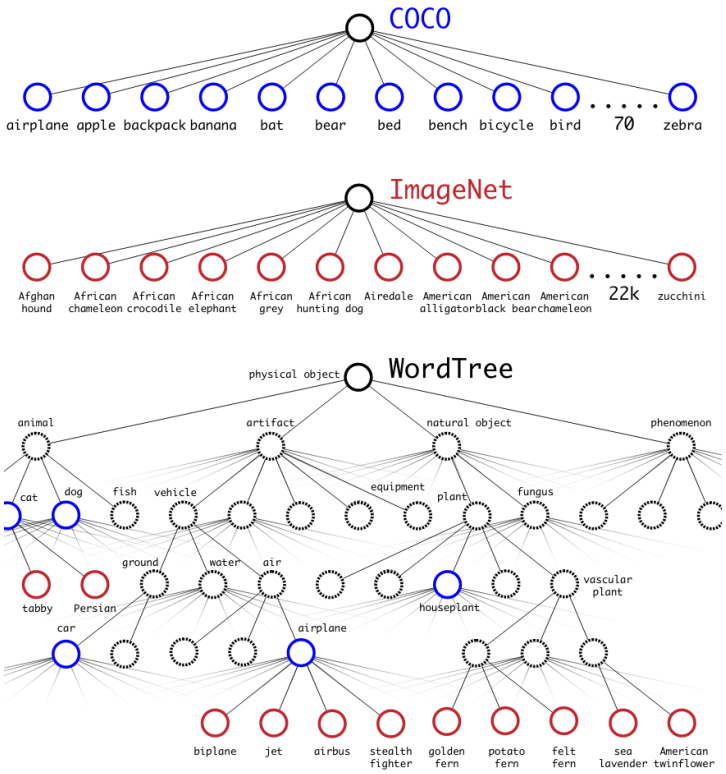


Figura 2.7: Comparación entre estructuras de etiquetado simples (arriba) y una agrupación semántica de WordTree bajo categorías (abajo). Esto permite seguir un proceso de formación de datos agnóstico, ya que las etiquetas pueden combinarse utilizando WordTree. Imagen de [3].

La última mejora de YOLO, *YOLOv3* [19], se basa en redes residuales [20], que abordan el problema de la desaparición de los gradientes cuando las redes se hacen más profundas. El apilamiento de varias capas resulta en gradientes que disminuyen su valor hasta un punto que la precisión aritmética de la máquina no es capaz de manejar. Los gradientes se cancelan, dificultando el proceso de formación, ya que los parámetros de las primeras capas tardan un tiempo sustancialmente mayor en converger. Las redes residuales añadidas en esta revisión del diseño añaden conexiones de atajo a través de las capas, centrando los gradientes de retropropagación en las diferencias entre la entrada y la salida de la capa. Como dice esta referencia [19], la combinación de estas capas residuales y las convolucionales permite entrenar arquitecturas mucho más profundas (53 capas convolucionales), capaces de producir una mayor generalización. Al igual que en los detectores de SSD, la arquitectura YOLO realiza detecciones a múltiples escalas, utilizando 3 escalas para dividir los mapas de características en cuadrículas de células. En el conjunto de datos COCO se realiza una agrupación similar a la de la figura 2.10, seleccionando 9 tamaños de ancla en lugar de 5, y agrupándolos en 3 escalas. Ahora, en cada una de las celdas, caben 9 cajas delimitadoras de anclas (3 formas de anclas $\times$ 3 escalas). Esto tiene como objetivo mejorar el pobre rendimiento de la versión anterior cuando se trata de objetos pequeños, así como producir una mejor generalización: en el *R-CNN* [16] y en el SSD [19] las formas de anclaje se seleccionan a mano. Estos cambios, con un ajuste en la función de error, conforman las mejoras de *YOLOv3* sobre las versiones anteriores.

Para (ancla, celda, escala) combinación, *YOLOv3* predice:

- Las coordenadas del objeto dentro del ancla. Los detalles se pueden visualizar en la figura 2.12.
- *Puntuación de objetividad*, que se calcula mediante una regresión logística a fin de maximizar la probabilidad de superposición con un cuadro delimitador de la verdad del terreno con respecto a la de cualquier otra ancla anterior.
- 80 puntos, ya que la implementación original está entrenada en el conjunto de datos del COCO, que contiene 80 clases. Estas clases pueden estar superpuestas (por ejemplo, "woman" "person"). Por lo tanto, estas puntuaciones son calculadas por clasificadores logísticos independientes y no se pasan por una operación de softmax.

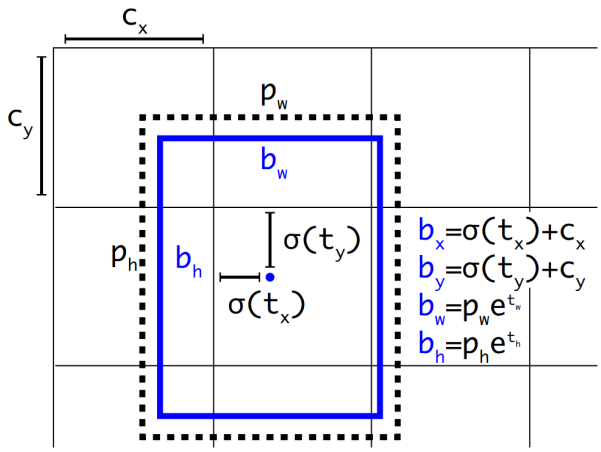


Figura 2.8: Salida en YOLO para cada ancla y célula. La línea discontinua representa el ancla anterior, mientras que la línea azul representa la detección que corrige esa ancla.

La arquitectura de una red de detección basada en YOLO puede compararse con la de una basada en SSD en la figura 2.13. Esto permite ver la diferencia fundamental en la etapa de extracción de características de cada enfoque.

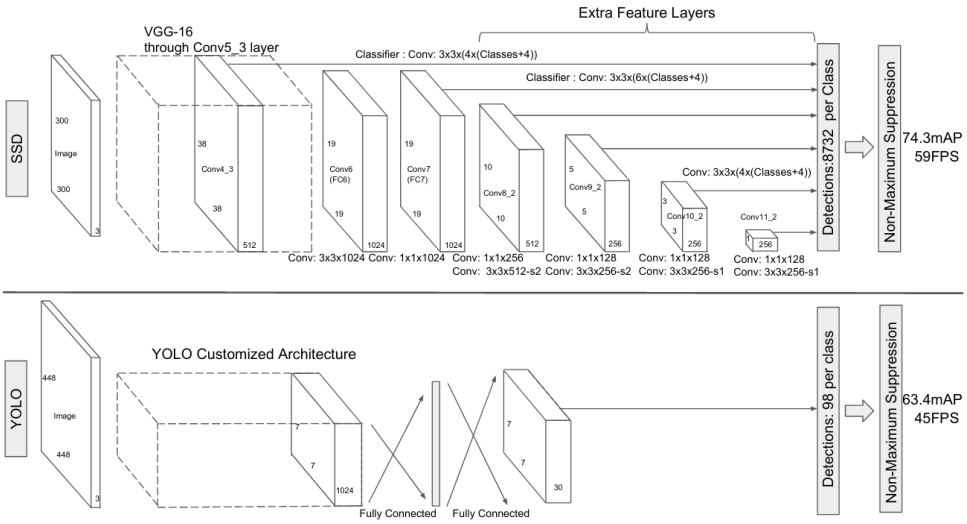


Figura 2.9: Arquitectura general de una red SSD (arriba) y una YOLO (abajo). Imagen de [2].

2.2. Bases de datos de detección visual de objetos

La detección visual de personas pretenden encontrar una persona en una imagen o en un vídeo. Dado que queremos encontrar la persona en cuestión bajo diferentes

circunstancias, es decir, en distintos entornos y diferentes iluminaciones, necesitaremos típicamente entrenar el modelo con un conjunto de imágenes representativo. Por este motivo, a lo largo de los últimos años han surgido en la comunidad internacional diferentes *datasets* con el fin de solucionar este problema.

2.2.1. Common Objects in Context (COCO)

El *dataset* de Common Objects in Context (COCO)[3][21] es uno de los conjuntos de imágenes mas usados para entrenar redes para detección de objetos. Esto se debe a que contiene mas de 200.000 imágenes etiquetadas (figura 2.10) con en torno a 80 clases de objetos diferentes. Además no solo se usa para detección, también para segmentación, clasificación y otros usos.

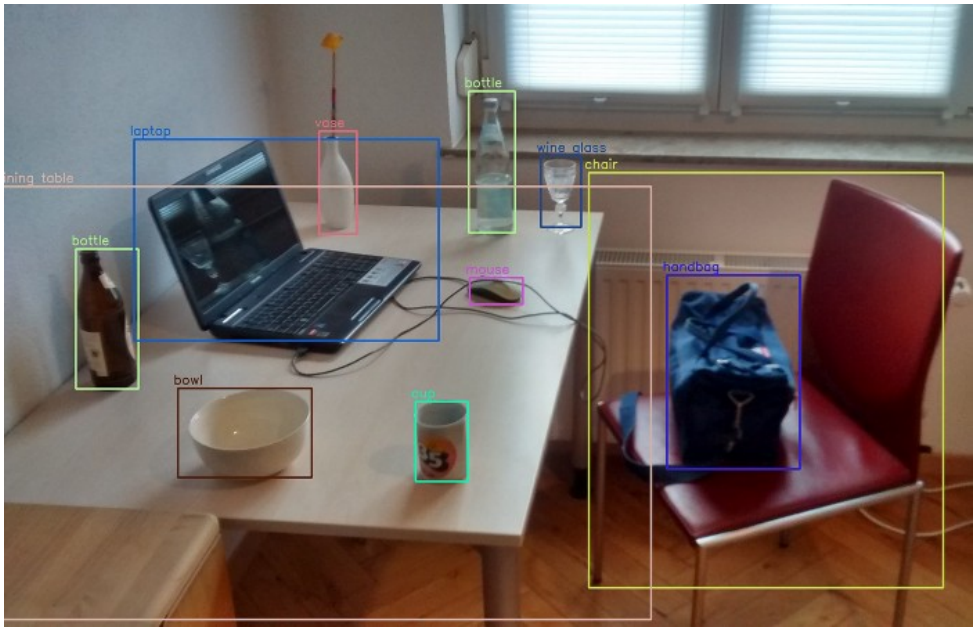


Figura 2.10: ejemplo de *COCO*

2.2.2. Open Images

Open Images[4] [22] es un conjunto de datos de aproximadamente 9 millones de imágenes anotadas (figura 2.11) con etiquetas de nivel de imagen, cajas delimitadoras de objetos, máscaras de segmentación de objetos, relaciones visuales y narraciones localizadas. Las

[3] `https://cocodataset.org`

[4] `https://storage.googleapis.com/openimages/web/index.html`

cajas han sido dibujadas en gran parte manualmente por anotadores profesionales para garantizar la precisión y la coherencia. Las imágenes son muy diversas y a menudo contienen escenas complejas con varios objeto.

También ofrece anotaciones de relaciones visuales, indicando pares de objetos en relaciones particulares (por ejemplo, "mujer tocando la guitarra"), propiedades de los objetos (por ejemplo, "la mesa es de madera") y acciones humanas (por ejemplo, "la mujer está saltando").

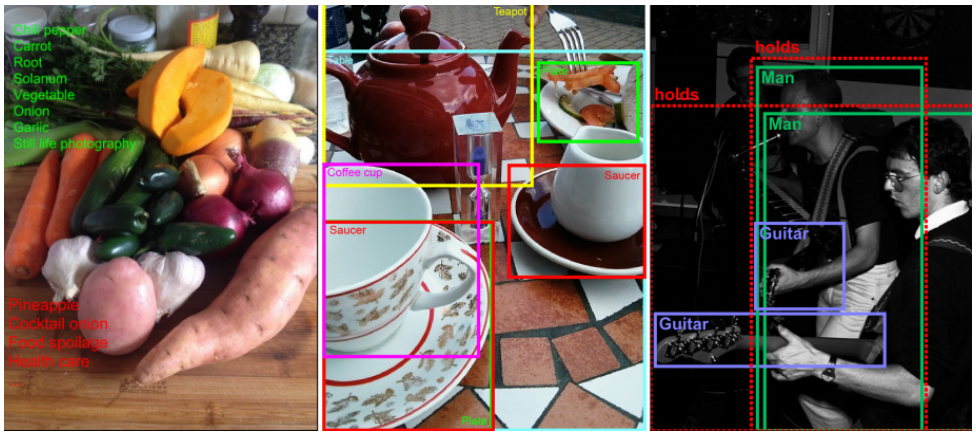


Figura 2.11: ejemplo de *Open Images*

2.2.3. PASCAL Visual Object Classes Challenge (VOC) 2012

Este conjunto de datos contiene los datos del *PASCAL Visual Object Classes Challenge (VOC)* 2012[5] [23], también conocido como *VOC2012*, correspondiente a las competiciones de Clasificación y Detección. Un total de 11540 imágenes están incluidas en este conjunto de datos, donde cada imagen contiene un conjunto de objetos, de 20 clases diferentes, haciendo un total de 27450 objetos anotados.

2.2.4. ImageNet

El proyecto ImageNet[6] es una gran base de datos visual diseñada para su uso en la investigación de software de reconocimiento visual de objetos. Contine más de 14 millones de imágenes anotadas a mano para indicar qué objetos se han fotografiado y en al menos

[5] `http://host.robots.ox.ac.uk/pascal/VOC/voc2012/`

[6] `http://image-net.org/index`

un millón de las imágenes también se proporcionan recuadros delimitadores. Además contiene más de 20.000 categorías.

Capítulo 3

Herramientas utilizadas

EL objetivo de este trabajo es preparar la infraestructura para desarrollar prácticas de visión con drones en la plataforma Kibotics[24] tanto con drone real como simulado. Para ello se han necesitado las siguientes herramientas software.

3.1. Plataforma educativa Kibotics

Kibotics [24] es una plataforma web de enseñanza de robótica infantil desarrollada por las Asociación de robótica e Inteligencia Artificial (IA) JdeRobot[1]

La batería de ejercicios (Figura 3.1) que incluye el entorno *Kibotics* [24] se realiza usando el simulador robótico *Websim*. Se trata de un simulador diseñado para el aprendizaje de conceptos básicos de programación de robots especialmente para niños y funciona íntegramente dentro de un navegador.

[1]https://jderobot.github.io

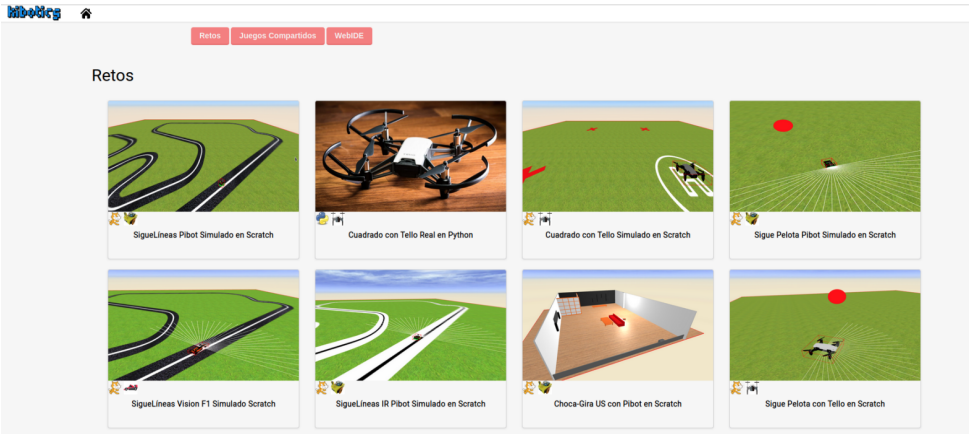


Figura 3.1: Ejercicios Kibotics

Este simulador permite que los usuarios puedan programar fácilmente los movimientos de los robots, ya que simplemente tienen que acceder a la información que recogen sus sensores y enviar las órdenes precisas a los actuadores del robot. Estas órdenes se deben programar en *Python* o *Scratch* (figura 3.2), dentro del editor que incorpora la interfaz de *Websim*.

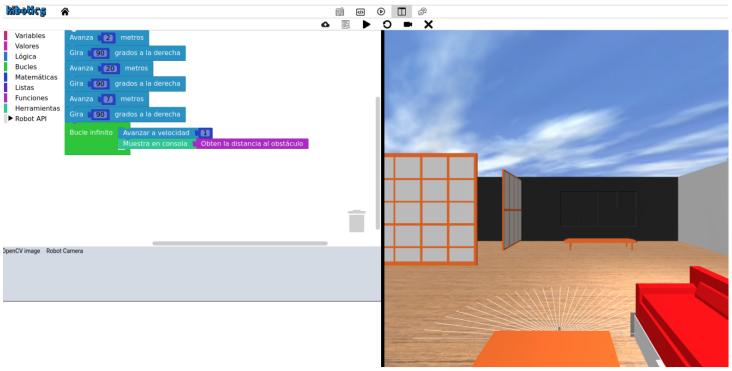


Figura 3.2: Ejercicio con Scratch

Tanto en la versión simulada como en la real, el API proporcionado por la plataforma para interactuar con el robot contiene una función *getImage* que devuelve una captura de la cámara de dicho robot en formato RGB. Además contiene una pequeña aproximación al mundo de la Visión Artificial (VA) con la función de su API *GetColoredObject*, que recibe un color en RGB y consiste en un pequeño filtro de color aplicado sobre la imagen y que devuelve a los niños la el **bounding box** del objeto de color indicado.

Se ha usado para la parte simulada.

3.2. Tensorflow y TensorflowJS

Es una herramienta y biblioteca de código abierto *end-to-end* para el *Machine Learning* que fue liberada bajo licencia de *Apache* 2 a finales de 2015 y que está disponible en github [25]. Fue desarrollada por el equipo de investigación en *Machine Learning "Google Brain"* en *C++* y *Python* y es usada en multitud de productos y servicios de Google como Gmail o *Google Translation*. Google ofrece en su plataforma *Cloud* ejecutar *Tensorflow* en *Tensor Processing Unit* (TPU), un nuevo tipo de procesadores en *Cloud* optimizados para ejecutar Inteligencia Artificial (IA). *TensorFlow* está orientado a problemas de *Deep Learning* y permite entrenar y construir redes neuronales.

TensorFlow puede correr tanto en CPUs como en GPUs (haciendo uso de *Compute Unified Device Architecture (CUDA)*). Está disponible en *Linux* de 64 bits, *MacOS*, y plataformas móviles que incluyen *Android* e *iOS*. Además ha incluido soporte para *Javascript*, por lo que puede ser usado en cualquier navegador. Actualmente es el entorno más popular en *Deep Learning*.

Puede ejecutar de forma rápida y eficiente gráficos de flujo. Un gráfico de flujo está formado por operaciones matemáticas representadas sobre nodos, y cuya entrada y salida es un vector multidimensional o tensor de datos, por este motivo recibe el nombre de *TensorFlow*.

Las ventajas de este software se extienden a muchas disciplinas aparte de la tecnología TIC. Se emplea en imágenes médicas para la detección de tumores por ejemplo, también se usa en la detección y combinación de estilos artísticos en la pintura, etc.

En este TFM se emplean la versión 1.15.0 y 2.2.0 de *Tensorflow* para *Python*, que se emplea en el drone real y la versión 2.0.1 para *Javascript*, empleada con el drone simulado.

3.2.1. Object Detection API

Object Detection API[2] es un *framework* de código abierto construido sobre *TensorFlow* que facilita la construcción, el entrenamiento y el despliegue de modelos de detección de objetos desarrollado por *Google*. Todas las redes de detección preentrenadas que se pueden descargar desde la web de *Tensorflow* han sido entrenadas con este API.

Hasta Julio de 2020, cuando publicaron la primera versión con soporte para *Tensorflow* 2.x, solo tenía solo soporte para la versión 1.x. En este trabajo se ha usado la versión 2.2.1

[2] `https://github.com/tensorflow/models/tree/master/research/object_detection`

de *Object Detection API*.

3.3. Biblioteca Opencv

OpenCV [3] es una librería de código abierto desarrollada por *Intel* y publicada bajo licenciade BSD. Esta librería implementa gran variedad de fuciones para la interpretación de la imagen. Sus siglas provienen de los términos anglosajones "*Open Source Computer Vision Library*", y tal y como se puede , es una librería destinada a aplicaciones de visión por computador en tiempo real. Puede ser empleada en MacOS, Windows y Linux, y existen versiones para *C#*, *Python* y *Java*, a pesar de que originalmente era una librería en *C/C++*. Además hay interfaces para *Ruby*, *Python*, *Matlab* y otros lenguajes.

En 2017 añadieron *soporte para Javascript* mediante *emscripten*[4] que es un software que permite convertir bibliotecas escritas en *C++* en binarios para *Javascript* (*OpenCVJS*). Esto permite tener muchas de las funcionalidades desarrolladas en *C++* en el navegador.

OpenCV implementa algoritmos para técnicas de calibración, detección de rasgos, rastreo, análisis de la forma, análisis del movimiento, reconstrucción 3D, segmentación de objetos y reconocimiento, etc. Los algoritmos se basan en estructuras de datos flexibles acopladas con estructuras IPL (*Intel Image Processing Library*), aprovechándose de la arquitectura de Intel en la optimización de más de la mitad de las funciones. Incorpora funciones básicas para modelar el fondo, sustraer dicho fondo y generar imágenes de movimiento MHI (*Motion History Images*). Además incluye funciones para determinar dónde hubo movimiento y en qué dirección.

[3] https://opencv.org/

[4] https://emscripten.org

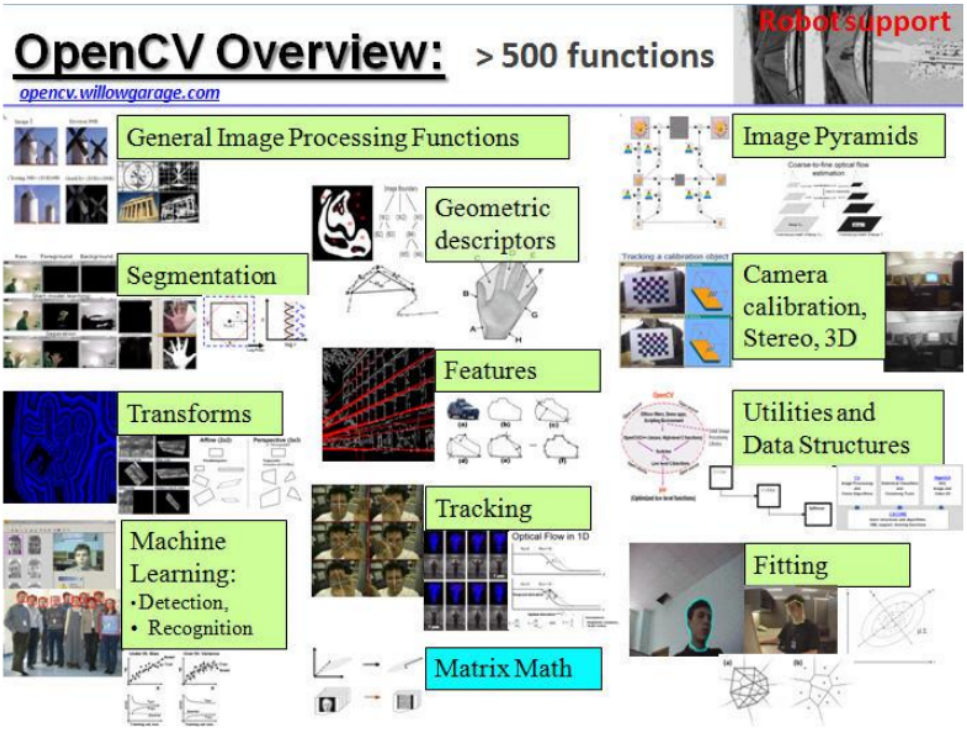


Figura 3.3: Funciones de OpenCV

Fue diseñado para tener una alta eficiencia computacional, está escrito en C y puede aprovechar las ventajas de los procesadores multinúcleo. Contiene más de 2500 funciones que abarcan muchas áreas de la visión artificial. También tiene una librería de aprendizaje automático (MLL, *Machine Learning Library*) destinada al reconocimiento y agrupación de patrones estadísticos.

Desde su aparición *OpenCV* ha sido usado en numerosas aplicaciones. Entre las cuales se encuentra la unión de imágenes de satélites y mapas web, la reducción de ruido en imágenes médicas, los sistemas de detección de movimiento, la calibración de cámaras, el manejo de vehículos no tripulados, el reconocimiento de gestos, etc. *OpenCV* es empleado también en reconocimiento de música y sonido, mediante la aplicación de técnicas de reconocimiento de visión en imágenes de espectrogramas del sonido.

Hay una gran cantidad de empresas y centros de investigación que emplean estas técnicas como IBM, Microsoft, Intel, SONY, Siemens, Google, Stanford, MIT, CMU, Cambridge e INRIA.

En este proyecto se hace uso de la versión *OpenCV* 4.2 de *Python* para el drone real y 3.3.1 de *OpenCVJS* para el drone simulado.

3.4. Google Colaboratory

Google Colaboratory o colab[5] (figura 3.4) es un servicio de *Google* que permite ejecutar código *Python* desde el navegador basándose en la tecnología *Jupyter notebook*[6]. En el servidor hay un intérprete de *Python* que recibe el código que se escribe en el navegador y devuelve el resultado para mostrarlo en su celda correspondiente.

La mayor ventaja de este servicio con respecto a ejecutar el propio código en local es que se tiene acceso tanto a *GPUs* como a *TPUs* proporcionadas por *Google*, es decir, se pueden hacer cálculos muy costosos computacionalmente desde ordenadores básicos.

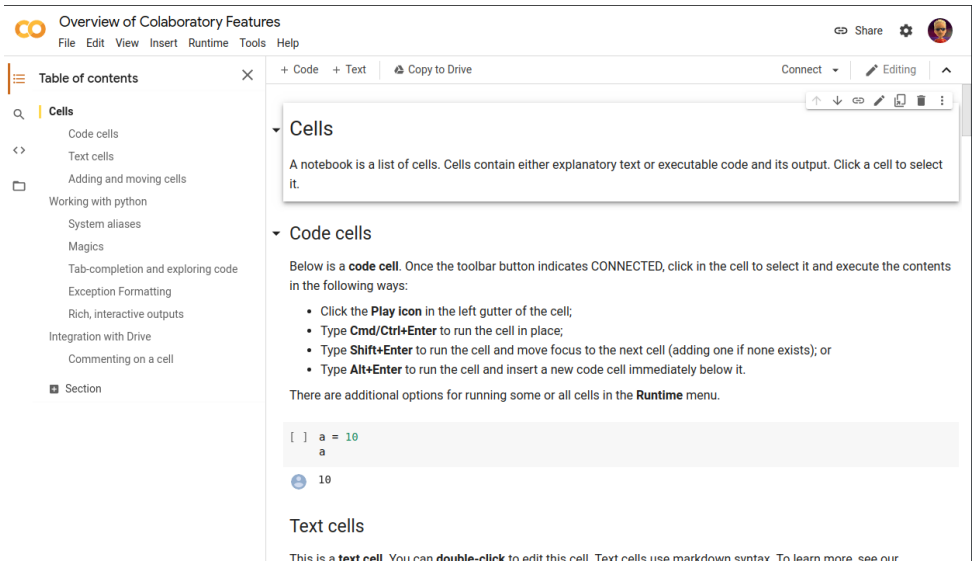


Figura 3.4: Google colab

3.5. Drone DJI Tello

Es un pequeño drone (figura 3.5) fabricado por la empresa DJI [7]. Tiene las ventajas de ser programable mediante un API además de tener un precio muy razonable, en torno a 100€, lo que le permite ser una muy buena opción de cara a la enseñanza.

[5] `https://colab.research.google.com`
[6] `https://jupyter.org`
[7] `https://www.dji.com/es`

Figura 3.5: Drone Tello

Este drone tiene una cámara frontal y crea una red *WiFi* para poder interactuar con él. Todo el procesado de imágenes no se realiza a bordo, sino, en el ordenador al cual está conectado. Tiene una segunda inferior que se usa para estabilización visual y que permite realizar movimientos precisos en distancia y rotación.

Además es muy pequeño (9.8 x 9.2 x 4,1 cm) y como se puede ver en la figura 3.5 tiene protecciones en las hélices de plástico frente a choques y si por casualidad reciben un choque saltan del drone para evitar problemas. Todas estas cosas lo hacen adecuado para niños.

Se ha usado para programar el seguimiento de una persona real.

3.6. Mixamo

Mixamo [26] es una empresa del grupo *Adobe Systems* que desarrolla y vende servicios basados en la web para la animación de personajes en 3D. Utilizan métodos de *Machine Learning* para automatizar los pasos del proceso de animación de personajes, incluyendo desde el modelado 3D hasta la animación 3D. Desde su web ofrecen modelos y animaciones gratuitos (3.6).

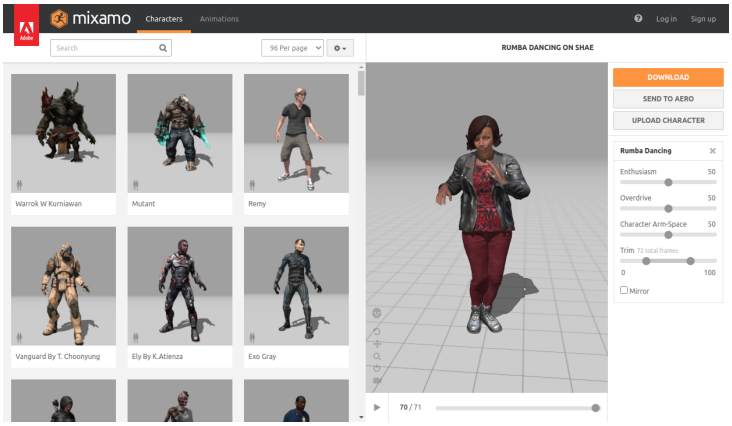


Figura 3.6: Ejemplos de Mixamo

En este proyecto se ha usado para obtener los modelos de las personas para la versión simulada.

3.7. Herramienta Blender

Blender[8] (figura 3.7) es un programa dedicado especialmente al modelado, iluminación, renderizado, animación y creación de gráficos tridimensionales.

Inicialmente fue distribuido de forma gratuita pero sin el código fuente. Posteriormente pasó a ser software libre. Actualmente es compatible con todas las versiones de Windows, macOS, GNU/Linux (incluyendo Android), Solaris, FreeBSD e IRIX.

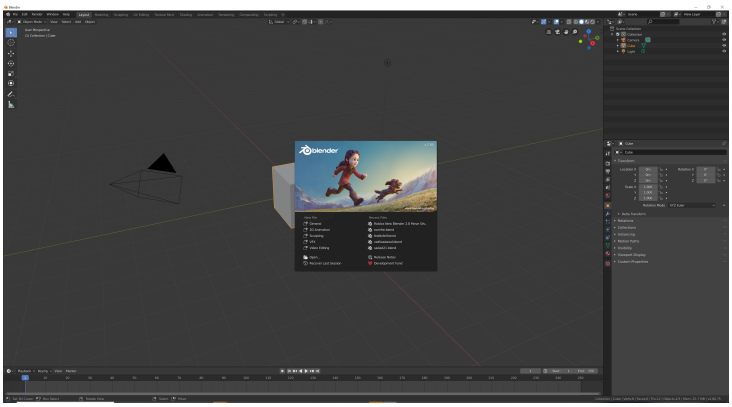


Figura 3.7: Blender

[8] `https://www.blender.org`

En este TFM se ha usado para modificar los modelos extraídos de *Mixamo* y hacerlos más compactos.

3.8. LabelMe

LabelMe [27] es un proyecto creado por el Laboratorio de Ciencias de la Computación e Inteligencia Artificial del MIT (CSAIL) que proporciona un conjunto de datos de imágenes digitales con anotaciones. Además también tiene una herramienta de etiquetado (Figura 3.8) de *datasets* que permite subir tu propio *dataset* a un servidor para poder etiquetar cada imagen desde el navegador.

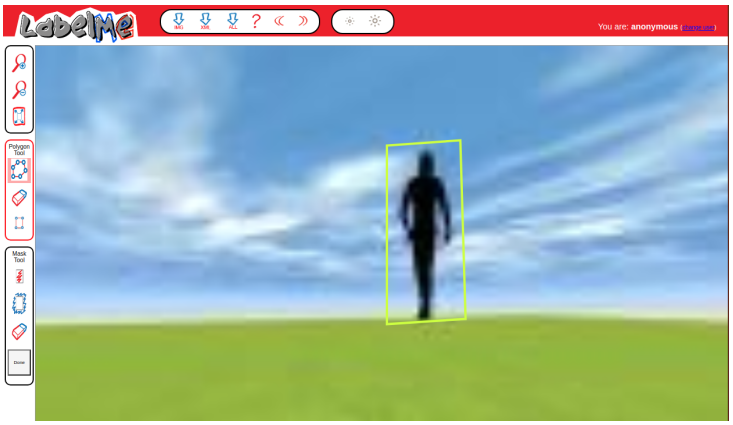


Figura 3.8: Etiquetado de LabelMe

En este proyecto se ha usado etiquetar el *dataset* para el reentrenamiento de la red para el simulador.

Capítulo 4

Comportamiento SiguePersona visual con Drone real

En este capítulo se explica el software que se ha desarrollado para conseguir un comportamiento sigue persona con un drone real con el objetivo de ser incluido en la plataforma *Kibotics* para enseñanza de robótica.

Para ello se ha tenido que perfeccionar el driver aportado por el fabricante [28]. Además, como va a ser usado por niños, se ha incorporado una nueva función al interfaz sencilla de usar, por ejemplo, en vez de tener que programar la detección de un objeto de un color indicado, se ha creado un método que recibe como parámetro el nombre del color y devuelve el cuadro que rodea dicho objeto.

4.1. Diseño

La aplicación desarrollada se compone de 2 partes claras, típicas en los comportamientos robóticos sencillos: una parte perceptiva y una parte de toma de decisiones de control, ambas se ejecutan continuamente, en un bucle infinito de iteraciones (Figura 4.1). La primera es la detección neuronal de objeto en la imagne usando *TensorFlowJS* y la segunda son tres controladores PID que gobiernan el movimiento del drone en tres ejes, el delante-detrás, el arriba-abajo y el giro izquierda-derecha..

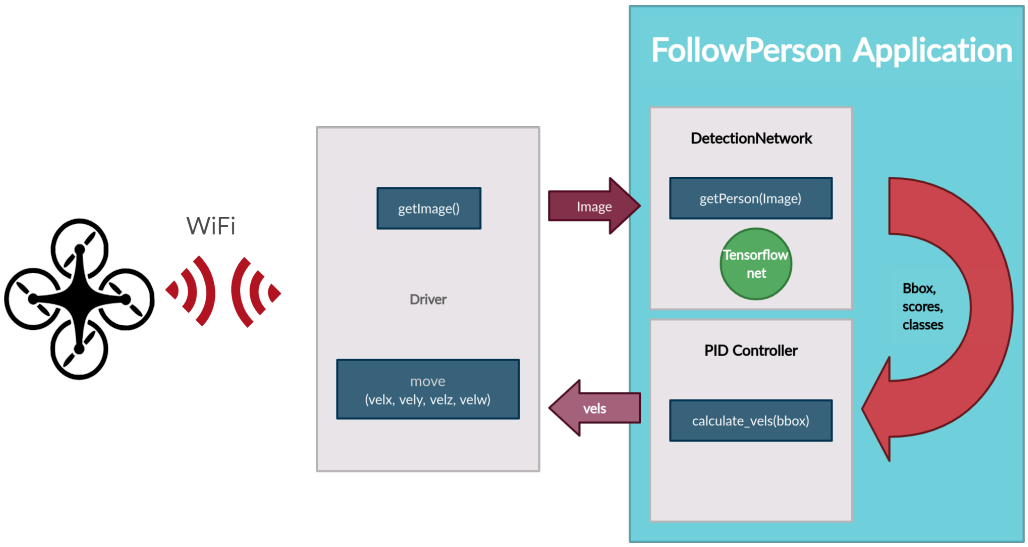


Figura 4.1: Diseño de la aplicación SiguePersona visual

La detección neuronal es un recubrimiento de la red neuronal que recibe las imágenes del driver y devuelve los *bounding boxes*, puntuaciones y clases detectadas. La parte de los controladores usando los *bounding boxes* calcula la velocidades y se las pasa al driver.

4.2. Desarrollo del *driver* real

El *driver* desarrollado para la plataforma *Kibotics* parte del proporcionado por el fabricante del drone. Se han añadido una serie de cambios por las limitaciones detectadas, además de necesitar simplificar el uso ya que está orientado a enseñanza infantil.

Este *driver* permite recibir imágenes de la cámara del drone, recibir informaciones básicas como batería restante o tiempo de vuelo. Además permite controlar el drone tanto en velocidad como en posición (avanza x metros, gira x grados,...), ...

El driver ejecuta en el ordenador, que a su vez está en continua comunicación con el drone *Tello* a través de la red *WiFi* que levanta de modo automático el propio drone. El driver no ejecuta a bordo del cuadricóptero.

4.2.1. Primera versión

Esta primera versión tiene el objetivo de subsanar los siguientes problemas:

- El *driver* no tiene control en velocidad, , sólo control en posición, que se consigue gracias a la cámara ventral y la estabilización visual que proporciona el fabricante, como mecanismo de seguridad.
- Si pasan 5 segundos sin recibir mensajes el drone se desactiva
- Las velocidades de entrada están en **cm/s** y **grados/s** de las funciones del driver del fabricante.

No tiene control en velocidad

El *driver* aportado por el fabricante en *Python* viene sin control en velocidad. Sí trae opciones para poder dar valor a las velocidades usadas por el drone y control en posición, pero no control en velocidad. Después de analizar el documento del SDK[1] se vio que sí existe un comando planeado para el control en velocidad, pero que no se ha implementado realmente en el *driver* aportado. Se implementa dicho método en el *driver* desarrollado. Además, se implementa un hilo que se encarga de enviar cada poco tiempo la velocidad (500ms) como recordatorio porque, si no, como mecanismo de seguridad el drone se detiene si no recibe más mensajes de velocidad.

Las velocidades de entrada están en cm/s y grados/s

La intención ha sido que se use el Sistema Internacional en todo momento y en este caso son **m/s** y **rad/s**. Esto es tan fácil como realizar las conversiones pertinentes en las funciones para establecer las velocidades.

Si pasan 5 segundos sin recibir mensajes el drone se desactiva

Este problema ha sido resuelto añadiendo un hilo de **Python** (perro de guarda, *watchdog*) que se encarga de enviar velocidades cada **500ms**. De esta manera evitamos que el *Drone* se desactive.

4.2.2. Segunda versión

Los problemas subsanados en esta versión han sido los siguientes:

[1] `https://terra-1-g.djicdn.com/2d4dce68897a46b19fc717f3576b7c6a/Tello%20%E7%BC%96%E7%A8%8B%E7%9B%B8%E5%85%B3/For%20Tello/Tello%20SDK%20Documentation%20EN_1.3_1122.pdf`

- *Driver* es necesario en Python 3.x pero en la primera versión está escrito en Python 2.7.
- Poca fiabilidad en el envío de mensajes del *driver* al drone.
- El control en velocidad es poco reactivo.

Convertir el driver de Python 2.7 a Python 3.x

Esto ha sido fácil porque casi todo el código de la aplicación funcionaba en Python 3, las únicas diferencias eran la manera de importar los módulos que varía de entre las dos versiones y que en Python 2 los bytes se representan como cadenas de texto(*String*) y en Python 3 son del tipo bytes.

Tipo de cambio	**Python 2**	**Python 3**
Representación de bytes	' '	b' '
Importar desde mismo repositorio	import module	import .module

Tabla 4.1: Cambios de Python 2 a Python 3 efectuados

En la tabla 5.1 se pueden ver los cambios efectuados.

Poca fiabilidad en el envío de mensajes del *driver* al drone

La comunicación con el Drone consiste en enviar un mensaje y recibir una respuesta que confirma que lo ha recibido, o en el caso de pedirle algún dato como la batería restante el dato en si.

El problema radica en que frecuentemente en la *WiFi* que comunica drone y ordenador o se pierde el mensaje que se envía o se pierde la respuesta. Por lo que se ha implementado un mecanismo de reenvío de mensajes. Si no se recibe respuesta se vuelve a enviar el mismo mensaje hasta tener respuesta o se alcance el número máximo de reintentos.

El control en velocidad es poco reactivo

Para mejorar este apartado solo ha hecho falta reducir el tiempo entre mensajes de velocidad a **25ms**. Además, no importa que se pierdan mensajes porque enseguida se envía otro con la velocidad actualizada.

4.3. Detección visual de la persona

Para elegir la red se han comparado tres del conjunto de redes preentrenadas con el *dataset* COCO de *Object Detection API*[2]:

- SSD Mobilenet v2 FPN de 320x320
- CenterNet resnet50 FPN de 512x512
- SSD Resnet50 FPN de 640x640

Para esta elección se ha grabado un vídeo con la cámara del drone real (720x960 píxeles), donde en todo momento hay una persona y se ha usado el mismo vídeo con las tres opciones, y en un ordenador con un procesador I7 de octava generación, 16 GB de RAM y con gráfica integrada. obteniendo los resultado de la tabla 4.2.

Red	**Tiempo**	**Detecciones**	**Score**
SSD Mobilenet v2	90ms	79 %	64 %
SSD Resnet50 v1	240ms	86 %	71 %
CenterNet Resnet50 v1	450ms	85 %	71 %

Tabla 4.2: Comparativa de redes

Se ha elegido *SSD Mobilenet v2 FPN* (Figura 4.2) porque es la única que funciona a una velocidad razonable para incorporar un comportamiento robótico reactivo.

[2] `https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md`

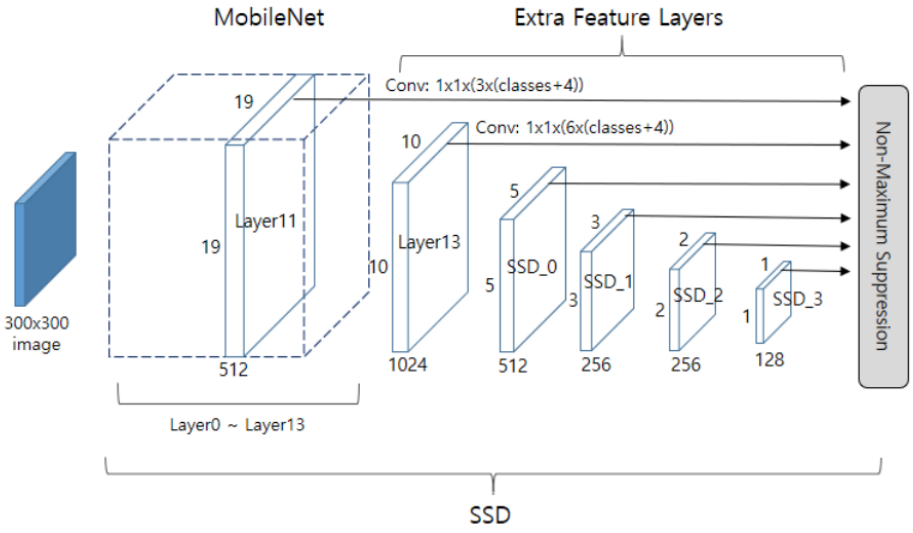


Figura 4.2: Esquema de una red Mobilenet SSD

Esta red tiene 2 componentes principales:

- *Feature Pyramid Networks (FPN) lite*[29]. Permite extraer características de la imagen con indiferencia del tamaño del objeto
- *Single-Shot Multibox Detector (SSD) Mobilenet v2*. Una red Single-Shot Multibox Detector (SSD) con una red base *Mobilenet v2*[30]

Para poder trabajar con ella se usa en *Tensorflow 2.0*. Además se ha tenido que desarrollar un recubrimiento para poder usar la red de manera fácil. Esta clase desarrollada permite cargar la red indicada por configuración ya sea un modelo de **Tensoflow** o un grafo. Además agrega un postprocesado de la información de salida de la red (figura 4.3):

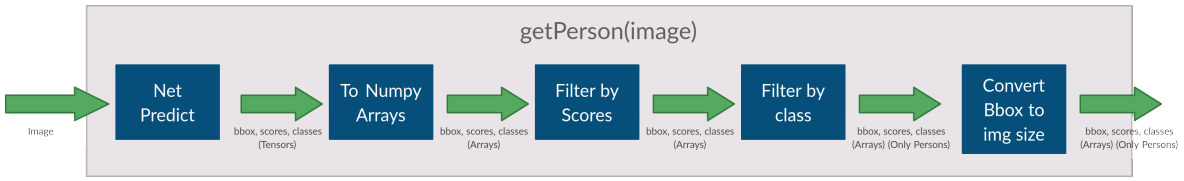


Figura 4.3: Función *getPerson*

- convirtiendo en *numpy arrays* los datos desde tensores
- Desechando los resultados con una puntuación menor al límite indicado por configuración.

- Filtrando los resultados para quedarse con las clases buscadas, en este caso personas.
- También se convierten los tamaños de las regiones de interés de porcentaje a píxeles.

Además, se ha creado también otro nivel de abstracción superior para la fuente de la imagen para ayudar a su uso. Así mediante configuración se indica si la fuente es el drone, una webcam, un directorio de imágenes o un vídeo y la fuente en sí y no hay que preocuparse de procesar la imagen para convertirla en RGB en caso de que sea la fuente no lo sea por ejemplo.

La red al final devuelve 3 *arrays*, el primero con la clase a la que pertenecen las detecciones, el segundo con las regiones de interés o *bounding boxes* (x mínima, y mínima, x máxima, y máxima) y el tercero con las puntuaciones, que indica la fiabilidad estimada de cada una de las detecciones.

Todo este procesamiento visual se ha incluido en el *driver*, mediante una función llamada *getPerson()*, para que pueda ser usado de manera sencilla desde las nuevas aplicaciones de robótica de los usuarios de *Kibotics*, quedando integrada en dicha plataforma.

4.4. Control PID

El control PID es un bucle de control que calcula la siguiente orden a comandar al drone real en cada iteración.

$$u(t) = K_p e(t) + K_i \int_t^0 e(t')dt' + K_d \frac{de(t)}{dt}$$

Donde **e** es el error con respecto al objetivo y las constantes K_p , K_i y K_d que deben calcularse experimentalmente. los objetivos son centrar en la imagen la persona horizontalmente, verticalmente y con un tamaño concreto, lo que conlleva que el drone está mirando directamente a la persona a una distancia razonable.

Una vez que se procesa la imagen recibida por el drone y se tienen las detecciones, se selecciona la persona con mayor puntuación y de su *bounding box* se extrae posición central, el área y la posición superior.

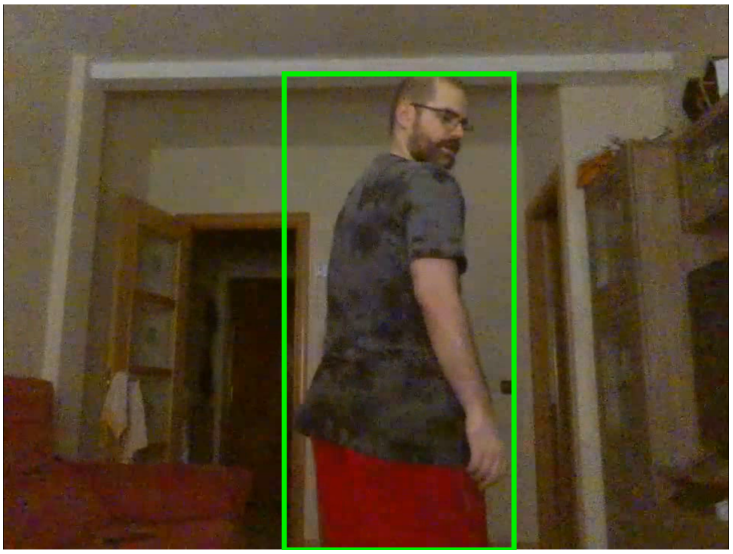


Figura 4.4: Detección de persona

Para controlar el drone se va a hacer con tres velocidades, avance, giro horizontal y movimiento vertical. Para ello se utiliza un control PID (en este caso debido al funcionamiento del drone solo proporcional y derivativo) para cada una de las velocidades.

Control de avance

En el caso de la velocidad de avance, se toma como referencia el área. Se fija un área objetivo para el *bounding box*, si la obtenida es menor, se avanza y si es mayor se retrocede. Las constantes **proporcional y derivativa** en este caso son **0.01 y 0** respectivamente.

Control de giro horizontal o guiñada

En este caso se toma como referencia la coordenada x del centro de la imagen. Si el *bounding box* se mueve a la izquierda hay que girar a la izquierda y si se va a derecha igual. Las constantes, ajustadas experimentalmente, **proporcional y derivativa** en este caso son **0.7 y 0.001** respectivamente.

Control de elevación

En el caso de la velocidad vertical se ha decidido usar como referencia la posición superior del *bounding box* posicionándolo en torno a un 10 % de la parte superior de la imagen con el objetivo que se vea la cara de la persona. Se probó también la manera fácil,

que es centrar verticalmente la persona en la imagen, pero puede ocurrir que solo se tenga una parte de la persona en la imagen y esté centrado, como se puede ver en las imágenes 4.5 y 4.6. En ambos casos el centro del rectángulo está muy próximo al centro vertical de la imagen, pero no por ello está bien.

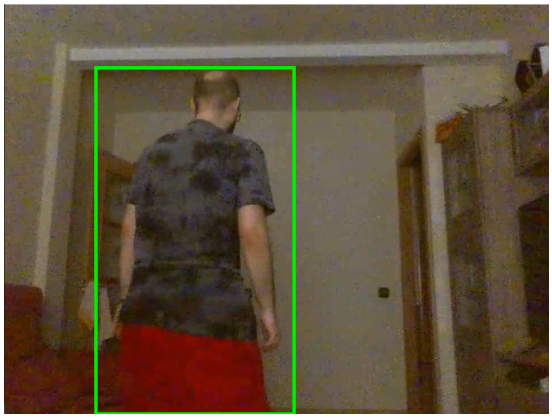


Figura 4.5: Persona seleccionada

Figura 4.6: Torso detectado

En cambio tomando como referencia el punto más alto del cuadrado se tiende a obtener la imagen 4.5, lo que además facilita la detección de la persona. Las constantes **proporcional y derivativa** en este caso son **3 y 0.5** respectivamente ajustadas de modo experimental. Hay que destacar además que en el caso de tener error por estar muy cerca del borde superior de la imagen se ha sumado 0.3 a dicho error para evitar que considere el borde superior de la imagen como valor aceptable.

4.5. Validación experimental

La validación experimental del desarrollo del software realizado se ha hecho en 4 casos, tres tests unitarios, uno por cada *PID* y un test global con los tres controladores integrados y funcionando en conjunto. En todos los casos el bucle de control funciona a 7 FPS.

4.5.1. Control de giro horizontal

Esta prueba consiste en moverse de izquierda a derecha delante del drone para ajustar las constantes del *PID* de giro horizontal (Figura 4.7).

Figura 4.7: Ejemplo de giro horizontal en drone real

4.5.2. Control de elevación

Esta prueba consiste en agacharse y levantarse delante del drone para ajustar las constantes del *PID* de elevación (Figura 4.8).

Figura 4.8: Ejemplo de control de elevación en drone real

4.5.3. Control de avance

Esta prueba consiste en avanzar y retroceder delante del drone para ajustar las constantes del *PID* de avance (Figura 4.9).

4.5.4. Ejecución típica completa

Esta prueba consiste en moverse por la habitación para comprobar si el comportamiento conjunto es el correcto (Figura 4.10). De no serlo, toca ajustar las constantes oportunas.

Figura 4.9: Ejemplo de avance en drone real

Figura 4.10: Ejemplo de ejecución típica en drone real

En el vídeo se ilustra que el comportamiento deseado del drone real siguiendo a una persona que se mueve de modo natural se ha conseguido satisfactoriamente una vez ajustados los tres controladores PID.

Capítulo 5

Comportamiento Sigue-Persona visual con drone simulado

En este capítulo se explica el software que se ha desarrollado para conseguir un comportamiento sigue persona con un drone simulado con el objetivo de ser incluido en la plataforma *Kibotics* para enseñanza de robótica.

5.1. Diseño

El comportamiento en el caso simulado es casi idéntico al real, cambiando cosas propias del lenguaje en cada caso. El caso de la versión simulada es igual a la real, se compone de 2 partes claras (figura 4.1), la primera es la detección neuronal y la segunda los controladores PID.

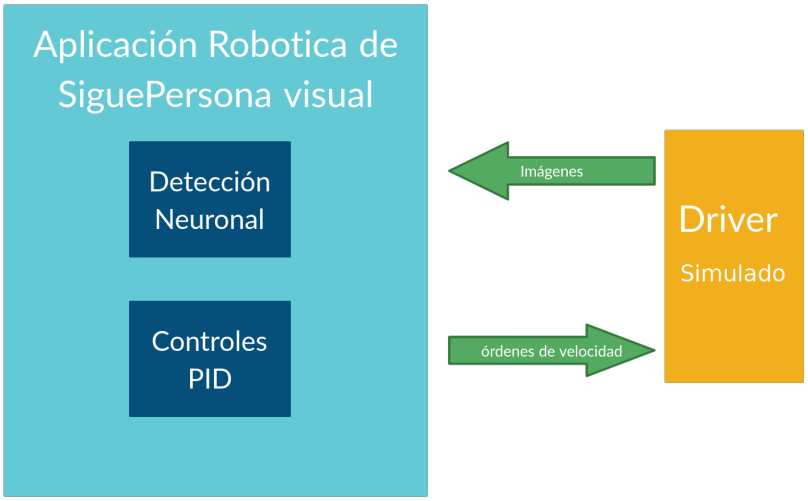


Figura 5.1: Diseño del comportamiento SiguePersona en su versión de simulador

La detección neuronal es un recubrimiento de la red neuronal que recibe las imágenes del driver y devuelve los *bounding boxes*, puntuaciones y clases detectadas. La parte de los controladores usando los *bounding boxes* calcula la velocidades y se las pasa al driver.

5.2. Creación del escenario

Lo primero es crear un escenario del simulador en el que haya un drone y una persona a la que seguir (figura 5.2).

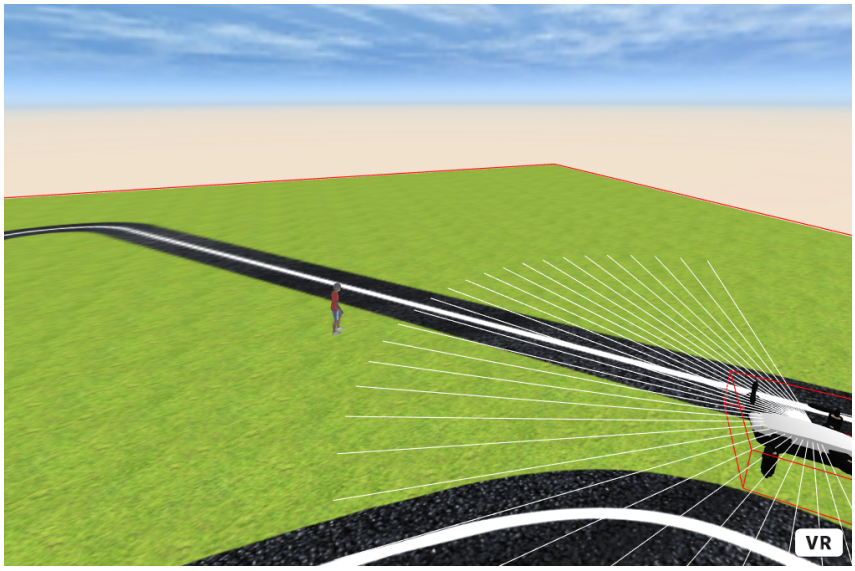


Figura 5.2: Mundo simulado

Para ello se ha necesitado descargar varios modelos 3d de personas de la web mixamo [26] (figura 5.3).

Figura 5.3: Modelo 3D de persona

Figura 5.4: Modelos 3D seleccionados

Los problemas que tenían estos modelos es que el simulador (*websim*) los cargaba muy oscuros y con textura metálica, además de que ocupan mas de 100 MB. Para solucionar estos problemas se han cargado los modelos en *Blender* para poder aclarar y quitar el

metalizado de las texturas. Además se ha reducido el tamaño es estas para reducir el tamaño final de cada modelo a 3 MB.

Otro problema era que el modelo viene en un directorio con la forma de la persona en un fichero y las texturas separadas en varias imágenes. Así que se ha decidido guardarlo en un solo fichero *binary graphic library transmission format (GLB)*. el resultado se puede ver en la figura 5.4.

5.3. Detección visual de la persona

Debido a las limitaciones en la capacidad de procesamiento propias del intérprete de *JavaScript* en este caso no se han probado más redes y se ha optado por la que mejores resultados ha dado en tiempos de ejecución de las pruebas reales, una *SSD Mobilenet v2* (figura 4.2).

Se trabaja con *TensorflowJS 2.0.1*. Además, se ha hecho un recubrimiento de la red para poderla usar más fácilmente permitiendo cargar la red siendo un grafo o un modelo de *TensorflowJS*. También añade un postprocesado a la salida de la red.

Las red usada en *Javascript* es básicamente las misma que en *Python* pero con tres cambios importantes para mejorar rendimiento (figura 5.5):

- Se elimina el postprocesado del modelo original.
- se suprime el *NonMaxSuppression* multiclase original y se sustituye por uno de una sola clase
- las operaciones de *NonMaxSuppression* se ejecutan en CPU para no perder tiempo en la carga de las texturas

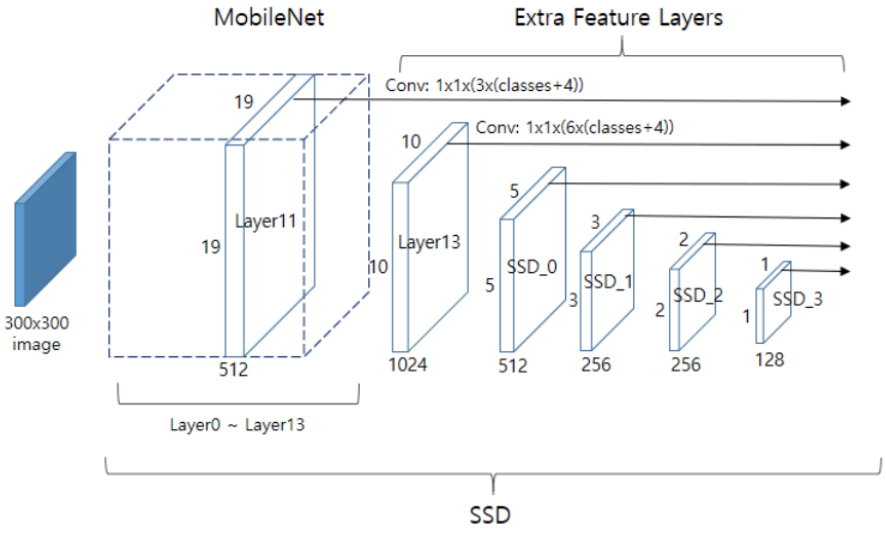


Figura 5.5: Esquema de una red Mobilenet ssd usada en *TensorflowJS*

EL postprocesado en el caso de la versión simulada es la siguiente:

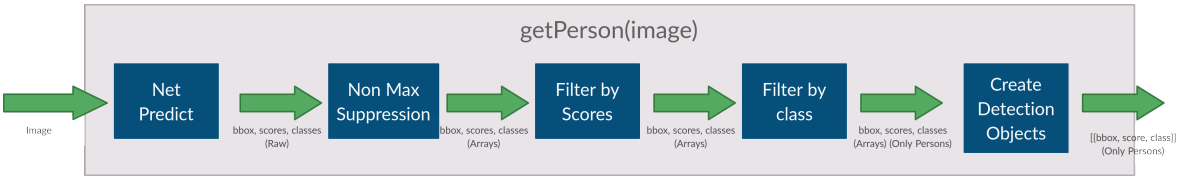


Figura 5.6: Función *getPerson en JS*

- Se pasa la salida de la red por una función *NonMaxSuppression* monoclase para obtener ya las detecciones, puntuaciones y clases.
- Se filtran detecciones para eliminar las que tengan menos que el limite indicado por configuración.
- Se filtra por clase para quedarnos solo con las personas.
- Se crea un *array* de objetos que representan las detecciones (*bounding box*, puntuación y clase).

Al final del postprocesado de la salida de la red se obtiene un *array* con las detecciones representadas por un objeto que contiene ***bounding box*, puntuación y clase**.

Todo este procesamiento visual se ha incluido en el *driver del drone simulado* para que pueda ser usado de manera sencilla desde las nuevas aplicaciones de robótica de los usuarios de *Kibotics* con el simulador.

Creación del *dataset*

En las primeras pruebas con el drone simulado usando la red preentranada disponible en la web se observó que la detección de la persona es muy deficiente. No es capaz de detectarla en más del 50 % de los fotogramas, lo que imposibilita un seguimiento aceptable. Por lo que se decide reentrenar la red de detección con un *dataset* nuevo, creado con capturas de imagen de cámara del drone [31] en el simulador de *Kibotics* y por ello muy adaptado a las necesidades concretas de este TFM(figuras 5.7 y 5.8).

Figura 5.7: Ejemplo 1 del *dataset*

Figura 5.8: Ejemplo 2 del *dataset*

Para generar las imágenes se coloca el modelo de la persona y se va cambiando mediante código la posición del drone alrededor del modelo a 3 radios diferentes (2,5 y 10 metros) y añadiendo un nivel de aleatoriedad en el momento final del posicionado para que las imágenes no sean regulares (figura 5.9). De esta manera se han obtenido 4000 imágenes aproximadamente. Siendo representativo de la mayoría de posiciones relativas desde las que un drone puede tener una persona en sus cercanías.

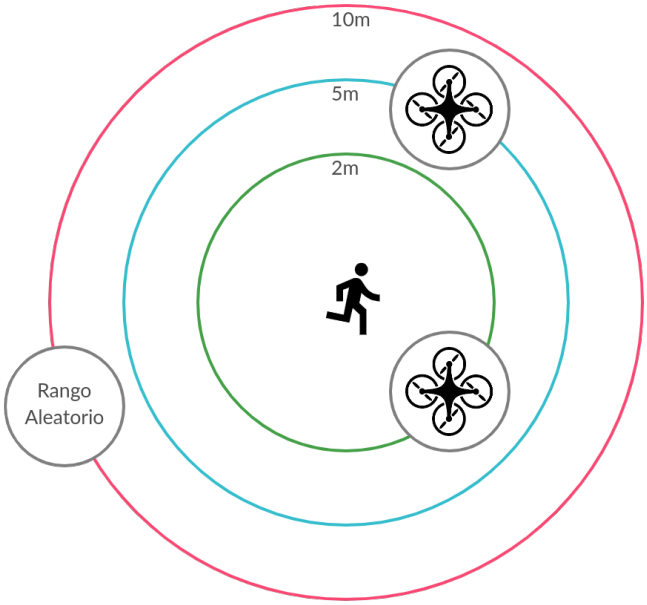


Figura 5.9: Posiciones relativas desde las cuales se capturan imágenes simuladas para entrenar la red de detección de personas

Una vez generadas todas las capturas se han etiquetado manualmente, para ello se han subido a la web *LabelMe*, que permite etiquetar las imágenes (Figura 5.10).

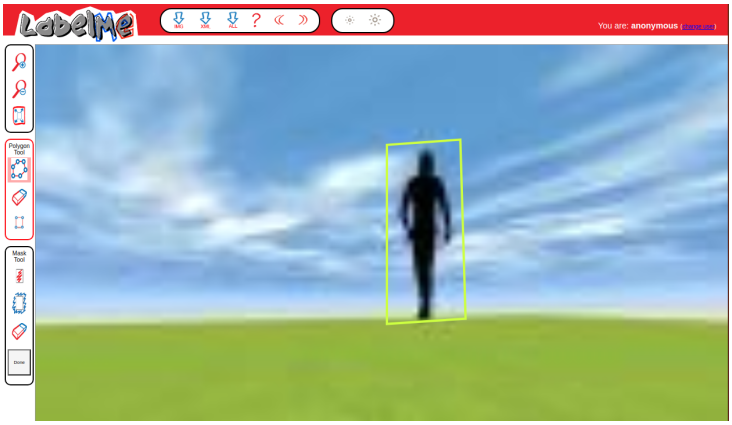


Figura 5.10: Etiquetado del *dataset* en *LabelMe*

Cuando están todas etiquetadas, se descargan, Son un conjunto de imágenes y ficheros *XML*.

Para aumentar el tamaño del dataset se ha seguido una técnica de ”data augmentation”, con idea que el entrenamiento de la red neuronal sea más rico. Se ha procedido a

espejar las imágenes ya etiquetadas mediante un *script* de *python*, doblando el tamaño de *dataset*. Después de esto se ha dividido en *train* y *test*, dejando un 10 % para la segunda parte mediante otro *script*.

Los dos últimos pasos han sido convertir esos ficheros *XML* en uno *CSV* para *train* y otro para *test* para poder manejar mejor el *dataset* y crear ficheros *record* (ficheros binarios que contienen tanto imágenes como anotaciones) para poderlos usar mejor en los entrenamientos de *Tensorflow*.

Reentrenar red

El reentrenamiento se ha efectuado en **Google Colab** mediante un cuadernillo *Python*. La primera intención fue hacerse con *Tensorflow 2.0* pero *Object Detection API* todavía no tiene soporte para guardar las redes entrenadas en formato grafo, por lo que no ha podido usarse en el navegador al tener unos tiempos de detección superiores al segundo.

Para subsanar este contratiempo se ha desarrollado uno equivalente para *Tensorflow 1.15*, donde sí hay soporte para grafos. Dicho entrenamiento ha consistido en 20.000 iteraciones con un *batch* de 24 imágenes. se han obtenido los siguientes resultados:

Resultados	
Precision	0.64
Recall	0.69

Tabla 5.1: Resultados del entrenamiento

Además se han obtenido una pérdida del 0.3. Con estos datos junto con una pequeña bajada del limite de confianza se ha conseguido que la detección en el simulador ronde el 80 %. De esta manera ya es posible seguir la persona sin problemas en el simulador que ejecuta en el navegador.

5.4. controles PID

Una vez que se procesa la imagen recibida de el drone y se tienen las detecciones de personas, se selecciona la persona con mayor puntuación y de su *bounding box* se extrae posición central y el área. En este caso no se hace control vertical porque las imágenes recibidas no permiten un control aceptable en este eje.

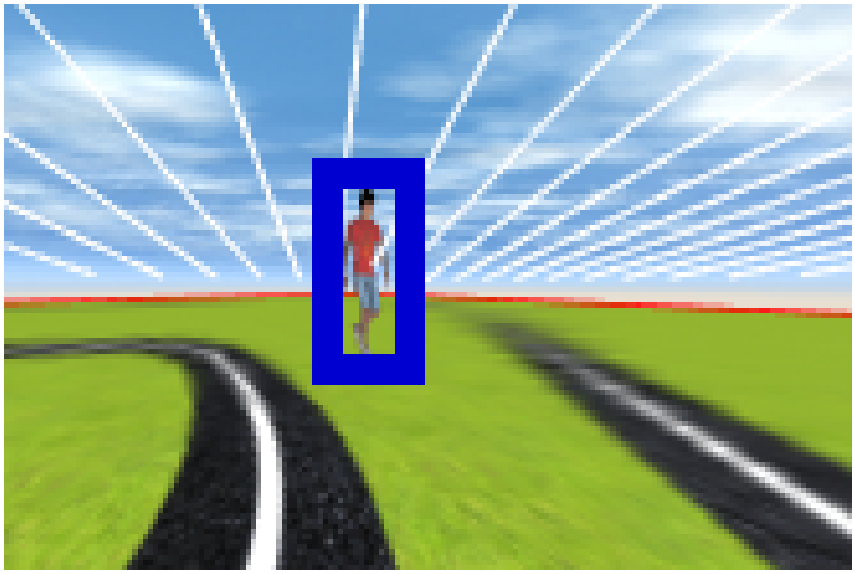


Figura 5.11: función *Detección ideal*

Para controlar el drone se va a hacer con dos velocidades, avance horizontal y giro horizontal. Para ello se utiliza un control PID (en este caso debido al funcionamiento del drone solo proporcional y derivativo) para cada una de las velocidades.

Control de avance horizontal

En el caso de la velocidad de avance, se toma como referencia el área. Se fija un área objetivo para el *bounding box*, si la obtenida es menor se avanza y si es mayor se retrocede. Las constantes **proporcional y derivativa** en este caso son **0.01 y 0** respectivamente. Los valores de las constantes son diferentes porque dependen del comportamiento del robot en cuestión, incluso en robots del mismo modelo pueden variar estos valores ligeramente.

Control de giro horizontal o guiñada

En este caso se toma como referencia la coordenada x del centro de la imagen. Si el *bounding box* se mueve a la izquierda hay que girar a la izquierda y si se va a derecha igual. Las constantes **proporcional y derivativa** en este caso son **0.002 y 0.0001** respectivamente.

5.5. Validación experimental

La validación experimental del desarrollo se ha hecho en 3 casos: un test unitario por cada *PID* un test global del conjunto. En todos los casos el bucle de control funciona a 7 FPS.

5.5.1. Control de giro horizontal

Esta prueba consiste en colocar uno de los modelos 3D moviéndose de izquierda a derecha delante del drone, mediante una animación de A-frame[1]. Permite ajustar las constantes del *PID* de giro horizontal[2] (figura 5.12).

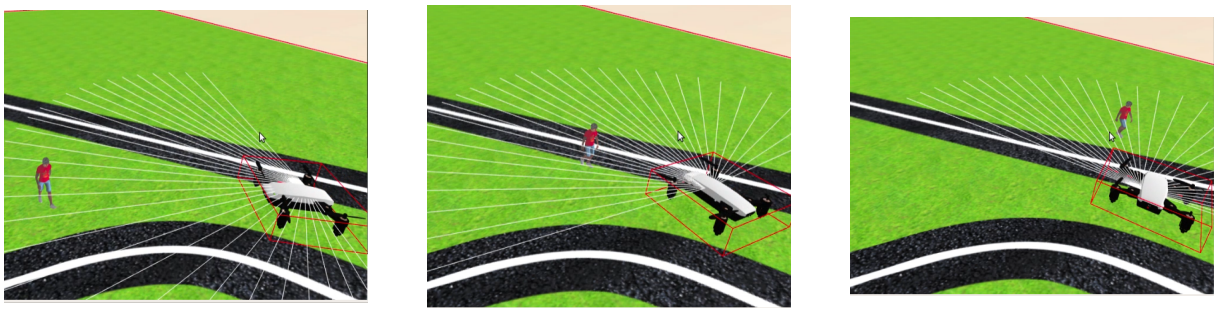


Figura 5.12: Ejemplo de giro horizontal

5.5.2. Control de avance

Esta prueba consiste en colocar uno de los modelos 3D de persona alejándose y acercándose al drone para ajustar las constantes del *PID* de avance[3] (figura 5.13).

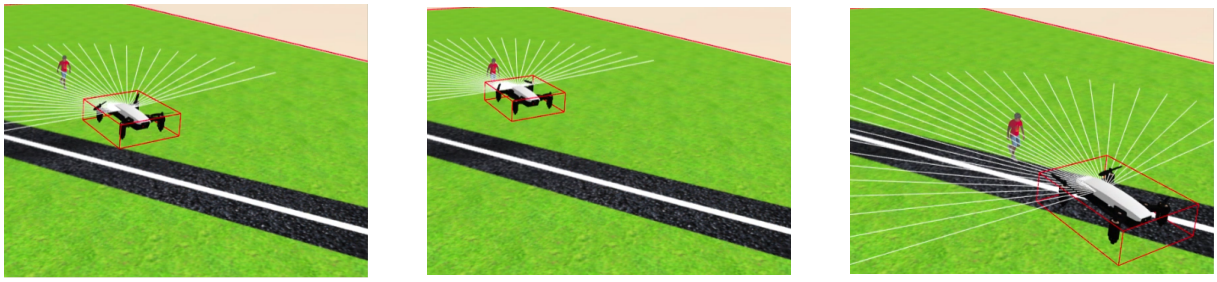


Figura 5.13: Ejemplo de avance

[1] `https://aframe.io`
[2] `https://www.youtube.com/watch?v=xLa3wSYIIuY`
[3] `https://www.youtube.com/watch?v=CNo6rionlcE`

5.5.3. Ejecución típica completa

En este caso en vez de usar una animación, se le ha conectado un teleoperador existente en *Websim*, que permite controlar un objeto en el mundo simulado con las flechas del teclado, para poder mover libremente el modelo 3D de la persona para que lo siga el *drone* (figura 5.14). Si el movimiento no es el esperado, toca ajustar las constantes oportunas.

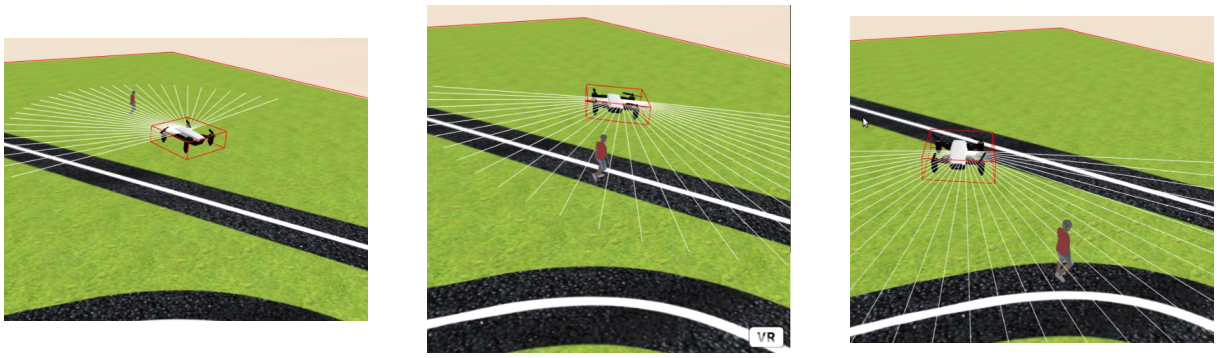


Figura 5.14: Ejecución Típica

Capítulo 6

Conclusiones

Una vez documentado el software diseñado y desarrollado a lo largo de este TFM y la funcionalidad que ofrece, se dedica este capítulo a la comprobación de los objetivos alcanzados, así como a la explicación de los conocimientos adquiridos y una breve exposición de posibles mejoras y de las líneas de actuación futuras.

6.1. Conclusiones

El objetivo principal, que era la creación de la infraestructura necesaria para añadir procesamiento visual complejo, de manera sencilla de usar, se ha cumplido, ahora ya se pueden desarrollar prácticas tanto con drone simulado como real de detección robusta de personas.

En cuanto a los subobjetivos concretos, se ha conseguido un comportamiento robótico *sigue*Persona visual real gracias al uso de *Tensorflow* y *Opencv* para detectar a la persona y el uso de varios controladores PID para las velocidades. Para lograr este subobjetivo se han creado clases de *Python* para contener las las redes y permitir un nivel de abstracción mayor además de simplificar su uso mediante métodos personalizados como *getPerson() en Python*. Igualmente se ha mejorado el driver de Python del drone real DJI Tello, que permite control en velocidad y un envío más rápido y fiable de comandos desde el ordenador al propio drone volador.

También se ha cumplido el subobjetivo del comportamiento *sigue*Persona visual simulado, teniendo que crear un mundo que se adaptara a la necesidades y creando clases de

Javascript para contener las las redes y permitir un nivel de abstracción mayor además de para simplificar su uso mediante el método *getPerson* en *JavaScript*. Además, para este subobjetivo se ha necesitado crear un *dataset* específico de 4000 imágenes y reentrenar una red de detección visual puesto que la red preentrenada disponible no funcionaba satisfactoriamente. Se ha utilizado TensorlfowJS de modo que la inferencia neuronal ejecuta en el navegador web en tiempo real.

De manera personal, también se ha alcanzado el objetivo de mejorar en conocimientos en el campo del desarrollo web y el modelado 3D. Gracias a ello se han adquirido conocimientos en tecnologías web de procesado de imagen como *OpenCVJS* y *TensorflowJS* que en un principio creía que era inviable por la necesidad de recursos que tiene y las limitaciones propias del intérprete de *JavasScript*.

6.2. Trabajos futuros

Las contribuciones de este TFM han abierto las posibilidades para desarrollar nuevas prácticas utilizando visión de manera sencilla de la plataforma *Kibotics* manteniendo la reobusted habitual de las redes neuronales. Pero esto es solo el principio, algunas mejoras pueden ser:

- **Creación de prácticas que utilicen esta tecnología**. Se ha asentado la base, pero ahora toca desarrollar nuevas prácticas de seguimiento de personas o de otros objetos como señales de tráfico, semáforos, etc.
- **Aumentar el número de opciones en cuanto a detección**, como por ejemplo un sigue persona que detecte caras para poder seguir a una persona concreta en un grupo
- Se puede explorar la identificación de personas particulares.

Bibliografía

[1] Rigoberto Vizcay. *Deep Learning para la Detección de Peatones y Vehículos sobre FPGA*. PhD thesis, Centro Universitario UAEM Valle de México, 2018. [Accedido 22 de Junio de 2019].

[2] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. *Lecture Notes in Computer Science*, page 21–37, 2016.

[3] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger, 2016.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[5] Anyu. "RNA – Redes Neuronales Artificiales", *Bitácoras de un Ingeniero*. `http://andrealezcano.blogspot.com.es/2011/04/rna-redes-neuronales-artificiales.html`, 2011. [Accedido 29 de Mayo de 2019].

[6] MIT Computer Science & Artificial Intelligence Lab. Using AI to predict breast cancer and personalize care. `https://www.csail.mit.edu/news/using-ai-predict-breast-cancer-and-personalize-care`, 2019. [Accedido 30 de Mayo de 2019].

[7] Scratch. `https://scratch.mit.edu/`.

[8] Lego. `https://www.lego.com/es-es/categories/coding-for-kids`.

[9] Kodu. `https://www.kodugamelab.com/`.

[10] Snap. `https://snap.berkeley.edu/`.

[11] Y. Abdullah, G. Mehmet, A. Iman and B. Erkan. A Vehicle Detection Approach using Deep Learning Methodologies. *Computer Engineering Department Ankara University*, 2018.

[12] Ignacio Arriola. *Detección de objetos basada en Deep Learning y aplicada a vehículos autónomos*. PhD thesis, Universidad del País Vasco. Ingeniería computacional y sistemas inteligentes, 2018.

[13] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *In Advances in neural information processing systems*, pages 91–99, 2015.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[15] Ignacio Condés Menchén. *Deep Learning Applications for Robotics using TensorFlow and JdeRobot*. PhD thesis, Escuela Técnica Superior de Ingeniería de Telecomunicación, Universidad Rey Juan Carlos, 2018. [Accedido 24 de Junio de 2019].

[16] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[17] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[18] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2015.

[19] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement, 2018.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[21] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[22] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, Tom Duerig, and Vittorio Ferrari. The open images dataset v4: Unified image

classification, object detection, and visual relationship detection at scale. *IJCV*, 2020.

[23] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[24] Asociación de robótica e inteligencia artificial JdeRobot. Kibotics. `https://kibotics.org`.

[25] Tensorflow. `https://github.com/tensorflow/tensorflow`.

[26] Adobe Systems Incorporated. Mixamo. `www.mixamo.com`.

[27] Labelme. `http://labelme.csail.mit.edu/Release3.0/`.

[28] DJI. Dji tello driver python. `https://github.com/dji-sdk/Tello-Python`.

[29] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2017.

[30] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019.

[31] *Dataset* de imágenes de *websim*. `https://github.com/aitormf/websim_person_dataset`.

[32] Marcos Pieras Sagardoy. *Visual people tracking with deep learning detection and feature tracking*. PhD thesis, Escuela Técnica Superior de Ingeniería de Telecomunicación, Universidad Rey Juan Carlos, 2017. [Accedido 24 de Junio de 2019].