Universidad Politécnica
de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

Máster Universitario en Inteligencia Artificial

Trabajo Fin de Máster

Sistema de detección y seguimiento de objetos con un dron basado en la red Yolo

Autor(a): Raúl Bonmatí Campello

Tutor(a): Javier de Lope Asiaín

Madrid, Julio 2024

Este Trabajo Fin de Máster se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Máster
Máster Universitario en Inteligencia Artificial
Título: Sistema de detección y seguimiento de objetos con un dron basado en la red Yolo
Julio 2024

Autor(a): Raúl Bonmatí Campello

Tutor(a):
Javier de Lope Asiaín
Departamento de Inteligencia Artificial
ETSI Informáticos
Universidad Politécnica de Madrid

Dedicado a toda mi familia, en especial a mis padres, mi hermana por apoyarme y aguantar las pruebas con el dron hasta altas horas de la madrugada, y a mi abuelito Diego por inculcarme con su pasión la importancia de la ciencia en este mundo.

También dedicado a mis amigos, por hacerme creer en los momentos difíciles, cuando a veces ni siquiera yo creía, que podía acabar con este trabajo, en especial a Alberto, Dani y Pilar.

A mis compañeros y amigos del trabajo por motivarme cuando después de nuestras largas jornadas laborales tenía que ponerme con el TFM al llegar a casa.

Tampoco quiero olvidarme de mi tutor, Javier, por sus consejos, su empatía y su paciencia durante estos años que no han sido especialmente fáciles.

Muchas gracias a todos.

Resumen

Hoy en día la detección y rastreo de objetos mediante vehículos u otros elementos autónomos aplicando la inteligencia artificial da lugar cada vez a más aplicaciones y usos que suponen una mejora en la seguridad y la calidad de vida de las personas.

Uno de los vehículos autónomos más empleado es, sin lugar a dudas, el dron. En torno a esta nueva necesidad, esta memoria recoge el desarrollo y posterior validación de un sistema de detección, localización y rastreo de objetos mediante un dron.

Para ello utilizamos el algoritmo de reconocimiento de objetos en tiempo real YOLO, You Only Look Once, concretamente, su versión 4, que implementado utilizando Darknet como arquitectura de red neuronal permite una rápida y precisa detección de los distintos objetos con los que ha sido entrenado.

Abstract

Nowadays, the detection and tracking of objects by means of autonomous vehicles or other elements applying artificial intelligence is giving rise to more and more applications and uses that improve people's safety and quality of life.

One of the most widely used autonomous vehicles is undoubtedly the drone. In response to this new need, this report includes the development and subsequent validation of a system for the detection, localization and tracking of objects using a drone.

For this purpose, we use the real-time object recognition algorithm YOLO, You Only Look Once, specifically, its version 4, which implemented using Darknet as neural network architecture allows a fast and accurate detection of the different objects with which it has been trained.

Tabla de contenidos

Índice de figuras

Índice de tablas

1 Introducción

Hoy en día el uso de drones está muy extendido, y todo este auge que se está experimentando en el ámbito de los vehículos aéreos no tripulados, también llamados UAVs, viene acompañado de un avance sin precedentes tanto en las tecnologías que se pueden implementar en ellos como en las diferentes tareas que pueden desempeñar.

De entre los muchos usos que se le pueden dar a los drones actualmente, podemos destacar sus aplicaciones en los siguientes campos:

- **Agricultura**, donde permiten realizar un monitoreo más completo que el que podrían hacer los humanos a nivel de suelo y así detectar malezas, enfermedades en cultivos de forma más temprana y poder hacer una estimación más correcta de los rendimientos que se puedan llegar a obtener [1].
- **Exploración y mapeo** del terreno, ya sea para la elaboración de mapas topográficos de gran detalle, la evaluación de los daños provocados por un desastre natural, o la lucha contra incendios [2].
- **Seguridad**, tanto por la parte humana en tareas como la búsqueda y rescate de personas desaparecidas, la vigilancia de fronteras y eventos multitudinarios y la prevención de delitos aplicando técnicas de reconocimiento facial. Como también a nivel de infraestructuras, mediante la revisión del estado de elementos como puedan ser puentes, carreteras, líneas y torres eléctricas, vías del tren, etc., con el fin de detectar con la mayor brevedad posible grietas, anomalías u otros daños que puedan comprometer tanto el funcionamiento como la seguridad de los mismos.
- **Logística**, donde algunas empresas como Amazon con Prime Air [3] o Alphabet con Wing [4] están comenzando a utilizar los drones como forma de transporte de mercancías pequeñas.

Cabe destacar que la mayoría de las aplicaciones mencionadas anteriormente, por no decir todas, requieren del empleo de la inteligencia artificial, concretamente de la rama de la visión artificial o visión por computador, aplicada sobre todo al reconocimiento de objetos y patrones.

Por este motivo, el desarrollo de algoritmos y modelos que permitan realizar dicha tarea de búsqueda, reconocimiento y rastreo de objetos en tiempo real es frenético, y constantemente aparecen nuevos avances y metodologías que hacen la tarea de forma más rápida y precisa, como puede ser el caso de YOLO.

1.1 Descripción del problema

En este TFM se pretende abarcar el tema del reconocimiento de objetos y su posterior rastreo mediante un dron, pero de una forma más generalista y, sobre todo, que se pueda realizar desde casa de forma más o menos sencilla.

De esta forma, se plantea desarrollar un sistema que consiste en la navegación de un dron por un entorno de interior que permita el reconocimiento de una serie de objetos previamente definidos, así como el seguimiento de los mismos en tiempo real.

1.2 Objetivos

Para el desarrollo del proyecto se establecen los siguientes objetivos:

- Seleccionar los objetos que constituirán las clases a detectar por el modelo y generar el dataset apropiado para ello.
- Definir el tipo de herramienta que vamos a utilizar para el reconocimiento de objetos.
- Entrenar y testear dicho modelo hasta conseguir la precisión necesaria.
- Desarrollar el código que permita el control y monitoreo del dron desde el PC.
- Implementar el modelo entrenado en el código de control del dron, concretamente en el módulo encargado del reconocimiento y rastreo de objetos.
- Determinar los valores de los parámetros que permitan un mejor seguimiento de los objetos.
- Conseguir que al enviarle al dron el comando de búsqueda del objeto X, éste inicie una maniobra de rastreo hasta encontrarlo y aproximarse a él hasta una distancia aproximada de 80 cm, además, en caso de que el objeto se mueva, el dron deberá seguirlo manteniendo dicha distancia.

1.3 Estructura del documento

Por lo que respecta a la estructura del documento, en el segundo capítulo se pone en contexto el estado actual de desarrollo de algunas de las técnicas empleadas para el desarrollo de este trabajo, como puedan ser las redes convolucionales (CNN), concretamente un tipo de arquitectura de red específica como es YOLO y su implementación en el framework Darknet.

A lo largo del tercer capítulo se definirán las distintas herramientas utilizadas en este trabajo, tanto a nivel hardware, como puedan ser los objetos de entrenamiento, el dron, como software para establecer la comunicación PC-dron. Además, en este capítulo también se expondrá el proceso seguido para la obtención del dataset de forma manual.

Durante el cuarto capítulo se desarrolla el entrenamiento del modelo de YOLOv4, incluyendo detalles acerca de la arquitectura del mismo, así como una breve explicación de los diferentes errores de entrenamiento que se han observado durante el desarrollo del trabajo.

En el quinto capítulo, se define la arquitectura software empleada en el trabajo y cada uno de los distintos módulos que entran en juego para poder realizar las distintas acciones necesarias para el control del dron.

En el sexto capítulo se exponen los resultados tanto de los test realizados con imágenes de vídeos aplicando el modelo entrenado, como algunas demos de vuelos resales del dron reconociendo y siguiendo objetos.

Finalmente, en el séptimo capítulo se muestran las conclusiones en base a los resultados obtenidos. Para, posteriormente, en el octavo presentar posibles trabajos futuros.

Los últimos dos capítulos estarán dedicados a bibliografía y anexos respectivamente.

2 Estado del arte

2.1 Detección de objetos en tiempo real

Como se ha mencionado previamente en este trabajo, el aumento en la utilización de drones para cada vez un mayor número de tareas, está estrechamente relacionado con el avance de los sistemas de detección y reconocimiento de objetos en tiempo real, que consiste en la localización y clasificación de un determinado objeto en una imagen que contenga múltiples objetos.

Otros sistemas de detección de objetos son RCNN y Faster RCNN. Estos sistemas se caracterizan por una alta precisión en la clasificación de los objetos a cambio de una velocidad de procesamiento muy lenta que los hace inviables para su uso en tiempo real. Se trata de detectores de dos etapas.

La búsqueda del equilibrio entre precisión (**mAP**) y una velocidad de procesamiento que permita el procesamiento de imágenes en **tiempo real**, es decir que sea capaz de procesar 30 imágenes o más por segundo, pone a **YOLO** en el punto de mira [5].

2.2 YOLO

YOLO, de sus siglas ***You Only Look Once***, representa el sistema más avanzado hoy en día en el estado del arte de los sistemas de reconocimiento de objetos en tiempo real. Su funcionamiento viene definido por una única red neuronal que se aplica a la **imagen completa**, lo que dota de una gran velocidad de inferencia al sistema de detección de objetos.

Una de las características que hace especial a YOLO es que abarca la detección de objetos como un problema de **regresión** en vez de clasificación como hacen el resto de métodos conocidos.

Una de las características principales de YOLO es que utiliza características extraídas del escaneo de la imagen al completo para localizar un objeto en concreto, es decir, aprende de forma global en vez de local con respecto a la imagen de entrenamiento. Este proceso, consiste en ir recorriendo la imagen con una ventana o máscara predefinida que devolverá un valor booleano indicando si el objeto a localizar está o no dentro de dicha área de escaneo. Gracias a esto YOLO tiene muchos menos errores de detección de objetos incorrectos en el fondo de las imágenes debido a su mayor conocimiento del contexto de la imagen al completo.

Además, una de las claves de YOLO consiste en que, al tratarse de un detector de imágenes de una sola etapa, es mucho más eficiente que otros de dos etapas como R-CNN y sus variantes, ya que permite procesar la imagen completa en un único paso.

Por otro lado, cabe destacar que YOLO ha ido desarrollándose y mejorando a lo largo del tiempo, habiendo ya varias versiones del mismo. A continuación, se detallan las principales características y mejoras implementadas en cada una de sus actualizaciones, hasta su versión 4 que será la utilizada en este trabajo [6] [7].

2.2.1 YOLOv1

La primera versión de YOLO [8] se caracterizó por el empleo de técnicas como *Deformable parts model* (**DFM**), la cual consiste en ir recorriendo la imagen con

una ventana o máscara predefinida que devolverá un valor booleano indicando si el objeto a localizar está o no dentro de dicha área de escaneo. Además, una de las novedades que supone este modelo es el hecho de introducir el concepto de los detectores de objetos de una sola etapa en lugar de los tradicionales de dos etapas.

En cuanto a su arquitectura, ésta se caracteriza por estar formada por 24 capas convolucionales y 2 capas totalmente conectadas.

2.2.2 YOLOv2

Para la segunda versión de YOLO [9], se testean una serie de cambios, hasta dar con aquellos que afectaban de manera favorable al **mAP**. De esta forma se acaban introduciendo mejoras como los 'anchor boxes' [10].

Por otra parte, su arquitectura cambia a **Darknet-19** que como su nombre indica consta de 19 capas convolucionales y 5 capas de maxpool (mejora de su 'backbone'), haciendo la red un poco más profunda que su versión inicial, lo que permite una mayor extracción de características de las imágenes.

2.2.3 YOLOv3

En la tercera versión de YOLO [11], se busca mejorar la detección de objetos pequeños, la detección multi escala y la extracción de características en diferentes capas, mediante el empleo de técnicas como 'skip connections'.

En cuanto a su arquitectura, se trata de una arquitectura **DarkNet-53**, con 53 capas convolucionales es mucho más poderoso y eficiente que DarkNet-19.

2.2.4 YOLOv4

Por último, para la cuarta versión [12], que además será la utilizada en este proyecto, podemos destacar que para el '*backbone'* de su arquitectura se probaron tres alternativas diferentes de red (CSPResNext50, CSPDarkNet53 y EfficientNet-B3), eligiéndose finalmente la arquitectura **CSPDarkNet53**. En cuanto al '*Neck'* de la red, se establece **PANet** junto con *Spatial Pyramid Pooling* (SPP) como métodos de agregación de parámetros o características de la red, que tendrá como objetivo preparar dichas características previo paso a la detección. Por último, el 'head', encargado de la detección de los objetos, será el mismo que utiliza YOLOv3.

Por lo que respecta a las modificaciones implementadas para mejorar el rendimiento, podemos destacar las siguientes:

- **Bag of Freebies**, se trata de un método de *data augmentation* que consiste en aumentar la variabilidad de las imágenes de input para hacer que el sistema sea más robusto como podemos observar en la *Figura 1*.
- **Bag of Specials**, hace referencia al conjunto de las diferentes funciones de activación e inferencia implementadas, que a cambio de un poco más de coste de inferencia proporcionan una mejora considerable en la presión de la detección.

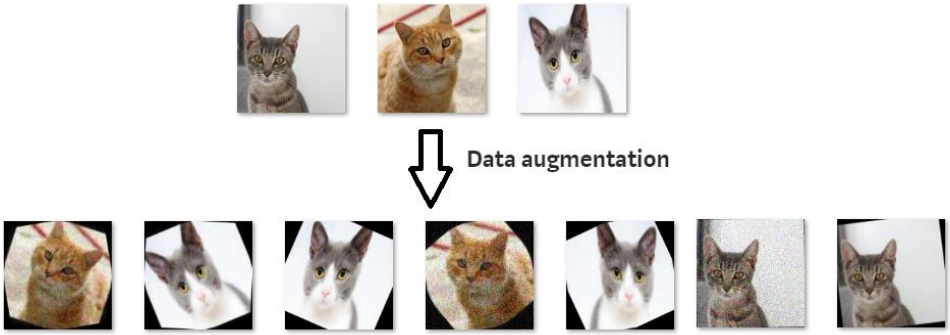

Figura 1 - Ejemplo de 'data augmentation' utilizado en Bag of freebies [20]

Para determinar que módulos de los estudiados en '*Bag of freebies'* y '*Bag of specials*' eran los mejores, se realizó un **estudio de ablación**, que consiste en ir eliminando algunos de estos inputs uno a uno para ver cuales afectan de forma relevante a los outputs de la red.

3 Herramientas utilizadas

Para llevar a cabo el desarrollo de este trabajo, se necesitarán diversas herramientas además de lo esperado cómo puedan ser un ordenador para poder entrenar al modelo de inteligencia artificial. Estas herramientas consistirán en el dron que utilizará dicho modelo para detectar los objetos a través de su cámara, así como el dataset obtenido mediante dicha cámara y que emplearemos para entrenar al modelo. A continuación, entraremos más en detalle en cada uno de ellos.

3.1 Dron

La percepción, en el ámbito de la inteligencia artificial que como siempre busca emular a los humanos, tiene que ver con el reconocimiento del entorno físico, además de la localización, el seguimiento y la monitorización de éste.

Para este trabajo el elemento del que dependerá la percepción y sobre el que aplicaremos el modelo de inteligencia artificial será un dron, concretamente el modelo **Tello** de la marca Ryze [13], del que hablaremos a continuación más en profundidad.

Se trata de un dron que incorpora tecnología de vuelo de una de las marcas más afamadas de este tipo de vehículos aéreos no tripulados como es DJI.

Figura 2 - Drone Tello de Ryze

Este dron se caracteriza, entre otras cosas, por ser un dron de pequeño tamaño con apenas 10 cm y 80 gramos de peso, diseñado sobre todo para entornos de interior por lo que carece de sistema GPS. Además, dispone de un sistema telemétrico y barómetros lo que permite recoger y transmitir información que puede resultar importante como posición, velocidad, altitud, temperatura interna, etc.

Aparte de todo esto, también está dotado de un módulo wifi y bluetooth que permiten conectar al dron con los dispositivos que se encarguen de su control. Estos dispositivos pueden ser tanto un smartphone mediante la propia aplicación del dron disponible para descargar en las aplicaciones de descarga correspondientes, como a través de un ordenador. Para este segundo caso, cabe destacar que se deberá desarrollar una interfaz con el dron siguiendo las

instrucciones que vienen definidas den el kit de desarrollo de software (SDK) propio que podemos encontrar en la web oficial [14].

Pero lo que más nos interesa del dron para el desarrollo de este trabajo son su **sistema de posicionamiento por visión** que gracias a una cámara y a un módulo 3D de infrarrojos situados en su parte inferior, permite reconocer patrones que sobrevuela el dron y de esta forma se auto estabiliza, así como su **cámara frontal** de 5 Mp que permite obtener imágenes en formato JPG o vídeos en HD a una resolución de 720p a 30 FPS.

El uso de este pequeño dron está muy extendido, por lo que hay numerosas librerías y controladores desarrollados en diferentes lenguajes basándose en el SDK, sin embargo, para este trabajo se ha decidido desarrollar todo el módulo de interfaz de control y monitorización del dron desde cero. Esto se debe a que el control del vehículo durante todo el trabajo se realizará desde un ordenador en el que se ejecutará dicho código encargado de las funciones previamente mencionadas, así como de comunicación con el mismo, que se detalla a continuación.

3.1.1 Comunicación

Para establecer la comunicación ordenador-dron se emplea la comunicación **Wi-Fi**, concretamente a través de diferentes puertos UDP, cada uno de ellos encargado de una función determinada:

- **Comunicación y comandado**, mediante una comunicación UDP hacia la dirección IP del dron (192.168.10.1) puerto 8889. Para inicializar la comunicación de control del dron se enviará el comando *'command'*, y una vez éste devuelva el *'ok'* ya se podrán enviar el resto de comandos necesarios para su control. Algunos de los comandos utilizados a lo largo del proyecto están reflejados en la Tabla 4.
- **Estado del dron**, que con una comunicación UDP través del puerto 8900 permite recibir continuamente un conjunto de variables que nos aportan información sobre el estado del dron y que nos llegan en forma de un string con el siguiente formato:

  "pitch:%d;roll:%d;yaw:%d;vgx:%d;vgy%d;vgz:%d;templ:%d;temph:%d;tof:%d;h:%d;bat:%d;baro: %.2f; time:%d;agx:%.2f;agy:%.2f;agz:%.2f;\r\n"

  Donde %d hace referencia a la parte entera de un número decimal, mientras que %.2f representa un float de dos cifras decimales. Por lo que descomponiendo este string podemos obtener los siguientes valores del dron:

  - *Pitch* (elevación).
  - *Roll* (alabeo).
  - *Yaw* (dirección).
  - *Vgx* (velocidad en el eje X).
  - *Vgy* (velocidad en el eje Y).
  - *Vgz* (velocidad en el eje Z).
  - *Templ* (temperatura mínima).
  - *Temph* (temperatura máxima).
  - *Tof* (distancia al suelo).
  - *H* (altitud).

- *Bat* (porcentaje de batería restante).
- *Baro* (medida barométrica, presión).
- *Time* (tiempo de motores encendidos).
- *Agx* (aceleración en el eje X)
- *Agy* (aceleración en el eje Y)
- *Agz* (aceleración en el eje Z)

- **Stream de vídeo**, se establecerá mediante una conexión a través del puerto UDP 11111 por el que irán llegando los frames de vídeo capturados por la cámara frontal del dron, que serán procesados gracias a un objeto del tipo OpenCV.VideoCapture.

Un aspecto importante a tener en cuenta es el Firewall del ordenador, que tendremos que bien desactivar por completo, o bien, crear las reglas necesarias para poder establecer las comunicaciones pc-dron comentadas anteriormente.

3.1.1.1 Control y comportamiento

Durante el desarrollo del proyecto, uno de los procesos que más problemas supuso fue el control del dron para que su comportamiento en el entorno de trabajo fuese el adecuado.

Esto es debido a que se trata de un dron sin GPS, que utiliza un sistema de **posicionamiento por visión** formado por un conjunto cámara-sensor de infrarrojos situados en su parte inferior para reconocer aquello que sobrevuela y utilizarlo como referencia para mantener la posición. Por lo que ante cualquier cambio en las condiciones que puedan afectar a la visualización de la superficie, su comportamiento se ve afectado, dando lugar a grandes problemas de inestabilidad.

Según el propio manual del dron, algunos de los aspectos a tener en cuenta para mejorar el reconocimiento de la superficie por parte del dron para evitar dichos problemas de inestabilidad son los siguientes [15].

- Volar sobre superficies monocromáticas.
- Volar sobre superficies reflectantes o transparentes.
- Cambios en la iluminación (exceso de claridad u oscuridad).
- Volar sobre superficies sin patrones o texturas claras.
- Volar sobre superficies con patrones muy repetitivos y parecidos.

Teniendo todo esto en cuenta se realizan una serie de pruebas con el objetivo de determinar las mejores condiciones para la estabilidad del dron. Algunas de las más destacas son:

- Cubrir el suelo que era considerablemente reflectante, con esterillas o cartones de color más mate.
- Colocar folios con distintos patrones a lo largo del espacio de trabajo.
- Jugar con los niveles de iluminación, tanto natural como artificial.
- Colocar objetos de diferentes formas y orientaciones.

Una vez realizadas las pruebas pertinentes, la configuración del entorno que **mayor estabilidad** proporciona al dron es aquella en la que colocamos un fondo mate sobre el que disponemos una serie de **patrones** de formas geométricas distintas entre sí y con la mayor cantidad de iluminación artificial posible, ya que la luz solar confunde al sensor de infrarrojos.

A pesar de mejorar considerablemente la estabilidad del dron y su capacidad de mantener la posición cuando no recibe comandos, hay ocasiones en las que éste se desplaza erráticamente. Esto es debido a que mientras el dron no recibe comandos, va realizando movimientos para mantener la posición con respecto a lo que el sistema de posicionamiento por visión determina. Por lo que se plantea como solución crear una interfaz gráfica como la de la *Figura 25* que permita el **control** del movimiento del dron tanto por botones como por teclado.

Gracias a esta implementación podemos ir enviando continuamente de forma manual comandos al dron y de esta forma 'vencer' a las posibles correcciones erróneas que pueda realizar el sistema de posicionamiento por visión. Además, este control manual será de vital importancia para el siguiente paso del proyecto, la elaboración del dataset.

3.2 Dataset

Como bien sabemos uno de los aspectos más importantes a la hora de entrenar un modelo de inteligencia artificial lo constituyen las imágenes utilizadas para su entrenamiento, también llamadas dataset.

Para este proyecto se consideró que la mejor forma de obtener las **imágenes** que iban a conformar el dataset era extrayéndolas directamente de **vuelos en tiempo real** del dron, debido a que el modelo de inteligencia artificial trabajará en estas condiciones. Para facilitar la obtención de dichas imágenes se desarrolla dentro del módulo de streaming de la interfaz de usuario como el de la *Figura* 3, una funcionalidad que permite guardar los **frames** del vídeo que captura el dron y guardarlos con el nombre del objeto de la clase a la que pertenecen, de esta forma se obtienen las imágenes en crudo o 'raw frames' de cada una de las clases y se clasificarán tal y cómo se explica en los siguientes apartados.

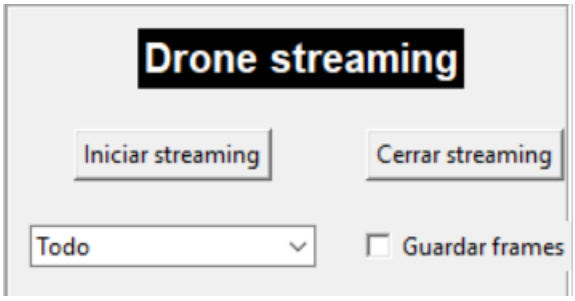

Figura 3 - Módulo de streaming de la interfaz gráfica de usuario

3.2.1 Objetos/clases

Para una primera versión del trabajo se propuso un modelo entrenado en base a 6 objetos o clases distintas: Cojín, Libro, MandoTV, Pelota, Planta y Termo.

Figura 4 - Imágenes objetos descartados (Libro y Cojín)

Por lo tanto, los objetos o clases elegidos finalmente para el entrenamiento del modelo fueron los siguientes:

- **MandoTV**
- **Pelota**
- **Planta**
- **Termo**

Figura 5 - Imagen del dataset que incluye los 4 objetos a clasificar

Una vez seleccionados los objetos, el siguiente paso para la generación del dataset es definir el tamaño del mismo que sería necesario para conseguir un modelo correctamente entrenado. Tras investigar un poco, se decide que el tamaño del dataset será de 2000 imágenes.

Un aspecto importante a tener en cuenta en la distribución de estas **2000 imágenes** es que, para un mejor entrenamiento, el número de imágenes de cada uno de los objetos o clases debería de ser lo más parecidos entre ellos. Por este motivo se decide dividir el dataset en **5 grupos de 400 imágenes** cada uno:

- Grupo 1, compuesto por 400 imágenes en las que aparece únicamente el objeto **MandoTV**.
- Grupo 2, compuesto por 400 imágenes en las que aparece únicamente el objeto **Pelota**.
- Grupo 3, compuesto por 400 imágenes en las que aparece únicamente el objeto **Planta**.
- Grupo 4, compuesto por 400 imágenes en las que aparece únicamente el objeto **Termo**.
- Grupo 5, compuesto por 400 imágenes en las que aparecen **todos o casi todos los objetos** simultáneamente.

Una cosa a tener en cuenta y que viene de lo aprendido en otro intento de entrenamiento fallido tal y como se explica más adelante en el capítulo dedicado a los errores de entrenamiento, es la importancia de los cambios de entorno, 'background', iluminación y distancia a los objetos en dichas imágenes que conforman el set de entrenamiento. Por esta razón, de las 400 imágenes de cada uno de los grupos mencionados anteriormente, 200 tienen lugar en un **entorno más iluminado** y con un fondo más claro como el de la *Figura 6*, y las **otras** 200 en un **entorno menos iluminado** y con un fondo o 'background' más oscuro como en la *Figura 7*.

Figura 6 - Imagen del dataset en entorno con buena iluminación

Figura 7 - Imagen del dataset en entorno con mala iluminación

3.2.2 Adquisición y elección de imágenes

Una vez definidos los requisitos más generales que deberá tener el dataset, el siguiente paso será la adquisición de las imágenes y su posterior clasificación manual para quedarnos con aquellas que consideremos más adecuadas para el entrenamiento. Para conseguir esto se llevan a cabo los siguientes pasos:

1. **Obtener** las imágenes en crudo (**raw frames**) para ambos entornos, a partir del vídeo capturado por el dron en tiempo real, mediante la funcionalidad implementada en el programa, tal y como se explica en el apartado anterior. Dichas imágenes son capturadas en formato **JPG** con una resolución de **960x720**.

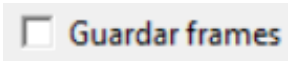

Figura 8 - Selector de guardado de frames del vídeo

2. Una vez obtenidas las imágenes suficientes para cada entorno, que para bien deberían de ser más de 200 para cada uno de los dos entornos, y para cada uno de los 5 grupos de objetos mencionados en el apartado anterior. El siguiente paso será analizar visualmente cada una de las imágenes y seleccionar aquellas que mayor diversidad y calidad aporten al entrenamiento. De entre los criterios de selección utilizados destacan:
    - Imágenes con diferentes perspectivas del objeto.

Figura 9 - Ejemplo imágenes dataset con distintas perspectivas del objeto

- Imágenes a diferentes distancias del objeto.

Figura 10 - Ejemplo imágenes dataset a distintas distancias del objeto

- Imágenes que contengan sólo parte del objeto o que estos aparezcan solapados con otros elementos, con el fin de evitar los errores aprendidos del tercer entrenamiento del modelo que se explicará más adelante en el apartado Errores entrenamiento.

Figura 11 - Ejemplo imágenes dataset con objetos solapados o recortados

Con las 400 imágenes por grupo ya seleccionadas manualmente, que conformarán las 2000 imágenes del dataset final, el siguiente paso será el etiquetado de las mismas.

Tabla 1 - Distribución imágenes dataset por clase

Clase	Nº de imágenes (etiquetas)
MandoTV	**400**
Pelota	**400**
Planta	**400**
Termo	**400**
Todo	**400**
Total	**2000**

3.2.3 Etiquetado

Es importante tener en cuenta, que el entrenamiento del modelo YOLOv4 requiere de **dos entradas** que guardan relación entre sí: por una parte, la **imagen** adquirida por el dron y posteriormente seleccionada manualmente como apta, y por otra, un archivo .txt con un formato definido que contendrá las **anotaciones** relativas a las posiciones de los objetos a clasificar dentro de la imagen.

Para el etiquetado de los objetos en las imágenes y la obtención de dichos archivos de anotaciones se decide utilizar la herramienta **LabelStudio** [16] en la que el etiquetado de cada objeto perteneciente a las clases para las que se entrenará el modelo se realizará manualmente dentro de su respectivo '*box*', tal y como podemos ver en la Figura 12 y, además, permite exportar dichas anotaciones directamente en formato YOLO.

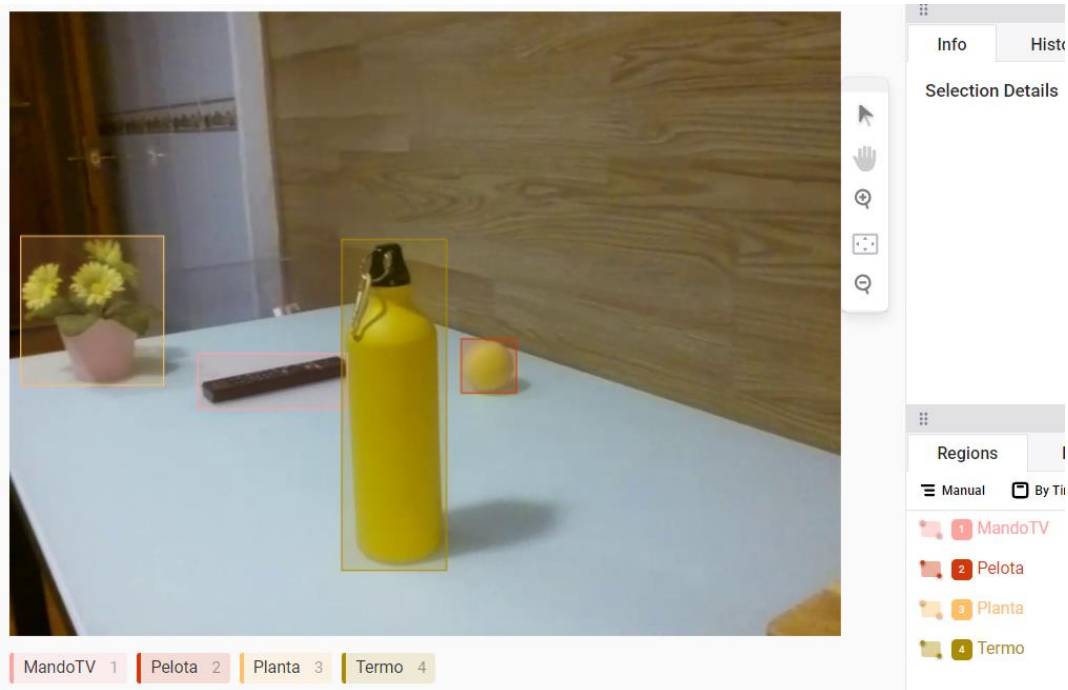

Figura 12 - Ejemplo etiquetado de todos los objetos con LabelStudio

El formato de anotaciones YOLO se caracteriza por recoger en cada una de sus líneas las siguientes 5 características de cada objeto etiquetado dentro de la imagen:

- **Object class**: define el ID de la clase a la que pertenece el objecto etiquetado.
- **X**: número de 0 a 1 que indica la posición del centro de la caja de etiquetado del objeto en el eje X de la imagen.
- **Y**: número de 0 a 1 que indica la posición del centro de la caja de etiquetado del objeto en el eje Y de la imagen.
- **Width**: indica la anchura en píxeles de la caja que contiene al objeto etiquetado.
- **Height**: indica la altura en píxeles de la caja que contiene al objeto etiquetado.

```
0 0.3161101083032489 0.5911552346570396 0.18005415162454871 0.08844765342960262
1 0.576037906137184 0.5667870036101084 0.0663357400722021 0.08664259927797852
2 0.09950361010830319 0.47833935018050566 0.17193140794223816 0.23826714801444068
3 0.46231949458483745 0.6290613718411552 0.1259025270758122 0.5288808664259927
```

Figura 13 - Ejemplo anotaciones en formato YOLO

Se decide utilizar el **etiquetado manual** ya que a pesar de ser mucho más costoso que el automático, proporciona una mayor precisión a la hora de ajustar lo máximo posible los bordes del 'box' con los del objeto, así como evitar que un objeto sea etiquetado incorrectamente como otro distinto.

Además, después de no obtener los resultados deseados en varios entrenamientos del modelo, tal y como se comenta también en el tercer error del apartado 4.3, se decide realizar una inspección visual de cada una de las imágenes ya etiquetadas con el objetivo de evitar que algún error humano en el etiquetado de los objetos conlleve un mal entrenamiento del modelo.

4 Entrenamiento del modelo de detección

A lo largo de este capítulo, se presentan las distintas alternativas que se han tenido en cuenta durante el entrenamiento del modelo, así como las decisiones que se han tomado finalmente con el objetivo de obtener los mejores resultados posibles, tanto en el entrenamiento del modelo como en el desarrollo del control autónomo del dron.

El modelo utilizado para la detección de objetos en tiempo real será **YOLOv4**, al tratarse de uno de los modelos que mejores resultados arrojan en cuanto a precisión (mAP) y velocidad de procesamiento de fotogramas (FPS) como se puede observar en la siguiente figura.

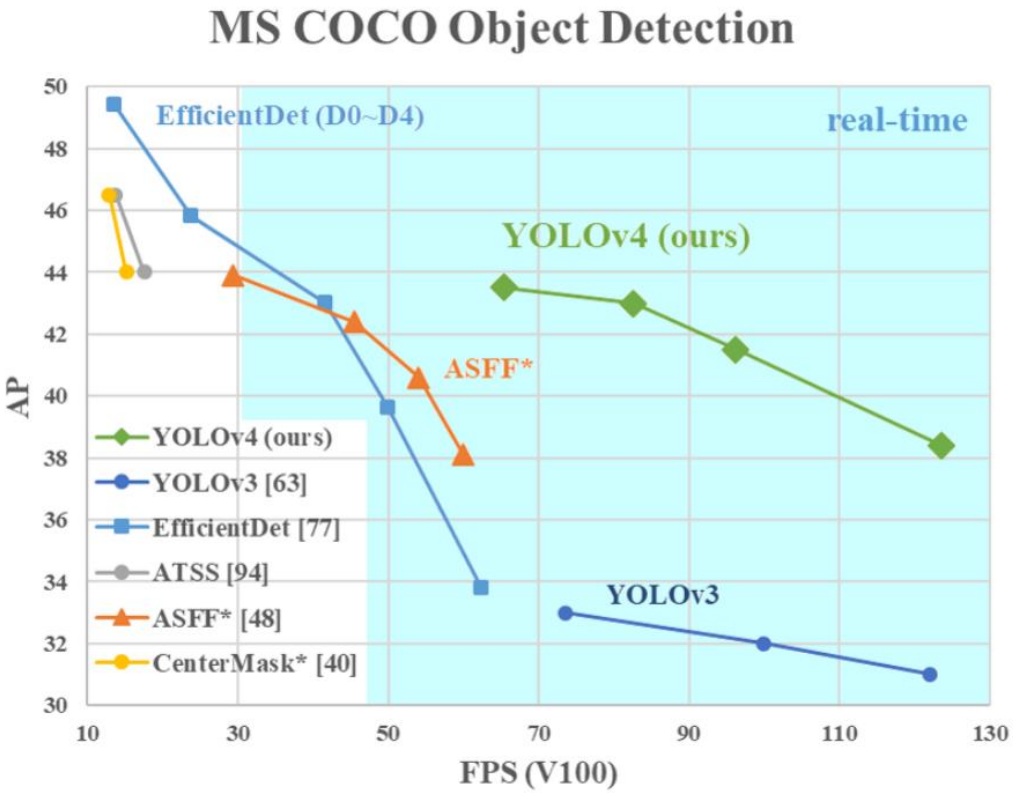

Figura 14 - Comparativa YOLOv4 con otros modelos de detección [12]

Además, se trata de un modelo con una arquitectura que permite numerosas configuraciones que proporcionarán al usuario la capacidad de desarrollar un modelo que se adapte de la mejor manera a sus necesidades, ya sea para conseguir un modelo de entrenamiento más rápido reduciendo la resolución de las imágenes de entrada, o bien, un modelo más de clasificación más preciso utilizando imágenes de mayor resolución como entrada.

Cabe destacar que el uso de las arquitecturas YOLO está muy extendido en el campo de la detección y reconocimiento de objetos, lo que facilita el encontrar información de interés para nuestro desarrollo. Dentro de dicha información, podemos encontrar diferentes implementaciones de YOLO en redes neuronales como puedan ser *Darknet* o *Darkflow*, se trata de implementaciones ya depuradas y de las cuáles es sencillo encontrar información, por lo que finalmente nos decantamos por utilizar *Darknet* [17] para nuestro modelo.

4.1 Arquitectura

Tal y como se ha detallado previamente en el apartado 2.2.4, la arquitectura de la red neuronal convolucional de YOLO en su versión 4 estará compuesta por los siguientes módulos:

- Para el *backbone* emplea **CSPDarknet53**, el mismo que utilizad la versión 3 de YOLO, pero sustituyendo las unidades lineales rectificadas (ReLU) por funciones de activación Mish y la integración de *Cross-Stage Partial connections* (CSP) en los bloques convolucionales.
- En cuanto al *neck* de la red, se caracteriza por el empleo de *Spatial Pyramid Pooling* (**SPP**) junto con una versión modificada de *Path Aggregation Network* (PAN) llamada **PANet** caracterizada por trasmitir mejor las características extraídas de cada imagen en sus distintas escalas, lo que mejorar la detección de objetos pequeños.
- Por último, el *head* será el mismo que utiliza la versión 3 de YOLO, caracterizado por trabajar sobre tres mapas de características distintos, obtenidos de diferentes escalados de la imagen, donde, entre otros métodos, se utilizan *anchor boxes* y la supresión de no-máximos para realizar la clasificación final.

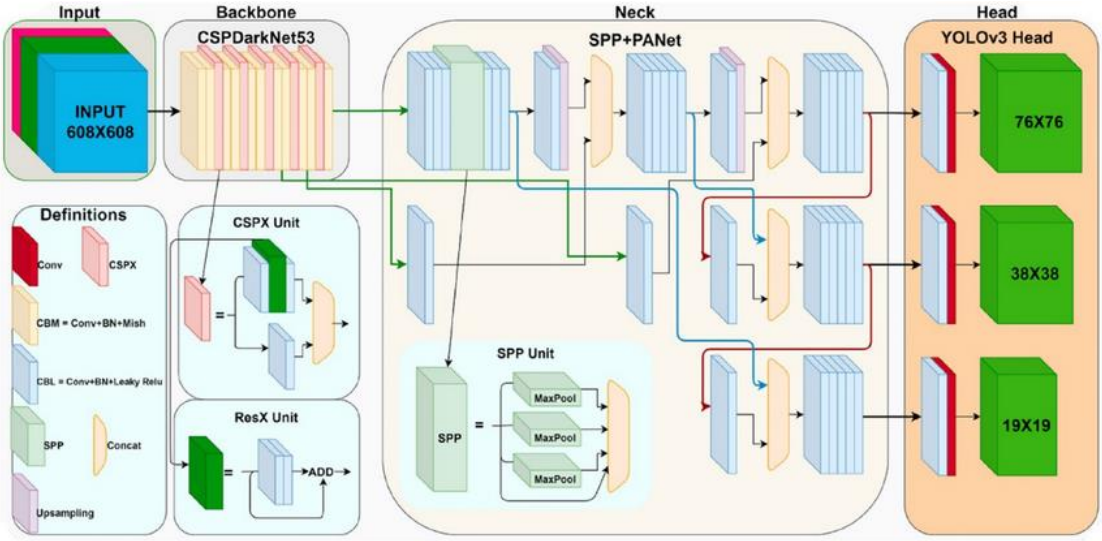

Figura 15 - Arquitectura YOLOv4 [21]

4.2 Entrenamiento

Una vez seleccionada Darknet, concretamente una versión actualizada de la versión inicial propuesta por Joseph Redmond [8] y publicada en GitHub por AlexeyAB [12], como la implementación que utilizaremos para el entrenamiento de **YOLOv4**, el siguiente paso consiste en definir el entrono para llevar a cabo dicho entrenamiento.

Para el entrenamiento se decide utilizar **GoogleCollab** ya que permite un entrenamiento del modelo mucho más rápido del que sería capaz de realizar mi portátil gracias al acceso a los recursos de computación en la nube de Google.

A continuación, se detallan los pasos a seguir para el entrenamiento en GoogleCollab:

1. Dentro de nuestro Google Drive creamos una carpeta llamada *CollabDev*, dentro de la cual creamos otra carpeta denominada *training*, dónde se guardarán los pesos del entrenamiento del modelo.
2. Crear un archivo de Jupyter Notebook que en nuestro caso llamamos *YOLOv4_model.ipynb* y vincularlo con nuestro Google Drive, concretamente con el path de la carpeta *CollabDev* creada en el paso anterior.
3. **Clonar** el repositorio *Darknet* de AlexeyAB [18] desde GitHub a la carpeta CollabDev.
4. Generar y subir a la carpeta *CollabDev* los archivos necesarios para el entrenamiento, que serán los siguientes:
    a. **obj.zip** que contiene las 2000 imágenes que componen el dataset, junto con sus respectivas 2000 anotaciones obtenidas de su etiquetado, tal y como se explica en el apartado *3.2.3*.

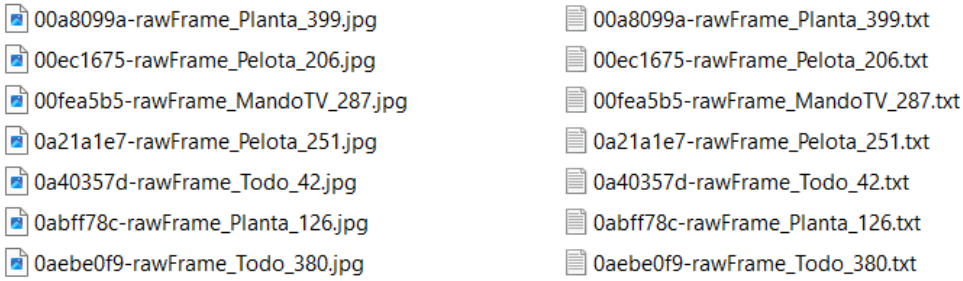

Figura 16 - Ejemplo de contenido del archivo obj.zip

    b. **yolov4-custom.cfg**, contiene la configuración de la arquitectura de la red neuronal convolucional de YOLO en la que modificamos los siguientes parámetros:
        - *Batch* = 64, indicando el número de imágenes cargadas por iteración.
        - *Subdivisions* = 16, para dividir el batch en 16 mini-batches con 4 imágenes cada uno.
        - *Width* y *height* = 416 para establecer el tamaño de la red y, por lo tanto, la resolución e las imágenes de entrada.
        - *max_batches* = 8000 (2000 * nº. de clases).
        - *steps* = 6400, 7200 (80% y 90% de max_batches).
        - *Filters* = 27 ((nº. clases + 5) *3) antes de cada capa YOLO.

- *Classes* = 4, indicando el número de clases a clasificar por el modelo.

```
batch=64 ★          [convolutional]   [convolutional]   [convolutional]
subdivisions=16 ★   size=1            size=1            size=1
classes = 4
train = data/train.txt
valid = data/test.txt
names = C:/Users/Raulbc/Documents/Projects/Dev/Git_repos/TFM_repo/Collab_Dev/darknet/data/obj.names
backup = /mydrive/ANE_RAUL/TFM_IA/Collab_Dev/training
```

Figura 18 - Contenido archivo obj.data

```
saturation = 1.5    [yolo]            [yolo]            [yolo]
exposure = 1.5      mask = 0,1,2      mask = 3,4,5      mask = 6,7,8
hue=.1              anchors = 12, 16, anchors = 12, 16, anchors = 12, 16,
                    classes=4 ★       classes=4 ★       classes=4 ★
learning_rate=0.001 num=9             num=9             num=9
burn_in=1000
max_batches = 8000★
policy=steps
steps=6400,7200 ★
scales=.1,.1
```

Figura 17 - Modificaciones archivo yolov4-custom.cfg

c. **obj.data** que contendrá información sobre el número de clases a clasificar y las rutas a los archivos de entrenamiento (*train.txt*), validación (*test.txt*), al archivo que contiene los nombres de las clases (*obj.names*) y a la carpeta *training* en la que se irán guardando los pesos del entrenamiento cada 1000 iteraciones.

d. **obj.names**, deberá contener los nombres de las clases (uno por fila) en el mismo orden que hemos utilizado en LabelStudio para etiquetar los objetos, que serán: Mando TV, Pelota, Planta y Termo, en se orden.

e. **process.py**, se trata del script que generará los archivos de *train.txt* y *test.txt* incluyendo de forma aleatoria en cada uno de ellos el porcentaje de imágenes del dataset deseado, en nuestro caso se destinará un 90% de las mismas a entrenamiento, y un 10% a testeo.

5. Modificar el archivo **Makefile** para habilitar OpenCV, GPU, CUDNN, CUDNN_HALF y LIBSO, para ello se utilizarán los comandos de la siguiente figura.

```
%cd darknet/
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
!sed -i 's/CUDNN_HALF=0/CUDNN_HALF=1/' Makefile
!sed -i 's/LIBSO=0/LIBSO=1/' Makefile
```

Figura 19 - Comandos configuración archivo Makefile

6. Una vez ya configurado el archivo Makefile, construimos la darknet con el comando *¡make*.

7. Una vez construida la red darknet, ya podremos copiar los archivos generados en paso 4 a la carpeta CollabDev/darknet/data, a excepción de **process.py** y **yolov4-custom.cfg.** Este último se copiará en el directorio CollabDev/darknet/cfg, mientras que process.py se copiará directamente en la carpeta darknet.
8. Ejecutar el script *process.py* para generar los archivos *train.txt* y *test.txt.*
9. Descargar los pesos pre-entrenados de YOLOv4 desde el repositorio de darknet de AlexeyAB [18] para de esta forma evitar empezar el entrenamiento desde cero.
10. Lanzar entrenamiento del modelo llamando al script *darknet.py* seguido de una serie de parámetros o flags que se detallarán a continuación:

```
!./darknet detector train data/obj.data cfg/yolov4-custom.cfg yolov4.conv.137 -dont_show -map
```

Figura 20 - Parámetros entrenamiento modelo YOLOv4

El parámetro '*detector*' indica que se cargará la red neuronal pasada como parámetro y con '*train*' indicamos que queremos entrenarla. El tercer (*obj.data*) y cuarto (*yolov4-cutom.cfg*) parámetro constituyen los ficheros de configuración de par nuestra red neuronal, y el último parámetro (*yolov4.conv.137*) hace referencia a los pesos pre entrenados con los que se iniciará el entrenamiento del modelo.

Por otro lado, tenemos el flag '*-dont_show*' para deshabilitar que el programa muestre las gráficas de error a lo largo del entrenamiento, ya que en GoogleCollab no se pueden mostrar imágenes. Y, por último, el flag '*-map*' que permite que se lleve a cabo una evaluación del entrenamiento aplicando el archivo *test.txt* cada X número de iteraciones y de esta forma calcular el mAP del modelo entrenado hasta el momento.

11. En caso de que el entrenamiento falle y se quede a medias, el hecho de que cada 1000 iteraciones se guarden los pesos del entrenamiento en la carpeta de training, permite reiniciar el entrenamiento desde dicho punto, lo que constituye una ventaja importante debido a los largos periodos de tiempo que lleva entrenar el modelo que pueden extenderse incluso varios días. Para retomar el entrenamiento habrá que ejecutar el mismo comando que en el paso 10, únicamente cambiando los pesos pre entrenados (yolov4.conv.137) por los que se hayan guardado de la última iteración antes de caerse el entrenamiento.

En el apartado siguiente se expondrán los distintos entrenamientos realizados a lo largo del proyecto hasta conseguir un modelo que daba unos valores de precisión cercanos al 100%.

4.3 Errores entrenamiento

Durante el desarrollo del proyecto se realizaron un total de 3 entrenamientos distintos del modelo, ya que en los tres primeros se detectaron diversos errores que podrías estar comprometiendo el resultado final. A continuación, se detallan estos res entrenamientos:

- Para el **primer** entrenamiento del modelo realizado, ya con los definitivos cuatro objetos a clasificar (MandoTV, Pelota, Planta y Termo) se genera un dataset con todas las imágenes captadas en un **mismo entorno**, con

una iluminación tenue y sin muchas variaciones tanto perspectiva como en distancia de los objetos en las imágenes adquiridas. Todo esto unido a que los objetos siempre aparecen sobre el mismo fondo en todas las imágenes de este entrenamiento, resulta en una clasificación pobre que hace que el modelo sea incapaz de reconocer los objetos cuando aparecen sobre un fondo más claro o a una mayor distancia del dron.

- Para el **segundo** entrenamiento y teniendo en cuenta lo aprendido en los dos anteriores, obtenemos el dataset de forma que la mitad de imágenes de cada objeto sean capturadas en un entorno iluminado y de fondo claro, mientras que la otra mitad lo sea sobre un fondo oscuro con menor nivel de iluminación, estos dos ambientes aparecen reflejados en la Figura 10. Además, se trata de mover conscientemente el dron durante los vuelos de adquisición de imágenes para tener capturas de los objetos con la mayor variedad de perspectivas y distancias posibles.

  Testeando dicho entrenamiento, nos damos cuenta de que bien cuando los objetos aparecen solapados con otros elementos, o bien cuando sólo aparece una parte de ellos en plano, el sistema no es capaz de reconocerlos, y no sólo eso si no que en ocasiones un objeto aparece clasificado como otro distinto, por lo que se decide revisar el etiquetado. Revisando el etiquetado salta a la vista que había **algunas etiquetas que no se correspondían con el objeto**, como se puede observar en la Figura 21, por lo que no queda otra que revisar todo el etiquetado y reentrenar el modelo de nuevo.

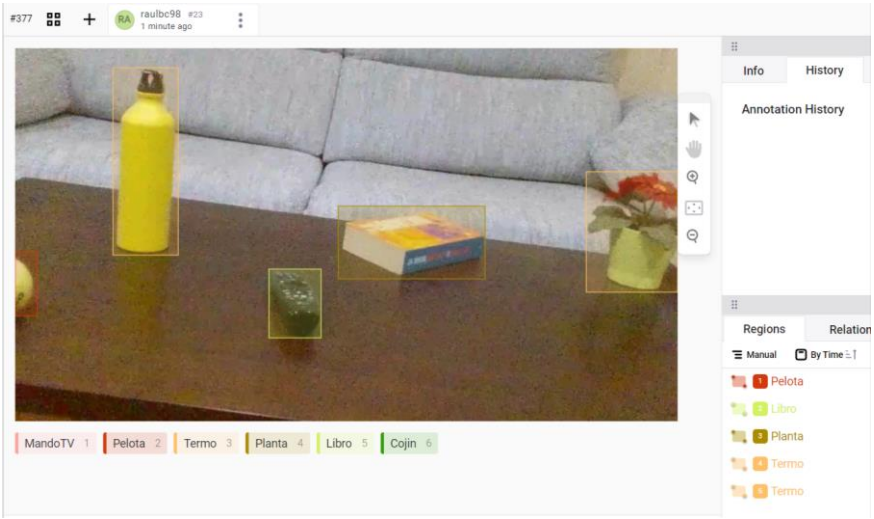

Figura 21 - Error en el etiquetado

- Para el **tercer** y último entrenamiento, no sólo se revisarán individualmente cada una de las etiquetas de todas las imágenes que componen el dataset, si no que, además, se sustituirán algunas de estas imágenes por otras en las que aparecen **fragmentos** entrecortados de los **objetos** (como en la *Figura 11*), para de esta forma, preparar al modelo para clasificar también a los objetos en las circunstancias de solape comentadas en el apartado anterior.

5 Arquitectura de control

En este apartado se expondrán las distintas etapas del desarrollo del proyecto llevadas a cabo una vez que el modelo ya ha sido entrenado y presenta unos resultados de clasificación correctos.

Estas etapas incluirán desde la importación del modelo a nuestro PC local donde se ejecutará el código final de control del dron, hasta dicho código que vendrá acompañado de una interfaz de usuario que lo hará más intuitivo y fácil de manejar para el usuario.

5.1 Importar modelo al PC local

El primer paso para poder ejecutar el modelo entrenado directamente en el dron en tiempo real, es importar dicho modelo y ser capaces de compilarlo y ejecutarlo en nuestro PC.

Para ello será necesario compilar darknet en el PC, que supone una tarea más complicada de lo que puede parecer en un principio debido a la cantidad de librerías y dependencias que existen entre ellas y que debemos instalar previamente en nuestro PC. Los pasos a seguir para compilar darknet aparecen reflejados en el apartado *10.2*.

5.2 Arquitectura de control software

En el momento de establecer una arquitectura para este proyecto se tienen en cuenta varias premisas principales como son: las distintas conexiones que se pueden establecer entre el PC y el dron (status, control y stream), así como la necesidad de que todo se ejecute lo más rápido posible con el fin de mantener un control en tiempo real del dron.

Teniendo en cuenta todo esto, se decide utilizar una arquitectura basada en **4 hilos** secundarios que cuelgan de un hilo padre principal y que se ejecutarán en paralelo según las necesidades en cada momento.

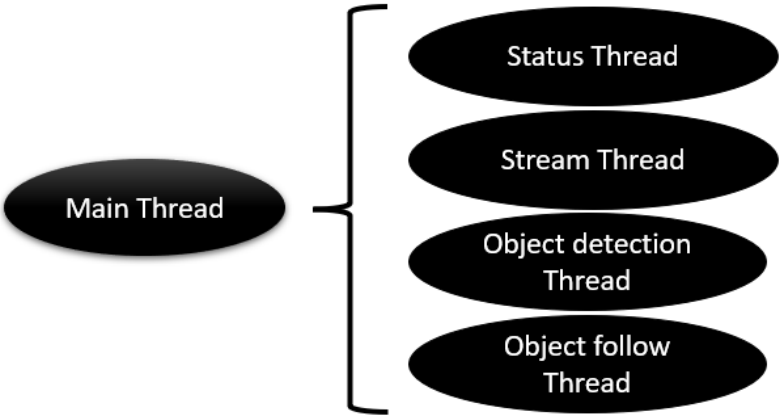

Figura 22 - Arquitectura hilos del software desarrollado

A continuación, de detallarán más en profundidad estos módulos principales:

5.2.1 Main

Se encarga lanzar la **ejecución continua** de la interfaz gráfica (**HMI**) desde la que el usuario enviará los comandos necesarios para realizar las acciones deseadas mediante los procesos definidos en 'My_Tello.py'. Además, gracias a procesos definidos en 'My_Tello.py' también se recibirá el feedback, ya sea en forma de confirmación de comando recibido, stream o estado de algunos de los parámetros del dron.

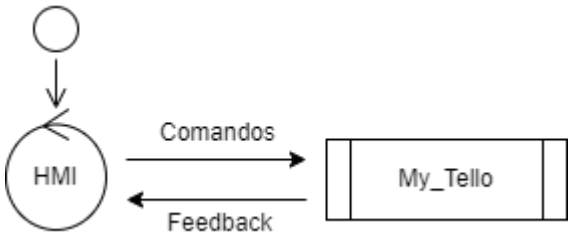

Figura 23 - Diagrama flujo del Main del programa

5.2.2 Status

El primer hilo del que vamos a hablar, y a su vez el más sencillo, será el encargado de ejecutar en segundo plano la actualización de los datos del dron a cada segundo, a través de la comunicación establecida con el dron explicada anteriormente en el *3.1.1*.

Para iniciar este hilo, el usuario tendrá que clicar en el botón '***Update status***' del HMI, para de esta forma lanzar su ejecución continua que actualizará los valores que se muestran en el HMI, mediante las funciones que se indican en el siguiente diagrama.

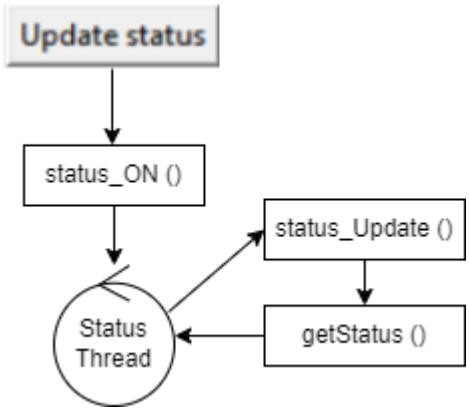

Figura 24 - Diagrama de flujo del hilo de actualización de estado del dron

5.2.3 Stream

Por otro lado, en cuanto el usuario haga clic en el botón '***Iniciar streaming***' se enviará el comando '*streamon*' al dron y se verificará si el vehículo lo ha recibido y ejecutado correctamente, todo ello a través de la función ***streamON()***. En caso de que el comandado haya sido exitoso y tengamos streaming en tiempo real desde la cámara del dron, se lanzará la ejecución del hilo de streaming ***StreamThread*** mediante la función **updateStream()**.

Este hilo ejecutará continuamente la función ***getStream()*** hasta que es usuario decida hacer clic en '***Cerrar streaming***'. En dicha función, se capturan los

frames de vídeo provenientes de la comunicación establecida con el dron a través del puerto 11111 explicada en el apartado *3.1.1* y, además, se lanzará la ejecución en paralelo del hilo encargado de llevar a cabo la detección de objetos ***ObjectDetectionThread*** del que daremos más detalles en el siguiente apartado.

En cada iteración del bucle de la función ***getStream()*** se procesará un frame proveniente del dron. En caso de que el usuario habilite la opción '***Guardar frames'***, dicho frame se guardará en la carpeta correspondiente y con su nombre predefinido por el usuario en el selector situado a la izquierda de dicho control. Seguidamente, si el '**Modo rastreo**' ha sido habilitado por el usuario y el objeto seleccionado para ser buscado es distinto de 'Todos', se habilitará el flag *ObjectDetectThreadEvent*, que cómo veremos en el siguiente capítulo habilita la detección de objetos en el hilo correspondiente para detectar en dicho frame el objeto seleccionado.

5.3 Interfaz gráfica

Con la finalidad de facilitar tanto la monitorización como el control del dron, se decide desarrollar una **interfaz de usuario (UI o HMI)** sencilla como podemos ver en la Figura 25 pero que acabará siendo de gran utilidad, cuyas funcionalidades principales se enumeran en este apartado y se expondrán de forma más detallada en el siguiente apartado.

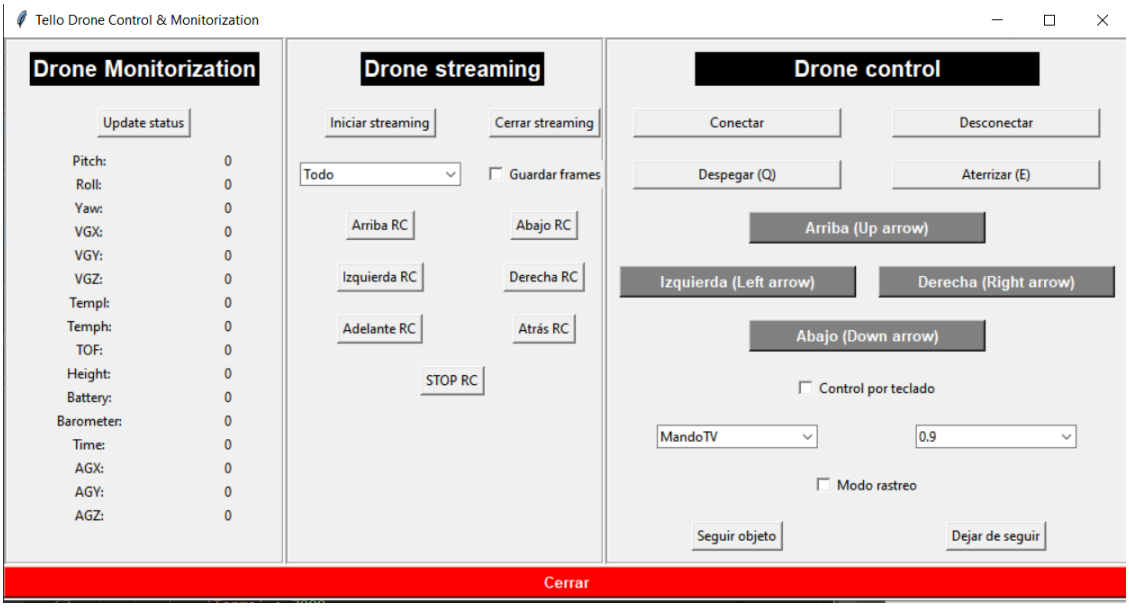

Figura 25 - Interfaz gráfica de control y monitorización del dron

Como podemos observar, dentro de la interfaz de usuario observamos **3 módulos** principales (monitorización, streaming y control), así como un botón de '***Cerrar'*** en la parte inferior.

Dentro del módulo de **monitorización**, destaca la presencia de un único botón llamado '***Update status***' con el que el usuario iniciará la ejecución continua del hilo encargado de actualizar los datos del estado dron que aparecen justo debajo de dicho botón, y cuya adquisición de detalla en el apartado *5.2.2*.

Figura 26 - Módulo de monitorización del dron del UI

En cuanto al módulo de **streaming**, cómo su nombre indica se encargará de iniciar o cerrar la transmisión de imágenes desde el dron en tiempo real a través de sus dos controles principales: '***Iniciar streaming***' y '***Cerrar streaming***', con los cuales el usuario lanzará o parará la ejecución del hilo de adquisición de vídeo del dron según sus necesidades.

Además, dentro de este mismo módulo, encontramos la opción de habilitar el **guardado de los frames** del vídeo del dron que serán guardados de acuerdo con la categoría elegida en el selector de la izquierda.

Por último, dentro del módulo de streaming también encontraremos los botones de **control RC** del dron que permiten un control más fluido del dron (sin necesidad de esperar la respuesta de comando recibido por su parte) y que el usuario empleará para desplazar al dron durante los vuelos de captura de imágenes para el dataset.

Figura 27 - Módulo de streaming del UI

Por lo que respecta al último módulo de la interfaz, el de **control** del dron, es el más completo de los tres. Lo primero que encontramos son los botones de '**Conectar**' y '**Desconectar**' con los que el usuario será capaz de iniciar o cerrar la conexión entre el PC y el dron. También encontramos los botones de control básico del dron (despegar, aterrizar, arriba, abajo, izquierda y derecha) mediante los cuales el usuario al clicar en ellos enviará al dron los comandos correspondientes para realizar el movimiento seleccionado. Este control también se podrá realizar por teclado, habilitando la opción de '**Control por teclado**' (las teclas asignadas a cada movimiento aparecen en el botón de la interfaz de usuario).

Dentro del módulo de control, también encontraremos la sección en torno a la que se desarrolla este trabajo, que será la encargada de controlar las secciones de código dedicadas al rastreo y seguimiento del objeto seleccionado en el selector de la izquierda, siempre y cuando la precisión de la clasificación de dicho objeto sea igual o mayor a la establecida en el selector de la derecha. Para iniciar la detección de objetos, el usuario deberá habilitar la opción '**Modo rastreo**', de esta forma el programa comenzará a mostrar en el streaming el objeto detectado, pero no lo seguirá. Para que el dron siga al objeto, el usuario tendrá que clica en el botón '**Seguir objeto**', o bien en '**Dejar de seguir**' para finalizar el seguimiento.

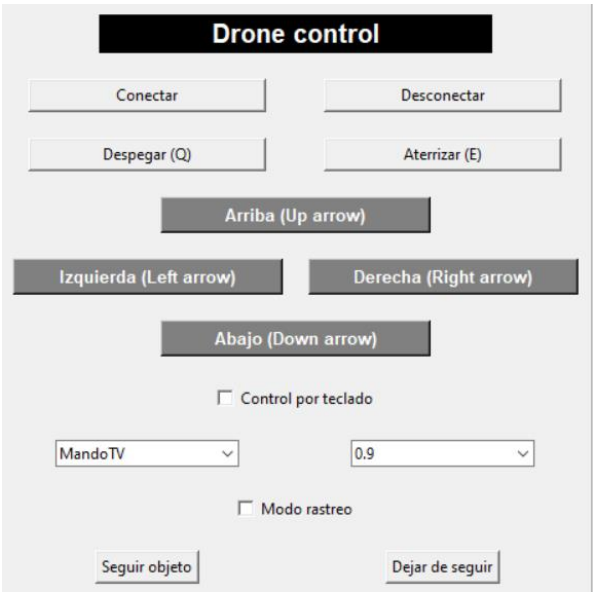

Figura 28 - Módulo de control del UI

Por último, pero no menos importante, encontramos el botón de '**Cerrar**' en la parte inferior de la interfaz de usuario. Clicando este botón, el usuario podrá realizar un cierre controlado del programa mediante el envío del comando '*emergency*' al dron que parará sus motores al instante y el cierre de todos los sockets de comunicación previamente establecidos.

Figura 29 - Botón de Cerrar del UI

Figura 30 - Diagrama de flujo del hilo de streaming

5.3.1 Detección de objetos

Tal y como hemos visto en el apartado anterior, el hilo encargado de aplicar el modelo de detección en cada uno de los frames detectados se comenzará a ejecutar continuamente, a través de la función ***detectObject()***.

Este hilo se iniciará en cuanto lo haga el hilo de streaming tal y como aparece en el diagrama anterior. Además, también desde el hilo de streaming se habilitará el evento *ObjectDetectThreadEvent* en caso de que el usuario haya seleccionado un objeto concreto a buscar y se lanzará la ejecución de la función ***TrackObject()*** que se encargará de aplicar el modelo entrenado para detectar el objeto seleccionado. En caso de detectarse un objeto, se obtendrá el centro geométrico del 'box' de detección que se mostrará sobre la imagen del streaming en tiempo real.

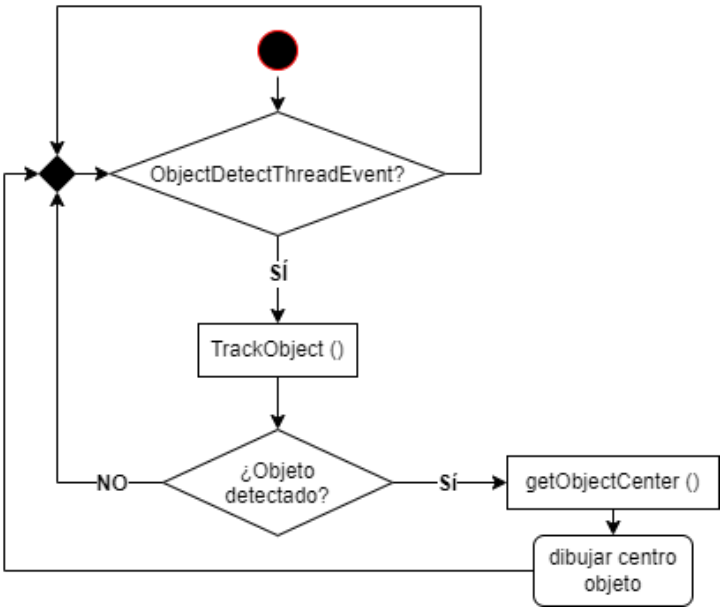

Figura 31 - Diagrama de flujo del hilo de detección de objetos

5.3.2 Búsqueda y seguimiento de objetos

Por lo que respecta al hilo encargado del seguimiento del objeto seleccionado en tiempo real, éste se iniciará o parará con la acción directa del usuario sobre los botones 'Seguir objeto' o 'Dejar de seguir' respectivamente, y ejecutará la función ***followObject()***, cuyo funcionamiento queda representado en el diagrama de flujo de la *Figura 35*.

Figura 32 - Controles de seguimiento de objeto del UI

El funcionamiento de dicha función podemos encontrar tres modos de funcionamiento: inicialización, búsqueda y seguimiento. Lo primero que ocurrirá, como su nombre indica, será la **inicialización**, en la que al iniciar el hilo se realizará el envío del comando de despegue al dron, seguido de otro comando de ascenso para alcanzar una altura aproximada de 1 metro. A continuación, el código entrará en modo **búsqueda**, dónde se enviarán periódicamente comandos para que el dron gire 30 grados sobre sí mismo en sentido horario hasta que se detecte el objeto seleccionado, momento en el que

se pasará a modo **seguimiento** en el que encontramos dos secciones. La primera de ellas en la que se dividirá el área de visión de la cámara del dron en diferentes zonas y determinará a través de un algoritmo en cuál de ellas encuentra el dron en función de la posición del objeto con respecto al centro de la imagen, y en la segunda se decidirá el comando a enviar al dron en función de la zona y cuadrante en la que se encuentre el objeto, con el fin de centrarlo en la cámara del dron.

Cabe destacar que si durante el seguimiento del objeto, se dejase de detectar al mismo, el código volvería al modo de búsqueda y el dron empezaría a girar de nuevo hasta encontrar el objeto.

5.3.2.1 Obtención de zona del objeto

Para determinar la posición del objeto con respecto al centro de la imagen, se dividirá la imagen en una zona central a la que llamaremos **ZonaA**, y cuatro cuadrantes, cada uno de ellos a su vez dividido en 3 zonas: **ZonaB1** (horizontal), **ZonaB2** (vertical) y **ZonaB3** (esquina), tal y como podemos ver en la *Figura 33.*

Cabe destacar que la amplitud tanto horizontal como vertical de la **ZonaA** es parametrizable desde la interfaz de usuario a través de las variables *Htrack* y *Wtrack.*

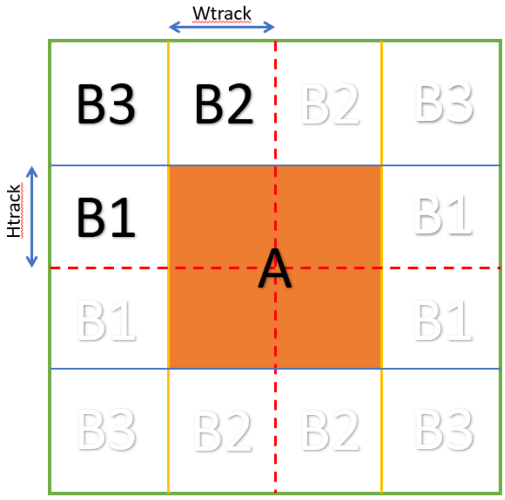

Figura 33 - Definición de zonas en la imagen de la cámara del dron

Para determinar en cuál de las zonas definidas anteriormente se encuentra el objeto detectado, recurriremos al output dado por la función ***getObjectCenter()*** del hilo de detección de objetos que nos proporción al posición relativa del centro geométrico de la caja de detección del objeto con respecto al centro de la imagen.

5.3.2.2 Comandado del dron según zona

Dependiendo de la zona en la que se encuentre el objeto detectado, el dron se tendrá que comportar de una u otra forma, con el objetivo de mantener dicho objeto lo más centrado posible, para evitar perderlo de vista y, además, a una distancia contante que para este proyecto se define en 80-100 centímetros. De esta forma el comando del vehículo según zona quedará de la siguiente manera:

- **ZonaA**: es considerada la zona central de la imagen, por este motivo, si el objeto se encuentra en ella, el único movimiento que hará el dron será el de alejarse o acercarse para mantener la distancia definida con el objeto. El procedimiento empleado para calcular dicha distancia será

utilizar el área del 'box' de detección específico de cada objeto medido experimentalmente manteniendo el dron a unos 80 centímetros de cada objeto. De esta forma se definen unos rangos de áreas que determinarán que el dron se encuentra a la distancia correcta del objeto, en caso contrario, si el área se excede de dicho rango significará que el objeto está demasiado cerca y se enviará el comando '*backward 20*' para que se desplace hacia detrás alejándose del objeto. Y, en caso de que el área sea menor que el rango, el objeto estará demasiado lejos por lo que habrá que desplazar al dron hacia delante con el comando '*forward 20'*.

Tabla 2 - Áreas box de detección a 80 cm

Objeto	**Área límite**
Mando TV	1000
Pelota	1500
Planta	12000
Termo	7000

- **ZonaB1**. Si el objeto se encuentra en alguna de las zonas B1 será sinónimo de que está desplazado hacia un lado, izquierda o derecha. En caso de que se esté a la izquierda, se enviará el comando '*ccw 20'*, mientras que si está a la derecha se enviará '*cw 20*'.
- **ZonaB2**. El hecho de que el objeto se encuentre en una de las zonas B2 significa que está ´por encima o por debajo de lo que debería. En caso de estar por encima, se enviará el comando *'up 20'*, y en el caso contrario, '*down 20'*.
- **ZonaB3**. Por último, esta zona es más compleja, ya que indica que el objeto se encuentra en una de las esquinas de la imagen, es decir, tanto más arriba o abajo, como más a la izquierda o derecha de cómo debería estar. Para estos casos en primer lugar se realizará un desplazamiento lateral como los que se haría si estuviese en zonaB1 y, a continuación, en caso de seguir estando en zona B3 se realizará un desplazamiento vertical como si se tratase de la zona B2. Si después de esto, el objeto sigue encontrándose en zona B3 volveremos al desplazamiento lateral, y así sucesivamente.

En la siguiente figura se representan gráficamente los movimientos que realizará el dron según la zona en la que se encuentre el objeto a detectar, tal y como se ha explicado anteriormente.

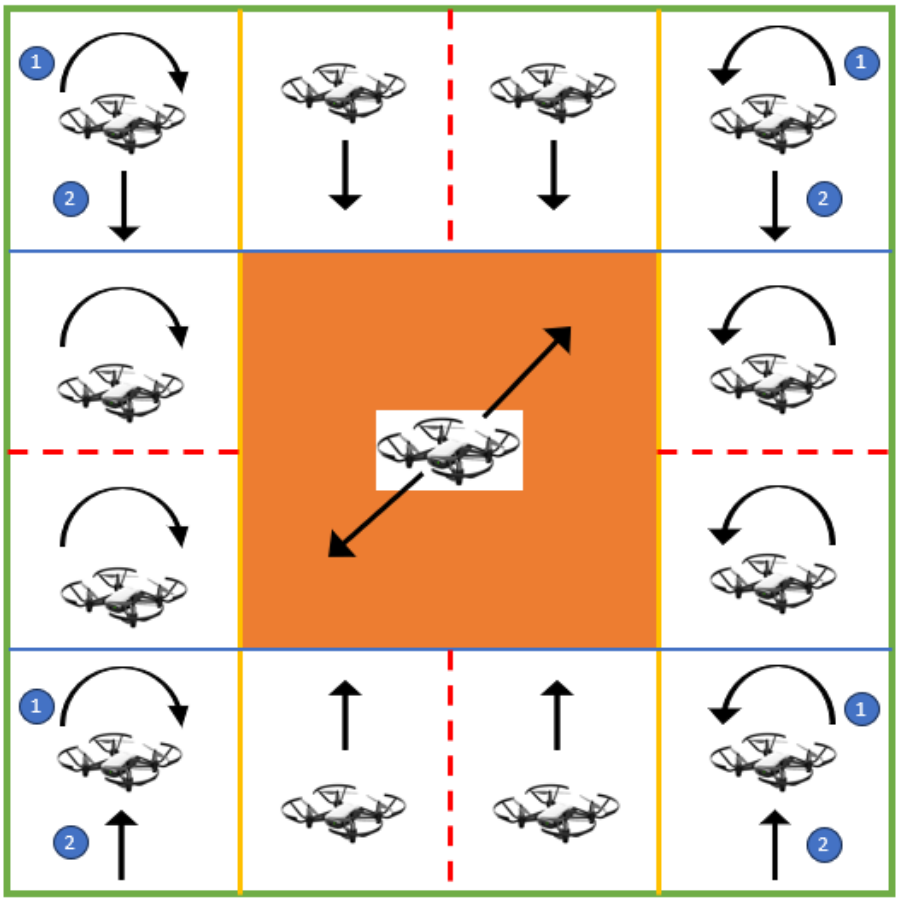

Figura 34 - Movimientos del dron según la zona en la que esté el objeto

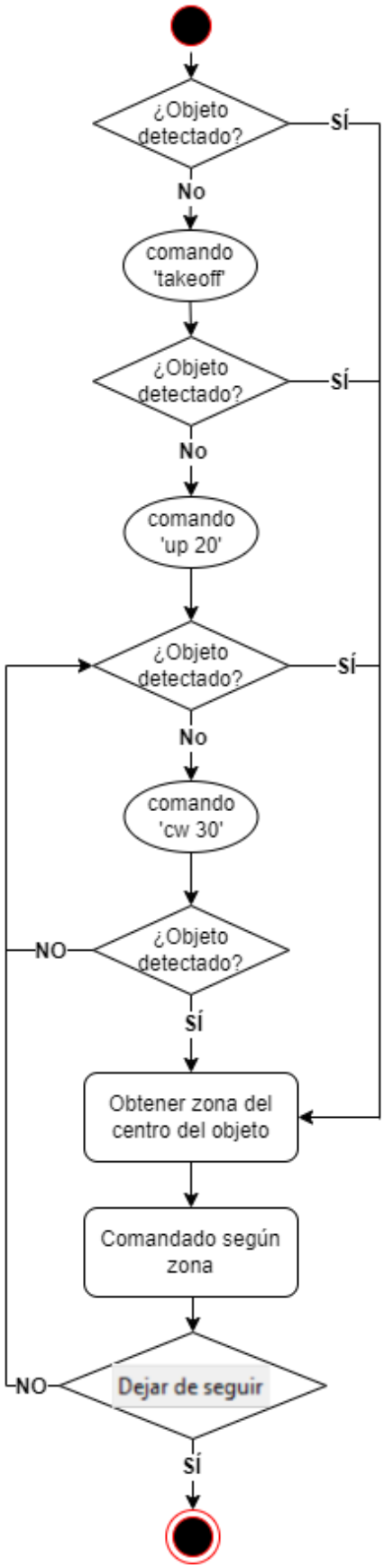

Figura 35 - Diagrama de flujo del hilo de detección y seguimiento de objetos

6 Resultados

A lo largo de este capítulo se expondrán los resultados obtenidos en las diferentes etapas o hitos que se han llevados a cabo durante el desarrollo del proyecto. Estas etapas son las siguientes:

- Mejorar la estabilidad del dron durante el vuelo.
- Entrenar un modelo que proporcione una alta precisión en la clasificación de los objetos definidos.
- Implementar el modelo entrenado en el código y que el dron sea capaz de detectar y seguir los objetos.

6.1 Estabilidad del dron

El primer problema importante con el que lidiar es el de la estabilidad del dron provocado por el sistema de auto posicionamiento del dron y las características de la superficie que sobrevuela.

Tras realizar varias pruebas utilizando distintos objetos, patrones, iluminaciones y demás, se consigue una combinación que mejora considerablemente la estabilidad del dron. Esta combinación consistirá en cubrir el suelo con un material mate o que reduzca los reflejos, sobre el que pondremos distintos patrones como los que podemos observar en la *Figura 36*, todo ello iluminado únicamente con luz artificial, pero sin excederse.

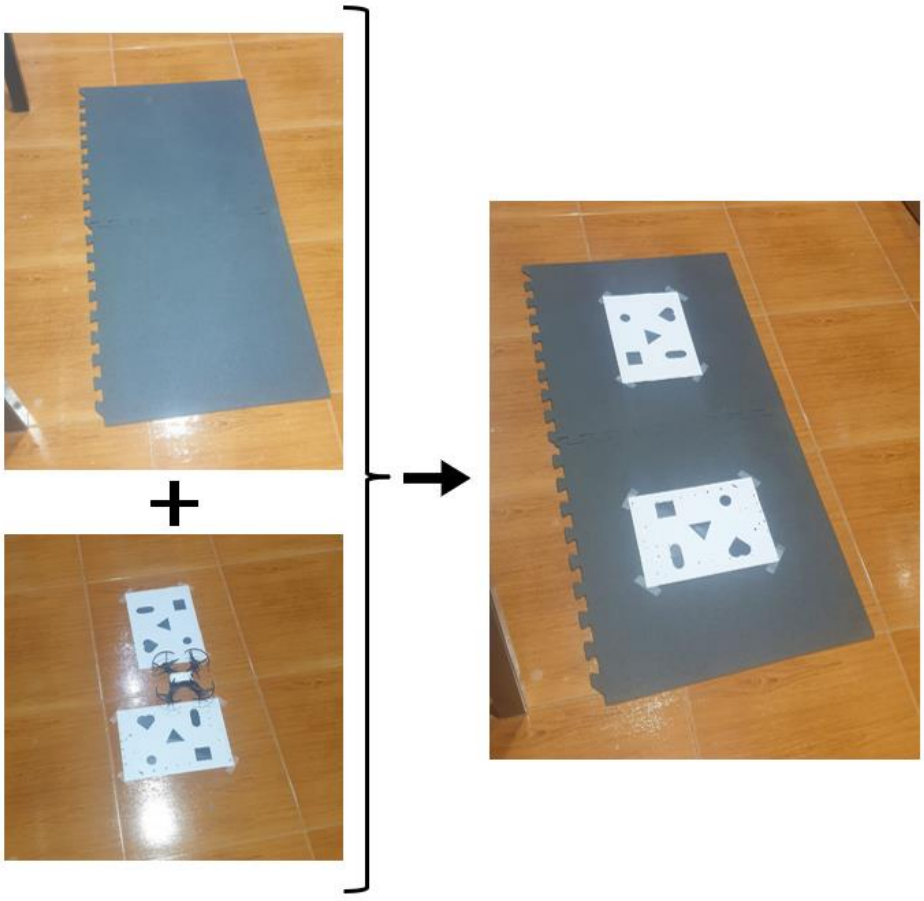

Figura 36 - Configuración superfície de vuelo para mayor estabilidad

6.2 Modelo entrenado de YOLOv4

Tal y como se detalla en el apartado 4.3 se llevan a cabo varios entrenamientos hasta que, en el cuarto entrenamiento, se obtiene un modelo que presenta unos resultados de clasificación que rozan la perfección a partir de las 6000 iteraciones de entrenamiento, tal y cómo se puede observar en la siguiente tabla, atendiendo a valores de precisión, mAP, recall, IoU y ratios de error y de acierto.

Tabla 3 - Resultados para los pesos de los distintos momentos del entrenamiento

Iteraciones	Precisión	Recall	F1-score	IoU	mAP@0.5
1000	0,92	0,99	0,96	72,33	0,980898
2000	0,98	1,00	0,99	85,18	0,999961
3000	0,98	1,00	0,99	87,59	0,999961
4000	0,97	1,00	0,99	88,98	0,999864
5000	0,97	1,00	0,99	89,26	0,999576
6000	0,97	1,00	0,99	90,17	0,999961
7000	0,97	1,00	0,99	91,33	0,999961
8000	0,97	1,00	0,99	91,36	0,999961

6.2.1 Objeto Mando TV

Como podemos observar en las etiquetas que aparecen en la siguiente figura, en la que vemos la detección del objeto Mando TV en dos entornos distintos, la precisión es del 100% para estas dos imágenes.

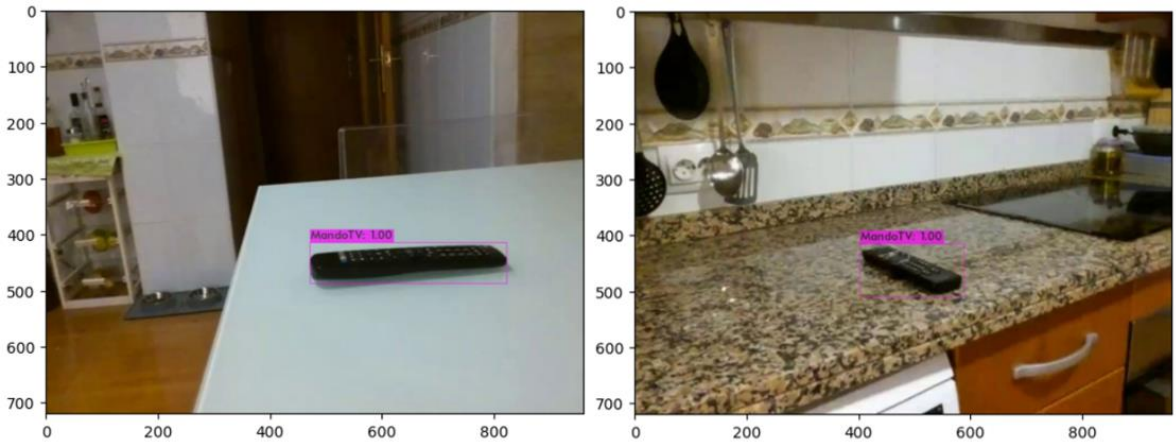

Figura 37 - Ejemplo detección mando en varios entornos

6.2.2 Objeto Pelota

Para el caso del objeto pelota, al igual que en el caso anterior, los resultados en cuanto a precisión en las imágenes de la figura son del 100% de precisión.

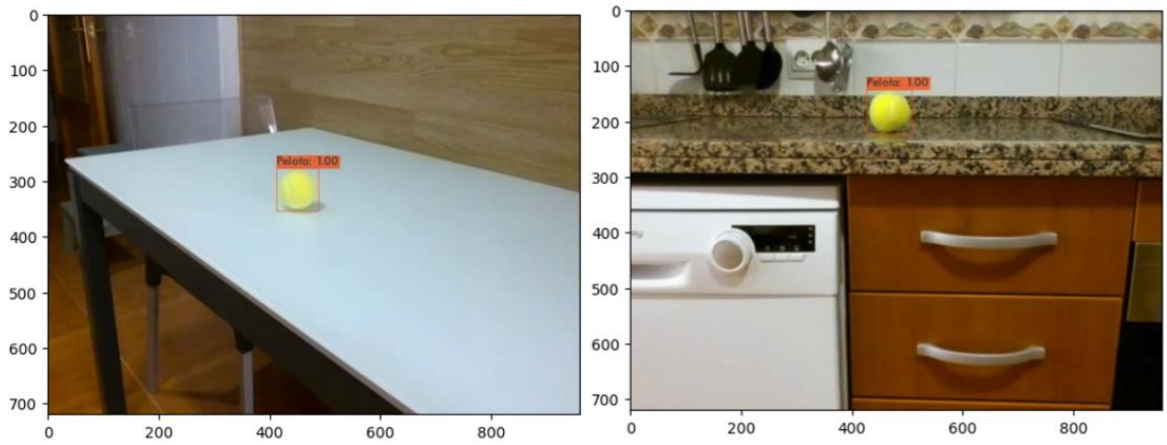

Figura 38 - Ejemplo detección pelota en varios entornos

6.2.3 Objeto Planta

En cuanto a las imágenes d detección del objeto Planta, la detección es igualmente inmejorable con un 100% de precisión.

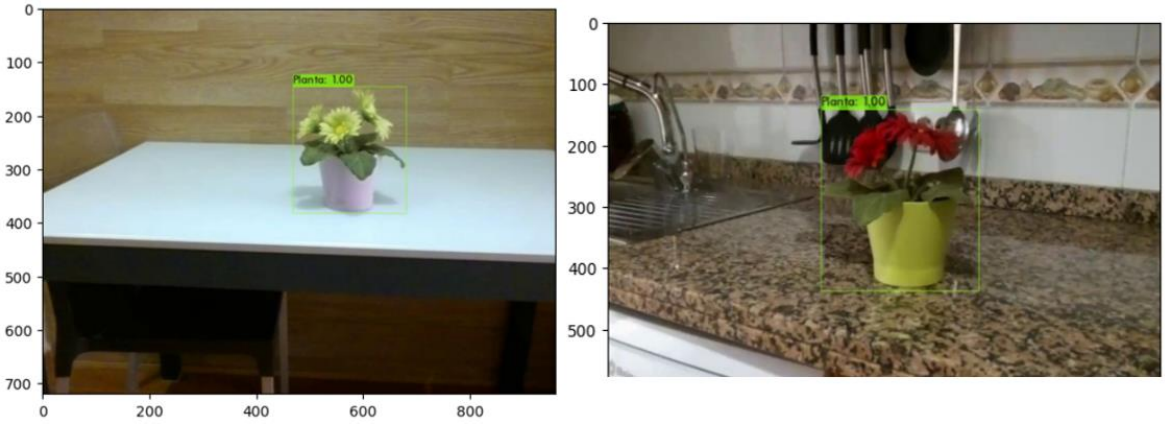

Figura 39 - Ejemplo detección planta en varios entornos

6.2.4 Objeto termo

Por lo que respecta al objeto termo, los resultados que arroja la detección de objetos también son del 100% de precisión en ambos entornos.

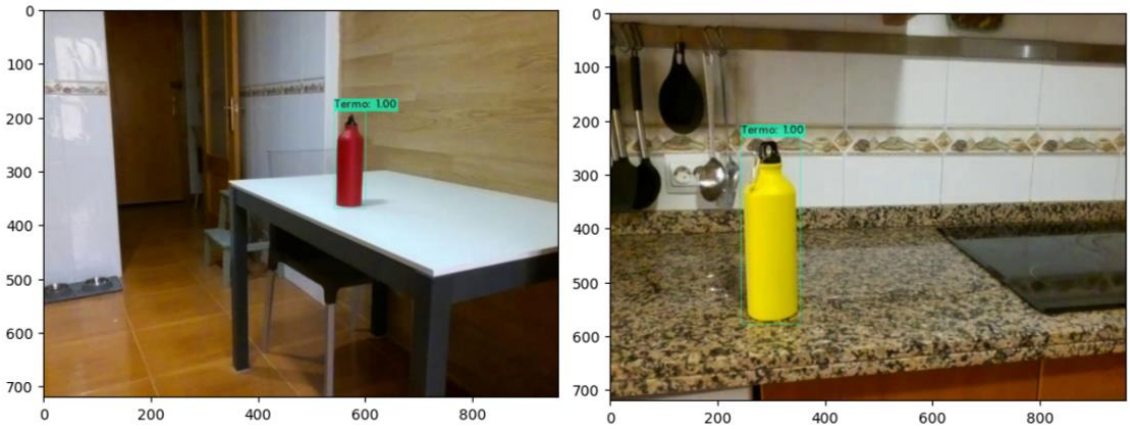

Figura 40 - Ejemplo detección de termo en varios entornos

6.2.5 Todos los objetos

En la siguiente figura observamos como los cuatro objetos son detectados con un 100% de precisión en ambos entornos.

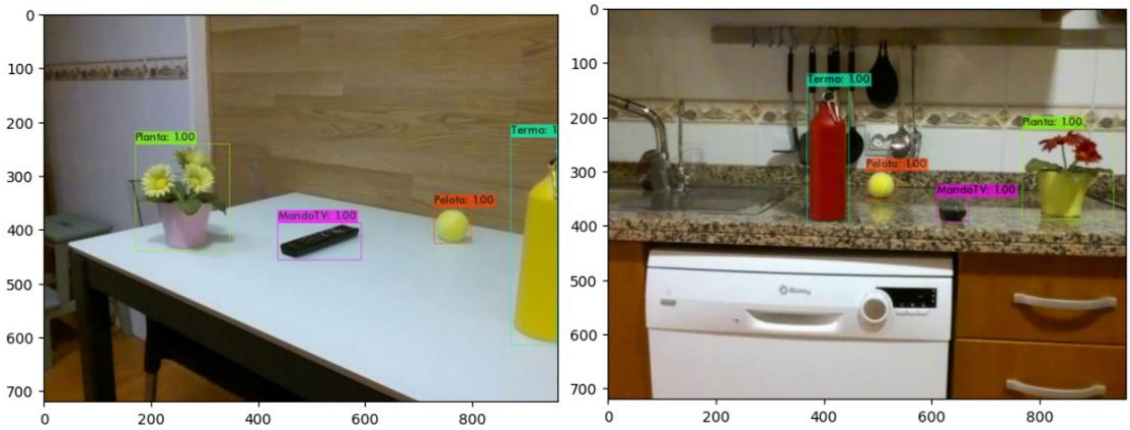

Figura 41 - Ejemplo de detección de todos los objetos en varios entornos

6.2.6 Objetos solapados o recortados

Este se trata de un caso especial en el que se busca que los objetos aparezcan solapados con otros como ocurre en la imagen de la izquierda de la figura con el mando, o también que sólo aparezca una parte de los mismos como sucede con el termo de las otras dos imágenes.

En ambos casos los resultados de precisión también son del 100%.

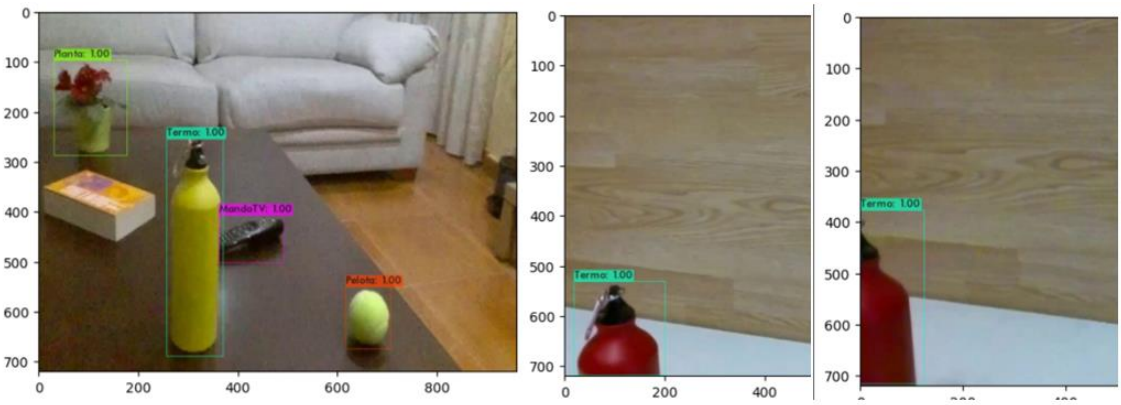

Figura 42 - Ejemplo de detección de objetos solapados o recortados

6.3 Vuelos de reconocimiento del dron

A través del siguiente enlace se puede acceder a una lista de reproducción de Youtube donde se encuentran los vídeos de algunos vuelos del dron reconociendo objetos.

Lista reproducción Youtube

7 Conclusiones

Como conclusión principal del proyecto, podemos decir que el modelo de detección de objetos YOLO, concretamente su versión 4, da un rendimiento de detección de objetos con una precisión superior al 95% e incluso en algunos casos cercana al 100%, dependiendo de la visión que se tenga del objeto a detectar para su aplicación en sistemas de detección en tiempo real como es el caso del dron utilizado en este trabajo.

Sin embargo, el proceso de conseguir estabilizar y manejar el dron en espacios reducidos como son el interior de una casa complica el trabajo, pero a base de prueba y error se consigue la estabilidad deseada.

Por último, el sistema de aproximación del dron a los objetos utilizando el área del 'box' de detección del mismo como variable que determina la distancia entre el dron y dicho objeto tiene sus fallas en determinados casos donde, por ejemplo, la perspectiva no permita observar al objeto en su totalidad, siendo el 'box' más reducido de lo que debería ser.

8 Trabajos futuros

Mejorar la aproximación a objetos mediante el área del box, ya que en caso de que haya algún solape que permita visualizar sólo una parte de ellos o algún cambio de perspectiva que los haga más grandes o pequeños de lo estipulado en las mediciones de área, puede conllevar que el dron no mantenga correctamente la distancia establecida con el objeto.

Además, también existiría la posibilidad de implementar un método de detección de objetos más actualizado como puedan ser las nuevas versiones de YOLO, cuya última versión, la novena, fue publicada en febrero de este mismo año.

9 Bibliografía

[1] Prompts para IA, «Visión por computadora en drones: aplicaciones, beneficios y mejores prácticas,» [En línea]. Available: https://prompt.uno/vision-por-computadora/vision-por-computadora-en-drones/.

[2] EagleDron, «Los 10 usos principales de los Drones,» [En línea]. Available: https://www.eagledron.es/los-10-usos-principales-de-los-drones/.

[3] Xataka, «Los drones de Amazon ahora pueden volar más lejos.,» [En línea]. Available: https://www.xataka.com/transporte/drones-amazon-ahora-pueden-volar-lejos-donde-no-alcanza-vista-para-ser-exactos.

[4] El Español, «El nuevo dron de reparto de Google puede llevar paquetes de hasta casi 3 kilos a más de 100 km/h,» [En línea]. Available: https://www.elespanol.com/omicrono/hardware/20240117/nuevo-dron-reparto-google-puede-llevar-paquetes-kilos-kmh/825667861_0.html.

[5] R. Cheng, «A survey: Comparison between Convolutional,» *Journal of Physics: Conference,* 2020.

[6] Roboflow, «What is YOLOv4? A Detailed Breakdown,» [En línea]. Available: https://blog.roboflow.com/a-thorough-breakdown-of-yolov4/.

[7] Medium, «YOLO- A State-of-the-Art object detection model,» [En línea]. Available: https://medium.com/@khwabkalra1/yolo-you-only-look-onc-523b01ec4f4d.

[8] J. Redmon, S. Divvala, R. Girshick y A. Farhadi, «You Only Look Once: Unified, Real-Time Object Detection,» 2016.

[9] J. Redmon y A. Farhadi, «YOLO9000: better, Faster, Stronger,» 2016.

[10] T. Yang, X. Zhang, Z. Li, W. Zhang y J. Sun, «MetaAnchor: Learning to Detect Objects with Customized Anchors».

[11] R. Joseph y F. Ali, «YOLOv3: An Incremental Improvement».

[12] A. Bochkovsky, C.-Y. Wang y H.-Y. M. Liao, «YOLOv4: Optimal Speed and Accuracy of Object Detection,» 2020.

[13] Ryze Robotics, «Tello,» [En línea]. Available: https://www.ryzerobotics.com/es/tello.

[14] Ryze Robotics, «Tello SDK,» [En línea]. Available: https://terra-1-g.djicdn.com/2d4dce68897a46b19fc717f3576b7c6a/Tello%20 编程相关 /For%20Tello/Tello%20SDK%20Documentation%20EN_1.3_1122.pdf.

[15] Ryze Robotics, «Manual de usuario Tello,» [En línea]. Available: https://dl-cdn.ryzerobotics.com/downloads/Tello/201806mul/Tello%20User%20Manual%20V1.0_ES.pdf.

[16] Label Studio, «Label Studio: Open Source data labelling platform,» [En línea]. Available: https://labelstud.io.

[17] pjreddie, «darknet,» [En línea]. Available: https://github.com/pjreddie/darknet.

[18] AlexeyAB, «Darknet, YOLOv4,» [En línea]. Available: https://github.com/AlexeyAB/darknet.

[19] Wikipedia, «CUDA,» [En línea]. Available: https://en.wikipedia.org/wiki/CUDA.

[20] D. Shah, «Medium YOLOv4 Bag of Freebies,» [En línea]. Available: https://medium.com/visionwizard/yolov4-bag-of-freebies-dc126623fc2d.

[21] I. Pacal y D. Karaboga, «A robust real-time deep learning based automatic polyp detection system,» *Computers in Biology and Medicine,* vol. 134.

10 Anexos

10.1 Comandos

En la siguiente tabla aparecen los distintos **comandos** definidos en el software development kit (**SDK**) del dron Tello que se utilizan en este proyecto para el control del mismo.

Tabla 4 - Comandos Software Development Kit (SDK) dron Tello

Comando	Finalidad
command	Inicia la comunicación con el dron.
takeoff	Hace despegar al dron.
land	Aterriza al dron.
streamon	Inicia la transmisión de streaming de vídeo por parte del dron.
streamoff	Acaba la transmisión de streaming de vídeo por parte del dron.
emergency	Apaga todos los motores al instante
up X	Desplaza el dron X cm hacia arriba, donde X puede ir de 20 a 500.
down X	Desplaza el dron X cm hacia abajo, donde X puede ir de 20 a 500.
left X	Desplaza el dron X cm hacia la izquierda, donde X puede ir de 20 a 500.
right x	Desplaza el dron X cm hacia la derecha, donde X puede ir de 20 a 500.
forward X	Desplaza el dron X cm hacia delante, donde X puede ir de 20 a 500.
back X	Desplaza el dron X cm hacia atrás, donde X puede ir de 20 a 500.
cw X	Rota al dron X grados sobre sí mismo en sentido de las agujas del reloj, donde X puede ir de 1 a 3600 grados.
ccw X	Rota al dron X grados sobre sí mismo en sentido contrario al de las agujas del reloj, donde X puede ir de 1 a 3600 grados.
rc a b c d	Envía un comando de radio control al dron, definido por cuatro canales: • a: izquierda/derecha (-100~100). • b: adelante/atrás (-100~100). • c: arriba/abajo (-100~100). • d: rotación o guiñada (-100~100).

10.2 Despliegue

Como se ha comentado en el apartado *5.1*, compilar ser capaces de compilar Darknet en nuestro PC resulta de vital importancia para el desarrollo del proyecto, los pasos llevados a cabo para conseguirlo fueron los siguientes:

1. **Descargar** la versión más reciente de **DarkNet** desde el repositorio de GitHub de AlaexeyAB [18].
2. **Instalar Microsoft Visual Studio 2019**, junto a su paquete llamado *'Desarrollo para el escritorio con C++'*, que utilizaremos más adelante para la compilación del programa.
3. **Instalar CMake** para Windows en su versión de 64 bits, concretamente la versión 3.28.5, que servirá para generar el código que le pasaremos más adelante a Microsoft Visual Studio para compilar DarkNet.
4. Chequear que los **drivers** de la tarjeta gráfica de nuestro PC están actualizados, en nuestro caso, el PC viene dotado con una tarjeta gráfica **NVIDIA GeForce RTX 3050Ti Laptop GPU.**
5. **Instalar CUDA**, que consiste en una plataforma de computación en paralelo que gracias a incluir un compilador junto con hunas herramientas de desarrollo creadas por Nvidia, permite codificar algoritmos en sus GPUs mediante un lenguaje de programación derivado de C, llamado CUDA C. En el momento de su instalación es de vital importancia seleccionar la versión que sea compatible con nuestra GPU, para ello podemos ir a [19]. En nuestro caso, acabamos instalando la versión **11.1**.
6. **Instalar CUDnn**, una biblioteca de primitivas con aceleración de GPU que se utiliza con redes neuronales profundas. En nuestro caso instalamos la versión **8.9.7.**
7. **Instalar OpenCV_v4.5.5** y añadir al path las variables necesarias.
8. **Ejecutar** el **Makefile** en CMake para generar el archivo de compilación de DarkNet en formato de 64 bits.
9. Lanzar la **compilación** en Microsoft Visual Studio del archivo de solución generado en el paso anterior, de esta forma, obtendremos el ejecutable **darknet.exe** necesario para cargar el modelo entrenado en nuestro PC.
10. Por último, dentro del directorio darknet, copiaremos el archivo **pthreadVC2.dll** desde la carpeta 3rdparty al directorio principal, donde también quedarán el ejecutable **darknet.exe** y los pesos obtenidos en el entrenamiento en GoogleCollab (**yolov4_custom_final.weights**).

Una vez realizados estos pasos ya será posible compilar y utilizar las funciones propias de darknet en nuestro proyecto, lo que permite cargar el modelo de YOLOv4 ya entrenado en la inicialización del programa, para que de esta forma podamos procesar todas las imágenes necesarias sin necesidad de volver a cargar el modelo.