Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Sistema de control de un dron para tareas en interiores

Control system for an indoor drone

Edgar Mesa Santana

La Laguna, 03 de julio de 2019

D. **Jonay Tomás Toledo Carrillo**, con N.I.F. 78698554Y profesor Titular de Universidad adscrito al departamento de ingeniería informática y de sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A (N)

Que la presente memoria titulada:

“Sistema de control de un dron para tareas en interiores”

ha sido realizada bajo su dirección por D. **Edgar Mesa Santana**,

con N.I.F. 45897555-C.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 21 de noviembre de 2018

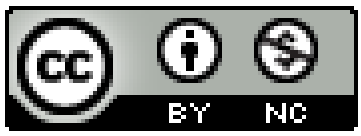
Agradecimientos

Primero agradecer a mi tutor Jonay Tomás Toledo Carrillo, porque sin su inestimable ayuda y guía en los momentos importantes.

Además quisiera agradecer a mi familia ya que sin ellos este sueño no se hubiera realizado.

Por último gracias a todos los compañeros y profesores de los que he aprendido mucho, y de los que me llevo un gran recuerdo.

Licencia



Resumen

El objetivo de este trabajo ha sido la construcción de un software de control de un micro dron. Concretamente se trata del modelo Tello, de la marca DJI. Un dron de pequeñas dimensiones que cuenta con conexión wifi para su control externo. El dron también posee una cámara de alta definición, que envía los fotogramas, a través de una conexión TCP/IP. Se pretende que el dron, mediante el software, sea capaz de reconocer su entorno y seguir un objetivo concreto.

El lenguaje utilizado para este propósito es python, concretamente la versión 2.7. El uso de este lenguaje y su versión, serán explicados en futuros apartados.

En cuanto al reconocimiento de objetos hemos utilizado un algoritmo ligero denominado SSD mobilenet, el cual está orientado para ser añadido a sistemas de poca capacidad de cómputo.

La conexión entre el dispositivo Tello y el ordenador, se realiza mediante conexión wifi, diferenciando entre dos puertos diferentes, uno en el que recibe los comandos y el dron envía una respuesta y otro en la que el dron envía los frames de la cámara.

Palabras clave: Dron, Autónomo, Detection, Tracking

Abstract

The objective of this project has been the construction of a micro drone controller software. The model we choose was the Tello, from the DJI brand. It's a small drone device that has a Wi-Fi connection for external control. The device also has a high-definition camera, which sends the frames, through a TCP/IP connection. It is intended that the drone, through software, be able to recognize its environment and follow a specific objective.

The language used for this purpose is python, specifically the 2.7 version, The use of this language and its version will be explained in future sections.

As for the recognition of objects, we have used a lightweight algorithm called SSD mobilenet, which is oriented to be added in systems with low computing capacity.

Keywords: Drone, Autonomous, Detection, Tracking

Índice general

Capítulo 1
Introducción **11**
Motivación 11
Antecedentes y estado actual 11
Características Técnicas del Dron 14
Objetivo Principal 15
Objetivo específico 15
Casos de uso 16

Capítulo 2
Metodología **17**
Metodología General 17
Herramienta SDK 17
Resumen 17
Navegación Automática **21**
Resumen 21
Librerías y Herramientas utilizadas 21
Implementación 22
Test para la elección del tracker **24**
Resumen 24
Librerías y Herramientas utilizadas 25
Algoritmos probados 25
Análisis de Resultados 26
Conclusiones 32

Capítulo 3
Resultados **33**
Resumen 33
Análisis de Resultados 34

Capítulo 4

Conclusiones y líneas futuras **35**

Capítulo 5
Summary and Conclusions **35**

Capítulo 6
Presupuesto **36**

Capítulo 7
Apéndice **37**

Código desarrollado detector 37

Código desarrollado de tracking 39

Código control 41

Índice de figuras

Figura 1: Funcionamiento R-CNN----------13
Figura 2: Estructura SSD----------14
Figura 3: Dron Tello DJI----------15
Figura 4: Interfaz video----------19
Figura 5: Interfaz control Manual----------20
Figura 6: Diagrama básico de funcionamiento----------22
Figura 7: Captura de los videos utilizados----------25
Figura 8: Captura prueba_01----------27
Figura 9: Captura prueba_02----------28
Figura 10: Captura prueba_c01----------30
Figura 11.1: Captura prueba_c02----------31
Figura 11.2 : Captura prueba_c02 oclusión parcial----------31
Figura 12 : Interfaz con el sistema de reconocimiento activo
y el objetivo seleccionado----------34
Figura 13 : Movimientos sesión pruebas----------35

Índice de tablas

Tabla 1: Prueba_01----------27
Tabla 2: Prueba_02----------28
Tabla 3: Prueba_c01----------30
Tabla 4: Prueba_c02----------32
Tabla 5 Presupuesto----------37

1. Capítulo 1 Introducción

1.1. Motivación

El uso de drones se ha generalizado en múltiples aplicaciones. Tanto es así que en la actualidad es usado como sistema de grabación, audiovisuales, vigilancia, carreras, etc. Sin embargo se basan en un sistema de control manual pilotado a través de visión directa del dron o mediante FPV. Los drones autónomos son todavía un campo en desarrollo y pueden facilitar muchas labores.

En este TFG se pretende desarrollar un software de control de un dron de interior de la marca DJI, en concreto el modelo tello.

1.2. Antecedentes y estado actual

Existen drones que ya poseen la capacidad de seguir, cómo el dron parrot bebop 2, que tiene el sistema follow-me. Con el modo de seguimiento follow-me, el dron será capaz de seguir el objetivo mediante reconocimiento visual y la localización GPS del smartphone. Esta tecnología realiza el encuadre mediante reconocimiento visual, el seguimiento horizontal desde el GPS y el seguimiento vertical mediante altímetro. Esto está pensado para exteriores amplios, sin demasiados obstáculos.

El dron Tello de la marca DJI ha sido elegido con la idea de realizar el seguimiento en espacios reducidos o de interior. Con esta idea se ha elegido por su tamaño reducido y por su comunicación vía wifi, posibilitando la comunicación con una gran variedad de dispositivos.

La visión por computador es una disciplina en auge, que se encarga de estudiar la manera de procesar, analizar e interpretar imágenes de forma autónoma. Este tipo de aplicaciones pueden ser usadas en muchos ámbitos, tanto para sistemas de seguridad, inspección automática, medicina o el sistema que nos ocupa, la navegación.

Dentro de estos sistemas, la detección de objetos es uno de los usos más importantes. Aunque el problema parezca sencillo para una persona, para una máquina no es un problema sencillo, debido a la forma en la que las máquinas codifican una imágen digital. Para una máquina, una imágen son matrices tridimensionales en las que se codifica de manera numérica el color de cada sección (RGB). Las máquinas buscan en estas celdas,

patrones que se corresponden a un objeto en particular. Esto no está exento de problemas, pensemos en un objeto cotidiano, un coche por ejemplo, los coches son de distintos colores, tamaños e incluso formas. No es lo mismo un deportivo que una pick up, además debemos añadir otras dificultades que dependen de la perspectiva, la iluminación, oclusiones parciales.

Para reconocer un coche o cualquier otro objeto, la máquina tiene que ser capaz de realizar una representación genérico del coche, y esta representación debe ser impasible a cambios.

Como todo esto de la clasificación de imágenes es un sistema complejo, se han creado sistemas de reconocimiento mediante deep learning, aprendizaje profundo. Estas redes neuronales, son implementadas con diferentes aproximaciones.

Como R-CNN o Region-based Convolutional Neural Network, que consiste en tres fases que se dividen en:

1. Escaneado de la imágen de entrada, para pasarle un algoritmo denominado Selective Search, de la que obtiene una serie de regiones de la imágen.
2. Una red neuronal convolucional (CNN) para cada región extraída.
3. Extracción de la salida de cada CNN para ser añadida en una máquina de vectores de soporte para calificar la región y para la realización de una regresión lineal, que restringirá el cuadro a delimitar del objeto.

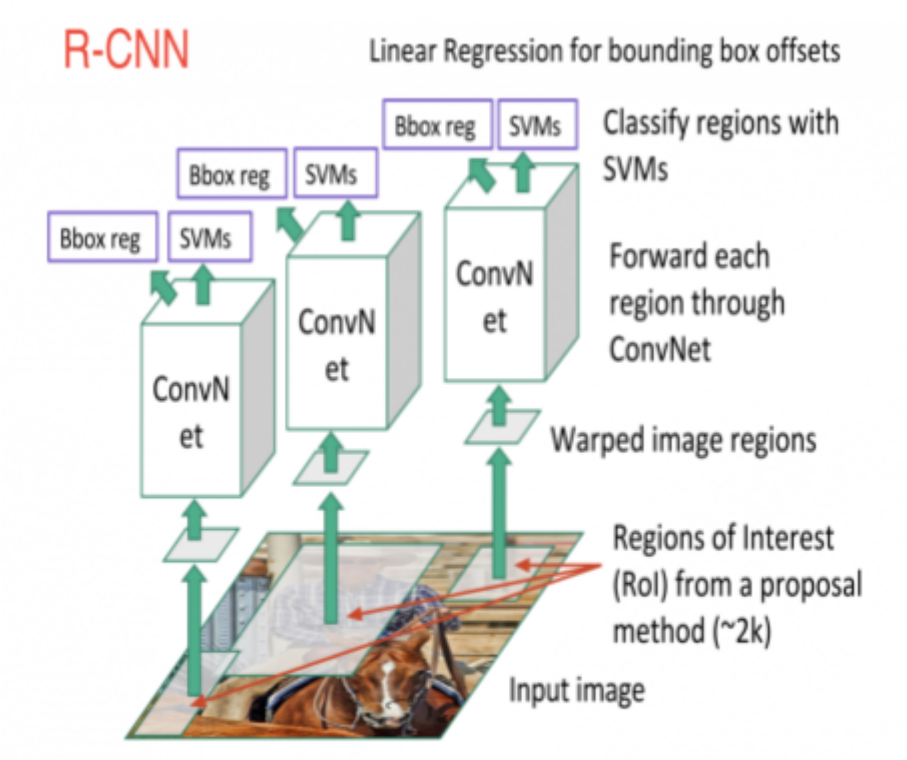


Figura 1: Funcionamiento R-CNN.

Otro modelo es el SSD, o también denominado Single-Shot Detector. Esta aproximación, combina los procesos de realizar la división de las regiones de la imágen y la clasificación de las regiones en un único paso, a la vez que brinda las cajas que delimitan los objetos, junto a la categoría a la que pertenecen.

Para realizar esto, la imágen pasa primero a través de una serie de capas, creando un mapa de características. Tras esto para cada capa de características, se utiliza un filtro convolucional de 3x3 para evaluar un pequeño conjunto predefinido. Por cada pequeño conjunto se establece al mismo tiempo el recuadro y la probabilidad para cada categoría.

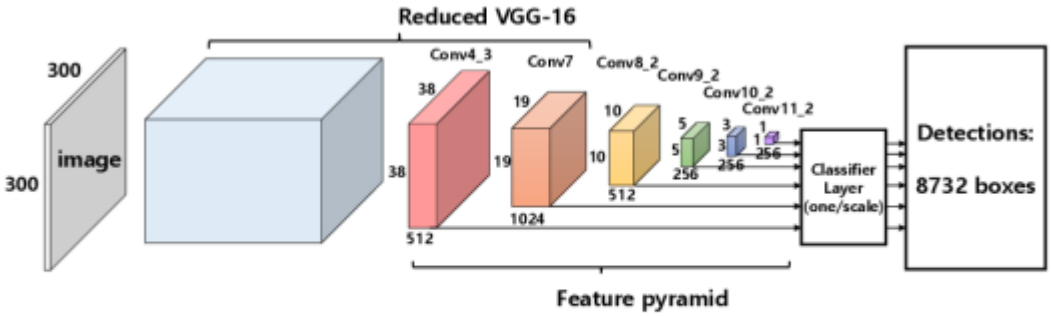


Figura 2: Estructura SSD

1.3. Características Técnicas del Dron

El dron tello de la marca DJI, es un quadcopter, con una distancia operativa máxima de 100 metros y una velocidad máxima de 8 m/s. Su conexión es vía wifi (IEEE 802.11n), banda de frecuencia de 2.4GHz.

Su cámara fotográfica realiza fotos de 5 megapíxeles. En modo vídeo su máxima resolución es de 1280x720p a 30fps en formato MP4. Ángulo de campo de visión 82.6º.

Funciones de despegue y aterrizaje automáticos.

En cuanto a la tecnología de su batería es de un polímero de ión-litio, con capacidad de 1100mAh/4.1Wh y 3.8 V, lo que le da una autonomía de 13 min.

Sus dimensiones son 92 mm de ancho, 98 mm de profundidad y 41 mm de altura. El diámetro del rotor es de 7,62 cm y el peso total es de 80 g con la batería.

La caja del dron incluye herramienta específica para realizar la extracción o colocación de las hélices, junto con dos hélices de repuesto. Un pequeño manual con instrucciones básicas. Además viene incluida una batería extra, haciendo un total de dos baterías. También incluye cuatro protectores, para los rotores y las hélices, acoplándose fácilmente con un click en cada uno de los rotores.

Figura 3: Dron Tello DJI

1.4. Objetivo Principal

El objetivo principal de este proyecto consiste en la realización de un software que ofrezca la capacidad de analizar los datos proporcionados por el dron y actuar en consecuencia, proporcionando un sistema de control autónomo con el que seguir un objetivo.

Este objetivo será especificado desde el mismo sistema de software, proporcionando flexibilidad a la par que fiabilidad en la realización de la tarea encomendada.

1.5. Objetivo específico

- Análisis de la herramienta SDK existente:

    1. Entendimiento exhaustivo del código inicial. Poniendo especial atención en el sistema de comunicación dron-ordenador, al igual que en los controles.

    2. Al existir tres ejemplos de uso del sdk en los que se distinguen: tipo de control y obtención de la imágen, facilita el entendimiento del

software. Aumentando en complejidad gradualmente. Las versiones se denominan, Single Tello Test, Tello Video y Tello Video With Pose Recognition. Como sus propio nombre indica, el primero es una pequeña prueba de funcionamiento, en la que debemos escribir los comandos en un archivo de texto y al ejecutarlo, el dispositivo seguirá esas instrucciones, con el inconveniente de no controlarlo durante el vuelo. El segundo es algo más complejo, el sistema ahora es una interfaz gráfica que nos muestra la imágen y además es controlable mediante el teclado, lo cual soluciona el inconveniente del primero. La tercera es igual que la anterior, pero agregando un reconocimiento de la postura del cuerpo, pero es sólo para tres movimientos, dado que el resto no están contemplados.

3. Analizar las versiones existentes y elegir sobre cuál construir el software autónomo de control.

- Desarrollo del software de control:

1. Tras elegir una de las versiones, decidir mediante un software de pruebas qué sistema de tratamiento de imágenes utilizaremos para el desarrollo del software de control autónomo.

2. Realizar las funciones pertinentes para ajustar el funcionamiento del sistema de movimiento junto al sistema de seguimiento y tratamiento de imágenes elegido.

   Mejoras en el software creado, aumentando la limpieza del código y ajustando los parámetros mediante pruebas con el dron.

1.6. Casos de uso

El objetivo principal de este proyecto consiste en la creación de un programa informático para el seguimiento de un objetivo concreto en zonas de interior de forma automática mediante el tratamiento de imágenes, asegurando su correcto funcionamiento, así como los requisitos descritos.

El sistema software desarrollado se desarrolla con la idea de un sistema autónomo de seguimiento, para un dron comercial. Este sistema software, además es capaz de identificar el entorno.

Contextualizando su uso, el sistema es desarrollado con fines inicialmente académicos, aunque puede ser usado en otras situaciones. Ejemplo de este es el ámbito recreativo, usado para grabar a una persona en movimiento, sin la necesidad de ejercer ningún tipo de control sobre el dron.

2. Capítulo 2 Metodología

2.1. Metodología General

En esta sección del informe discutiremos sobre los distintos lenguajes presentes en el software y las librerías utilizadas para el desarrollo de la misma. Así como las decisiones tomadas con el propósito de dar solución a los retos acontecidos durante la realización del trabajo.

En líneas generales el programa principal está desarrollado en el lenguaje de programación python, concretamente en la versión 2.7. La razón esencial del uso de este lenguaje, es su implementación inicial del sdk en este lenguaje y además posee librerías para el tratamiento y reconocimiento de imágenes.

Como librerías utilizadas se encuentran opencv, una librería de utilidades denominada imutils, con la que realizamos medidas de FPS, librería threading para manejo de hilos.

Una parte importante del código es el decodificador h264decoder escrito en c++, cuya función es la de decodificar el video proporcionado por el aparato de DJI.

Además de una interfaz gráfica basada en Tkinter y PIL, con la que mostramos en el dispositivo de control distintas opciones de control y el video en tiempo real.

2.2. Herramienta SDK

2.2.1. Resumen

El SDK, se encuentra en la clase Tello, dentro del archivo tello.py. Esta clase es la encargada de la interacción con el dron. La conección con el dispositivo, se realiza nada más iniciar la clase. Las dos funciones más importante de esta clase, aparte de la conexión, son las funciones de lectura del frame y la función de envió de los comandos.

```
def send_command(self, command):
        """
        Send a command to the Tello and wait for a response.
        :param command: Command to send.
        :return (str): Response from Tello.
        """

        print (">> send cmd: {}".format(command))
        self.abort_flag = False
        timer = threading.Timer(self.command_timeout, self.set_abort_flag)

        self.socket.sendto(command.encode('utf-8'), self.tello_address)

        timer.start()
        while self.response is None:
            if self.abort_flag is True:
                break
        timer.cancel()

        if self.response is None:
            response = 'none_response'
        else:
            response = self.response.decode('utf-8')

        self.response = None

        return response
```

Como podemos observar, la función de envío, recibe un comando, se formatea, y es enviado por el socket de conexión y espera por la respuesta durante un tiempo.

Existen dos versiones de el sdk, una inicial en la que sólo es capaz de enviar los comandos de movimiento al dron

En esta primera versión del programa, que utiliza la primera versión del sdk, los comandos serán escritos en un archivo de texto, y el software, irá enviando cada comando de uno en uno, enviará el primer comando, esperará respuesta del quadcopter y seguirá con el siguiente comando. En esta versión el SDK, no posee el socket necesario para la recepción de video.

Mientras que la segunda versión del SDK, existen dos versiones de uso:

- En la primera versión de uso del segundo SDK, tenemos una

interfaz gráfica, en la que es posible el control del dron en tiempo real. Además desde la propia interfaz, podremos ver imágenes en tiempo real del propio dron.

- En la tercera versión, se ha agregado un nuevo sistema de control mediante la posición del cuerpo. Pulsando un botón en la interfaz, accedemos a este nuevo sistema de control. Aunque solo posee tres comandos de control, mover hacia atrás, hacia adelante y aterrizar.

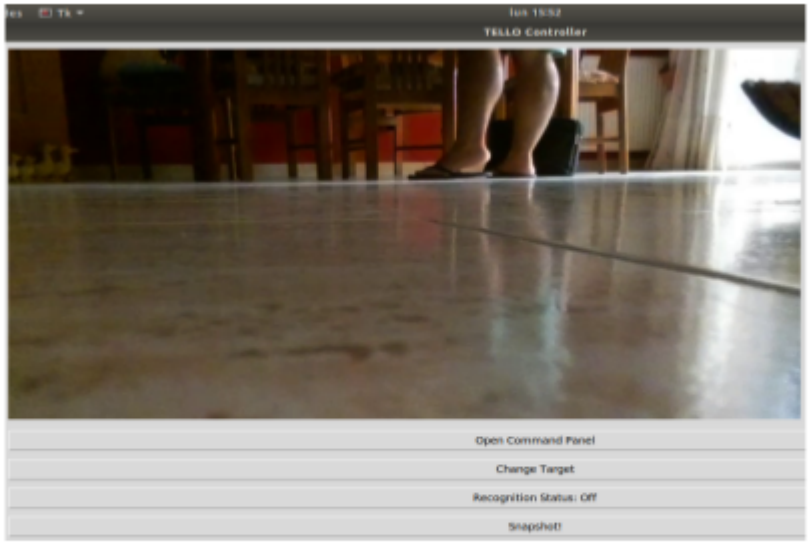


Figura 4: Interfaz video

Por tanto al ir añadiendo complejidad desde la primera versión hasta la tercera, nos vamos a centrar en esta última, la cual posee una interfaz gráfica, y un sistema de reconocimiento del cuerpo mediante puntos, que simulan las articulaciones de un sujeto.

Mediante esta versión es posible controlar el dispositivo mediante la interfaz gráfica, junto con el teclado :

- W : El dron realiza un movimiento de subida.
- S : El dron realiza un movimiento de bajada.
- A : El dron realiza un movimiento de rotación en contra de las agujas del reloj.
- D : El dron realiza un movimiento de rotación a favor de las agujas del reloj.

- Flecha Arriba : El dron se mueve hacia adelante.
- Flecha Abajo : El dron se mueve hacia atrás.
- Flecha Izquierda : El dron realiza un movimiento lateral hacia la izquierda.
- Flecha Derecha : El dron realiza un movimiento lateral hacia la derecha.

Además de mediante el teclado, también tenemos botones en la interfaz. De estos botones, los más importante son Takeoff, que se encarga del despegue automático del quadcopter y Land, que se encarga del aterrizaje.

Con los sliders laterales, ajustamos la velocidad de movimiento y de giro. Para hacer estos cambios, después de ajustar en los sliders, debemos pulsar los botones centrales Reset Distance y Reset Degree, respectivamente.

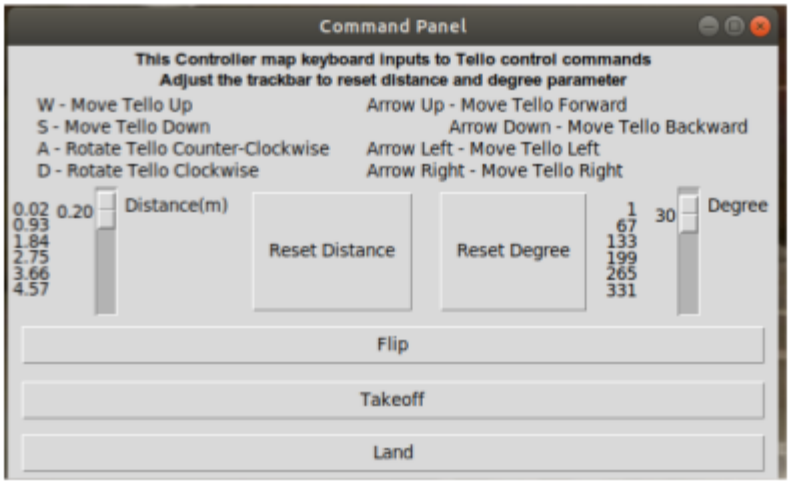


Figura 5: Interfaz control manual

La otra forma de ser controlado es mediante la opción denominada pose_mode, la cual habilita al dispositivo de detectar el cuerpo de una persona, y mediante la posición de los brazos realizar tres comandos:

	Si los brazos del sujeto se encuentran a 45º, el dron se moverá hacia delante.
	Si la persona, tiene los brazos formando un ángulo de 0º, el dron se moverá hacia atrás.
	En este último caso, el objetivo tiene los brazos haciendo una vv, por tanto el dron aterrizará.

2.3. Navegación Automática

2.3.1. Resumen

El sistema de navegación, está creado con la finalidad de perseguir un objetivo. Si bien, el problema es más complejo de lo que aparenta en un inicio. Aunque en la actualidad existen sistemas de seguimiento, siempre existe mejora en su rendimiento y comportamiento.

Para resolver este problema, necesitamos conocer cierta información:

- Localización: Localización del objeto a perseguir
- Estado: Estado actual del dron (reposo, en el aire)

2.3.2. Librerías y Herramientas utilizadas

Con el objetivo de la realización de este apartado, se ha utilizado las siguientes herramientas y librerías.

- **Python** : es un lenguaje de programación interpretado y multiparadigma. Paradigmas cómo programación orientada a objetos,

imperativa y programación funcional. Concretamente hemos utilizado la versión 2.7 de python. En futuros apartados hablaremos de el porqué de esta versión.

- **Opencv :** Es una librería libre, creada para el tratamiento de imágenes. Esta librería es usada para una infinidad de operaciones, modificación del tamaño de la imágen, aplicación de máscaras de suavizado, detección de objetos mediante SDD, diversos sistemas de tracking de los que hablaremos más adelante.

- **Caffe:** Arquitectura convolucional para la incorporación rápida de funciones. Es un framework escrito en c++, con una interfaz escrita en python. Hemos utilizado una red ya entrenada para el reconocimiento de objetivos, basado en SSD (Single Shot Multibox Detector).

- **h264decoder:** es un módulo para python, que provee de un decodificador simple para vídeo capturado desde un cámara. Inicialmente fue escrito para ser usado con proyectos de raspberry pi. Este decodificador está escrito por DaWelter https://github.com/DaWelter/h264decoder

2.3.3. Implementación

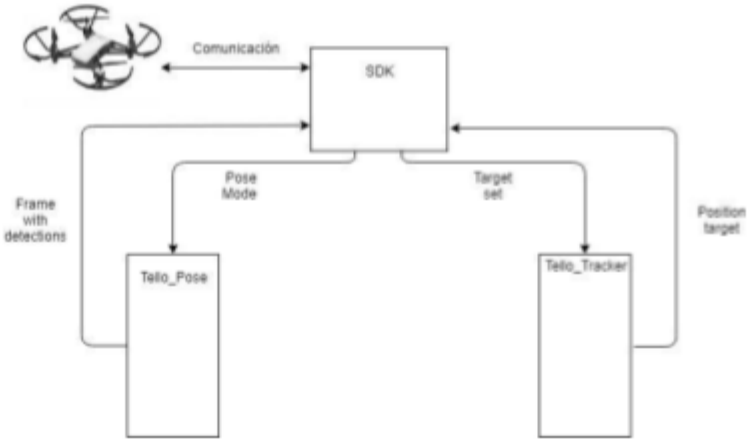


Figura 6: Diagrama básico de funcionamiento

Para la creación del software controlador, se modificó el archivo tello_pose.py, que se encuentra en la tercera versión del software

creado por dji. Esta implementación dibujaba una serie de puntos en el cuerpo, y mediante el ángulo que formaba estos puntos realizaba los movimientos. Tenía tres movimientos, en caso de tener los brazos levantados, el dispositivo se movía hacia atrás, los brazos hacia abajo le indicaba un movimiento hacia adelante y los brazos formando una V le ordenaban realizar un aterrizaje.

Este sistema ha modificado por un sistema basado en reconocimiento de imágenes y el sistema pre entrenado caffe, lo que amplía la cantidad de objetivos.

Este sistema caffe es un SSD desarrollado para dispositivos portátiles y de baja potencia denominado MobileNet.

Con este sistema, el dron pinta rectángulos verdes en los objetos detectados, mientras que el rojo será el objetivo a seguir.

En principio el desarrollo el sistema se basó inicialmente en este sistema de detección, obteniendo un objetivo basado en un identificador del objeto. Este identificador es común para todo objeto del mismo tipo. Toda persona tendrá como identificador el número 15. Esto es un problema, el sistema podría confundir de objetivo fácilmente. El código aquí descrito se encuentra en el apéndice [1].

Para esto, se añade al código inicial una nueva clase, en el archivo tello_tracker.py. Los sistemas de tracking no tienen ninguna idea preconcebida de qué es el objeto a observar. Se le pasa una posición en un frame inicial y es seguida por el tracker en la imágen. Con esto se soluciona el problema derivado de utilizar un sistema de detección. [2]

El tracking es más rápido que la detección, La razón es simple, cuando realizas seguimiento de un objeto, al decirle qué debe seguir, ya posee mucha información, como su forma, la localización en el espacio observado y la dirección y velocidad a la que se mueve. Con esta información se puede predecir el movimiento del objeto en el siguiente frame y realizar una pequeña búsqueda por la zona en cuestión. Mientras que los algoritmos de detección empiezan desde cero.

Sin embargo, lejos de desechar el sistema de detección, se ha mantenido para tratar de crear un sistema, no sólo capaz de detectar y seguir un objetivo, sino que además sea capaz de esquivar obstáculos y mantener persistencia en el sistema de tracking, el cual puede fallar al existir oclusión entre el objetivo y la cámara del dron.

Para el movimiento del dron, se ha añadido en el hilo de control dentro del archivo tello_control_ui.py. Este hilo se encargaba de enviar al dispositivo un comando erróneo para mantener el dron en vuelo (en caso de no recibir nada el dron automáticamente aterriza. Ha sido modificado como sistema de movimiento basado en la posición del objetivo con respecto a su posición en la imágen. La posición del objetivo es suministrada por el sistema de tracking. El sistema de control trata de mantener el objetivo centrado en la imágen. Para ello el dron utiliza la diferencia del objetivo con respecto al centro y proceder a la realización de movimientos de corrección. Los movimientos que es capaz de realizar son seis, girara derecha e izquierda, subir y bajar y movimiento hacia delante y hacia atrás.

Estos movimientos están acotados, para mantener el quadcopter lo más estable posible, evitando movimiento innecesarios si el objetivo apenas se desplaza del centro de la imágen. Mientras que cuanto más se aleja de él, mayor es la fuerza que se le agrega al movimiento. [3]

Para el funcionamiento del tracker, es necesario pasarle un recuadro, en el que se encuentre el objetivo. Para esto, hemos modificado el botón pause, que era el encargado de pausar la imagen. En lugar de esto, ahora llama a la función setTarget, la cual muestra un solo frame, y con el ratón seleccionamos la zona a realizar el tracking. Tras esto debemos pulsar la tecla enter. Con esto el sistema podrá realizar el tracking. Podemos llamar a setTarget, cada vez que queramos cambiar de objetivo.

2.4. Test para la elección del tracker

2.4.1. Resumen

Las simulaciones llevadas a cabo, han sido creadas para la obtención de datos sobre el mejor algoritmo de tracking, para cada situación. Las pruebas se realizaron con distintos tipos de vídeo y realizando la búsqueda sobre distintos objetos o personas en cada vídeo.

Los primeros vídeos han sido obtenidos de internet, mientras que los otros dos han sido grabados desde el propio dron, utilizando la aplicación para ios de la propia marca.

En el primero podemos observar una zona peatonal de una ciudad, mientras que el segundo, podemos observar que es un centro comercial abierto. Los dos grabados desde el dispositivo son calles de San Cristóbal de La Laguna.

2.4.2. Librerías y Herramientas utilizadas

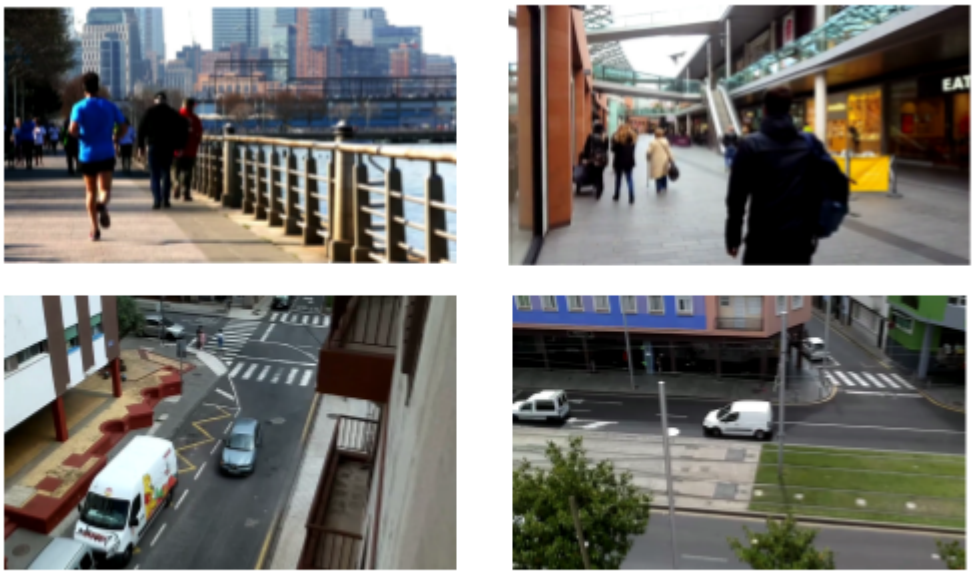


Figura 7: Capturas de los vídeos utilizados

Para realizar este test, debemos utilizar, además de las librerías anteriores en el sistema de tracking, la librería **imutils**. Esta librería contiene funciones para realizar operaciones básicas con imágenes como traslación, rotación. Pero para nosotros la función que nos interesa es la de fps, que nos permitirá medir en pantalla los fotogramas por segundo, que son capaces de procesar cada uno de los algoritmos de tracking utilizados.

2.4.3. Algoritmos probados

Los algoritmos utilizados, son los propios que contiene la librería opencv, estos algoritmos son:

- **CSRT :** Utiliza un filtro de discriminación correlacionado.

- **KCF:** También denominado tracking with correlation filters, la idea básica de este tracker es la de estimar un filtro de imagen óptimo de manera que la filtración, junto con la imagen de entrada, produzca la respuesta deseada. Esta respuesta deseada es típicamente de una gaussiana centrada en la ubicación del objetivo.

- **Boosting** : Consiste en combinar varios clasificadores débiles para obtener una clasificación conjunta mejor. Estos clasificadores débiles, se les añade un peso en función de si sus predicciones son acertadas.

- **Mil :** Multiple Instance Learning, utiliza un clasificador discriminador para separar el objeto del fondo de la imágen. Este clasificador utiliza el estado actual del tracker, para extraer ejemplos positivos y negativos del frame actual.

- **TLD:** Es un sistema de tracker, galardonado como el algoritmo de tracking en tiempo real para objetos desconocidos. De manera simultánea realiza el seguimiento, aprende de su apariencia y detecta cada vez que aparezca en el video. Como resultado de esto, funciona mejor cuanto mayor es el tiempo de uso.

- **Medianflow:** Este algoritmo utiliza una serie de puntos en el objeto a seguir, los cuales utiliza junto al siguiente frame para realizar el tracking, estimar el error, filtrar los valores atípicos y actualizar el recuadro. Funciona muy bien si el movimiento es esperado.

- **Mosse:** Es posiblemente de lejos el algoritmo más rápido de tracking que usa como método filtros correlacionales. Estos filtros correlacional comprenden tres pasos, primero utiliza la transformada rápida de Fourier en la imágen y en el objeto, tras esto una operación convolucional en el objeto y la imágen, pasando ambos resultados por la inversa de la transformada rápida de Fourier. Dando como resultado de esto la posición del objetivo en la imágen.

2.4.4. Análisis de Resultados

Para el vídeo denominado prueba_01, en el que aparecen personas caminando sobre un puente, hemos usado como sujeto de pruebas al señor de azul, que va corriendo de espaldas. Hemos utilizado a este señor como objetivo debido a que es ocultado totalmente por otra personas durante la grabación, y con ello pretendemos analizar la capacidad de recuperación de los algoritmos.

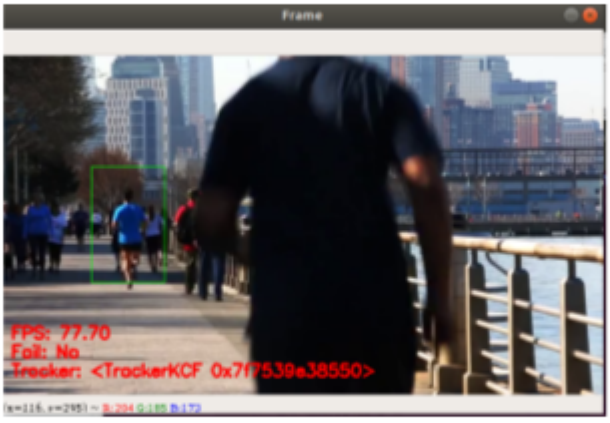


Figura 8: Captura prueba_01

	Media de FPS	Número de fallos	Observaciones
CSRT	27	427	El algoritmo falla, cuando el sujeto es ocultado.
KCF	80	20	El algoritmo se recupera perfectamente tras oclusión del sujeto
Boosting	30	0	El algoritmo no reporta fallo, aunque observamos como cambia el objetivo.
MIL	18	0	Mismo problema que el algoritmo de Boosting
TLD	25	5	Falla a mayor distancia del objetivo, cambios descontrolados.

Medianflow	130	34	Al ser ocultado el objetivo, falla y cambia el objetivo.
Mosse	165	723	Falla con facilidad, pero reporta el fallo.

Como podemos observar de la tabla para esta prueba, el algoritmo **CSRT** falla cuando el objetivo es ocultado, aún así notifica del fallo lo que está bien. Los algoritmos **Boosting**, **MIL** y **Medianflow** fallan y no reportan fallo, por lo que se reportan de la peor manera posible. En el caso de **TLD** falla cuando el objetivo se aleja una cierta distancia, lo que puede seguir siendo útil. De todos los algoritmos el que mejores impresiones nos ha dado es el **KCF**, demostrando que es capaz de recuperar el objetivo que ha sido ocultado.

Para la prueba_02, en la que podemos observar a varias personas caminando en un centro comercial. Para este hemos usado de sujeto de pruebas una mujer que que al realizar un giro a la izquierda desaparece del video. Con esto queremos observar la situación en la que el objetivo desaparezca de la pantalla.

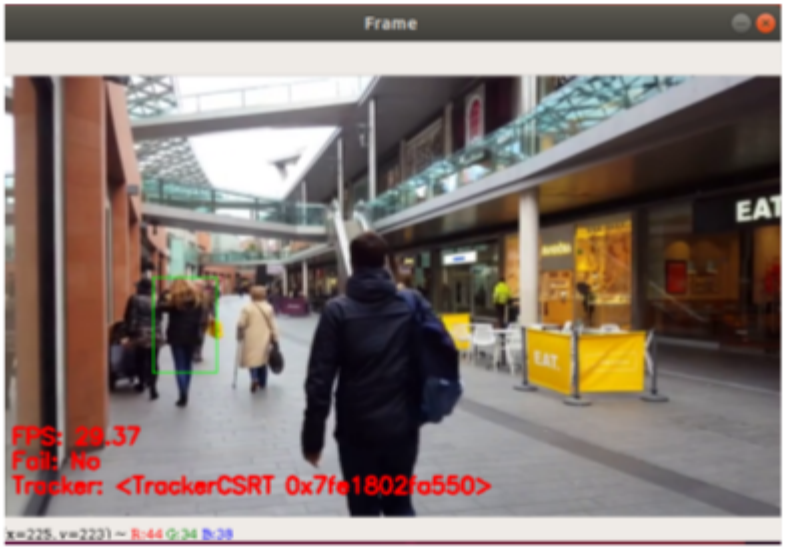


Figura 9: Captura prueba_02

	Media de FPS	Número de fallos	Observaciones
CSRT	33	164	El algoritmo falla, cuando el sujeto abandona la escena.
KCF	46	197	Sucede lo mismo que en el anterior algoritmo.
Boosting	50	0	El algoritmo no reporta fallo, sin embargo al perder el objetivo se mantiene en la misma zona.
MIL	18	0	Mismo problema que el algoritmo de Boosting
TLD	20	0	Falla incluso antes de que el objetivo abandone la escena, apuntando a otras personas en movimiento.
Medianflow	142	15	Se mantiene en la posición original, sin seguir al objetivo.
Mosse	184	277	Falla con facilidad, pero reporta el fallo.

En este caso que nos ocupa, los mejores resultados observados son los de los algoritmos **CSRT** y **KCF** Que nos reportan que el objetivo ha desaparecido de escena y mantienen una tasa superior a los 30 fps.

Para la prueba_c01, vídeo obtenido desde el propio dron, apuntando hacia una calle transitada de La Laguna. Tratamos de seguir el movimiento de un vehículo de color gris. No existe ningún tipo de oclusión, ni parcial ni total. Con este realizamos una prueba en la que el

vehículo realiza un cambio de orientación, por tanto su forma será modificado, con respecto al objetivo.

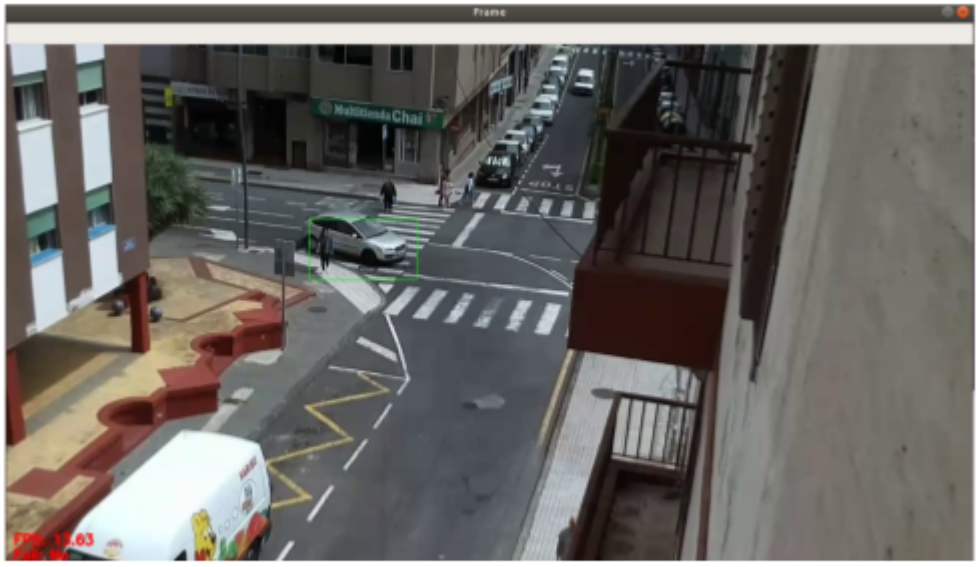


Figura 10: Captura prueba_c01

	Media de FPS	Número de fallos	Observaciones
CSRT	28	0	El algoritmo funciona perfectamente
KCF	54	133	Falla al cambiar el vehículo de orientación.
Boosting	40	0	El algoritmo funciona bien, algo descentrado.
MIL	16.2	0	En algunos caso falla al girar el vehículo, sin reportar el fallo.
TLD	10	0	Sigue teniendo fallos al moverse el objetivo.
Medianflow	65	49	Aunque informa de que ha fallado, la

			mayor parte del tiempo se queda apuntando hacia una zona en la que el objetivo se encontraba antes.
Mosse	93	393	Falla con facilidad, pero reporta el fallo.

Para este ejemplo en lo que tratamos es de mantener el objetivo aunque modifique su dirección de movimiento y con ello su forma, el algoritmo que mejores resultados ha dado es el de **CSRT**, sin fallos y funcionando perfectamente sin perder el objetivo.

Para la prueba_c02, es un vídeo en la trinidad, en la que el vehículo es tapado parcialmente por el tranvía, además, movemos el dron para que deje de observar el vehículo durante un instante y luego volvemos hacia la zona en la que se encuentra.

Con esto tratamos de observar los fallos de oclusión parcial y de pérdida del objetivo.

Figura 11.1: Captura prueba_c02

Figura 11.2: Captura prueba_c02 oclusión parcial

	Media de FPS	Número de fallos	Observaciones
CSRT	25	0	El algoritmo funciona en oclusión parcial, sin embargo al desaparecer el coche de escena, cambia el objetivo.
KCF	44	186	Notificación correcta de fallos, recuperación del objetivo correcta.
Boosting	29	0	Aunque no lo notifique, el algoritmo falla al desaparecer el objetivo, cambia de objetivo.
MIL	16.2	0	Falla cuando el objetivo desaparece de pantalla, aunque no lo notifica.
TLD	15	0	Tiene muchos fallos que no notifica, el objetivo es marcado de forma errática.
Medianflow	80	0	Aunque falla en oclusión parcial y en la desaparición momentánea del objetivo, no reporta fallo.
Mosse	97	77	Reporta fallos, en oclusión parcial funciona bien, no es muy fiable cuando el objetivo deja de aparecer

			en pantalla.

En este caso, aunque el algoritmo de **CSRT** funciona en oclusión parcial, en el caso de desaparecer el objetivo, elige otro vehículo de características iniciales, lo que lo hace un peor candidato que el algoritmo **KCF**.

2.4.5. Conclusiones

Para el objetivo de este sistema de seguimiento en concreto, debemos tener en cuenta que los fotogramas por segundo entregados por el propio dron, tienen un máximo de 30 fps. Con esto en mente si elegimos un sistema de tracking con alta tasa de fotogramas, estaremos desperdiciando totalmente la velocidad de procesamiento del algoritmo, además cómo podemos comprobar en la tabla anterior, algoritmos con alta tasa de fotogramas por segundo, posee un rendimiento peor que otros más lentos. Sobre todo si el objetivo en mente es un objetivo de tamaño inferior a un coche.

Por tanto descartamos usar en el sistema, algoritmos de tracking como **Mosse** y **Medianflow.**

En otros casos con sistemas que tengan una tasa de fotogramas más contenida pero con una alta tasa de fallos, tanto reportados por el propio sistema, como los observados mediante la prueba, también serán descartados. Descartamos los algoritmos **Tld**, **Boosting**, **MIL** y **CSRT**.

El que mejor resultados ha reportado en todas las pruebas, ha sido el algoritmo de **KCF**, tanto en media de fps (siendo superior al soportado por el dron), como por su capacidad de recuperación del objetivo en caso de ser ocluido. Por tanto es el que usaremos para el sistema de seguimiento del dron.
Aunque haya sido el elegido, debemos tener en cuenta que para objetos grandes, otros sistemas funcionan perfectamente.

3. Capítulo 3
Resultados

3.1. Resumen

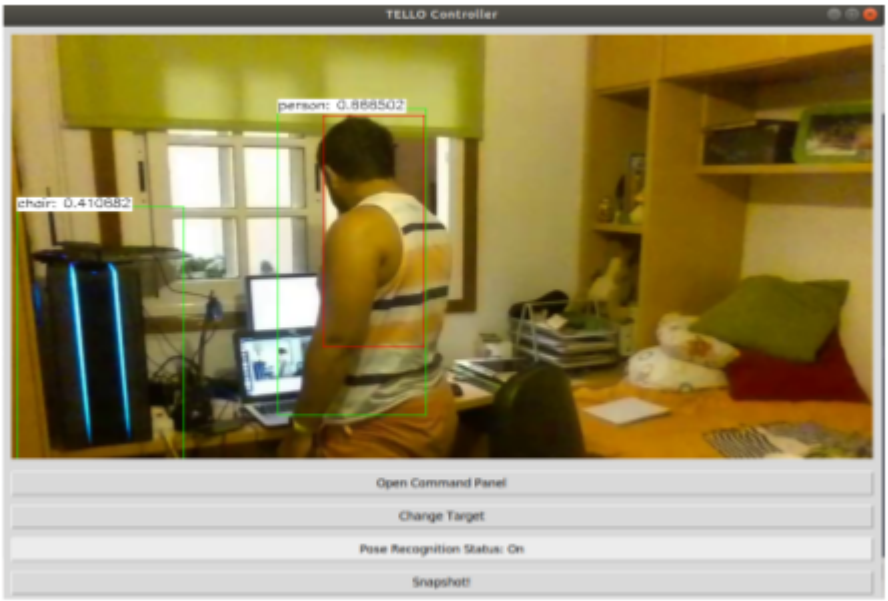


Figura 12: Interfaz SDK con el sistema de reconocimiento activo y el objetivo seleccionado

Los resultados obtenidos por el sistema de control autónomo desarrollado sobre la herramienta proporcionada (SDK), nos dan un sistema bastante capaz como sistema de seguimiento en zonas de interior.

En la siguiente imágen podemos observar mediante la consola los movimientos realizados durante una sesión de pruebas.

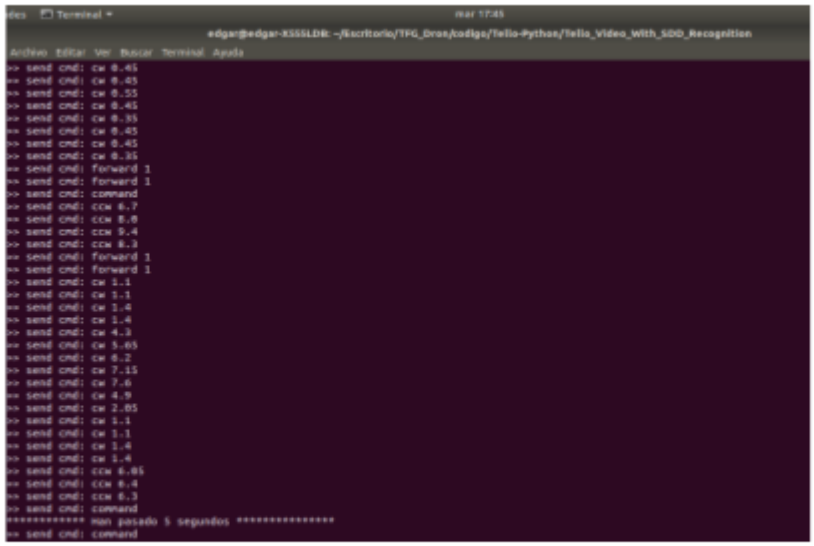


Figura 13: Movimientos sesión de pruebas

En la que se puede observar los comandos enviados desde la aplicación al dron, junto con un número, que es el movimiento que debe hacer.

3.2. Análisis de Resultados

Durante el proceso de creación de este sistema, hemos observado la gran cantidad de sistemas de seguimiento existentes en el mercado, distinguiendo entre dos grupos, la detección y el tracking. Utilizando ambos hemos podido resolver los problemas derivados de cada uno de ellos. En caso de haber usado simplemente el tracking, hubiera sido imposible distinguir el resto de objetos que pueden interferir el sistema. En el caso de haber empleado el sistema de detección, no hubiéramos podido preservar la identidad del objeto a seguir.

Con esto el uso de las dos sistemas de forma paralela creemos que es la mejor aproximación posible del sistema.

4. Capítulo 4
Conclusiones y líneas futuras

Durante el desarrollo de este proyecto, se han implementado una serie de herramientas aprovechables en futuras investigaciones sobre la navegación automática en vehículos aéreos, del tipo quadcopter. Mediante estas herramientas, hemos descubierto la posibilidad de utilizar el tratamiento de imágenes para localizar un objetivo a seguir, ya sea mediante tracking o mediante detección.

Tras el análisis de los algoritmos de tracking implementados dentro de la librería opencv, hemos establecido un sistema acorde con nuestro dispositivo y sus especificaciones.

Así mismo hemos mantenido el sistema de detección, posibilitando en un futuro la agregación de un agente inteligente, capaz de utilizar la información proporcionada del detector, siendo este agente inteligente el futuro encargado del movimiento automático del dispositivo, en lugar de las condiciones escritas en el programa. Con esto además de seguir el objetivo con la información proporcionada por el tracking, debemos agregarle la información obtenida por el detector, evitando así posibles colisiones.

5. Capítulo 5
Summary and Conclusions

During the development of this project, a series of useful tools have been implemented for future research on automatic navigation in aerial vehicles, like quadcopter type. Through these tools, we have discovered the possibility of using image processing to locate targets to follow, either by tracking or by detection.

After analyzing the tracking algorithms implemented within the opencv library, we have established a system in accordance with our device and its

specifications.

We have also maintained the detection system, enabling in the future the aggregation of an intelligent agent, capable of using the information provided by the detector. This intelligent agent will be in charge of the automatic movement of the device, instead of the conditions written in the program. With this in additions to following the objective with the information provided by the tracking, we must add the information obtained by the detector, with this the agent could avoid possible collisions.

6. Capítulo 6 Presupuesto

El presupuesto requerido para la realización de este proyecto calculado según el coste aproximado de una persona realizando este tipo de tareas en Canarias con proyectos de este tipo, equivale a 40€.

Tarea	Horas empleadas	Coste
Gestión del conocimiento	150	6000
Desarrollo del sistema de detección	50	2000
Pruebas de funcionamiento	15	600
Pruebas algoritmos de tracking	30	1200
Análisis de resultados tracking	20	800
Implementación sistema de tracking	40	1600

Análisis de resultado sistema autónomo	30	1200
Total	335	13400

Tabla 7.1: Resumen de tipos

7. Capítulo 7 Apéndice

7.1. Código desarrollado detector

Algoritmo de detección

```
/*****************************************************************************
*
* tello_pose.py
*
*****************************************************************************
*
* Edgar Mesa Santana
*
*
* 09/09/2019
*****************************************************************************
def detect(self, frame, box):
        """
        Funcion principal que realiza la deteccion de objetos en el frame dado
        :param frame: Recibe el frame del video
        :param box: Es la localizacion del objeto pasado por el tracker, Es
usado solo para pintarlo en el frame
        :return : Devuelve el frame con los recuadros de los objetos detectados.
```

```
        """
        frameWidth = frame.shape[1] # 960
        frameHeight = frame.shape[0] # 720

        frame_resized = cv2.resize(frame, ( self.inWidth, self.inHeight ))
        inpBlob = cv2.dnn.blobFromImage(frame_resized, 0.007843,
(self.inWidth, self.inHeight),
                                (127.5, 127.5, 127.5), False)
        self.net.setInput(inpBlob)

        # Prediction of network
        detections = self.net.forward()

        for i in range(detections.shape[2]):
                confidence = detections[0, 0, i, 2] # Confidence of prediction
                if confidence > self.confidence:
                classId = int(detections[0, 0, i, 1])  #name object

                        #Object location
                xLeftBottom = int(detections[0, 0, i, 3] * self.inWidth)
                yLeftBottom = int(detections[0, 0, i, 4] * self.inHeight)
                xRightTop   = int(detections[0, 0, i, 5] * self.inWidth)
                yRightTop   = int(detections[0, 0, i, 6] * self.inHeight)

                        # Factor for scale to original size of frame
                heightFactor = frameHeight/self.inHeight
                widthFactor  = frameWidth/self.inWidth
                # Scale object detection to frame
                xLeftBottom = int(widthFactor * xLeftBottom)
                yLeftBottom = int(heightFactor * yLeftBottom)
                xRightTop   = int(widthFactor * xRightTop)
                yRightTop   = int(heightFactor * yRightTop)
                # Draw location of object
                if box is not None:
                xb, yb, wb, hb = self.targetPosition(box)
```

```
            cv2.rectangle(frame, (xb, yb), (wb, hb),
                          (255, 0, 0))
            cv2.rectangle(frame, (xLeftBottom, yLeftBottom), (xRightTop,
yRightTop),
                  (0, 255, 0))
            # Draw label and confidence of prediction in frame resized
            if classId in self.classNames:
            label = self.classNames[classId] + ": " + str(confidence)
            labelSize, baseLine = cv2.getTextSize(label,
cv2.FONT_HERSHEY_SIMPLEX, 0.5, 1)
            yLeftBottom = max(yLeftBottom, labelSize[1])
            cv2.rectangle(frame, (xLeftBottom, yLeftBottom - labelSize[1]),
                          (xLeftBottom + labelSize[0], yLeftBottom + baseLine),
                          (255, 255, 255), cv2.FILLED)

            cv2.putText(frame, label, (xLeftBottom, yLeftBottom),
                  cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0))

      return frame
*
*
*******************************************************************************/
```

7.2. Código desarrollado de tracking

Algoritmo de tracking

```
/*******************************************************************************
 *
 * tello_tracker.py
 *
 *******************************************************************************
 *
 * Edgar Mesa Santana
 *
 * 09/09/2019
```

```
*****************************************************************************
 def track(self, frame):
        """
        Funcion principal de la clase, esta funcion es la encargada de recibir el frame,
        y realizar el tracking del objetivo marcado
        :param frame: Recibe el frame del video
        :return : Devuelve la posicion del objeto
        """
        if frame is not None:
        # grab the frame sizes, for later
        ( frameHeight, frameWidth ) = frame.shape[:2]
                # resize the frame (so we can process it faster) and grab the
                # new frame dimensions
        frameResized = cv2.resize(frame, ( self.inWidth, self.inHeight ))

        if (self.initBBSet is False) & (self.targetBox is not None):
                targetBox = list(self.targetBox)
                targetBox = ( targetBox[0]*self.inWidth/frameWidth,
targetBox[1]*self.inHeight/frameHeight,
                                targetBox[2]*self.inWidth/frameWidth,
targetBox[3]*self.inHeight/frameHeight )
                targetBox = tuple(targetBox)
                self.initBB = targetBox
                self.tracker.init(frameResized, self.initBB)
                self.initBBSet = True

        else:
                        # grab the new bounding box coordinates of the object
                (success, box) = self.tracker.update(frameResized)
                        # check to see if the tracking was success
                if success:
                        (x, y, w, h) = [int(v) for v in box]
                        self.box = (x*frameWidth/self.inWidth, y*frameHeight/self.inHeight,
                                        w*frameWidth/self.inWidth +
x*frameWidth/self.inWidth,
                                        h*frameHeight/self.inHeight +
y*frameHeight/self.inHeight )
                else:
                        self.box = None
```

```
        return self.box

 ****************************************************************************/
```

7.3. Código control

Algoritmo de control

```
/*****************************************************************************
 *
 *tello_control_ui.py
 *
 *****************************************************************************
 *
 * Edgar Mesa Santana
 *
 * 09/09/2019
 *****************************************************************************
 def _sendingCommand(self):
        """
        start a while loop that sends 'command' to tello every 5 second
        """
        while True:
                if (self.tracking is True) & (self.target is not None):
                newBox = self.target
                (x, y, w, h) = [int(v) for v in newBox]   # Contiene la nueva posicion
                xR = w + x
                yR = h + y
                xObjectCenter = abs((x + xR)/2.0)
                yObjectCenter = abs((y + yR)/2.0)
                if self.initialArea is None:
                    self.initialArea = abs(x-xR)*abs(y-yR)
                newObjectArea = abs(x-xR)*abs(y-yR)
                xDif = abs((xObjectCenter - 480)/30.0)
                yDif = abs((yObjectCenter - 360)/1000.0)
                areaDif = abs(newObjectArea - self.initialArea)/1000000.0
#               print ("xDif: %f, yDif: %f, areaDif: %f" %(xDif, yDif, areaDif)) #0.9
```

```
            if (areaDif*10) > xDif:
                if newObjectArea < self.initialArea:
                    self.telloMoveForward(areaDif)
                elif newObjectArea > self.initialArea:
                    self.telloMoveBackward(areaDif)
            elif xDif > (yDif*20):
                if xObjectCenter < 300:
                      self.telloCCW(xDif)
                elif xObjectCenter > 420:
                      self.telloCW(xDif)
            elif xDif < (yDif*20):
                 if yObjectCenter < 420:
                    self.telloUp(yDif)
                 elif yObjectCenter > 540:
                    self.telloDown(yDif)
            else:
                 self.tello.send_command('command')
                 time.sleep(5)
            else:
                 self.tello.send_command('command')
                 print ("********** Han pasado 5 segundos ***************")
                 time.sleep(5)

*****************************************************************************/
```

Bibliografía

Dji-sdk----------https://github.com/dji-sdk/Tello-Python

Python 2.7 ----------https://docs.python.org/2/index.html

Opencv----------https://docs.opencv.org/master/d6/d00/tutorial_py_root.html

Tkinter----------https://www.datacamp.com/community/tutorials/gui-tkinter-python#tkinter

Pil----------https://recursospython.com/guias-y-manuales/instalar-pil-pillow-efectos/

Numpy ----------https://es.wikipedia.org/wiki/NumPy

Imutils----------https://pypi.org/project/imutils/

Visión por computador----------http://informatica.blogs.uoc.edu/2012/04/19/la-vision-por-computador-una-disciplina-en-auge/

Reconocimiento de objetos----------https://www.deeplearningitalia.com/uso-del-aprendizaje-profundo-para-el-reconocimiento-de-objetos/

Caffe (software)----------https://en.wikipedia.org/wiki/Caffe_(software)

Tracking----------https://www.pyimagesearch.com/2018/07/30/opencv-object-tracking/

Kcf----------https://cw.fel.cvut.cz/b172/courses/mpv/labs/4_tracking/4b_tracking_kcf

Mil----------http://vision.ucsd.edu/~bbabenko/old/project_miltrack.shtml

Tld----------http://kahlan.eps.surrey.ac.uk/featurespace/tld/

Medianflow----------https://summerofhpc.prace-ri.eu/breaking-the-4th-dimensional-wall-of-object-tracking/

MOSSE----------https://www.oreilly.com/library/view/practical-computer-vision/9781788297684/5515bcd3-ed2e-4fbf-9396-6bfe79356d0e.xhtml