



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

¡Muchas gracias a todos!

Resumen

Resumen

Índice general

Índice de figuras	VI
Índice de tablas	VII
1. Introducción	2
2. Objetivos	3
3. Estado del arte	4
3.1. Redes neuronales para detección	4
3.1.1. Redes neuronales convolucionales	4
3.2. Bases de datos de detección	6
3.2.1. Common Objects in Context (COCO)	7
3.2.2. Open Images	7
3.2.3. PASCAL Visual Object Classes Challenge (VOC) 2012	8
4. Herramientas utilizadas	9
4.1. Kibotics	9
4.2. Tensorflow	10
4.2.1. Object Detection API	11
4.3. Opencv	11
4.4. Google Colaboratory	13
4.5. DJI Tello	13
4.6. Mixamo	14
4.7. Blender	15
4.8. LabelMe	15
5. Comportamiento SiguePersona visual con Drone real	17
5.1. Diseño	17
5.2. Desarrollo del <i>driver</i>	18
5.2.1. Primera versión	18

5.2.2. Segunda versión	19
5.3. Detección visual de la persona	20
5.4. control PID	23
5.5. Validación experimental	25
6. Drone simulado	28
6.1. Diseño	28
6.2. Creación del escenario	29
6.3. Comportamiento siguePersona visual	30
6.4. Creación del <i>dataset</i>	33
6.5. Reentrenar red	35
6.6. Validación experimental	35
7. Conclusiones	38
Bibliografía	40

Índice de figuras

3.1.	CNN Rigoberto Vizcay [1]	4
3.2.	Fases de R-CNN	5
3.3.	Fases de Faster R-CNN	6
3.4.	ejemplo de <i>COCO</i>	7
3.5.	ejemplo de <i>Open Images</i>	8
4.1.	Ejercicios Kibotics	9
4.2.	Ejercicio con Scratch	10
4.3.	Funciones de OpenCV	12
4.4.	Google colab	13
4.5.	Drone Tello	14
4.6.	Ejemplos de Mixamo	14
4.7.	Blender	15
4.8.	Etiquetado de LabelMe	16
5.1.	Diseño Aplicación	18
5.2.	Esquema de una red Mobilenet ssd	21
5.3.	función <i>getPerson</i>	22
5.4.	Detección de persona	23
5.5.	Persona seleccionada	24
5.6.	Torso detectado	24
5.7.	Persona seleccionada	25
5.8.	Torso detectado	25
5.9.	Persona seleccionada	26
5.10.	Torso detectado	26
5.11.	Persona seleccionada	26
5.12.	Torso detectado	26
5.13.	Persona seleccionada	27
5.14.	Torso detectado	27
6.1.	Diseño versión de simulador	28

6.2.	Mundo simulado	29
6.3.	Modelo 3D de persona	30
6.4.	Modelos 3D seleccionados	30
6.5.	Esquema de una red Mobilenet ssd usada en <i>TensorflowJS</i>	31
6.6.	función <i>getPerson</i> en <i>JS</i>	31
6.7.	función <i>Detección ideal</i>	32
6.8.	Ejemplo 1 del <i>dataset</i>	33
6.9.	Ejemplo 2 del <i>dataset</i>	33
6.10.	función <i>Esquema de capturas</i>	34
6.11.	Etiquetado del <i>dataset</i> en <i>LabelMe</i>	34
6.12.	Modelo a la izquierda	36
6.13.	Modelo a la derecha	36
6.14.	Modelo cerca	36
6.15.	Modelo lejos	36
6.16.	Ejecución Típica	37

Índice de tablas

5.1. Cambios de Python 2 a Python 3 efectuados	20
5.2. Comparativa de redes	21
6.1. Resultados del entrenamiento	35

Acrónimos

API Application Programming Interface.

CNN Convolutional Neural Network.

COCO Common Objects in Context.

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 Interseccion Over Union.

NMS Non-Maximum Suppression.

OpenCV Open Source Computer Vision Library.

R-CNN Region-based Convolutional Neural Network.

SDK software development kit.

SSD Single Shot Detector.

SVM Support Vector Machine.

TFM Trabajo de Fin de Máster.

VA Visión Artificial.

VOC PASCAL Visual Object Classes Challenge.

Capítulo 1

Introducción

introduccion

Capítulo 2

Objetivos

Objetivos

Capítulo 3

Estado del arte

3.1. Redes neuronales para detección

3.1.1. Redes neuronales convolucionales

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 3.1 se puede ver la estructura de la red.

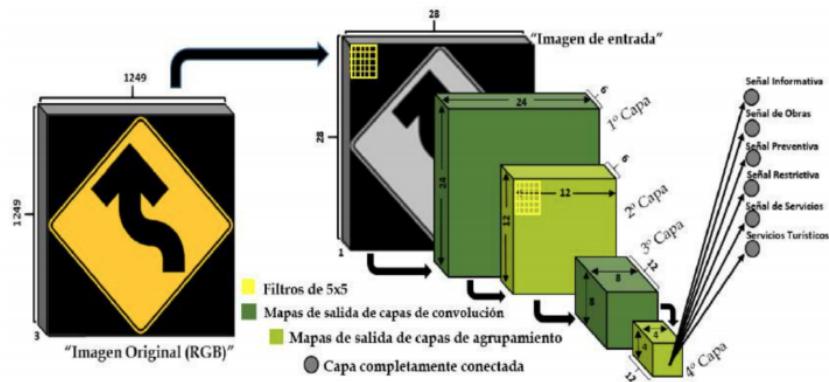


Figura 3.1: CNN Rigoberto Vizcay [1]

Y. Abdullah, G. Mehmet, A. Iman and B. Erkan [2] 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 3.2:

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.

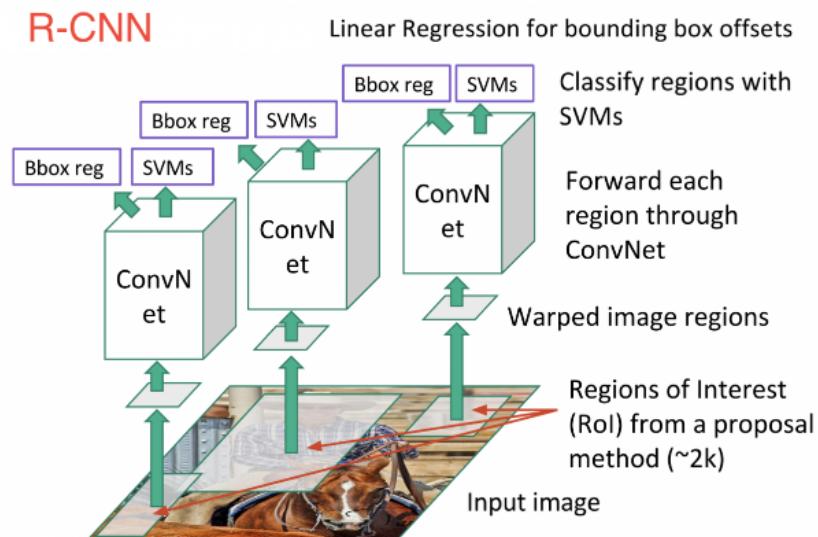


Figura 3.2: Fases de R-CNN

Por otro lado, Faster R-CNN (Figura 3.3) 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 [3] 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 [4] y una red convolucional Resnet 101 [5] 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 KITTI²). Tras realizar las pruebas concluyeron que el caso que peores resultados obtenía era el que tenía valores iniciales aleatorios. También se ha analizado un caso práctico de detección de baches en carretera con los mismos modelos empleados en la detección de peatones.

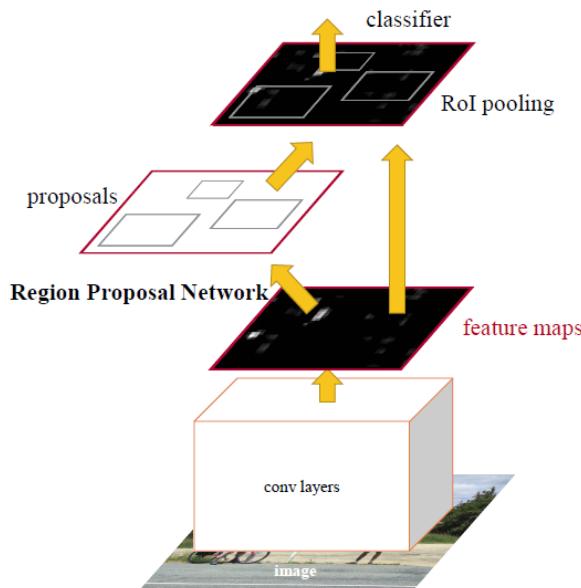


Figura 3.3: Fases de Faster R-CNN

3.2. Bases de datos de detección

La detección 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.

¹<http://cocodataset.org/#home>

²<http://www.cvlibs.net/datasets/kitti/>

3.2.1. Common Objects in Context (COCO)

El *dataset* de Common Objects in Context (COCO)³[6] 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^{3.4} con en torno a 80 clases de objetos diferentes. Además no solo se usa para detección, sino, para segmentación, clasificación y otros usos.

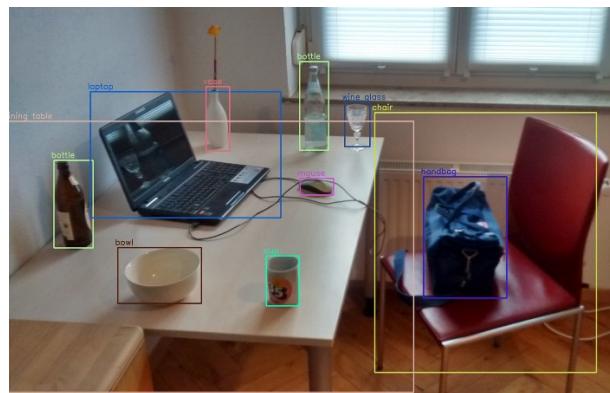


Figura 3.4: ejemplo de *COCO*

3.2.2. Open Images

*Open Images*⁴ [7] es un conjunto de datos de aproximadamente 9 millones de imágenes anotadas(figura 3.5) con etiquetas de nivel de imagen, cajas delimitadoras de objetos, máscaras de segmentación de objetos, relaciones visuales y narraciones localizadas. Las 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 objetos.

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").

³<https://cocodataset.org>

⁴<https://storage.googleapis.com/openimages/web/index.html>

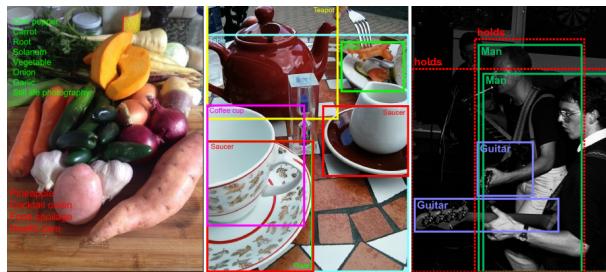


Figura 3.5: ejemplo de *Open Images*

3.2.3. PASCAL Visual Object Classes Challenge (VOC) 2012

Este conjunto de datos contiene los datos del *PASCAL Visual Object Classes Challenge (VOC)* 2012⁵ [8], 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.

⁵<http://host.robots.ox.ac.uk/pascal/VOC/voc2012/>

Capítulo 4

Herramientas utilizadas

EL objetivo de este trabajo es preparar la infraestructura para desarrollar prácticas de visión con drones en la plataforma Kibotics[9] tanto con drone real como simulado. Para ello se han necesitado las siguientes herramientas.

4.1. Kibotics

Kibotics [9] es una plataforma de enseñanza de robótica infantil desarrollada por las Asociación de robótica JdeRobot¹

La batería de ejercicios(figura 4.1) que incluye el entorno *Kibotics* [9] se simula usando el simulador robótico *Websim*. Se trata de un simulador diseñado para el aprendizaje de conceptos básicos de programación de robots especialmente para niños.

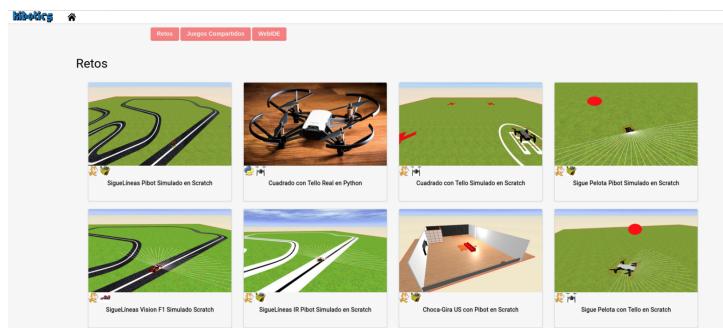


Figura 4.1: Ejercicios Kibotics

El simulador permite que los usuarios puedan programar fácilmente los movimientos

¹<https://jderobot.github.io>

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 ordenes se deben programar, en *Python* o *Scratch* (figura 4.2), dentro del editor que incorpora la interfaz de *Websim*.

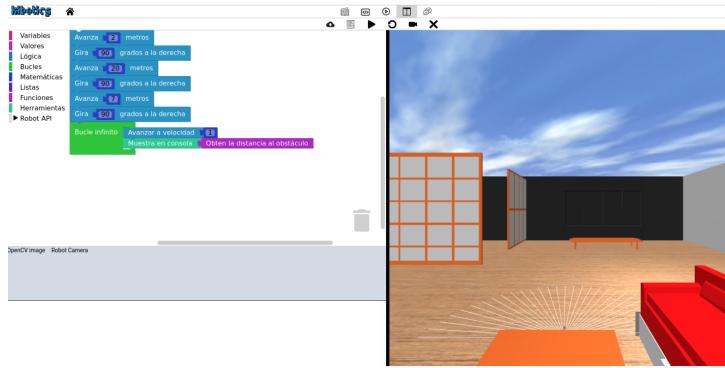


Figura 4.2: Ejercicio con Scratch

Se ha usado para la parte simulada.

4.2. Tensorflow

Es una plataforma 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* [10]. 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 a parte 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* y la versión 2.0.1 para *Javascript*.

4.2.1. 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.

4.3. OpenCV

*OpenCV*³ es una librería de código abierto desarrollada por *Intel* y publicada bajo licencia de BSD. Esta librería implementa gran variedad de herramientas para la interpretación de la imagen. Sus siglas provienen de los términos anglosajones “*Open Source Computer Vision Library*”, y tal y como se puede deducir 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*. Esto permite tener muchas de las funcionalidades desarrolladas en *C++* en el navegador.

OpenCV implementa algoritmos para técnicas de calibración, detección de rasgos,

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

³<https://opencv.org/>

⁴<https://emscripten.org>

CAPÍTULO 4. HERRAMIENTAS

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.

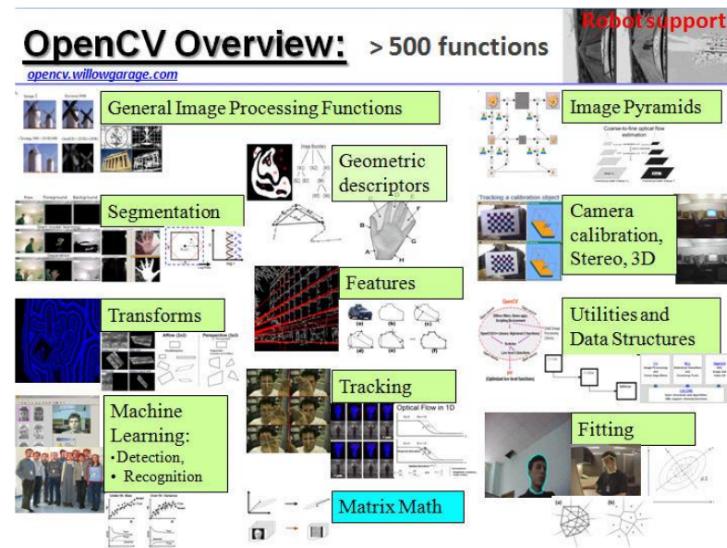


Figura 4.3: Funciones de OpenCV

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* y 3.3.1 de *Javascript*.

4.4. Google Colaboratory

*Google Colaboratory o colab*⁵ (figura 4.4) 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 interprete 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 en ordenadores básicos.

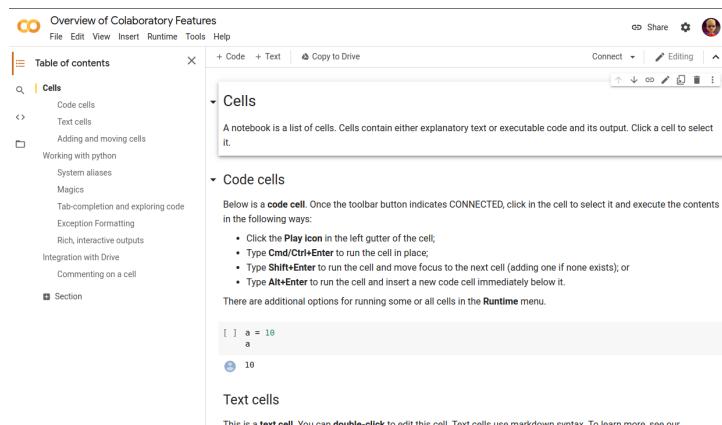


Figura 4.4: Google colab

4.5. DJI Tello

Es un pequeño drone (figura 4.5) fabricado por la empresa DJI⁷. 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 4.5: Drone Tello

4.6. Mixamo

Mixamo [11] 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 (4.6).

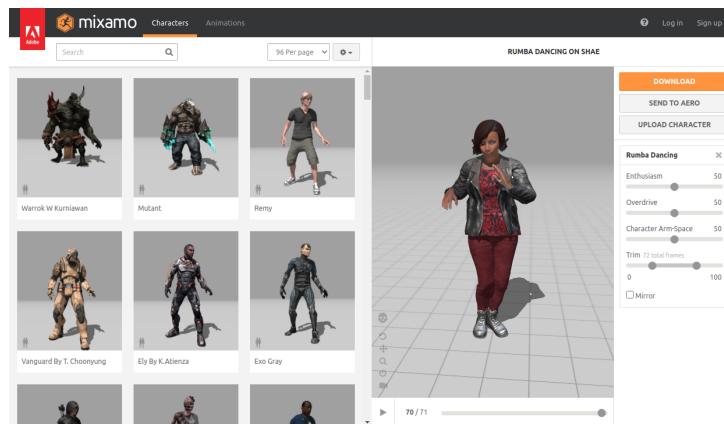


Figura 4.6: Ejemplos de Mixamo

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

4.7. Blender

*Blender*⁸ (figura 4.7) 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.

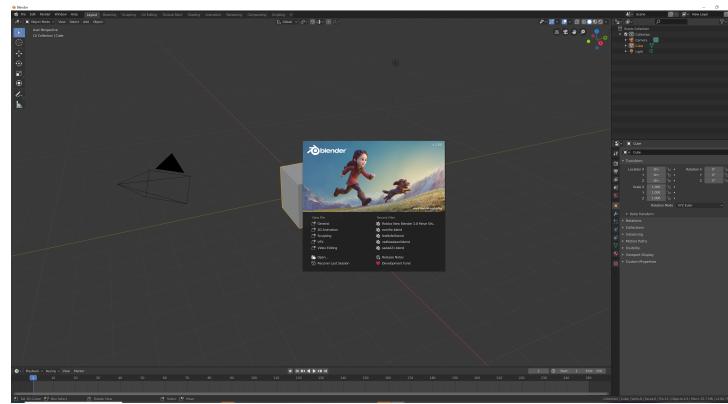


Figura 4.7: Blender

En este TFM se ha usado para modificar los modelos extraídos de *Mixamo*

4.8. LabelMe

LabelMe [12] 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^{4.8} de *datasets* que permite subir tu propio *dataset* a un servidor para poder etiquetar cada imagen desde el navegador.

⁸<https://www.blender.org>

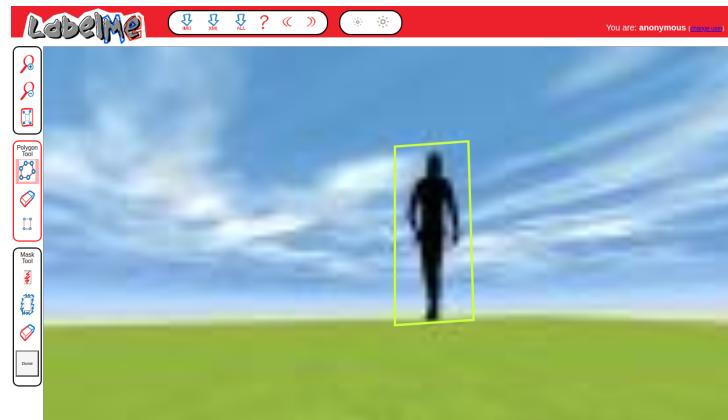


Figura 4.8: Etiquetado de LabelMe

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

Capítulo 5

Comportamiento SiguePersona visual con Drone real

En este capítulo se explica lo 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[13]. Además como va a ser usado por niños, se ha tenido que hacer más amigable el interfaz, 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.

5.1. Diseño

La aplicación desarrollada se compone de 2 partes claras (figura 5.1), la primera es la detección neuronal y la segunda los controladores PID.

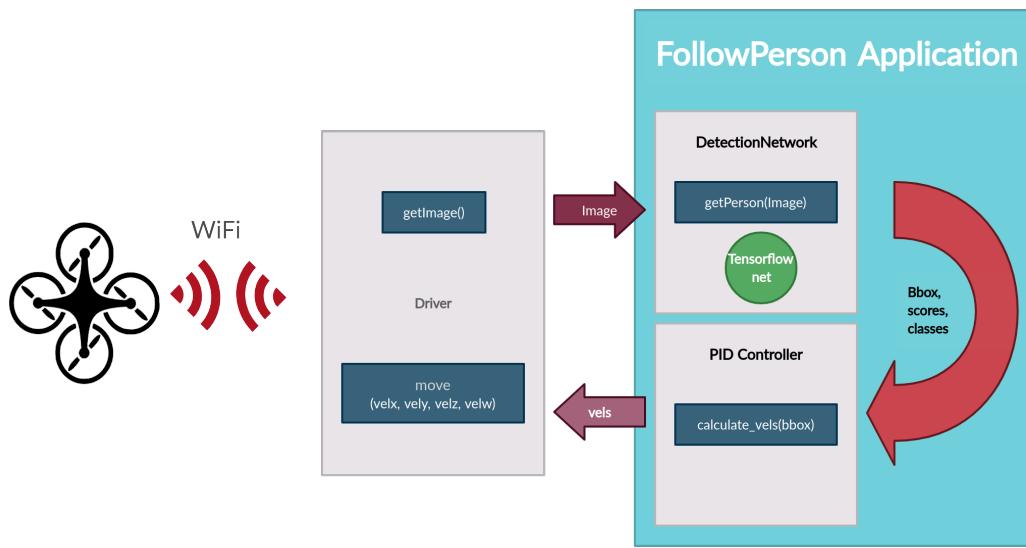


Figura 5.1: Diseño Aplicación

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. Desarrollo del *driver*

El *driver* desarrollado para la plataforma **Kibotics** parte del proporcionado por el fabricante del drone. Añadiendo 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, 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,...),...

5.2.1. Primera versión

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

- El *driver* no tiene control en velocidad.
- Si pasan 5 segundos sin recibir mensajes el drone se desactiva
- Las velocidades de entrada están en **cm/s** y **grados/s**.

No tiene control en velocidad

El *driver* aportado por el fabricante en **Python** viene sin control en velocidad, si 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 SDK¹ se vio que si existe un comando para el control en velocidad, pero que no se ha implementado en el driver aportado. Se implementa dicho método en el *driver*. 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 abordado añadiendo un hilo de **Python**(perro de guarda, *watchdog*) que se encargue de enviar velocidades cada **500ms**. De esta manera evitamos que el *Drone* se desactive.

5.2.2. Segunda versión

Los problemas subsanados en esta versión han sido los siguientes:

- *Driver* necesario en **Python 3.x** pero 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

¹https://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

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 5.1: Cambios de **Python 2** a **Python 3** efectuados

En la tabla 6.1 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 indica 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 en 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.

5.3. 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

²https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2detection_zoo.md

- SSD Resnet50 FPN de 640x640

Para esta elección se ha grabado un vídeo con la cámara del drone (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 5.2.

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

Tabla 5.2: Comparativa de redes

Se ha elegido *SSD Mobilenet v2 FPN*(figura 5.2) porque es la única que funciona a una velocidad razonable.

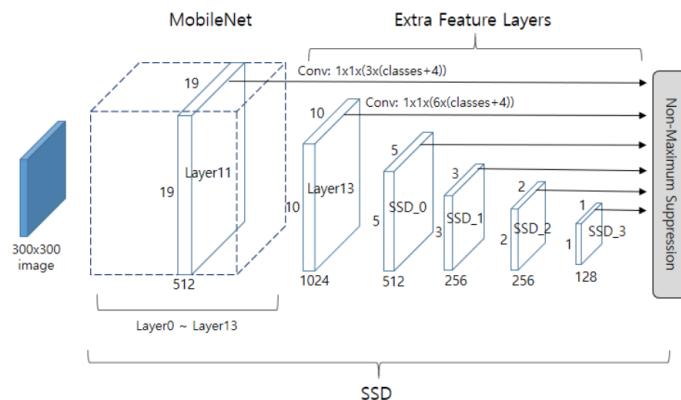


Figura 5.2: Esquema de una red Mobilenet ssd

Esta red tiene 3 componentes principales:

- *Feature Pyramid Networks (FPN)* [14]. Permite extraer características de la imagen con indiferencia del tamaño del objeto
- *Mobilenet v2*[15]. Es una red convolucional de extracción de características.
- *Single Shot Detector (SSD)*[16]. Comprueba si hay cada clase en las regiones de interés seleccionadas y permite un mayor ajuste de dicha región.

Para poder trabajar con ella en se usa **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 5.3):

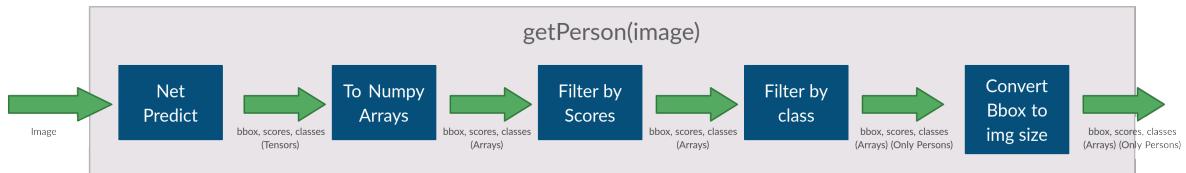


Figura 5.3: función *getPerson*

- 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* para que pueda ser usado de manera sencilla desde las nuevas aplicaciones de robótica de los usuarios de *Kibotics*.

5.4. control PID

El control PID es un bucle de control que calcula la siguiente operación 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 K constantes que deben calcularse experimentalmente.

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.

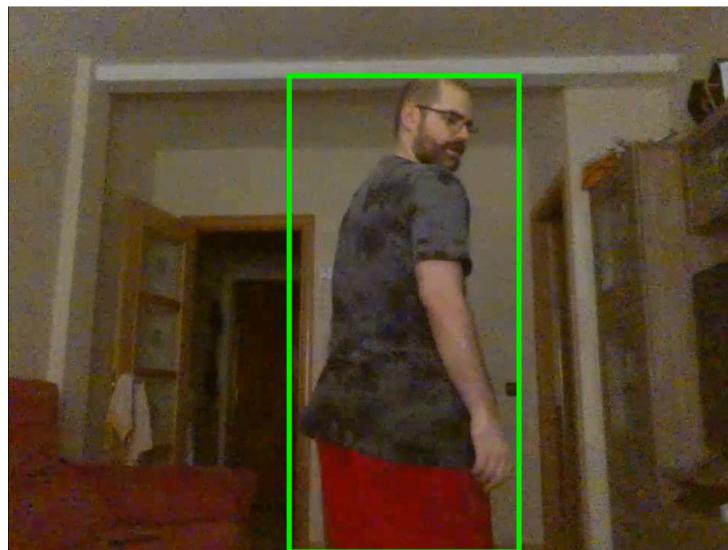


Figura 5.4: Detección de persona

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 **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 y esté centrado. como se puede ver en las imágenes 5.13 y 5.14 en ambos casos el centro del cuadrado está muy próximo al centro vertical de la imagen, pero no por ello estar bien.

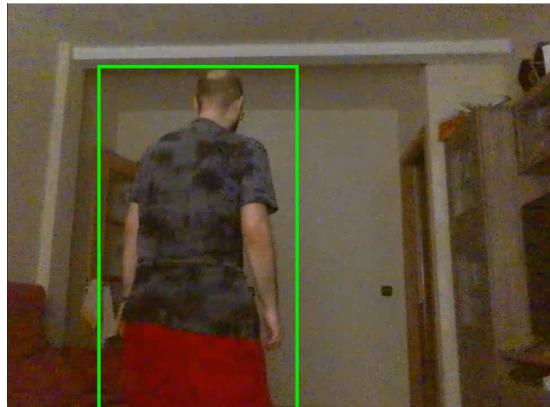


Figura 5.5: Persona seleccionada



Figura 5.6: Torso detectado

En cambio tomando como referencia el punto más alto del cuadrado se tiende a obtener la imagen 5.13 que además facilita la detección de la persona. Las constantes **proporcional y derivativa** en este caso son **3 y 0.5** respectivamente. 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.

5.5. Validación experimental

La validación experimental del desarrollo se ha hecho en 4 casos, uno por cada *PID* y el conjunto. En todos los casos el bucle de control funciona a 7 FPS.

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.

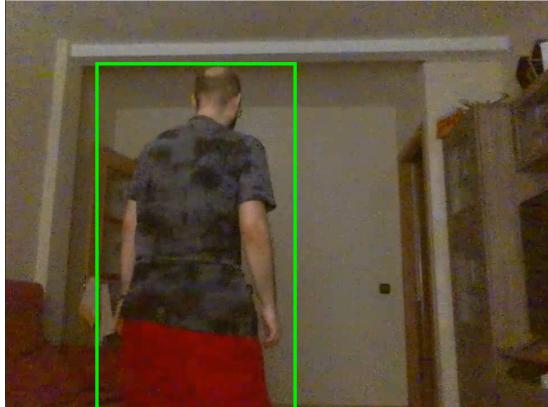


Figura 5.7: Persona seleccionada



Figura 5.8: Torso detectado

Control de elevación

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

Control de avance

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

Ejecución típica completa

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



Figura 5.9: Persona seleccionada



Figura 5.10: Tórso detectado

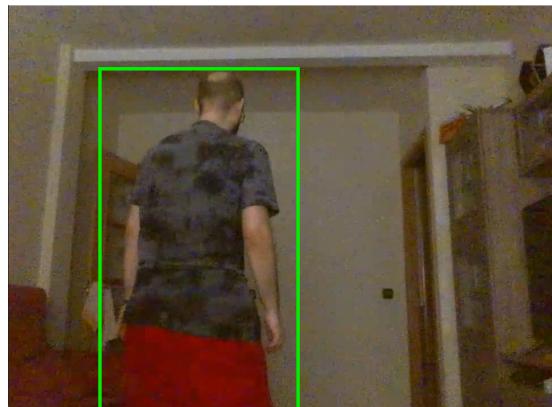


Figura 5.11: Persona seleccionada



Figura 5.12: Tórso detectado

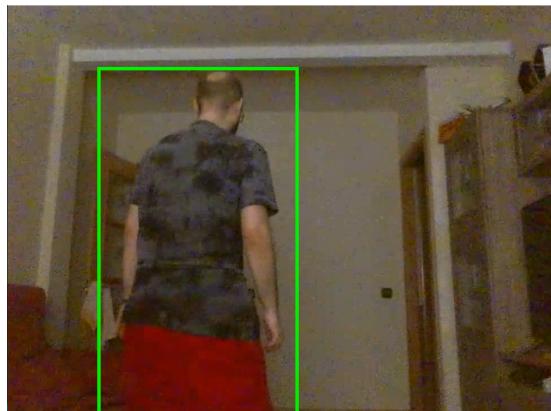


Figura 5.13: Persona seleccionada



Figura 5.14: Torso detectado

Capítulo 6

Drone simulado

En este capítulo se explica lo 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.

6.1. Diseño

El caso de la versión simulada es igual a la real, se compone de 2 partes claras (figura 5.1), la primera es la detección neuronal y la segunda los controladores PID.

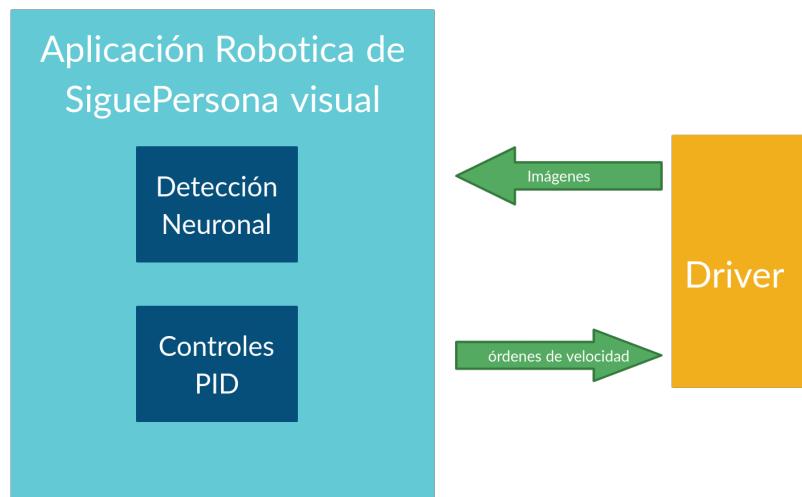


Figura 6.1: Diseño versión de simulador

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.

6.2. 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 6.2).

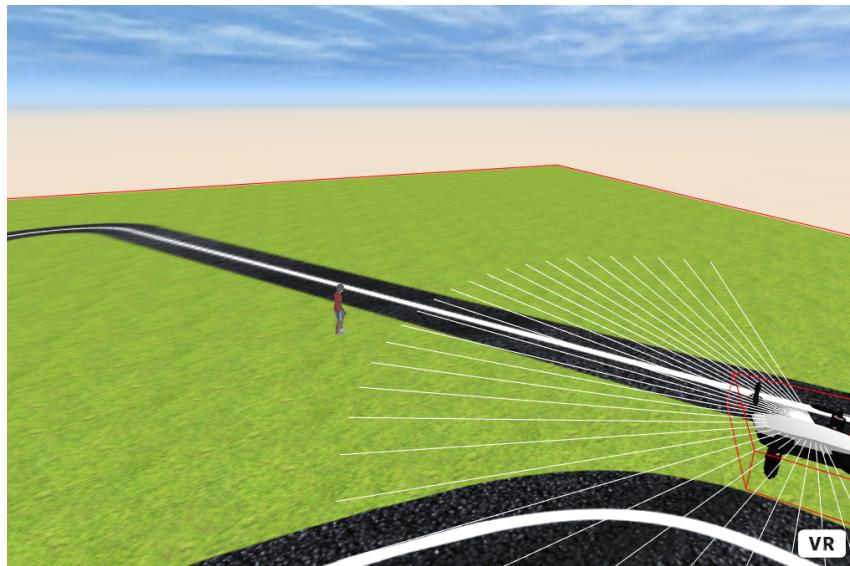


Figura 6.2: Mundo simulado

Para ello se ha necesitado descargar varios modelos 3d de personas de la web mixamo [11] (figura 6.3).

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 guardarla en un solo fichero *binary graphic library transmission format (GLB)*. el resultado se puede ver en la figura 6.4.



Figura 6.3: Modelo 3D de persona



Figura 6.4: Modelos 3D seleccionados

6.3. Comportamiento siguePersona visual

EL comportamiento en el caso simulado es casi idéntico al real, cambiando cosas propias del lenguaje en cada caso.

Detección visual de la persona

Debido a las limitaciones en la capacidad de procesamiento propias del interprete de *Javascript* en este caso no se han probado con mas 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 5.2).

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 la misma que en *Python* pero con tres cambios importantes para mejorar rendimiento (figura 6.5):

- Se elimina el postprocesado del modelo original.
- se suprime el *NonMaxSuppression* multiclas 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

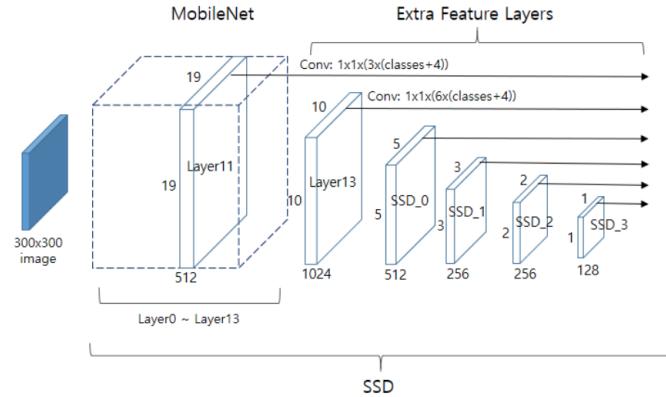


Figura 6.5: Esquema de una red Mobilenet ssd usada en *TensorflowJS*

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

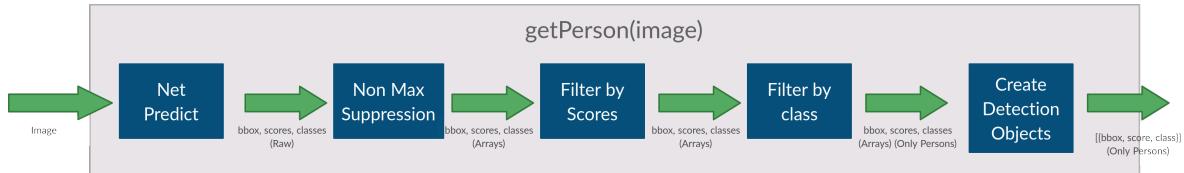


Figura 6.6: función *getPerson* en JS

- Se pasa la salida de la red por una función *NonMaxSuppression* monoclas para obtener ya las detecciones, puntuaciones y clases.
- Se filtran detecciones para eliminar las que tengan menos que el límite indicado por configuración.
- Se filtra por clase para quedarnos 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.

controles PID

Una vez que se procesa la imagen recibida de el drone y se tienen las detecciones, 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.

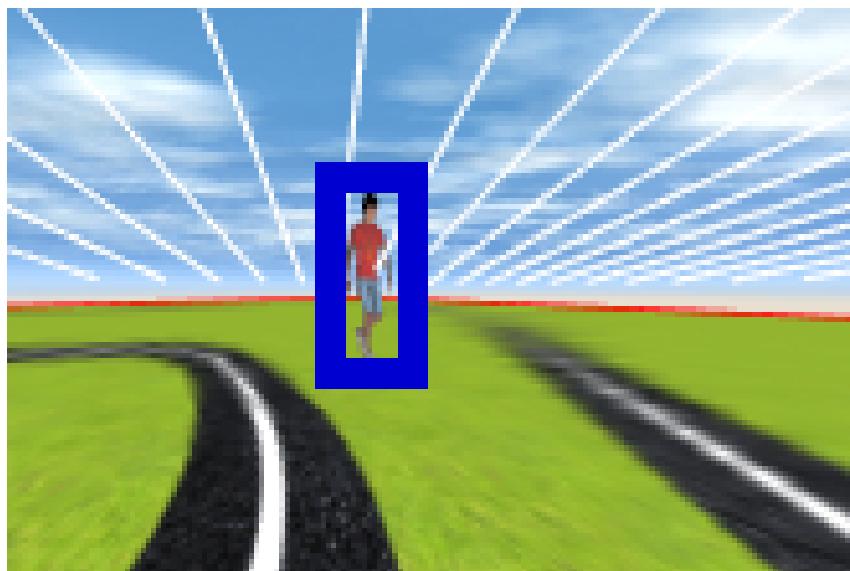


Figura 6.7: función *Detección ideal*

Para controlar el drone se va a hacer con dos velocidades, avance 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

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 **proporcional y derivativa** en este caso son **0.002 y 0.0001** respectivamente.

6.4. Creación del *dataset*

En las primeras pruebas con el drone simulado se detecta que la detección del modelo es muy deficiente. No es capaz de detectarlo 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* creado con capturas de imagen de cámara del drone[17] (figuras 6.8 y 6.9).



Figura 6.8: Ejemplo 1 del *dataset*



Figura 6.9: 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 6.10). De esta manera se han obtenido 4000 imágenes aproximadamente.

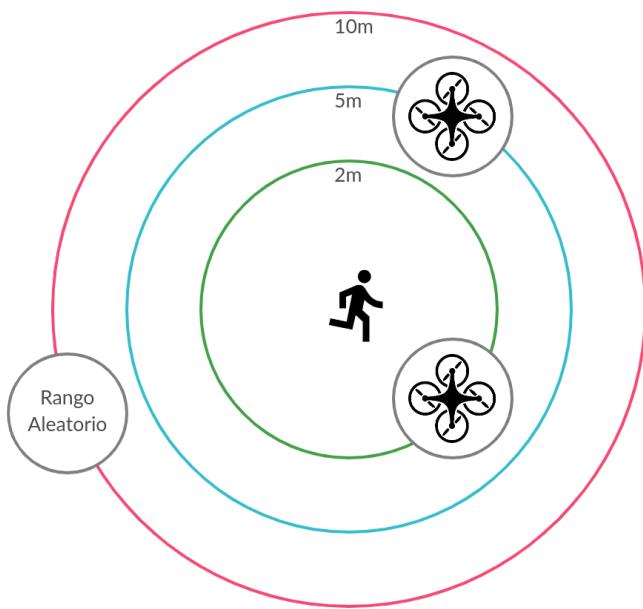


Figura 6.10: función *Esquema de capturas*

Una vez generadas todas las capturas se han etiquetado manualmente, primero se han subido a la web *LabelMe* que permite etiquetar las imágenes (figura 6.11).

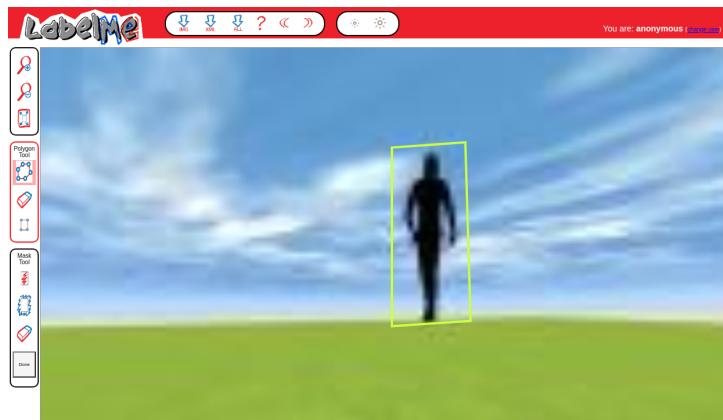


Figura 6.11: Etiquetado del *dataset* en *LabelMe*

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

Para aumentar el tamaño 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.

6.5. 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. 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 6.1: Resultados del entrenamiento

Además se han obtenido una perdida del 0.3. Con estos datos junto con una pequeña bajada del límite 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.

6.6. Validación experimental

La validación experimental del desarrollo se ha hecho en 3 casos, uno por cada *PID* y el conjunto. En todos los casos el bucle de control funciona a 7 FPS.

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. Permite ajustar las constantes del *PID* de giro horizontal (figuras 6.12 y 6.13).



Figura 6.12: Modelo a la izquierda



Figura 6.13: Modelo a la derecha

Control de avance

Esta prueba consiste en colocar uno de los modelos 3D alejándose y acercándose al drone para ajustar las constantes del *PID* de avance (figuras 6.14 y 6.15).



Figura 6.14: Modelo cerca

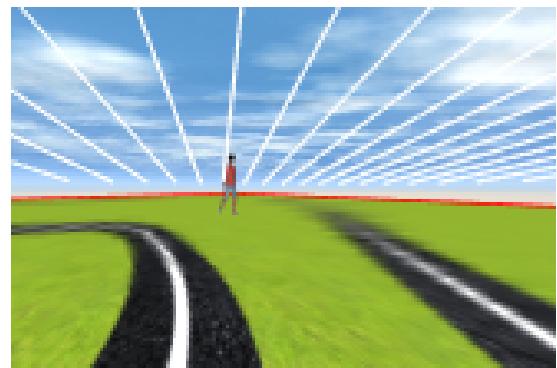


Figura 6.15: Modelo lejos

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 con las flechas del teclado, para poder mover libremente el modelo 3D para que lo siga el *drone* (figura 6.16). si el movimiento no es el esperado, toca ajustar las constantes oportunas.

CAPÍTULO 6. DRONE SIMULADO

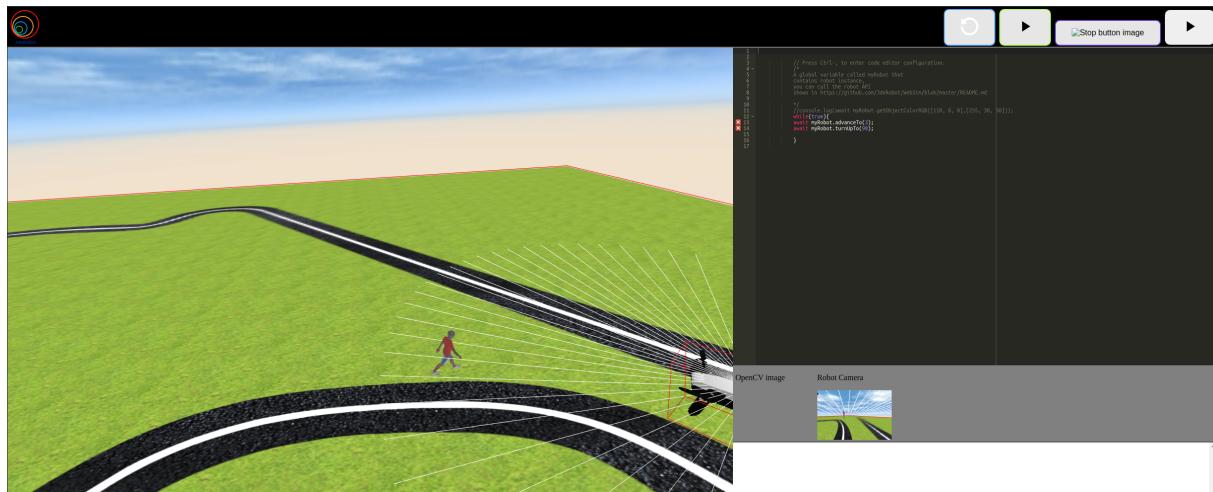


Figura 6.16: Ejecución Típica

Capítulo 7

Conclusiones

conclusiones

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] Y. Abdullah, G. Mehmet, A. Iman and B. Erkan. A Vehicle Detection Approach using Deep Learning Methodologies. *Computer Engineering Department Ankara University*, 2018.
- [3] 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.
- [4] 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.
- [5] 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.
- [6] 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.
- [7] 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.
- [8] 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>.

CAPÍTULO 7. CONCLUSIONES

- [9] Asociación de robótica e inteligencia artificial JdeRobot. Kibotics. <https://kibotics.org>.
- [10] Tensorflow. <https://github.com/tensorflow/tensorflow>.
- [11] Adobe Systems Incorporated. Mixamo. www.mixamo.com.
- [12] Labelme. <http://labelme.csail.mit.edu/Release3.0/>.
- [13] DJI. Dji tello driver python. <https://github.com/dji-sdk/Tello-Python>.
- [14] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2017.
- [15] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019.
- [16] 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.
- [17] Dataset de imágenes de websim. https://github.com/aitormf/websim_person_dataset.
- [18] 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].
- [19] 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].