



ugr | Universidad
de **Granada**

TRABAJO FIN DE GRADO
INGENIERÍA EN INFORMÁTICA

Seguimiento Activo de Personas con dron Parrot Bebop2

Integración en ROS

Autor

José Javier Alonso Ramos

Directores

Fran Barranco Expósito
Eduardo Ros Vidal



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
Granada, Julio de 2020

Seguimiento Activo de Personas con dron Parrot Bebop2: Integración en ROS

José Javier Alonso Ramos

Palabras clave: parrot, bebop, bebop2, autonomy, autonomía, dron, drone, UAV, track, tracking, YOLO, YOLOv3, python, python2, python2.7, Deep, SORT, Deep-SORT, CNN, convolucionar, convolutional, neural, neuronal, network, red, ROS, rospy, opencv, opencv2, detection, detección, fly

Resumen

Este proyecto de fin de grado presenta el desarrollo de un sistema de video vigilancia activo y un estudio de la eficacia y la eficiencia del mismo. Como elemento activo se utiliza el dron Parrot Bebop 2 que hace la función de cámara móvil. Todo el trabajo se implementa sobre el framework ROS (Robot Operating System)

El objetivo principal es la automatización de movimiento del dron dotándolo de visión por computador e inteligencia artificial. Se implementa un comportamiento reactivo que obtiene como input el vídeo proporcionada por la cámara de a bordo. A esta entrada se le realiza, previamente, detección y seguimiento de personas.

Explicaremos las funcionalidades de ROS, el controlador del dron en cuestión (Bebop Autonomy) y el la implementación del comportamiento reactivo.

Por último mostraremos los resultados obtenidos durante las pruebas del sistema.

El proyecto será liberado, contribuyendo así a la comunidad de software libre. El código abierto en el campo de la robótica es relativamente escaso, por lo tanto se espera que esta contribución resulte útil para cualquier investigador o desarrollador.

Active Person Tracking using Parrot Bebop2 drone: Running in ROS

José Javier Alonso Ramos

Keywords: parrot, bebop, bebop2, autonomy, autonomía, dron, drone, UAV, track, tracking, YOLO, YOLOv3, python, python2, python2.7, CNN, Deep, SORT, Deep-SORT, CNN, convolucional, convolutional, neural, neuronal, network, red, ROS, rospy, opencv, opencv2, detection, detección, fly

Abstract

This Bachelor thesis presents the development of an active video surveillance system and a study of its effectiveness and efficiency. Parrot Bebop 2 drone is used as a system's active element which works as a mobile camera. The entire project has been implemented in ROS (Robot Operating System).

The main objective is to automate the drone's movement by providing it with computer vision and artificial intelligence. A reactive behavior is implemented that takes as input the video provided by the on-board camera. At this entrance, people are previously detected and tracked.

ROS functionalities will be explained as well as the drone driver (Bebop Autonomy) and the code that implements the reactive behavior.

The results of testing the project will be shown at the end.

The project will be released under a free software license, thereby contributing to the open source community. Open source in the field of robotics is relatively scarce, therefore this contribution is expected to be useful for any researcher or developer.

Yo, José Javier Alonso Ramos, alumno de la titulación INGENIERÍA INFORMÁTICA de la Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada, con DNI *, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: José Javier Alonso Ramos

Granada a 01 de junio de 2020.

*

D. **Francisco Barranco Expósito**, Profesor del Área de Ingeniería Informática del Departamento Arquitectura y Tecnología de Computadores de la Universidad de Granada.

D. **Eduardo Ros Vidal**, Profesor del Área de Ingeniería Informática del Departamento Arquitectura y Tecnología de Computadores de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado *Seguimiento Activo de Personas con dron Parrot Bebop2, Integración en ROS*, ha sido realizado bajo su supervisión por **José Javier Alonso Ramos**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 15 de junio de 2020.

Los directores:

Francisco Barranco Expósito Eduardo Ros Vidal

Agradecimientos

A mi padre, que ha sido un gran colaborador en el proyecto, siempre dispuesto a realizar las pruebas de vuelo conmigo y a obligarme a dar lo mejor de mí mismo.

A mi madre, siempre entregada a su familia, dispuesta a escuchar tanto penurias como alegrías y capaz como nadie de diluir las primeras y ensalzar las segundas.

A mi hermana, que siempre supone un apoyo incondicional y cada día muestra interés en mis progresos tanto académicos como personales.

A mi hermano, que como mi padre, siempre ha estado dispuesto a realizar las pruebas del proyecto conmigo, y que ha vivido los avances en el mismo con la misma cercanía y alegría que yo.

Gracias también a mi familia en Madrid, que a pesar de la distancia que nos separa, me han hecho llegar todo su cariño y apoyo.

A mi tía Araceli, que se alegra como la que más por verme finalizar esta etapa y verme cumplir un sueño.

A mi primo Pedro, que siempre le entusiasman mis inventos caseros y grandes proyectos y cuya pasión por la ciencia nos une.

A mi primo Samuel, que siempre se preocupa de la felicidad de *"sus primos de Granada"* y que tiene un corazón enorme.

Y por supuesto, a mi abuela, que nos une a todos como nadie más podría hacerlo, y que siempre está pendiente y feliz de mis avances.

Por supuesto, dar las gracias a mis dos tutores, Eduardo y Francisco, por confiar en mí y darme la oportunidad de desarrollar este proyecto. En especial dar las gracias de nuevo a Fran y a su compañero Juan Ignacio, por su ayuda y plena disposición durante todo el proceso de desarrollo.

Por último, pero no menos importante, tengo que agradecer a todos mis amigos el conseguir sacarme una sonrisa en momentos de frustración, el conseguir que me despeje y cambiar el enfoque de la situación. Gracias a todos por apoyarme y hacerme feliz.

A todos ellos, que en todo momento me han apoyado y se han preocupado por mis avances y mi felicidad, gracias.

Declaración de autoría y originalidad del TFG

Yo, José Javier Alonso Ramos, con DNI *, declaro que el presente documento ha sido realizado por mí y se basa en mi propio trabajo, a menos que se indique lo contrario. No se ha utilizado el trabajo de ninguna otra persona sin el debido reconocimiento. Todas las referencias han sido citadas y todas las fuentes de información y conjuntos de datos han sido específicamente reconocidos.

Fdo: José Javier Alonso Ramos

Granada a 29 de junio de 2020.

Índice

1. Introducción	13
1.1. Objetivos	13
1.2. Estado del Arte	13
1.3. Motivación	15
2. Entorno Operacional	16
2.1. Ubuntu 16.04	16
2.2. Robotic Operating System	16
2.2.1. Conceptos	17
2.3. Driver <i>bebop-autonomy</i>	20
2.3.1. Comandos ROS	22
2.4. Bebop 2	23
2.4.1. Especificaciones	23
2.5. Ordenador	25
3. Gestión del Proyecto	26
3.1. Metodología de Desarrollo	26
3.2. Planificación	26
3.3. Restricciones del Sistema	27
3.3.1. Restricciones Harware	27
3.3.2. Restricciones Software	27
3.4. Hitos	28
3.5. Problemas	29
3.5.1. Virtualización	29
3.5.2. Equipo de Trabajo	29
3.5.3. Límite de Distancia de Conexión	31
3.5.4. Python y DeepSORT	31
3.5.5. Instalación de ROS2 junto a ROS Kinetic	31
3.6. Presupuesto	32
3.6.1. Recursos Hardware	32
3.6.2. Recursos Software	32
3.6.3. Recursos Humanos	33
3.6.4. Coste Total	33
4. Diseño	34
5. Implementación	36
5.1. Visión por Computador	36
5.1.1. Captación de imágenes del dron	36
5.1.2. Captación de imágenes de la webcam	37
5.1.3. Lectura de archivo de vídeo almacenado	37
5.1.4. Escritura de vídeo	38

5.1.5. Tracker - DeepSORT	38
5.1.6. Detector - YOLOv3	39
5.2. Comportamiento del Dron	40
5.2.1. Pilotaje Manual	40
5.2.2. Pilotaje Automático	41
5.2.3. Opciones Menú Principal	42
6. Experimentos	43
6.1. Eficiencia del tracker DeepSORT en MOT Challenge	43
6.1.1. MOT17_05	43
6.2. Pruebas del Detector y Tracker	53
6.2.1. Oclusión	53
6.2.2. Consistencia del Tracker	54
6.3. Pruebas con el Dron	56
6.3.1. Vuelo con app	56
6.3.2. Ejecución en Equipo Portatil	59
6.3.3. Vuelo automático	61
6.3.4. Experimentos Varios	69
7. Conclusiones y Futuras Mejoras	71

Índice de figuras

1.	2015 - Uso de drones en la industria	14
2.	Esquema de flujo de información mediante Tópics en ROS	17
3.	Estructura de mensaje Image	18
4.	Estructura de mensaje Twist	19
5.	Esquema de desplazamiento de Bebop 2 con velocidades positivas	20
6.	Organización de directorios de bebop_autonomy	21
7.	Interfaz de conexión Bebop2 - ROS - PC para la obtención de imagen del dron.	22
8.	Parrot Bebop 2	23
9.	Captura del tablero de Trello	26
10.	La imagen muestra la evolución de la cantidad de FPS y tracks detectados a lo largo de los frames obtenidos. Como vemos, los FPS rondan los 0.7. Analizaremos esta gráfica en mayor profundidad más adelante en el apartado de experimentación.	30
11.	Esquema de nodos ROS en la aplicación	34
12.	Esquema de comunicación entre los módulos de la aplicación	35
13.	Ejemplo de archivo txt generado	38
14.	Rendimiento de YOLOv3 frente a otros algoritmos de detección. Paper: [1]	39
15.	Menú principal + Menú de pilotaje manual	40
16.	Diálogo de selección de objetivo	41
17.	Interfaz HUD (Head-Up Display) del modo pilotaje manual de la aplicación Free Flight Pro	56
18.	Interfaz HUD (Head-Up Display) del modo pilotaje automático de la aplicación Free Flight Pro	57
19.	Dron siguiendo al objetivo de manera activa	58
20.	Evolución de FPS y número de tracks a lo largo de los frames	59
21.	Estado del equipo portátil durante la ejecución de la aplicación	60
22.	FPS en modo manual	61
23.	FPS en modo automático	61
24.	Inicio del sistema	62
25.	Selección de ID en cámara fija	63
26.	Salida de escena	64
27.	Despegue automático	64
28.	Selección de objetivo en vídeo del dron	65
29.	Detección a contra luz - Bebop	65
30.	Detección a contra luz - Servidor	66
31.	Detección en alto contraste - Bebop	66
32.	Detección en alto contraste - Servidor	67
33.	Evolución en vídeo Bebop	68

34. Evolución en vídeo Cámara fija	69
--	----

Índice de cuadros

1.	Tabla de costes hardware.	32
2.	Tabla de costes software	32
3.	Tabla de costes totales.	33

1. Introducción

1.1. Objetivos

En este proyecto se realiza un estudio sobre la eficacia que presenta el dron Parrot Bebop 2 como elemento de un sistema de video vigilancia activo. Durante el desarrollo del trabajo se pretenden cumplir los siguientes objetivos:

- Estudiar el estado del arte de los drones y los sistemas de video vigilancia.
- Aprender las funcionalidades y herramientas de las que dispone el framework ROS que nos permiten abstraernos de la interconexión y comunicación de nodos heterogéneos.
- Acercarnos al mundo de la Inteligencia Artificial utilizando un detector y *tracker* basados en *deep learning*: YOLOv3 y DeepSORT respectivamente.
- Implementar un sistema de control manual para pilotar el dron.
- Implementar un comportamiento reactivo que permita al dron perseguir un determinado objetivo.
- Dar un enfoque más genérico al proyecto de manera que pueda aplicarse a más campos de trabajo y no sólo a la video vigilancia.
- Desarrollar software libre en forma de un paquete ROS alojado en GitHub.
- Aprender nuevos programas de planificación y comunicación como *Trello* y *Slack*.

1.2. Estado del Arte

Cada vez son más los sectores en los que el uso de drones se ha estandarizado y es que es una tecnología muy versátil que ha llegado para quedarse.

Uno de los campos en el que es más fácil ver su amplio uso es en la industria del cine o filmografía más *amateur* donde son muy utilizados para tomar amplios planos de un paisaje o para grabar persecuciones.

Actualmente se está intentando implantar su uso en empresas de reparto de *paquetes*, pero debido a la falta de legislación y a los grandes cambios de infraestructura que requiere, este cambio está tardando en llegar aunque, muchas empresas (entre las que se encuentra la española *Correos*) ya tienen sus propios modelos de drones listos.

Otro sector del que no puede faltar mención, es sin duda el más avanzado en el tema: el sector militar. Es aquí donde se producen las mayores inversiones en temas de investigación y desarrollo y donde podemos encontrar la tecnología más puntera. Desde drones de tamaño minúsculo, hasta drones gigantescos del tamaño de una avioneta.

También son utilizados como método de control de planes de urbanismo, tráfico, plantaciones agrícolas y, recientemente, se han empezado a utilizar también en sistemas de video vigilancia, ya que permiten obtener un estado de la situación prácticamente en tiempo real y nos brindan la oportunidad de movernos con libertad para poder adecuarnos mejor a la situación.

Estos son tan sólo unos pocos usos de todos los que nos podemos encontrar en la actualidad. Para ver hasta 38 situaciones en las que usamos hoy en día los drones recomiendo echar un vistazo a este artículo de *CB INSIGHTS* [2] que ofrece una visión general del tema sin entrar mucho en detalle.

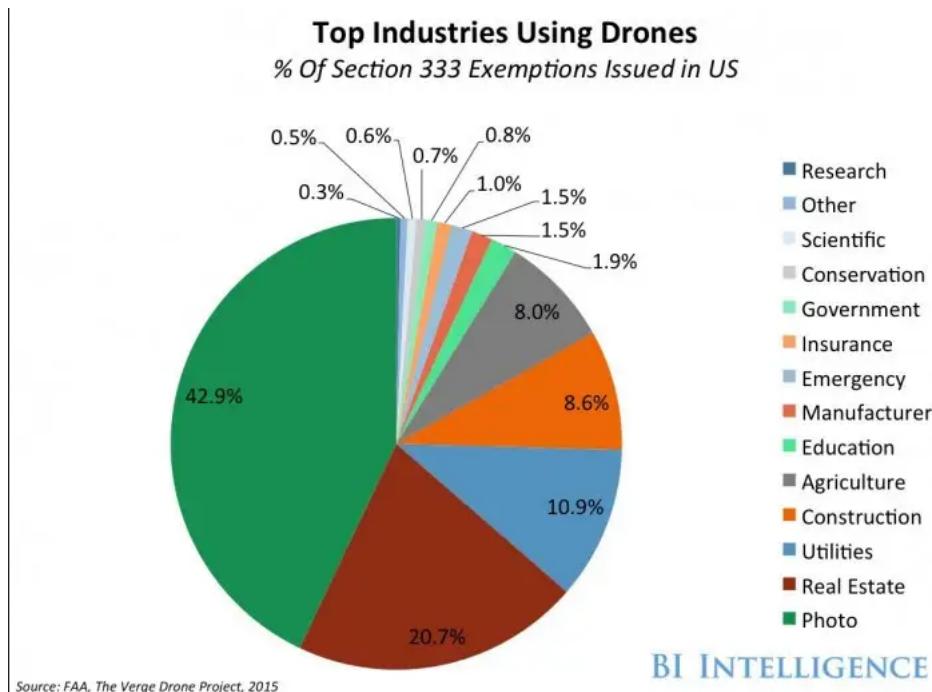


Figura 1: 2015 - Uso de drones en la industria

La figura expuesta en el artículo [3] de *Business Insider* recoge los datos de la *Federal Aviation Administration - FAA* [4] de 2015 sobre el uso de los drones en las distintas industrias en Estados Unidos.

Como podemos ver, el mayor uso de drones (42.9 %) se da en la industria de la fotografía como hemos hablado antes. En segunda posición con apro-

ximadamente la mitad de uso (20.7 %), aparece el sector relacionado con propiedades inmueble y planes de urbanismo. El tercer puesto lo ocupa el sector de servicios públicos y supone la mitad de uso respecto al segundo puesto (10.9 %). En el cuarto y quinto puesto aparecen los sectores de construcción y agricultura respectivamente, siendo el uso en el sector de construcción mínimamente superior (0.6 % mayor). El uso en el resto de sectores es mínimo **lo cual es una pena ya que entre ellos se encuentran investigación y ciencia (0.3 % y 0.6 % respectivamente)**, sectores que impulsarían y mejoraría esta tecnología con mayor rapidez.

1.3. Motivación

Este trabajo de fin de grado está enfocado al sector de *video vigilancia*, más específicamente a la video vigilancia activa de infraestructuras críticas, es decir, utilizar uno o más drones como sistema de vigilancia de una determinada zona o edificio que, por norma general, se encontrará aislado, será poco accesible y, por tanto, será poco frecuentada por personas. Además su seguridad, integridad y buen funcionamiento son de una **importancia relativamente alta**.

El hecho de que se encuentre aislada y poco accesible es lo que hace muy interesante la incorporación de un sistema inteligente para vigilar el lugar. Esto es porque, actualmente, son personas que se encuentran físicamente allí, las que se encargan de la vigilancia bien mirando las pantallas que muestran la imagen de cámaras estáticas o bien patrullando la zona. Estas dos tareas asociadas a unos cambios de guardia poco frecuentes pueden generar fatiga en los trabajadores y dar lugar tanto a falsos positivos como a **falsos negativos**.

En el peor de los casos (falso negativo) pondremos en riesgo la integridad de la zona, y deberemos asumir las repercusiones que puede tener para la sociedad (por ejemplo la caída de una central eléctrica puede suponer que una parte de la población se quede sin luz), además de los costes de arreglar los daños ocasionados.

En el caso más leve (**false** positivo) supondrá el despliegue de un equipo policial hasta la zona y acarrear **dichos costes**.

Todo esto nos hace pensar que sería muy útil un sistema inteligente que nos asegure una amplia probabilidad de acierto a la hora de detectar intrusos. Supondría un ahorro de personal, y su efectividad sería constante ya que no se vería afectado por ningún tipo de fatiga. De esta manera podríamos prepararnos ante posibles falsos positivos y buscar un suplemento (si es que fuera necesario), para los falsos negativos. Además al incorporar un elemento activo como es el dron, nos abre la posibilidad de disminuir el número de cámaras fijas en el entorno y seguir la pista de un posible intruso hasta zonas más inaccesibles para informar de su posición en tiempo real.

2. Entorno Operacional

Vamos a hablar de ROS (el framework utilizado), la versión de Ubuntu seleccionada, así como de los componentes hardware involucrados más destacables como es el ordenador que ha actuado como servidor y centro de cómputo y el dron Parrot Bebop 2. En definitiva vamos a tratar los puntos mencionados en el primero de los dos grandes grupos de **objetivos** mencionados en 3.4.

En esta sección aprovecharé para hablar de los problemas con los que nos hemos topado ya que la gran mayoría de ellos se han dado durante la preparación de este escenario base sobre el que desarrollar el proyecto.

2.1. Ubuntu 16.04

El proyecto se ha desarrollado en esta distribución de Linux por dos decisivos motivos. El primero es que ROS ofrece una de sus versiones más robustas para esta distribución: ROS Kinetic. Esta versión de ROS tiene una gran comunidad de soporte y por tanto podemos encontrar gran cantidad de paquetes, bibliotecas y ayuda para casi cualquier problema que tengamos. El segundo motivo ha sido el que realmente ha volcado la balanza hacia ROS Kinetic y no hacia cualquier otra versión, y es que el paquete que ofrece los drivers para controlar a nuestro dron se ha desarrollado en esta versión de ROS y es donde su funcionamiento está garantizado.

2.2. Robotic Operating System

Este framework libre, más conocido por su acrónimo ROS [5] [6], está orientado hacia el desarrollo de aplicaciones para robots. Nos brinda una gran colección de bibliotecas y herramientas que nos permiten desarrollar software complejo que garantice un comportamiento robusto en los robots.

ROS surge como la necesidad de estandarizar la manera de afrontar la programación del comportamiento de los robots. Hasta la tarea más trivial desde una perspectiva humana nos resultará compleja de diseñar e implantar en un robot en cuanto empecemos a tener en cuenta la enorme cantidad de factores y decisiones que entran en juego y que nosotros, como humanos, tomamos casi inconscientemente.

Por su filosofía de software libre, cualquiera puede contribuir en ROS creando paquetes que resuelvan problemas "*generales*" de manera que otra persona pueda usarlo para resolver otro problema más complejo e ir escalando en la dificultad. Así un laboratorio formado por expertos podría desarrollar un paquete dedicado a mapear interiores, otro equipo podría usar ese paquete para crear orto que permita desplazarse por el mapa y un tercero que implemente visión por computador para reconocer objetos mientras se mueve.

2.2.1. Conceptos

Hay una serie de conceptos básicos de ROS que es necesario definir para entender bien su funcionamiento:

- **Paquetes:** Son la unidad básica de organización de ROS. Un paquete puede contener nodos, una biblioteca independiente, un conjunto de datos, archivos de configuración, un software de terceros o cualquier otra cosa que constituya un módulo útil. El objetivo de estos paquetes es proporcionar esta funcionalidad útil de una manera fácil de consumir para que el software pueda reutilizarse fácilmente. La definición de un paquete de ROS perfecto es aquel que aporta suficiente funcionalidad para resultar útil pero que no pesa demasiado para resultar difícil de usar en otro proyecto.
- **Nodos:** Son los procesos de ROS que **realizan computación**. Los nodos pueden comunicarse entre sí por medio de *stream topics* o tópicos de transmisión, servidores RPC (Remote Procedural Call) o servidores de parámetros.
- **Temas(Tópicos):** Los *topics* son **buses** etiquetados sobre los que los nodos pueden escribir y leer mensajes. Las publicaciones y suscripciones en los buffers son anónimas de manera que un nodo con el rol de "publisher" no sabe con qué otro nodo se está comunicando. Puede haber varios suscriptores y publicadores en un mismo buffer pero la comunicación siempre es unilateral. 2

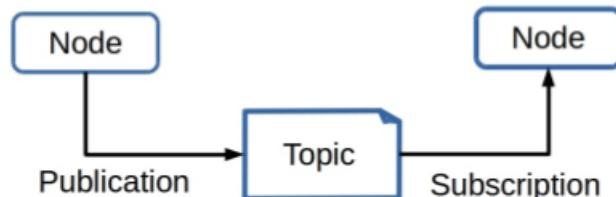


Figura 2: Esquema de flujo de información mediante Tópicos en **ROS**

- **Servidores RPC:** El paradigma de los *topics* es muy flexible pero la comunicación unidireccional puede llegar a ser un problema. Para las comunicaciones bidireccionales utilizamos los servidores RPC en los que un nodo proveedor ofrece un servicio al que un nodo cliente llama enviando un mensaje y esperando una respuesta.
- **Servidores de Parámetros:** Un servidor de parámetros es un diccionario compartido al que se puede acceder a través de las API de red. Los nodos usan este servidor para almacenar y recuperar parámetros

en tiempo de ejecución. Como no está diseñado para un alto rendimiento, se utiliza mejor para datos estáticos, no binarios, como los parámetros de configuración. Está destinado a ser visible globalmente para que las herramientas puedan inspeccionar fácilmente el estado de configuración del sistema y modificarlo si es necesario.

- **Mensajes:** Son las estructuras de datos con la que se comunican los nodos. Existen varios tipos pero los que utilizamos especialmente en el proyecto son *sensor-msgs/Image.msg* 3 y *geometry-msgs/Twist.msg* 4 que presentan la siguiente estructura:



Figura 3: Estructura de mensaje **Image**

- **header** es otro tipo de mensaje ROS que contiene un identificador del frame asociado, un contador que indica su posición en la secuencia de imágenes y una marca de tiempo que indica en qué momento se ha captado la imagen.
- **height** y **width** hacen referencia a la resolución de la imagen (W x H píxeles)
- **encoding** es el "formato.^{en} el que se representa internamente la imagen. Número de canales, su orden y significado de cada uno.
- **is_bigendian** es suficientemente autodescriptivo.

- **step** indica en bytes la longitud que hay entre el primer píxel de una fila de la imagen y el primer píxel de la siguiente fila. Dado que los canales de color y la resolución pueden variar, step es un atributo muy útil para recorrer las filas de una imagen.
- Por último **data** contiene la matriz que representa la imagen.

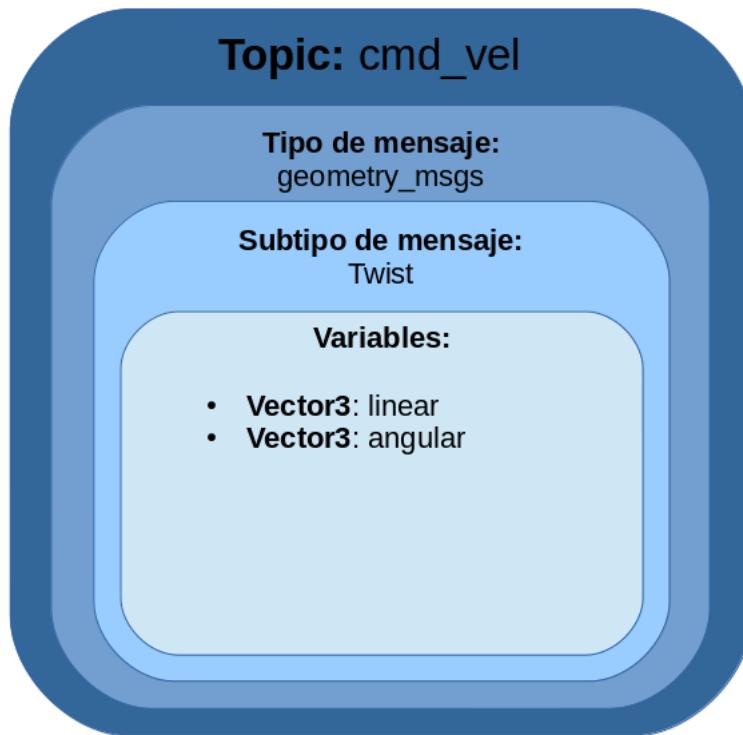


Figura 4: Estructura de mensaje **Twist**

Vector3 es un tipo de dato que consta de tres atributos de tipo float64 (x,y,z) para representar vectores en un espacio 3D. Para controlar el dron Bebop 2 debemos asignar valores a estos atributos en el dominio [0,1] siendo 0 la ausencia de velocidad y 1 la entrega de máxima potencia.



Figura 5: Esquema de desplazamiento de Bebop 2 con velocidades **positivas**

- **Bolsas(Bags):** Se trata de un formato especial que permite al usuario la opción de guardar o reproducir mensajes de datos de ROS.
- **Comandos:** Son una serie de órdenes para navegar por el sistema de ficheros y modificar o mostrar tópicos y mensajes.

2.3. Driver *bebop_autonomy*

bebop_autonomy es un paquete de ROS que actúa como driver para los drones Bebop 1 y Bebop 2. Está basado en el SDK *ARDroneSDK3* oficial de Parrot. Este driver ha sido desarrollado por el *autonomy lab* en la universidad Simon Fraser por Mani Monajjemi además de otros colaboradores [7].

Al ejecutar el driver *bebop_autonomy* se crean una serie de Topics a los que nos podemos suscribir para recibir información como */bebop/image_raw* del que leemos las imágenes captadas por la cámara, o */bebop/states/common/CommonState/BatteryStateChanged* que nos informa de cambios en el estado de la batería.

También se crean Topics en los que debemos hacer publicaciones para controlar el dron como */bebop/cmd_vel* al que indicamos en qué dirección debe moverse o rotar y con qué velocidad, o */bebop/takeoff* y */bebop/land* a los que debemos mandar un mensaje estandar vacío (tipo Empty) para

hacer despegar o aterrizar al dron respectivamente. En el proyecto también se hace uso del topic `/bebop/reset` en el cual, mandando un mensaje tipo Empty, provocaremos la detención inmediata del dron (modo de emergencia). El topic `/bebop/record` nos sirve para iniciar o detener la grabación de lo que capta la cámara (se graba en la memoria a bordo del dron).

Para ejecutar el driver escribimos lo siguiente:

```
$ rosrun bebop_driver bebop_nodelet.launch
```

Este comando inicia los topics y prepara la comunicaciones con el dron. Si suponemos que a nuestro directorio de trabajo lo hemos llamado 'TFG' la organización de carpetas sobre la que se encuentran los drivers tendrá la siguiente forma:

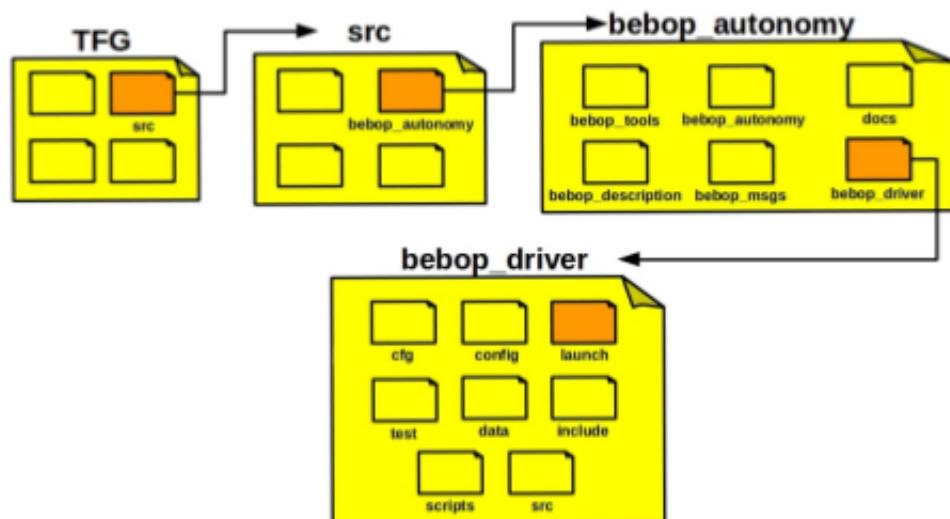


Figura 6: Organización de directorios de `bebop_autonomy`

Existe otro driver que a parte de iniciar los topics, se suscribe a uno de ellos (`/bebop/image_raw`) y abre una ventana por la que muestra la imagen que recibe. Para utilizar este driver tecleamos en la terminal lo siguiente:

```
$ rosrun bebop_tools bebop_nodelet_iv.launch
```

Este *launch file* es un buen primer contacto con *bebop_autonomy* ya que nos permite ver de manera mucho más visual que la instalación, tanto de ROS como del paquete de drivers, ha tenido éxito y que la interfaz de conexión con el dron funciona y recibimos imagen.

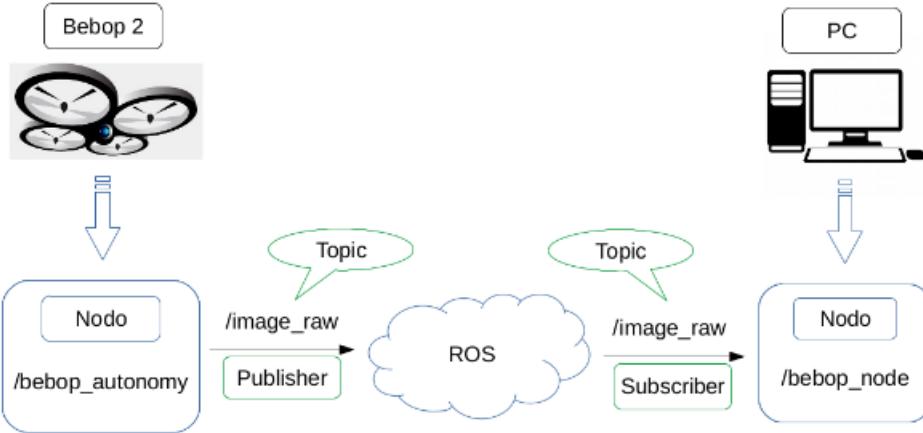


Figura 7: Interfaz de conexión Bebop2 - ROS - PC para la obtención de imagen del dron.

2.3.1. Comandos ROS

- **roscore:** Inicializa el núcleo de ROS y permite la comunicación entre nodos, servicios, parámetros, etc.
- **roscreate-pkg:** Con esta herramienta podemos crear paquetes ROS, pudiendo además especificar las dependencias de los mismos.
- **rospack:** Muestra información de un paquete.
- **rosstack:** Información de la “pila” que se deseé.
- **roscd:** Permite movernos dentro del directorio de trabajo.
- **rosls:** Muestra el contenido de la carpeta que indiquemos dentro del directorio de trabajo.
- **rosnode:** Muestra una lista los nodos en ejecución, información de un nodo o permite eliminar nodos de la ejecución.
- **rosdep:** Permite descargar dependencias de paquetes.
- **rostopic:** Herramienta útil para controlar Topics. Se puede obtener una lista de todos los Topics en ejecución, así como publicar mensajes en ellos, obtener información o leer los datos de dichos Topics.
- **rosservice:** Al igual que los Topics, se puede obtener una lista de los distintos servicios que se están ejecutando, información y utilizar dichos servicios para un fin.

- **rosmsg:** Obtiene la información de un tipo de mensaje.
- **rosrun:** Ejecuta un archivo ejecutable de un paquete específico.
- **roslaunch:** ROS permite ejecutar varios nodos a la vez. Mediante esta herramienta se puede ejecutar archivos con formato .launch, en estos archivos se puede configurar varios nodos para poder ejecutarlos a la vez facilitando al usuario el uso del sistema global de ROS.

2.4. Bebop 2



Figura 8: Parrot Bebop 2

La imagen muestra el modelo de dron utilizado para desarrollar el proyecto y hacer las distintas pruebas y experimentos. Se trata de la segunda revisión del modelo Bebop (Bebop 2) de la marca Parrot [8].

2.4.1. Especificaciones

- **Imagen:** Cuenta con una cámara de **14.0 Megapíxeles** que permite hacer fotografías con una resolución de **3800 x 3188 píxeles** y grabar vídeo a **30 fotogramas por segundo** en **full HD** (1920 x 1080 píxeles). Además cuenta con **estabilización digital** en los tres ejes un ángulo de visión de **180º**. Estas resoluciones de imagen las obtenemos al grabar los datos en la memoria interna del propio dron, pero si enviamos los datos a través de la interfaz wifi mediante la que nos conectamos, obtenemos una resolución de imagen de 856 x 480 píxeles. Esto se debe principalmente a que necesitamos tener un tiempo de transmisión de imágenes pequeño que nos garantice una comunicación fluida con el dron; si transmitiera imágenes en full HD es posible que la

latencia de transmisión y postprocesamiento de la imagen introdujera un delay incómodo a la hora de controlar el dron.

- **Conectividad:** Es un quadrotor que se comunica vía **Wi-Fi** con smartphones o tablets por medio de una aplicación propia tanto en la banda de **2.4GHz** como en la **5GHz** y da la opción de seleccionar el canal por el que transmitir para asegurarnos de escoger el que menos interferencias tenga. También nos permite establecer una contraseña con seguridad WPA2 para evitar que otra persona se conecte al dispositivo.

Cuenta con GPS pero desarrollaremos esta característica en el apartado de *movimiento*.

También dispone de un puerto **micro USB** para conectarlos al microcontrolador de abordo para poder actualizar o modificar el firmware del dispositivo.

- **Movimiento:** Alcanza unas velocidades máximas de **60km/h en horizontal** y **21km/h en vertical** en 14 segundos y es capaz de frenar por completo en 4 segundos. Además soporta ráfagas de viento de hasta 60km/h.

Tiene su propio sistema de **tracking** que funciona realmente bien. El movimiento autónomo es muy preciso y se puede ajustar la distancia que separa al dron del objeto de grabación así como las velocidades de rotación y movimiento lineal. Además existen kits de **FPV** para volar el dron de manera manual a mayor distancia que permiten ver desde la perspectiva del dron.

Incluye su propio sistema **GPS** lo cual permite hacer **planings** de vuelo sobre el terreno y que el dron ejecute los movimientos secuencialmente hasta acabar el recorrido. También permite la opción *"volver a casa"* que hace volver al dron hasta el punto donde inició el vuelo.

▪ **Hardware y características físicas:**

- Batería de 2700mAh que suponen 25min de autonomía
- Hélices flexibles que se bloquean en caso de contacto
- LED trasero visible a larga distancia
- GPS
- Procesador de 2 núcleos
- GPU de 4 núcleos
- 8GBs de memoria flash
- 4 motores sin escobillas
- Peso de 500g
- Estructura de fibra de vidrio y grilamid

2.5. Ordenador

La instalación y ejecución del proyecto se ha realizado en un ordenador de sobremesa con las siguientes características:

- CPU: i7-4970K
- GPU: NVIDIA GTX 970
- RAM: 16GB DDR3
- WIFI: Banda dual 2.4GHz y 5GHz

Lo más apropiado para el proyecto sería utilizar un ordenador portátil para poder tener mayor movilidad a la hora de volar el dron, pero debido a que el equipo con el que contábamos no disponía de tarjeta gráfica, tuvimos que optar por utilizar el ordenador de sobremesa.

La tarjeta gráfica es un componente esencial en este proyecto ya que es la encargada de calcular las detecciones y **tracks** en las imágenes obtenidas del dron. Este proceso es, computacionalmente, muy costoso; de hecho, en el equipo descrito, se ha conseguido un máximo de **13fps** durante la ejecución teniendo una media de 12fps. Esta cantidad es suficiente para probar el proyecto y realizar pequeños experimentos pero se recomienda utilizar tarjetas gráficas más potentes o un mayor número de ellas. En la web oficial de YOLOv3 [9] nos indican que, con una gráfica Pascal Titan X, se obtienen 30 **FPS** en el procesado del subconjunto de prueba *test-dev* del conjunto de datos *COCO* con una precisión media (mAP) del 57.9 %.

3. Gestión del Proyecto

3.1. Metodología de Desarrollo

Se ha utilizado la metodología de *Desarrollo de Prototipos* [10] en la que nos centramos en construir prototipos o versiones funcionales del proyecto de manera asidua para poder obtener una retroalimentación e ir refinando así el resultado final.

Esta metodología resulta muy útil cuando conocemos los objetivos generales que queremos alcanzar pero no identificamos con exactitud los requisitos de entrada, procesamiento o salida. También ofrece un mejor enfoque cuando estamos inseguros sobre la eficacia de algún algoritmo (como nos ocurre con DeepSORT), sobre la adaptabilidad de un sistema operativo o, en este caso, el framework ROS, o si no tenemos clara la forma de interacción humano-máquina (modo de pilotaje manual).

Por todos estos motivos nos ha parecido un modelo de desarrollo bastante adecuado que, además, nos permite saber si progresamos adecuadamente.

3.2. Planificación

Para organizar el proyecto se ha hecho uso de la plataforma *Trello* [11] conectada con la herramienta de comunicación en equipo *Slack* [12] de manera que tanto yo, como mis tutores tenemos acceso a la planificación y avances del trabajo. Además, también conectado al canal de Slack, se encuentra el repositorio del proyecto alojado en GitHub [13].

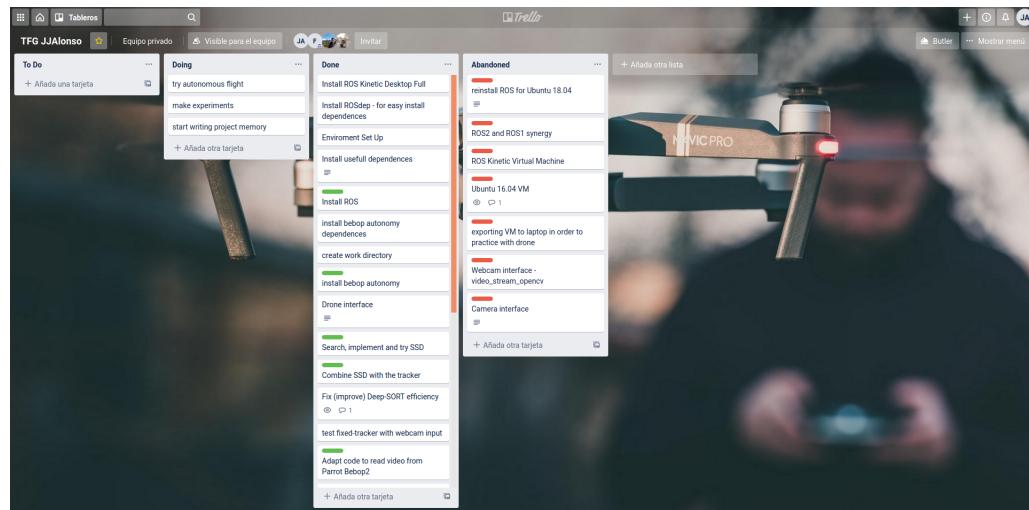


Figura 9: Captura del tablero de Trello

3.3. Restricciones del Sistema

3.3.1. Restricciones Harware

- El rango de conexión Wi-Fi es limitado. La área en la que podremos volar el dron será, aproximadamente, una circunferencia de 30 metros de radio con centro el servidor.
- **El servidor**, en mi caso, es un ordenador de sobremesa por lo que no puedo transportarlo a mi gusto y llevarlo a zonas amplias y despejadas en las que volar el dron, si no cuento con un suministro eléctrico.
- La iluminación debe ser adecuada para permitir que la detección y tracking sean adecuados.
- El dron Bebop funciona con baterías por lo que, eventualmente, habrá que cambiarla para proseguir con la ejecución o, en caso de no disponer de una de repuesto, inhabilitar el sistema hasta que se haya recargado.
- Necesitamos una gráfica NVIDIA para poder realizar los cálculos de los algoritmos de visión por computador.
- La interacción humano-máquina se ha desarrollado a través de la terminal de Linux por lo que obligatoriamente debemos disponer de un teclado.

3.3.2. Restricciones Software

- Para ejecutar el proyecto es necesario hacerlo sobre ROS Kinetic.
- ROS Kinetic solo está disponible para Ubuntu 16.04 por lo que deberemos disponer de este Sistema Operativo.
- Necesitamos Python 2.7 para la ejecución del proyecto tanto para la parte de visión por computador como para la que implementa el comportamiento del dron.

3.4. Hitos

Los **objetivos** han variado mucho a lo largo del desarrollo ya que se han ido planteando distintos enfoques hasta dar con el más indicado, sencillo y efectivo. Por ello los objetivos aquí definidos son los correspondientes con el desarrollo final. En un posterior apartado se explicarán las distintas maneras que se han estudiado para afrontar el proyecto y por qué finalmente se dejaron de lado.

1 Instalar Ubuntu 16.04

Es la distribución de linux en la que se encuentra disponible ROS Kinetic.

2 Instalar ROS Kinetic

Es la versión de ROS en la que funciona el driver Bebop Autonomy.

3 Instalar Bebop Autonomy

Es el driver que nos permitirá establecer conexión con el dron y comunicarnos con él.

4 Escoger algoritmo de tracking de personas

Buscar un algoritmo de **tracking** que funcione en nuestra instalación.

5 Crear una interfaz para leer imágenes del dron

Desarrollar una interfaz software que transforme los mensajes ROS en datos útiles que podamos utilizar para el desarrollo.

6 Desarrollar un control manual para el dron

Implementar un modo de vuelo manual para realizar una primera toma de contacto al enviar comandos de movimiento al dron. También nos permite tomar el control del aparato en caso de que haya algún problema.

7 Desarrollar un control autónomo para el dron

Desarrollar el objetivo principal del proyecto. Dotar al dron de un comportamiento reactivo para perseguir a un objetivo designado.

8 Implementar paquete ROS con el proyecto

Transformar el repositorio del proyecto en un paquete ROS de manera que se pueda usar mediante comandos del framework.

9 Crear launchfile para ejecutar el proyecto desde ROS

Crear un archivo ejecutable por ROS que lance automáticamente el driver del dron y el software del proyecto.

Estos objetivos podemos sintetizarlos en dos grandes grupos:

- Conseguir un sistema estable en el que llevar a cabo el desarrollo
Objetivos: 1,2,3
- Establecer una conexión dron-servidor, y programar el comportamiento del dron
Objetivos: 4,5,6,7,8,9

3.5. Problemas

A lo largo del proyecto nos hemos encontrado con diversos problemas de naturaleza tanto física como a nivel de **software**. Los descritos a continuación son los que han ocasionado largas pausas durante el desarrollo y que han supuesto un mayor trabajo de investigación para intentar solventarlos.

3.5.1. Virtualización

En un primer momento si intentó desarrollar el proyecto en una máquina virtual con *VirtualBoX* [14] pensando en la portabilidad del proyecto y la posibilidad de ejecutarlo en un equipo Windows. El problema de la virtualización es que no podemos aprovechar los recursos en su totalidad y en especial el uso de la tarjeta gráfica y la instalación de *CUDA* [15] es realmente complicado desde una VM con Ubuntu 16.04. Estas trabas durante la instalación de controladores y paquetes, y la incomodidad intrínseca de trabajar en una VM, hicieron que descartara esta idea.

3.5.2. Equipo de Trabajo

Como se ha sugerido en 2.5 lo más apropiado es realizar la instalación del proyecto en un equipo portátil para poder tener mayor libertad de movimiento. Fue así como se inició el trabajo. El portátil consta de:

- CPU: i5-7200U
- GPU: integrada
- RAM: 4GB DDR3
- WIFI: Banda dual 2.4GHz y 5GHz

El gran problema, como dije antes, es que carece de una tarjeta gráfica que realice los cálculos de detección y tracking. Ejecutando el proyecto final en este ordenador hemos llegado a obtener picos de 0.8fps pero la media ronda los 0.7fps. Claramente esto es insostenible y es imposible realizar

pruebas y experimentos si no llegamos ni a recibir 1 frame por segundo. La reactividad del dron es inexistente.

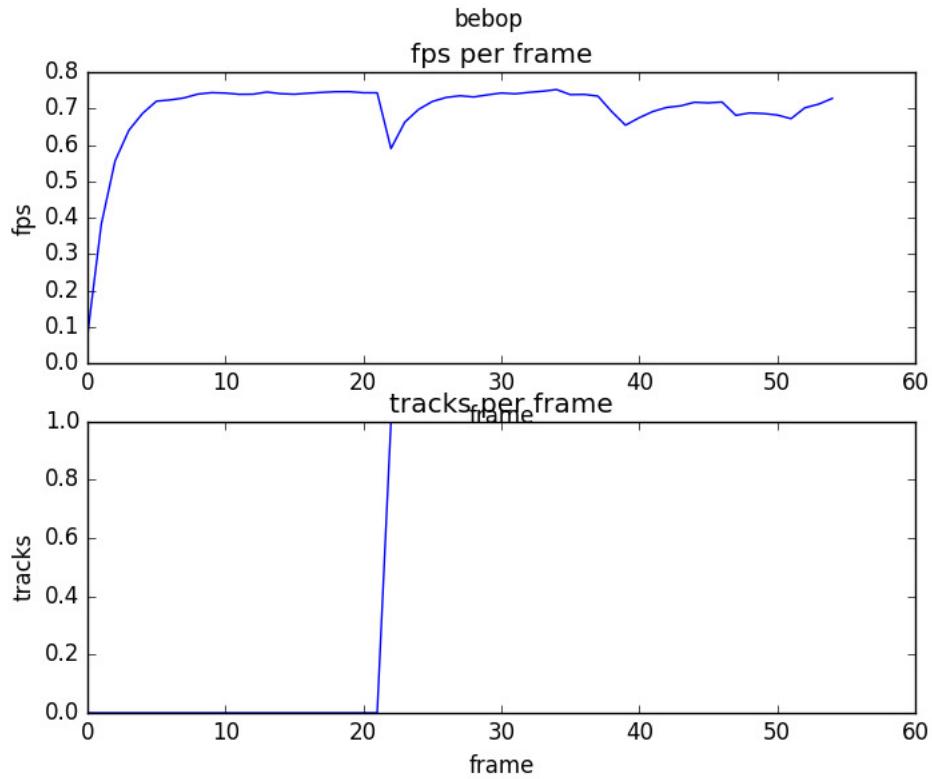


Figura 10: La imagen muestra la evolución de la cantidad de FPS y tracks detectados a lo largo de los frames obtenidos. Como vemos, los FPS rondan los 0.7. Analizaremos esta gráfica en mayor profundidad más adelante en el apartado de experimentación.

Una posible solución a estudiar sería realizar el cómputo más pesado en la nube: alquilar un servidor que cuente con herramientas que permitan la ejecución de algoritmos de *Machine Learning* (que cuenten con bibliotecas específicas como tensorflow o keras, y unas prestaciones hardware a la altura de las necesidades del proyecto). Respecto a esta opción, habría que estudiar cómo realizar el intercambio de información (tipos de datos, cantidad de información, frecuencia de peticiones...) y cómo afectaría esta implementación a los tiempos de respuesta del dron.

Este enfoque no se ha llevado a cabo debido al poco tiempo disponible y a limitaciones económicas, pero es una opción interesante que me gustaría retomar.

Finalmente, para continuar con el desarrollo, adquirí una tarjeta Wi-Fi para el ordenador de sobremesa. Con ella fue posible conectarnos de nuevo al dron y continuar con las pruebas.

3.5.3. Límite de Distancia de Conexión

Con la resolución del punto anterior nos surge un nuevo problema: dado que nuestro equipo servidor es local y estático, las posibles zonas de vuelo del dron se reducen a los alrededores de mi vivienda y, aunque por suerte, hay un descampado justo al lado, la distancia dron-servidor es límite, provocando que eventualmente se pierda la conexión durante las pruebas.

Esta situación de experimentación ha sido excepcional, al igual que lo ha sido la situación sanitaria en la que nos hemos visto envueltos estos últimos meses. El desarrollo de las pruebas debería haberse realizado en el **CITIC [16]**, que cuenta con una jaula específica para el vuelo de drones.

3.5.4. Python y DeepSORT

ROS Kinetic trabaja con python2.7, el cual ha dejado de tener soporte desde principios de este año 2020. Esto ha supuesto muchos problemas ya que los algoritmos de detección y tracking desarrollados para Python2.7 tienen dependencias de paquetes que ya no es posible descargar. Finalmente encontramos un algoritmo de detección + tracking casi totalmente funcional con python2: una implementación del algoritmo Deep SORT [17] por parte del usuario *Qidian213* en GitHub [18]. El algoritmo nos da un error por la falta de una biblioteca pero esto está resuelto en uno de los *issues* del repositorio [19].

Antes de encontrar este código intentamos dar otra solución, específicamente la comentada en el siguiente apartado.

3.5.5. Instalación de ROS2 junto a ROS Kinetic

Como hemos dicho, Python2 ha dejado de tener soporte y gran cantidad de software ha quedado obsoleto debido a ello. Además, se ha desarrollado nuevo software y nuevas versiones de antiguos algoritmos para Python3 (que si cuenta con soporte) mucho más eficientes.

Por esto tratamos de construir el proyecto sobre un entorno que nos brinda una mayor libertad en el desarrollo. Este entorno se trata del framework ROS2, la nueva versión de ROS que permite trabajar, como decimos, con Python3 y que además está disponible para versiones de Ubuntu más actualizadas, con mayor número de paquetes, y que ofrecen soporte a más largo plazo: Ubuntu 18.04 y Ubuntu 20.04.

El problema comienza a la hora de instalar el driver *Bebop Autonomy* ya que sólo está disponible para la versión *Kinetic* de ROS. Para intentar suplir esto, se intentó realizar una instalación conjunta de ambas versiones del framework ROS y ROS2 y establecer un puente entre ellas de manera que el driver del dron se ejecutara en ROS y el algoritmo reactivo en ROS2 haciendo uso de Python3.

Tras numerosos intentos fallidos tanto en la instalación conjunta de los frameworks, como en el establecimiento del puente entre los mismos, decidimos volver al enfoque inicial: buscar un tracker o, en su defecto, desarrollarlo, para que funcione con python2 y así poder usarlo en una instalación limpia de ROS Kinetic.

3.6. Presupuesto

3.6.1. Recursos Hardware

El presupuesto de este proyecto se ha calculado de la siguiente manera:

$$(D/V) * C$$

Donde:

- D es el número de meses que se ha utilizado el producto para el proyecto.
- V es la vida útil del producto en **número de meses**.
- C es el coste total del producto en euros.

Artículo	Coste (€)	Dedicación (meses)	Vida Útil (meses)	Coste Aplicable (€)
Bebop2	550	4	120	18.3
Ordenador de sobremesa	1200	4	60	80
Total				98.3

Cuadro 1: Tabla de costes hardware.

3.6.2. Recursos Software

Artículo	Coste (€)
Ubuntu 16.04	0.0
ROS	0.0
Visual Studio Code	0.0
Total	0.0

Cuadro 2: Tabla de costes software

3.6.3. Recursos Humanos

Este trabajo ha requerido, aproximadamente, 100 días de trabajo en los que se han invertido unas 4 horas diarias.

Suponiendo que un ingeniero informático recién graduado en España representa un coste de 2000 euros al mes teniendo una jornada laboral de 8 horas diarias (aproximadamente 12.5€/h), las 400 horas del proyecto serían valoradas en 5000€.

3.6.4. Coste Total

COSTE TOTAL	
Recursos Harware	98.3€
Recursos Software	0.0€
Recursos Humanos	5000€
Coste Total	5098.3€

Cuadro 3: Tabla de costes totales.

Si añadimos un 10 % referido a gastos de luz, recambio de piezas, nuevos componentes para el ordenador, etc. el coste total del proyecto ascendería a:

PRECIO TOTAL: 5608.13€

4. Diseño

La aplicación cuenta con únicamente dos nodos ROS. Uno implementa los drivers del dron y el segundo contiene el proyecto en sí mismo.

Este último nodo cuenta con el software de visión por computador, el software que implementa el comportamiento reactivo, y un pequeño script que nos permite capturar el vídeo de una webcam. Estas tres funcionalidades tan bien diferenciadas podrían (y deberían) haber sido separadas en tres nodos diferentes. ¿Por qué no se hizo entonces? Como hemos mencionado anteriormente, el proyecto se comenzó a desarrollar en un portátil de limitada potencia. El aumento de nodos implica un aumento de procesamiento paralelo que añadido a la sobrecarga intrínseca de la CPU al realizar el procesamiento gráfico, sería fulminante para el rendimiento. Por ello se diseñó el sistema con únicamente dos nodos (los estrictamente necesarios) y se mantuvo así para proceder lo antes posible con la implementación del código y estar seguros de tener una versión funcional del proyecto a tiempo para la fecha de entrega.

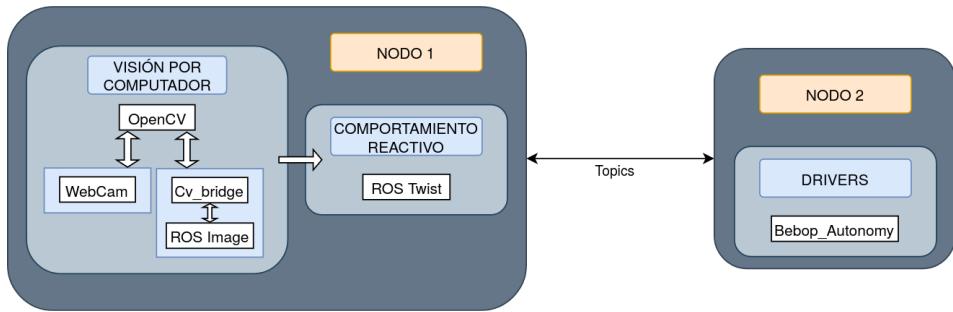


Figura 11: Esquema de nodos ROS en la aplicación

Ambos nodos son emisores y receptores de mensajes ROS dependiendo del *topic* en cuestión.

- El *NODO 1* publica en los *topics*: cmd_vel, takeoff, land, reset y record. A su vez está suscrito a: imageraw y BatteryStateChanged.
- El *NODO 2* tiene la configuración contraria al *NODO 1*. Esta suscrito donde él publica y viceversa.

La comunicación entre los módulos funcionales de la aplicación es cíclica siendo el nodo del controlador el que inicia el intercambio de mensajes.

El nodo que implementa el driver transmite la imagen captada por el dron al otro nodo del sistema. Aquí entra al módulo de Visión por Computador donde se procesa realizando la detección y tracking de las personas que aparezcan en escena. Posteriormente se extraen una serie de atributos referidos a las medidas y coordenadas de los Bounding Boxes y al identificador asignado a cada uno de ellos. Estos datos son pasados como parámetros al software de comportamiento del dron donde se tomarán las decisiones de movimiento oportunas basándose en la valoración de los datos obtenidos.

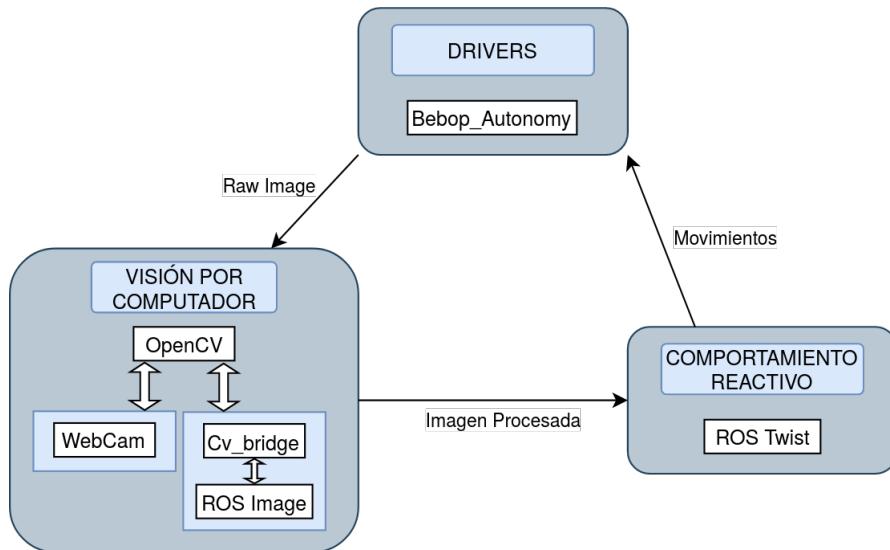


Figura 12: Esquema de comunicación entre los módulos de la aplicación

Si incluimos el análisis del video de la webcam lo que hacemos es realizar el tracking también de esta escena, permitiendo seleccionar un objetivo de manera que si este se sale del campo de visión, se manda una orden de despegue al dron.

Si utilizásemos un sistema de posicionamiento 3D que nos proporcionara la localización espacial del objetivo, podríamos llevar al dron directamente hasta la zona y empezar la búsqueda y el seguimiento activo. Este es, realmente, el enfoque más interesante del proyecto ya que se aproxima más a la idea de un sistema de video vigilancia plenamente autónomo en el que no se depende de ninguna persona física que inicialice el vuelo.

Como no disponemos de dicho sistema de posicionamiento nos remitimos, únicamente, a despegar el dron.

Si utilizamos el modo de pilotaje manual, los módulos de *Visión por Computador* y *Comportamiento Reactivo* quedan casi totalmente inutilizados. El módulo de visión tan sólo se encarga de mostrar por pantalla la cámara del dron para poder pilotarlo, y el módulo de comportamiento habilita una interfaz de usuario para poder indicar qué movimiento debe realizar el dron.

5. Implementación

Nos vamos a centrar en explicar la implementación de las funcionalidades que presenta el *NODO 1* de la figura 11 ya que es el que se ha desarrollado a lo largo del proyecto.

5.1. Visión por Computador

En este módulo nos encontramos con varias funcionalidades. Vamos a tratarlas en el orden en que se ejecutan cuando iniciamos la aplicación.

5.1.1. Captación de imágenes del dron

Para poder trabajar con las imágenes captadas por el dron debemos suscribirnos al topic `/bebop/image_raw` mediante la biblioteca *rospy* de python.

```
rospy.Subscriber("/bebop/image_raw", Image, callback_function)
```

Con esta línea nos suscribimos al topic y obtenemos en la función *callback_function* el mensaje ROS tipo `Image` pero este no es un formato válido con el que podamos trabajar. Es aquí donde entra en juego la biblioteca *cv_bridge* que obtiene, del mensaje `Image`, la imagen en sí misma mediante el método *imgmsg_to_cv2*.

```
self.bridge = CvBridge()
img = self.bridge.imgmsg_to_cv2(data, "bgr8")
```

De esta manera la imagen *img* se actualiza cada vez que hay una nueva publicación en el topic por parte del dron. Si quisieramos procesar todos y cada uno de los frames que obtenemos del bebop debemos guardar las imágenes obtenidas a medida que las transformamos. Debemos almacenarlas porque casi con total seguridad, la velocidad de captación de imágenes será mayor que la velocidad a la que las procesamos. Lo más lógico es usar un contenedor tipo cola dada su naturaleza para mantener el orden de los datos insertados y su estructura FIFO (First In First Out). En nuestro caso, no queremos procesar todos los frames ya que esto puede generar un retraso entre el frame que se está procesando y el frame recién captado que, además, irá aumentando durante la ejecución. Esto hará que la reactividad del dron disminuya o que, directamente, desaparezca. Por esto, la lectura de frames será asíncrona y cada vez que recibamos un frame del dron será considerado como el siguiente a tratar.

5.1.2. Captación de imágenes de la webcam

Dado que no hemos implementado un nodo que albergue la webcam, el proceso de obtención de las imágenes es mediante el método *VideoCapture(cam_path)* de OpenCV. Si implementamos la webcam como un nodo de ROS podemos usar el paquete *usb_cam* que habilita el topic '*camera_name*'/*image_raw* que utiliza el mismo tipo de mensaje ROS que el topic de Bebop Autonomy (*sensor_msgs/Image*). Por lo tanto el método de lectura del streaming de vídeo sería igual que en el apartado anterior.

Como decimos, en este caso utilizamos directamente los métodos de lectura de OpenCV. Con la webcam volvemos a tener el mismo problema que antes: necesitamos una lectura asíncrona que nos permita tener el menor retraso posible entre el frame captado y el frame procesado. Para ello creamos una nueva hebra que se encarga de leer constantemente el path de la cámara y sobrescribir el frame que consideramos como siguiente a procesar.

5.1.3. Lectura de archivo de vídeo almacenado

En pos de realizar pruebas sin tener que volar el dron, el software de visión nos permite abrir archivos de vídeo almacenados sobre los que realizamos detección y tracking de personas como si se tratase de un vuelo real. Cuando realizamos este tipo de pruebas, en vez de enviar comandos al dron, lo que hacemos es mostrar por pantalla cuáles serían esos comandos.

Al analizar un archivo de vídeo almacenado, sí realizamos una lectura síncrona de los frames ya que no necesitamos tener la reactividad que sí es crucial en un vuelo real. Lo que sí podemos hacer es establecer un intervalo según el cual iremos leyendo los frames a saltos. Si el intervalo es 1 la lectura será totalmente secuencial y síncrona. Si, por ejemplo, el intervalo es 3, significa que leeremos los frames 0,3,6,9,... En definitiva los frames que sean múltiplo del intervalo indicado. Esto permite que la ejecución dure menos tiempo y también que el experimento se asemeje más a la situación de un vuelo real en el que la lectura de imágenes es asíncrona y los frames no son consecutivos.

5.1.4. Escritura de vídeo

Disponemos de una funcionalidad para grabar localmente en el equipo servidor los fotogramas procesados (aparecen los Bounding Boxes) durante la ejecución del proyecto. Además, se crea un archivo de texto (cuyo nombre es el path de vídeo de entrada indicado al ejecutar la aplicación) donde indicamos el número de ejecución, la resolución a la que se han procesado las imágenes, el máximo, el mínimo y la media de FPS obtenidos, y el total de tracks obtenidos.

```

1 Ejecución: 1
2 res: (854, 480)
3 Max FPS: 6.04919506618
4 Min FPS: 0.0847444394786
5 Mean FPS: 5.71171140017
6 Max track: 1
7 *1*
8 Ejecución: 2
9 res: (854, 480)
10 Max FPS: 6.13637612589
11 Min FPS: 0.085257671313
12 Mean FPS: 5.40591759105
13 Max track: 1
14 *2*
15 Ejecución: 3
16 res: (854, 480)
17 Max FPS: 6.40755262922
18 Min FPS: 0.087412801942
19 Mean FPS: 6.06396340888
20 Max track: 10
21 *3*
-- -- . .

```

Figura 13: Ejemplo de archivo txt generado

5.1.5. Tracker - DeepSORT

Utilizamos DeepSORT [17] como algoritmo de tracking por ser puntero en la tarea que ocupa. Se trata de una actualización, una mejora del algoritmo SORT (Simple Online and Real-time Tracker) en la que aplicamos *deep learning*. Utiliza filtros de *Kalman* para predecir la posición de futuros tracks basándose en las detecciones actuales y tracks pasados. Para asignar estos nuevos tracks a los antiguos utiliza dos algoritmos: *Distancia Mahalanobis* [20] y *Algoritmo Húngaro* [21].

Aplicamos *Deep Learning* usando otra métrica, a parte de la distancia de Mahalanobis, para la re-asignación de trackers. Esta nueva métrica se basa en la apariencia de los tracks. Extraemos un vector de características de cada track y lo enlazamos (*match*) con aquel cuyo vector sea más parecido (no puede ser consigo mismo). Este vector de características es el obtenido en la última capa de la CNN de nuestro detector, que en este caso es *YOLOv3*.

5.1.6. Detector - YOLOv3

Como algoritmo de detección utilizamos la red neuronal convolucional (CNN) YOLOv3, un sistema de detección en tiempo real de última generación, que nos aporta gran precisión y rapidez a la hora de detectar y clasificar objetos.

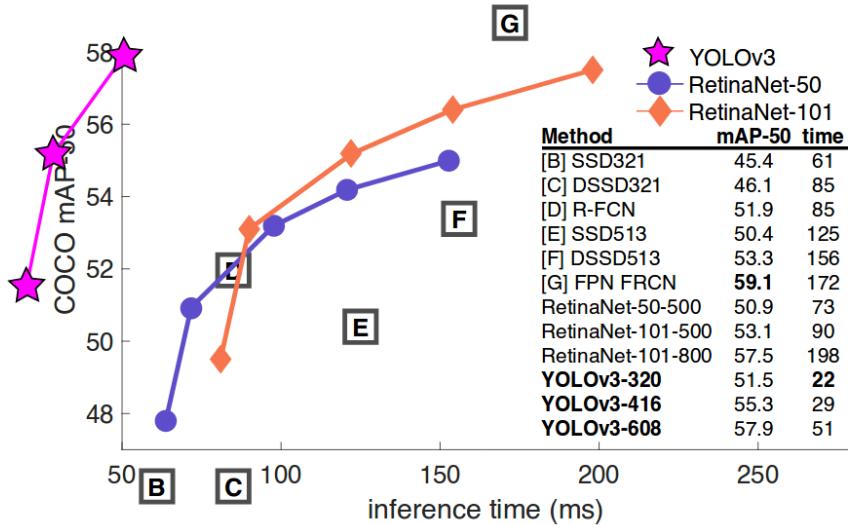


Figura 14: Rendimiento de YOLOv3 frente a otros algoritmos de detección.
Paper: [1]

Mediante YOLO obtenemos los Bounding Boxes de los objetos (personas) detectados junto con la probabilidad de que el objeto detectado pertenezca a X clase; en nuestro caso, a la clase *persona*. Son los datos relativos a los Bounding Boxes los que se transmiten al tracker: coordenadas en la imagen de la esquina superior izquierda de la detección y la altura y la anchura del rectángulo.

5.2. Comportamiento del Dron

5.2.1. Pilotaje Manual

Como decíamos en 4, el módulo de visión es útil sólo para mostrar por pantalla la imagen captada por el dron para así poder pilotarlo.

La aplicación despliega un menú de ayuda que muestra los controles asociados a cada movimiento del dron.

```
[0] Modo manual
[1] Seleccionar target video principal (solo funciona en modo automatico)
[2] Seleccionar target video secundario
[3] Seleccionar velocidad [0.0 - 1.0]
[4] Empezar/Parar grabacion
[5] Mostrar % bateria
[p] Parada de emergencia
[q] Salir del programa
[h] Help

0

[ ] Despega
[l] Aterrizar
[w] Avanza
[a] Desplaza izda
[s] Retrocede
[d] Desplaza dcha
[8] Asciende
[4] Gira izda
[6] Gira dcha
[2] Desciende
[5] Para
[e] Exit. Modo auto
[p] Parada de emergencia
[h] Help
```

Figura 15: Menú principal + Menú de pilotaje manual

Las acciones $w,a,s,d,8,4,6,2$ se ejecutan de manera continuada una vez se seleccionan, es decir, si pulsamos, por ejemplo, w el dron avanzará indefinidamente sin necesidad de mantener o pulsar repetidamente la tecla. Para detener el movimiento debemos pulsar 5 para que el dron quede flotando estático en el aire.

Si se pulsa e salimos del modo manual y, en caso de que hubiésemos indicado un track ID al que perseguir y este siga activo, el dron retomaría la persecución. La acción p envía un mensaje al topic *reset* lo que provoca la parada inmediata de los cuatro motores del dron.

5.2.2. Pilotaje Automático

En este modo necesitamos seleccionar el ID del track que marca la persona a la que queremos perseguir. Para ello, desde el menú principal 15 tecleamos 1 para acceder al diálogo de selección. En este diálogo se nos pide el ID de la persona a seguir y sólo podemos introducir un identificador válido o '-1' en caso de querer salir del diálogo sin seleccionar ningún objetivo.

```
[0] Modo manual
[1] Seleccionar target video principal (solo funciona en modo automatico)
[2] Seleccionar target video secundario
[3] Seleccionar velocidad [0.0 - 1.0]
[4] Empezar/Parar grabacion
[5] Mostrar % bateria
[p] Parada de emergencia
[q] Salir del programa
[h] Help

1
'-1'Exit.
Select target:
2
Bad Target. Select one of this:
[1]
'-1'Exit.
Select target:
1
Target Updated
```

Figura 16: Diálogo de selección de objetivo

En el momento en que indicamos un identificador válido, los atributos del Bounding Box (BB) asociado se envían al método de seguimiento. Aquí se toman las decisiones sobre qué movimiento realizar dependiendo de los valores de estos atributos.

Hemos fijado unos márgenes tanto en el eje X (40 % y 60 % del ancho de la imagen) como en el eje Y (40 % y 60 % de la altura de la imagen) entre los cuales intentaremos mantener siempre el *centroid* (punto medio) del BB. Si el centroid pasa a estar en [0,40) % en el eje X, rotaremos el dron hacia la izquierda para tratar de llevar el punto medio de nuevo entre los márgenes. Si, por el contrario se encuentra en (60,100] %, rotaremos hacia la derecha. Si sobrepasamos los límites verticales del eje Y ascenderemos en el caso de estar en [0,40) % y descenderemos si se encuentra en (60,100] %. Con la altura del BB, controlamos la distancia a la que nos encontramos del objetivo. También disponemos de unos márgenes entre los cuáles puede estar. La implementación de esta histéresis en el sistema evita que el dron esté continuamente moviéndose descontroladamente y permite tener un movimiento más fluido para obtener una imagen más clara. Si la altura del BB es demasiado grande, el dron se alejará del objetivo y si es más pequeña, se acercará.

5.2.3. Opciones Menú Principal

- **0:** Entra en modo manual y deja de perseguir al objetivo (si es que lo tenía).
- **1:** Permite introducir el ID del objetivo que vemos por el streaming de vídeo principal.
- **2:** En caso de ejecutar la aplicación sobre el vídeo proporcionado por el dron y el vídeo proporcionado por una webcam o cámara fija simultáneamente, permite introducir el ID del objetivo que vemos por el streaming de vídeo de la webcam.
- **3:** Nos permite introducir la velocidad de movimiento lineal y de rotación del dron. Por defecto la velocidad lineal es de 0.1 y la de rotación 0.07.
- **4:** Empieza/Termina la grabación en la memoria flash del dron de la imagen captada por el mismo.
- **5:** Nos informa del porcentaje de batería restante.
- **p:** Realiza una parada de emergencia. Funciona igual que en el pilotaje manual.
- **q:** Salimos de la aplicación.
- **h:** Muestra por pantalla las opciones disponibles del menú.

6. Experimentos

6.1. Eficiencia del tracker DeepSORT en MOT Challenge

MOT Challenge o *Multiple Object Tracking Benchmark* [22] es una prueba de rendimiento que se aplica comúnmente a los algoritmos de tracking para compararlos entre sí. Hemos realizado las pruebas sobre tres de los catorce benchmarks disponibles en el desafío de 2017 (MOT17): MOT17_05, MOT17_06, MOT17_12.

6.1.1. MOT17_05

Como vamos a demostrar analizando las distintas pruebas realizadas, la fluidez de la ejecución depende mucho de la resolución de los frames con los que trabajemos, el número de tracks que aparezcan en pantalla y el intervalo de lectura del vídeo (cuantos frames saltamos entre una lectura y la siguiente).

La resolución de la imagen afecta a la eficiencia porque es el área que tenemos que recorrer en busca de detecciones. Cuanto mayor sea la resolución más iteraciones debe hacer el algoritmo en busca de personas.

El número de tracks en pantalla influye debido a los costes computacionales que conlleva el cálculo de cada uno de ellos. Cuantos más tracks en un mismo frame, más cálculos se han de hacer y más tiempo se tarda en procesar.

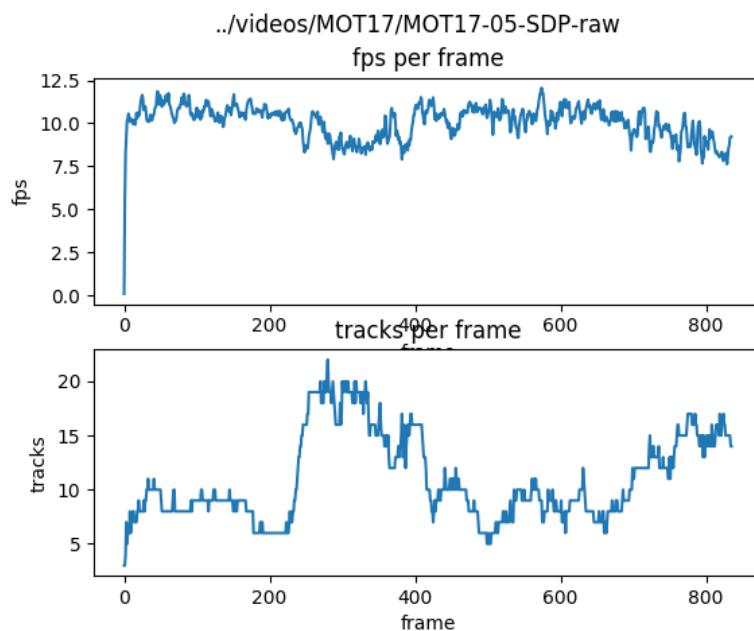
El intervalo de lectura del vídeo afecta ya que aumenta o disminuye el número de frames a procesar. De un vídeo de 60 frames, procesaríamos todos ellos si aplicamos un intervalo de 1 salto. Pero si aplicamos un intervalo de 3 saltos, los frames procesados disminuyen a 20, lo que se traduce en un aumento de **FPS** porque procesamos el mismo fragmento de vídeo en menos tiempo.

Para cada prueba analizaremos **sú** gráfico correspondiente, y tras comentarlos todos, compararemos los resultados entre sí. Los gráficos se dividen en dos: la parte inferior muestra cuántos tracks aparecen en pantalla por cada frame, mientras que la parte superior muestra la evolución de los **FPS por frame**.

Nos centramos tanto en los FPS debido a la escasez de los mismos. Si contásemos con un equipo hardware que, de base, nos proporcionase un mayor número de cuadros por segundo, podríamos centrarnos en mejorar otras características como obtener el mayor número de detecciones (y por ende tracks) posibles o intentar usar el vídeo a mayor resolución para que la visualización de los resultados fuese más cómoda. Como este no es el caso, y contamos con un número reducido de FPS, necesitamos ajustar los parámetros de ejecución de manera que aumentemos esta variable para que la reactividad de nuestro dron sea la mayor posible.

■ Prueba 1:

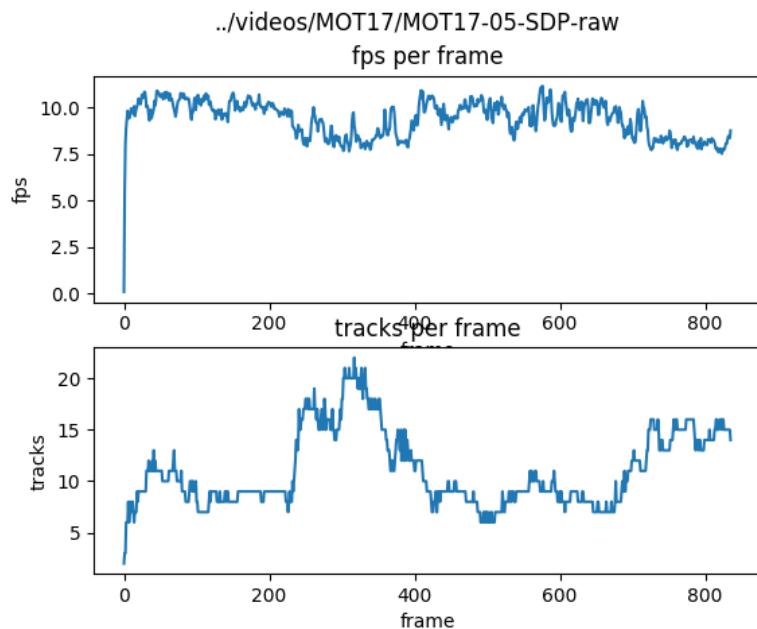
- **Resolución:** 426x240
- **FPS máximos alcanzados:** 12.0590228226
- **FPS mínimos alcanzados:** 0.0829128391446
- **FPS medios:** 10.0657791094
- **Máximo track alcanzado:** 231
- **Intervalo de lectura:** 1



Con un intervalo de 1 frame, conseguimos una lectura completamente secuencial, y logramos leer algo más de 800 frames. Como hemos dicho al principio, el número de tracks que aparezcan en pantalla influye en la fluidez de la ejecución, o lo que es lo mismo, en el número de FPS que obtenemos. Podemos ver que en los intervalos donde el número de tracks es pequeño el número de FPS es ligeramente superior como, por ejemplo, en los tramos [0,200] o [400,700]. Por el contrario, si el número de tracks en pantalla se dispara, el número de FPS disminuye como pasa en los tramos [200,400] y [700,+800]. Por esto podremos ver cierta simetría entre ambos subgrafos de manera que cuando uno sube el otro baja y viceversa.

■ Prueba 2:

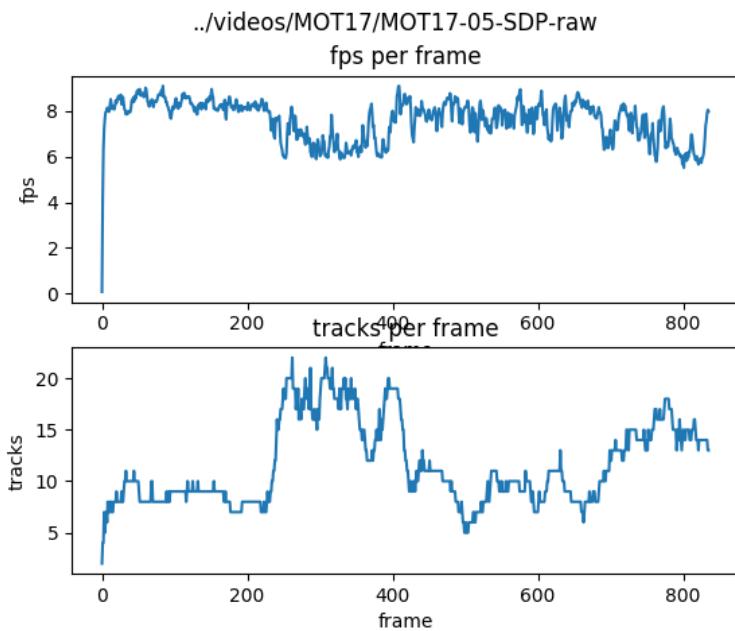
- **Resolución:** 640x480
- **FPS máximos alcanzados:** 11.1044242182
- **FPS mínimos alcanzados:** 0.0848866147599
- **FPS medios:** 9.35959613484
- **Máximo track alcanzado:** 240
- **Intervalo de lectura:** 1



Al subir la resolución podemos notar ciertos cambios en los resultados. El número de FPS medio ha descendido de 10 a 9.4, hemos perdido 0.6 cuadros por segundo. Vemos también, que al usar la que es la resolución nativa del vídeo, somos capaces de detectar más tracks (240 respecto a los 231 de la Prueba 1). Esto se debe a que al no haber sido retocada (no ha sufrido un *downsampling* como en la anterior prueba), la imagen conserva su nitidez y permite al algoritmo de detección dar menos falsos negativos.

■ Prueba 3:

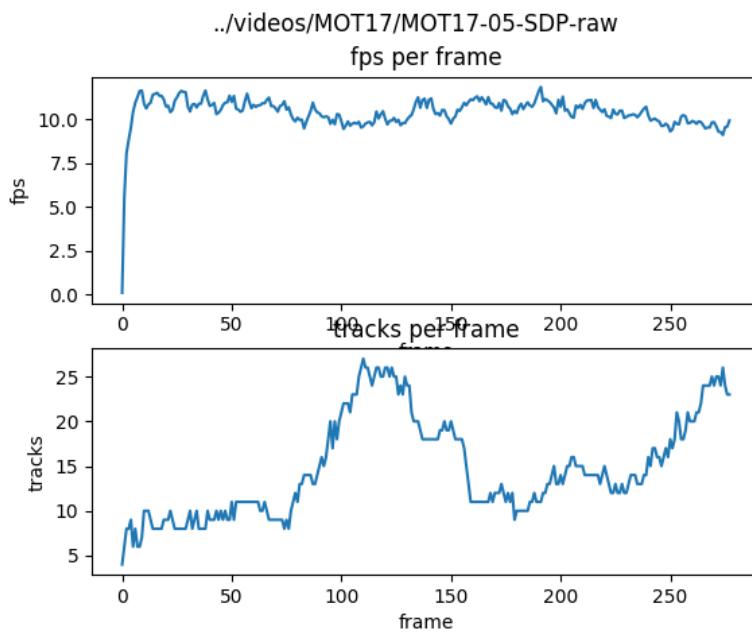
- **Resolución:** 1280x720
- **FPS máximos alcanzados:** 9.08981954517
- **FPS mínimos alcanzados:** 0.0849724390375
- **FPS medios:** 7.57638393757
- **Máximo track alcanzado:** 245
- **Intervalo de lectura:** 1



Al hacer un *upsampling* (aumentar la resolución) de la imagen vemos que se repiten los hechos anteriores. El número de FPS medio ha vuelto a descender, esta vez de 8.6 a 7.6. Hemos perdido 1 frame por segundo de media. También hemos detectado más tracks (245 frente a 240 en la prueba anterior). Esto puede deberse a dos motivos: El primero es que al aumentar la imagen, personas que antes eran figuras demasiado pequeñas y confusas para el detector, ahora no lo sean o, que al aumentar la imagen y provocar que pierda nitidez, el algoritmo de falsos positivos con figuras que realmente no son personas.

■ Prueba 4:

- **Resolución:** 426x240
- **FPS máximos alcanzados:** 11.8429932287
- **FPS mínimos alcanzados:** 0.0932298185409
- **FPS medios:** 10.4022314636
- **Máximo track alcanzado:** 208
- **Intervalo de lectura:** 3



Al igual que las tres anteriores pruebas se hicieron con un intervalo de 1 frame, las tres siguientes se ejecutaron con un **intervalo de 3 frames**. Así podemos comparar, los resultados obtenidos con el mismo vídeo, a la misma resolución pero con distintos saltos de fotogramas.

La diferencia más palpable es la disminución de frames procesados en la ejecución. Ahora vemos que se ha aplicado el algoritmo de tracking sobre algo más de 250 fotogramas sin llegar a los 300. Antes, sin embargo, sobrepasábamos los 800. Como vemos, estos números guardan una relación de 1:3 debido al intervalo utilizado.

El subgrafo inferior (tracks por frame) tiene picos más altos. Esto se debe, principalmente, al cambio brusco entre fotogramas. Supongamos dos fotogramas A y B consecutivos (por ejemplo $A=\text{frame_0}$ y $B=\text{frame_3}$). En el fotograma A aparece una persona en el extremo izquierdo de la imagen, durante los fotogramas 1 y 2 se traslada hacia

la derecha de manera que en el frame B aparece en el extremo derecho de la fotografía. El track asignado a la persona en A no reconoce a la persona de B como la misma debido a la larga distancia que las separa, de modo que, a la persona de B se le asigna un nuevo identificador (nuevo track) a la vez que el track asignado a esa misma persona en A perdura en el tiempo por si vuelve a aparecer en escena una figura que coincida con las características buscadas.

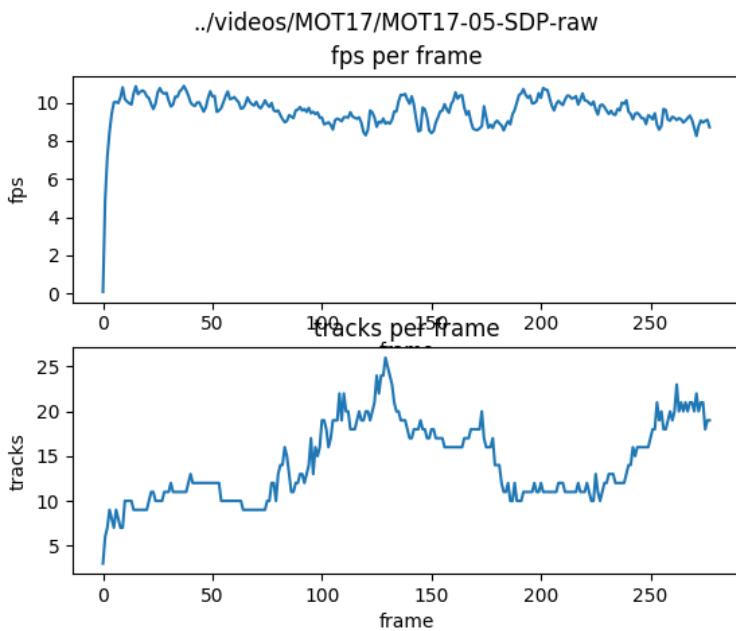
Por este motivo, en una situación dónde solo tendríamos un track si hiciésemos un estudio totalmente secuencial con un intervalo de 1 frame, obtenemos 2 identificadores debido al salto entre fotogramas que no nos permite hacer un seguimiento fluido de las figuras '*trackeadas*'.

La relación entre ambos subgrafos es menos notable debido a la separación temporal de los frames pero, aún así, distinguimos que los valles del grafo superior coinciden con los picos del grafo inferior y viceversa.

El número máximo de tracks es menor que en la Prueba 1 ya que al saltarnos frames perdemos los tracks que aparecerían en ellos (208 vs 231), y los FPS medios son ligeramente superiores (10.4022314636 vs 10.0657791094).

■ Prueba 5:

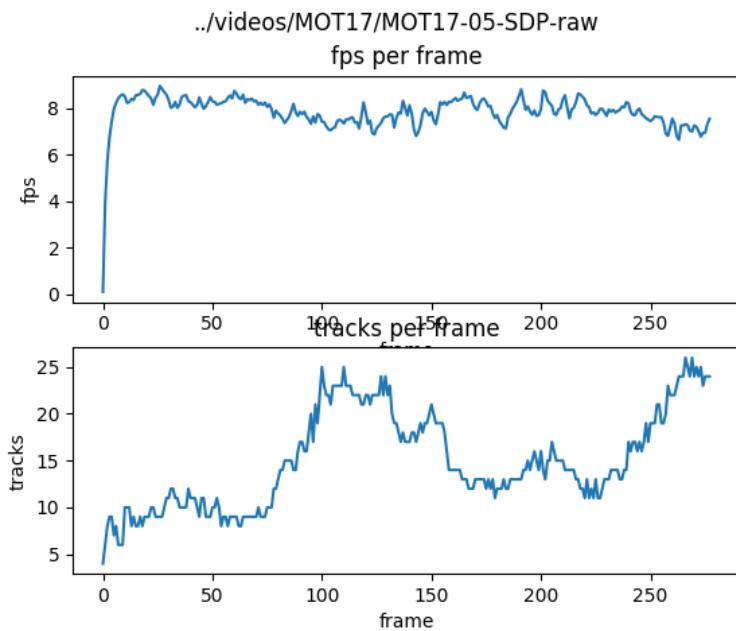
- **Resolución:** 640x480
- **FPS máximos alcanzados:** 10.8670977337
- **FPS mínimos alcanzados:** 0.092763326242
- **FPS medios:** 9.56315059173
- **Máximo track alcanzado:** 212
- **Intervalo de lectura:** 3



Vemos que las características cambiantes se repiten también en la resolución nativa del vídeo. El máximo número de tracks en un mismo frame también ha ascendido a alrededor de 25 tracks. El número total de tracks calculados es también menor que el de su homólogo con intervalo de 1 frame (212 vs 240), así como también es superior el número de FPS medios obtenidos aquí que en la Prueba 2 (9.56315059173 vs 9.35959613484). La diferencia sigue siendo poco más que sutil.

■ Prueba 6:

- **Resolución:** 1280x720
- **FPS máximos alcanzados:** 8.96106172524
- **FPS mínimos alcanzados:** 0.092512961794
- **FPS medios:** 7.85037875834
- **Máximo track alcanzado:** 225
- **Intervalo de lectura:** 3

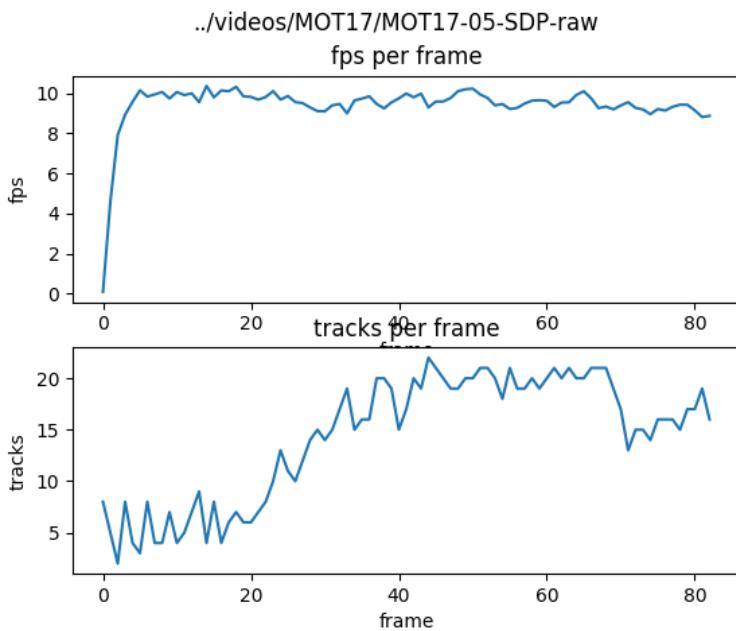


En esta prueba no hay nada nuevo que destacar más allá de las mismas diferencias que se dan en las dos pruebas anteriores.

Los FPS medios son ligeramente superiores pero no suponen una diferencia sustancial, el total de tracks calculados es menor por haber procesado menos tracks y haber perdido información (225 vs 245) y el pico de tracks por frame ha subido también alrededor de los 25.

■ Prueba 7:

- **Resolución:** 640x480
- **FPS máximos alcanzados:** 10.3682136894
- **FPS mínimos alcanzados:** 0.0862593791535
- **FPS medios:** 9.41332305856
- **Máximo track alcanzado:** 171
- **Intervalo de lectura:** 10



En esta última prueba tratamos de exagerar el intervalo escogido para observar qué cambia en los resultados.

Los FPS se mantienen muy cercanos a los obtenidos en las otras dos pruebas con la misma resolución, siendo estos de menor a mayor intervalo: 9.35959613484 vs 9.56315059173 vs 9.41332305856.

El número de tracks calculados desciende mucho. En el orden en que se han visto las pruebas podemos comparar: 240 vs 212 vs 171.

El pico de tracks en un mismo frame se acerca más a la primera ejecución siendo cercano a 20 tracks en un mismo frame.

Con tanta distancia entre frames en un vídeo de tan corta duración no se puede apreciar la correlación existente entre ambos subgrafos.

■ Conclusiones

Hemos visto que los distintos parámetros que podemos ajustar en la ejecución modifican de manera diferente los resultados:

- **Resolución:** Afecta de manera significativa a los FPS. A menor resolución mayor cantidad de cuadros por segundo obtendremos. También afecta al número de tracks encontrados en escena aunque de una forma no tan notable.
- **Intervalo:** El intervalo de lectura de frames no interviene en el número de fotogramas por segundo obtenidos pero si marca la diferencia en la cantidad total de tracks detectados.

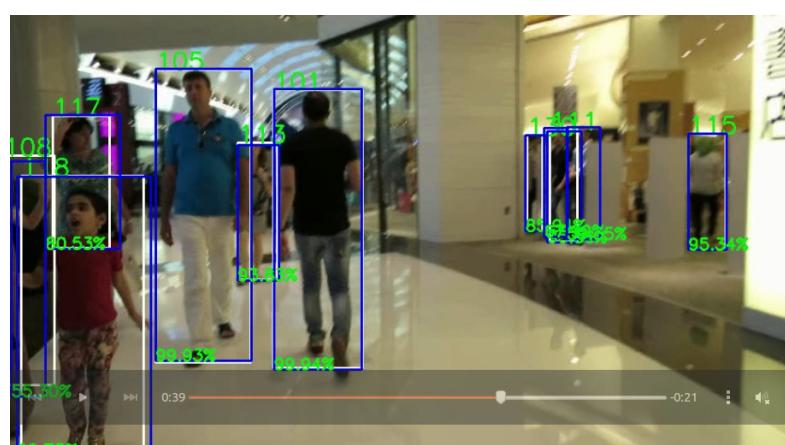
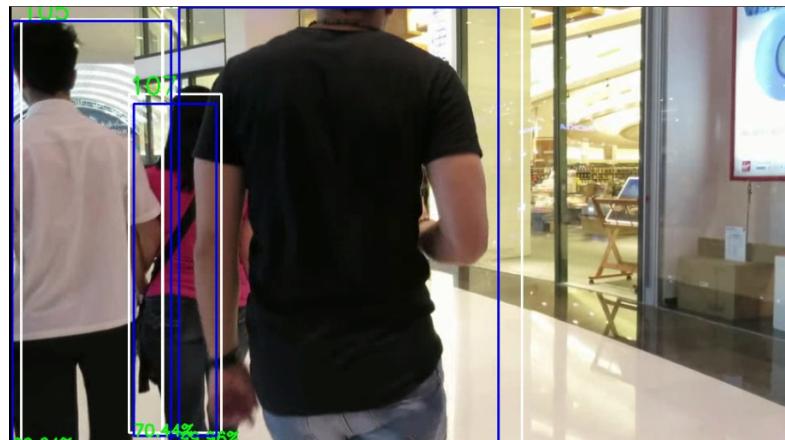
Considerando estos resultados, a la hora de ejecutar la aplicación sobre el vídeo obtenido del dron, hemos decidido utilizar una baja resolución para garantizar que contamos con todos los FPS posibles. Vamos a utilizar una resolución de 426x240 píxeles que nos proporciona un buen ratio de detecciones en escena ($\simeq 20$ en las pruebas MOT17_05) y nos brinda también una buena cantidad de FPS.

Respecto al intervalo de ejecución hemos decidido concederle un carácter dinámico, para ajustarse siempre al tiempo de procesamiento de manera que nos permita obtener siempre el último frame mandado por el dron. Dicho de una manera más clara, vamos a realizar una captura de vídeo asíncrona para garantizar la reactividad del dron.

Las demás pruebas del Benchmark MOT17 de las que hablamos al inicio de la sección se pueden ver en la siguiente carpeta de MEGA [23]. Encontraremos los vídeos con las detecciones y tracks, los gráficos obtenidos y los datos de las ejecuciones en un archivo .txt.

6.2. Pruebas del Detector y Tracker

6.2.1. Oclusión



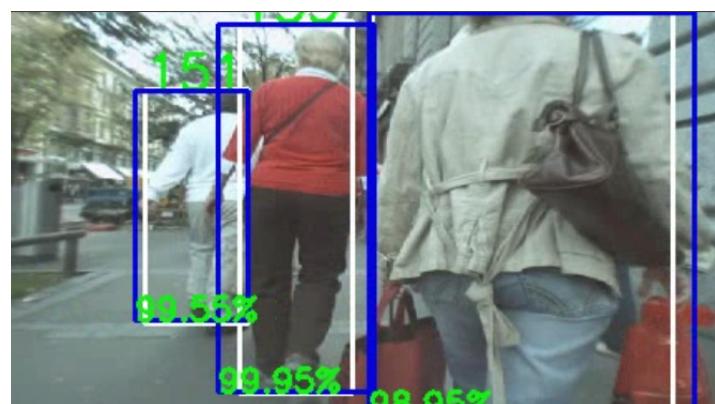
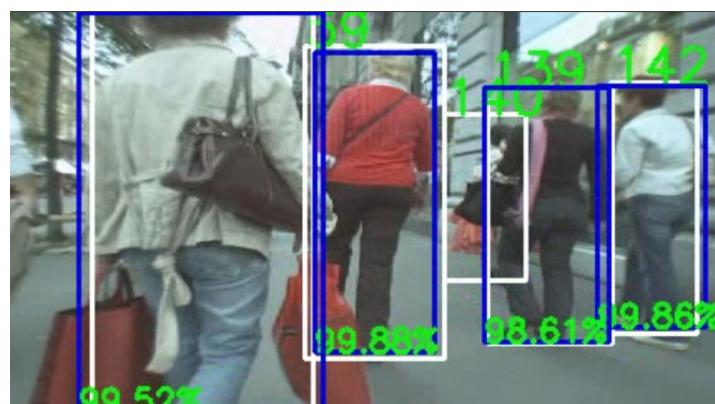
El algoritmo de detección se desenvuelve bien ante la oclusión de figuras.

En la primera imagen podemos ver como detecta a la mujer marcada con el identificador 107 que se encuentra entre dos personas que tapan parte de su cuerpo.

En la segunda fotografía vemos que detecta al hombre etiquetado con 105 a pesar de que un niño tapa la mitad inferior de su cuerpo. También vemos la detección de un niño que todavía no ha sido traqueado que se encuentra detrás del carro de bebés.

Por último, en el tercer frame vemos una persona identificada con 113 tapada por el hombre 105, y a la mujer 117 tapada por la niña 118.

6.2.2. Consistencia del Tracker





La consistencia del tracker se demuestra cuando tras perder el track de una persona es capaz de reasignar el identificador a la misma figura cuando reaparece en escena.

En la sucesión de imágenes de arriba vemos como se mantiene la consistencia.

En la primera imagen aparecen dos personas con identificadores 139 y 142.

En la segunda, otra persona las tapa por completo y perdemos su detección, pero los tracks siguen activos buscando figuras que encajen con su descripción.

En la última, las dos personas vuelven a aparecer en escena y el algoritmo es capaz de identificar de que se tratan de las mismas figuras que antes y les reasigna el mismo ID.

6.3. Pruebas con el Dron

6.3.1. Vuelo con app

- Modo Manual:

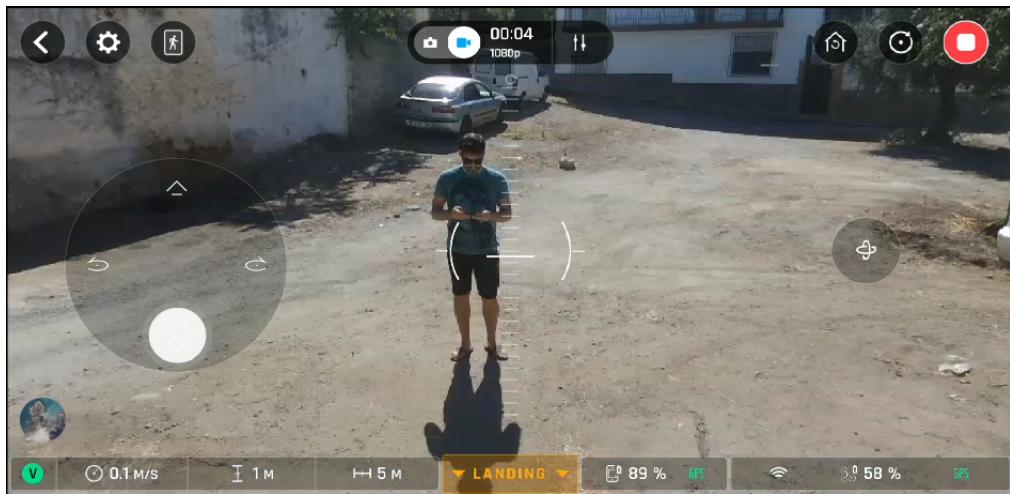


Figura 17: Interfaz HUD (Head-Up Display) del modo pilotaje manual de la aplicación Free Flight Pro

En pantalla aparece gran cantidad de información: velocidad actual de movimiento, altura a la que vuela, tiempo de grabación e, incluso, porcentajes de batería tanto del *smartphone* o *tablet* como del propio dron.

También aparecen todos los controles con los que podemos interactuar con el dron. Los dos principales son la cruceta/*joystick* de la izquierda que permite movimientos en el eje Y (ascenso y descenso) y rotar en ambas direcciones. A la derecha se encuentra el control de movimiento en los ejes X y Z que utiliza el giroscopio del dispositivo con el que nos conectamos al dron para recibir las órdenes. Si inclinamos el móvil hacia delante el dron avanzará, si lo inclinamos hacia atrás el dron retrocederá, e igual sucede hacia los lados.

En la parte superior de la pantalla, en el centro, se encuentran los controles para cambiar entre modo fotografía y vídeo. También arriba pero a la derecha se encuentran comandos de movimiento autónomo que permiten la vuelta del dron al punto de partida, y realizar un barrido circular de la zona. También en esta esquina está el control para iniciar o detener la grabación de vídeo. Por último, en la esquina contraria, encontramos un acceso a los ajustes del dron, y otro acceso al modo de tracking. Este modo lo describimos a continuación.

■ Modo Automático:



Figura 18: Interfaz HUD (Head-Up Display) del modo pilotaje automático de la aplicación Free Flight Pro

En este modo automático aparecen dos controles nuevos en pantalla abajo a la derecha. Nos permite bloquear la interacción con los controles para no interferir en el vuelo de manera no intencionada y también superponer en pantalla los controles del control manual para tomar el control del dron en cualquier momento.

En esta pantalla podemos seleccionar con un recuadro un objeto al que perseguir. Este recuadro lo tiene que dibujar manualmente el usuario. La aplicación extrae las características del área que hay dentro del rectángulo y *trackea* la figura por la escena. Esto nos permite seguir a cualquier tipo de objeto (personas, animales, vehículos).

Una vez seleccionada la figura a seguir el dron se mantendrá estático y seguirá al objetivo tan solo rotando. Para comenzar una persecución activa debemos pulsar sobre la opción *FOLLOW* que aparece en el centro de la pantalla, abajo. Ahora el dron pasa a perseguirnos si nos alejamos y a alejarse de nosotros si nos acercamos para mantener siempre cierta distancia. Si nos movemos en el plano horizontal, el dron se moverá junto a nosotros horizontalmente en vez de rotar para centrarnos en su visión y seguirnos de frente.

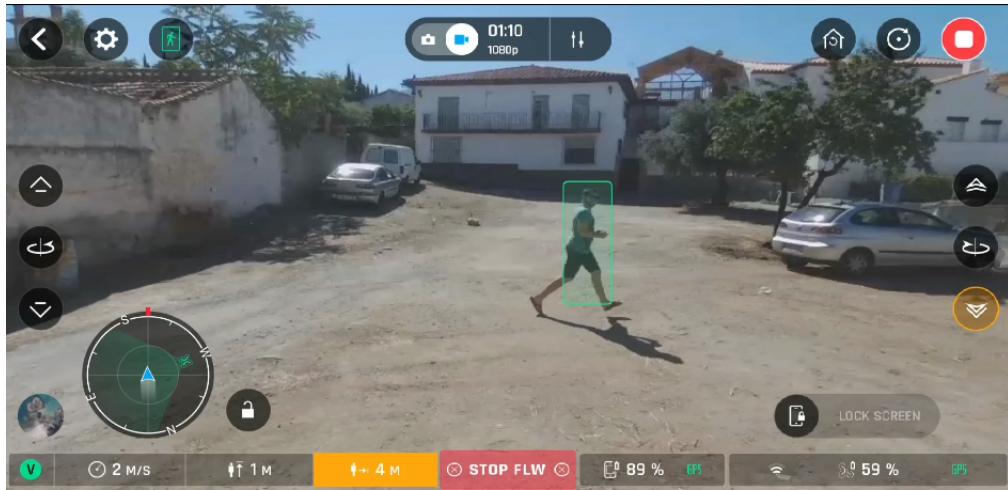


Figura 19: Dron siguiendo al objetivo de manera activa

Con todas estas características, ¿por qué querríamos desarrollar otra aplicación, que a priori, realiza la misma función? El objetivo de la aplicación creada, es aportar modularidad e inteligencia al sistema. Con este trabajo de fin de grado, hemos realizado un sistema de seguimiento al que podemos acoplar más de un dron, podemos incorporar imagen proveniente de cámaras estáticas, los objetos detectados son reconocidos y clasificados por el sistema de manera que sabemos qué es lo que estamos siguiendo. El algoritmo de detección cuenta con una gran lista de objetos reconocibles en la que, además de personas, encontramos, coches, motocicletas, camiones, gatos, perros, vacas, y un largo etcétera. También podemos entrenar la red neuronal para que reconozca cualquier objeto que nos interese. Además realizamos un *multitracking* de todos los objetos de la escena que nos permite cambiar entre objetivos de manera rápida, y por supuesto existe la opción de expandir la aplicación y dotar al dron de un comportamiento más complejo que tenga en cuenta más variables del entorno. Por ejemplo, podríamos incorporar un software de detección de profundidad de imagen 2D para evitar así chocar con obstáculos e incluso poder evitarlos.

Por todo esto creo que la aplicación desarrollada tiene mucho potencial y supone una muy buena base de la que partir para realizar proyectos verdaderamente complejos con grandes capacidades.

6.3.2. Ejecución en Equipo Portátil

La experimentación con el equipo portátil fue de corta duración. La eficiencia es ínfima y no proporciona una experiencia fluida.

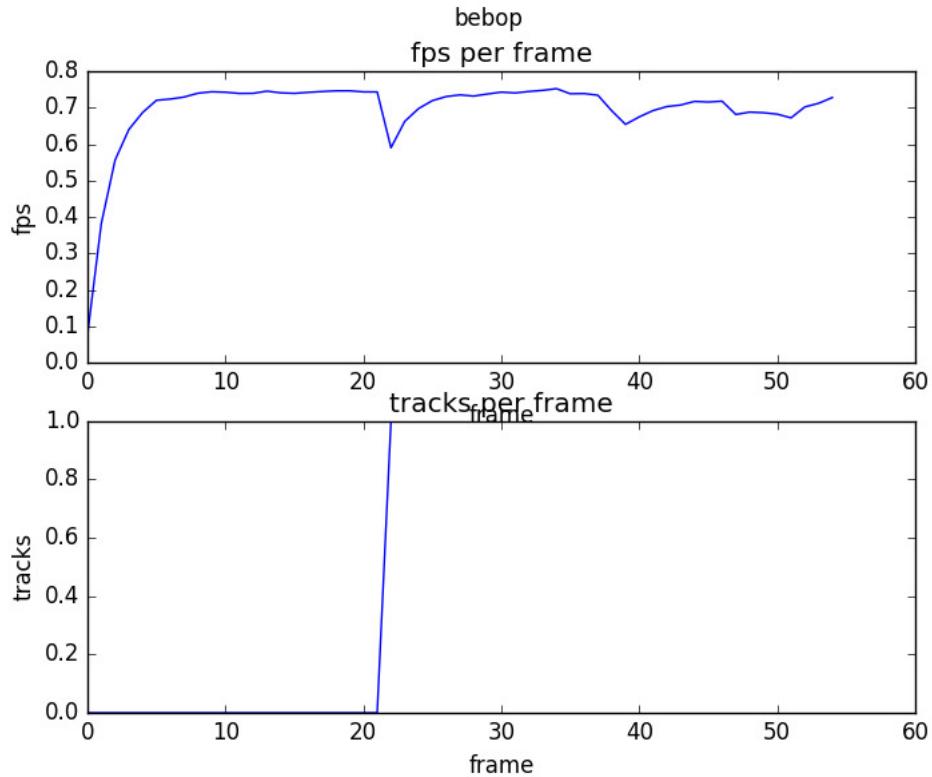


Figura 20: Evolución de FPS y número de tracks a lo largo de los frames

Durante la ejecución no conseguimos alcanzar los 0.8 FPS y cuando aparece una persona en escena y se calcula su track, los cuadros por segundo caen en picado hasta los 0.6 e intentan estabilizarse en torno a los 0.7. Estos valores se deben a que la ejecución se realiza en la CPU y su procesador gráfico integrado (el equipo no dispone de tarjeta gráfica dedicada) y estos no disponen de la potencia para realizar el procesamiento requerido por el sistema.

La siguiente imagen 21 muestra el estado del equipo durante la ejecución de la aplicación en la que vemos que los cuatro núcleos del procesador y la memoria RAM están trabajando al 100 % de su capacidad y aún así solo logramos obtener los resultados vistos en la gráfica superior 20.

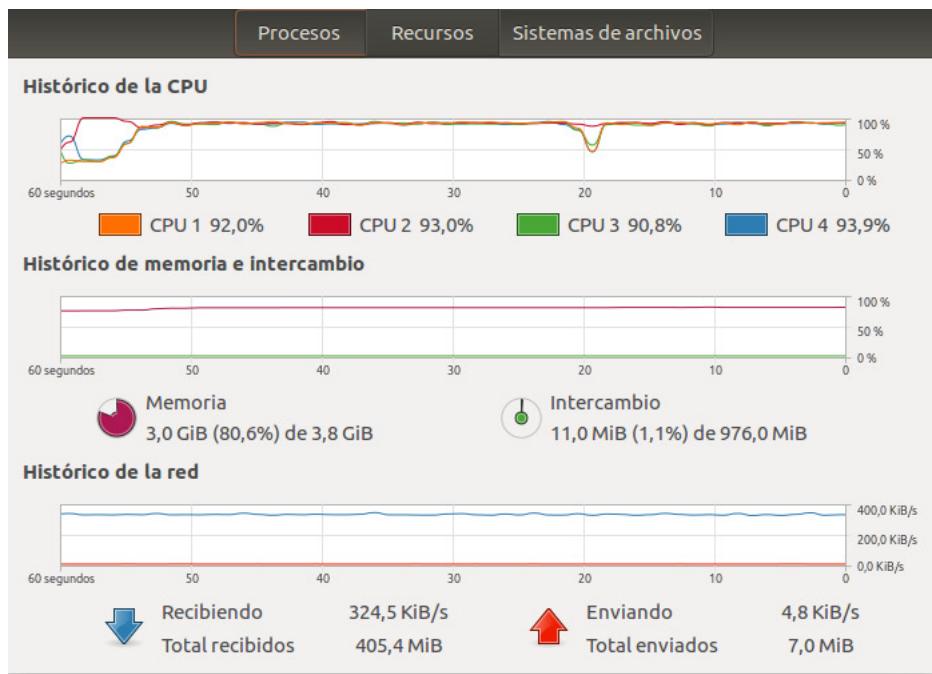


Figura 21: Estado del equipo portátil durante la ejecución de la aplicación

Estos resultados los obtenemos tan sólo iniciando el módulo de visión por computador. Si iniciásemos el módulo de movimiento autónomo, que supone la creación de una nueva hebra, los resultados serían incluso peores.

6.3.3. Vuelo automático

Como hemos dicho en la sección de ejecución con el equipo portátil, el hecho de iniciar o no el vuelo automático, repercute en la fluidez de la ejecución ya que crea una nueva hebra. Esto lo podemos ver en el equipo de sobremesa comparando los resultados obtenidos en vuelo manual y automático.

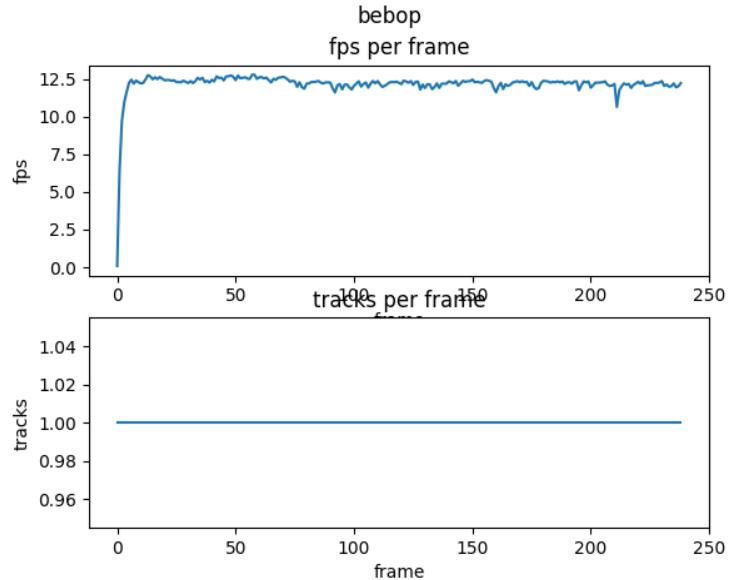


Figura 22: FPS en modo manual

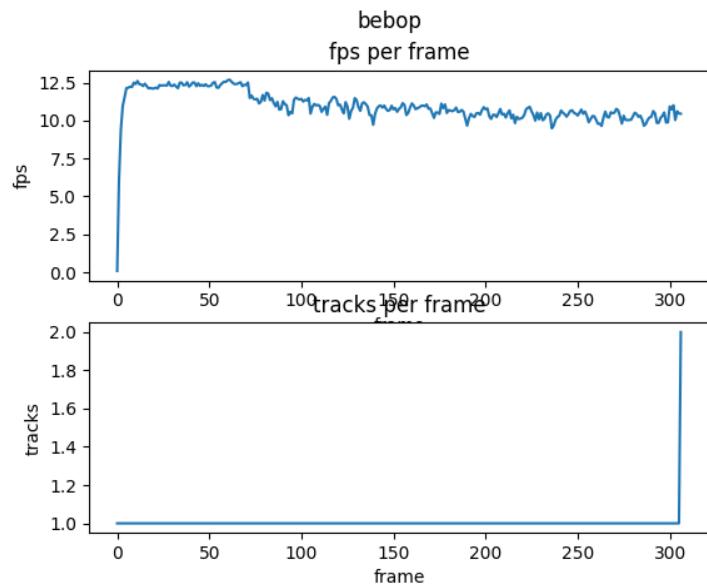


Figura 23: FPS en modo automático

Se aprecia en el gráfico del modo de vuelo manual 22 que los FPS son muy estables y rondan los 12 FPS, específicamente tiene una media de 12.1598130319 FPS durante toda la ejecución.

En cambio en el modo automático 23, vemos que a partir del track 75 (aproximadamente), que es cuando iniciamos el modo automático, los FPS desciden de 12.5 a, aproximadamente, 11. Los FPS medios obtenidos, teniendo en cuenta el inicio en modo manual, son 10.950283759.

Estos casi 11 FPS nos permiten obtener buenos resultados durante la ejecución aunque, por supuesto, sería deseable contar con mayor cantidad de fotogramas. Estos dos gráficos corresponden a pruebas estáticas, es decir, el dron se encuentra parado y no le enviamos comandos de movimiento pero sí activamos la nueva hebra que se encarga del movimiento autónomo.

A continuación vamos a ver los resultados de ejecutar el sistema al completo. Incorporamos la imagen de una cámara estática junto al vídeo proporcionado por el dron. En primer lugar, fijamos un objetivo en la cámara fija, cuando este objetivo salga de plano y el track se desactive, se manda una orden de despegue al dron. Cuando el dron ya está en el aire seleccionamos un identificador de los captados por esta entrada de vídeo y comienza la persecución del objetivo indicado.

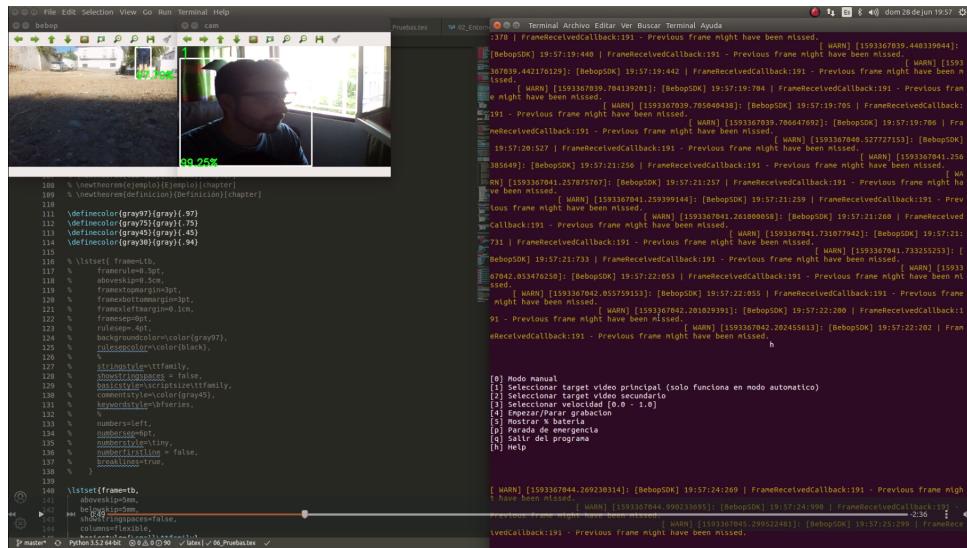


Figura 24: Inicio del sistema

En esta captura vemos que al iniciar el sistema contamos con la visión de la cámara fija en la que aparece una figura trackeada. También vemos la imagen del dron que se encuentra reposando en el suelo y, a la derecha, la ejecución de la aplicación en la terminal. Los mensajes de warning son

debidos a pérdida de fotogramas entre el dron y el servidor. Recordemos que estos dos equipos están separados ~~al rededor~~ de unos 25 metros y además hay obstáculos que interfieren en la señal.

```
[1] Seleccionar target video principal (solo funciona en modo automatico)
[2] Seleccionar target video secundario
[3] Seleccionar velocidad [0.0 - 1.0]
[4] Empezar/Parar grabacion
[5] Mostrar % bateria
[p] Parada de emergencia
[q] Salir del programa
[h] Help

[ WARN] [1593367053.047605485]: [BebopSDK] 19:57:33:047 | FrameReceivedCallback:191 - Previous frame might have been missed.
[ WARN] [1593367053.149095082]: [BebopSDK] 19:57:33:149 | FrameReceivedCallback:191 - Previous frame might have been missed.
[ WARN] [1593367053.690150678]: [BebopSDK] 19:57:33:690 | FrameReceivedCallback:191 - Previous frame might have been missed.
[ WARN] [1593367053.691264180]: [BebopSDK] 19:57:33:691 | FrameReceivedCallback:191 - Previous frame might have been missed.
[ WARN] [1593367053.727031127]: [BebopSDK] 19:57:33:726 | FrameReceivedCallback:191 - Previous frame might have been missed.
[ WARN] [1593367053.192610318]: [BebopSDK] 19:57:35:192 | FrameReceivedCallback:191 - Previous frame might have been missed.
[ WARN] [1593367055.195537223]: [BebopSDK] 19:57:35:195 | FrameReceivedCallback:191 - Previous frame might have been missed.
[ WARN] [1593367055.238027054]: [BebopSDK] 19:57:35:237 | FrameReceivedCallback:191 - Previous frame might have been missed.
[ WARN] [1593367055.451501018]: [BebopSDK] 19:57:35:451 | FrameReceivedCallback:191 - Previous frame might have been missed.
[ WARN] [1593367055.453817927]: [BebopSDK] 19:57:35:453 | FrameReceivedCallback:191 - Previous frame might have been missed.
[ WARN] [1593367055.710246894]: [BebopSDK] 19:57:35:710 | FrameReceivedCallback:191 - Previous frame might have been missed.
[ WARN] [1593367055.712119139]: [BebopSDK] 19:57:35:712 | FrameReceivedCallback:191 - Previous frame might have been missed.
[ WARN] [1593367056.402799101]: [BebopSDK] 19:57:36:402 | FrameReceivedCallback:191 - Previous frame might have been missed.
[ WARN] [1593367056.405123471]: [BebopSDK] 19:57:36:405 | FrameReceivedCallback:191 - Previous frame might have been missed.
[ WARN] [1593367056.410029234]: [BebopSDK] 19:57:36:409 | FrameReceivedCallback:191 - Previous frame might have been missed.
[ WARN] [1593367056.412623991]: [BebopSDK] 19:57:36:412 | FrameReceivedCallback:191 - Previous frame might have been missed.
2
'-1' Exit.
Select target:
[ WARN] [1593367056.609123536]: [BebopSDK] 19:57:36:609 | FrameReceivedCallback:191 - Previous frame might have been missed.
[ WARN] [1593367056.611751325]: [BebopSDK] 19:57:36:611 | FrameReceivedCallback:191 - Previous frame might have been missed.
[ WARN] [1593367057.638281453]: [BebopSDK] 19:57:37:638 | FrameReceivedCallback:191 - Previous frame might have been missed.
[ WARN] [1593367057.639267505]: [BebopSDK] 19:57:37:639 | FrameReceivedCallback:191 - Previous frame might have been missed.
[ WARN] [1593367057.640361066]: [BebopSDK] 19:57:37:640 | FrameReceivedCallback:191 - Previous frame might have been missed.
1
```

Figura 25: Selección de ID en cámara fija

Si ampliamos la captura de pantalla superior 25, vemos que estamos seleccionando la opción del menú que nos permite indicar un objetivo a perseguir en la entrada de vídeo que nos proporciona la cámara fija. Seleccionamos el identificador 1 que es el único que aparece en pantalla.

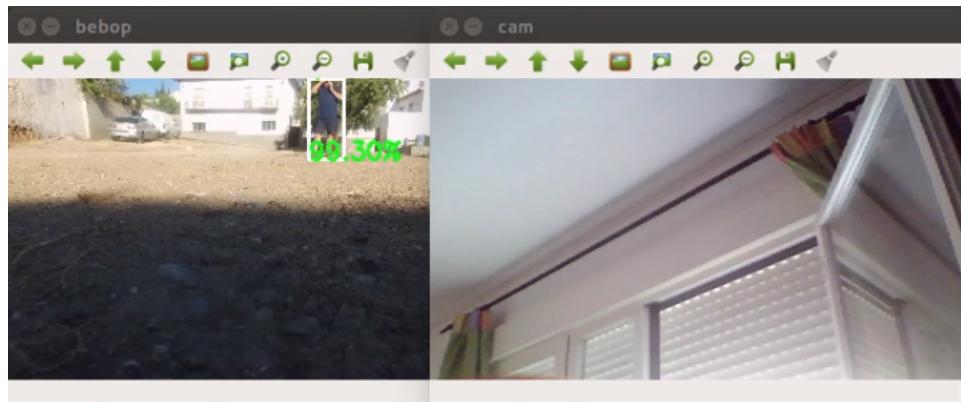


Figura 26: Salida de escena

El siguiente paso es simular que el objetivo marcado se sale de escena. En este caso movemos la cámara para que deje de ver el objetivo marcado anteriormente. Tras unos segundos en los que la figura no aparece en escena, su track se desactiva y se manda un comando al dron para que despegue.

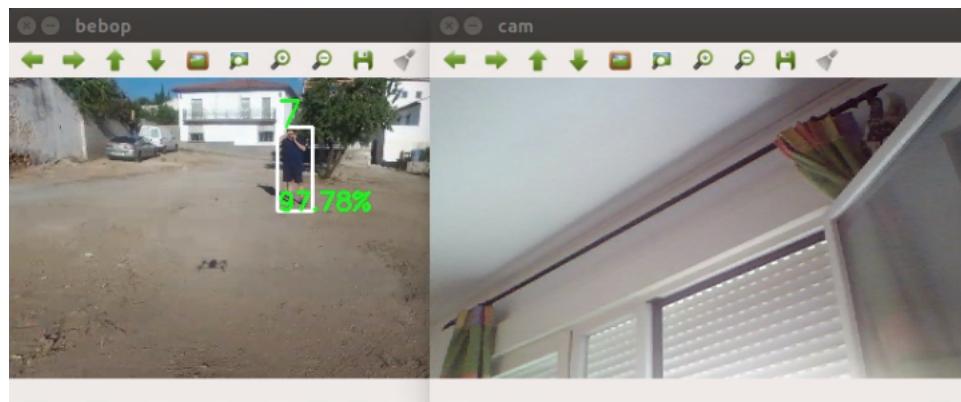


Figura 27: Despegue automático

Aquí podemos ver al dron ya en el aire pero aún estático. A continuación indicamos el ID del track que debe empezar a perseguir.



```

h

[0] Modo manual
[1] Seleccionar target video principal (solo funciona en modo automatico)
[2] Seleccionar target video secundario
[3] Seleccionar velocidad [0.0 - 1.0]
[4] Empezar/Parar grabacion
[5] Mostrar % bateria
[p] Parada de emergencia
[q] Salir del programa
[h] Help

1
'-1'Exit.
Select target:
7
Target Updated
■

```

Figura 28: Selección de objetivo en vídeo del dron

Seleccionamos la opción del menú para indicar el track que aparece en el vídeo del dron e introducimos el track 7. En este momento se actualiza el track al que debe perseguir y se crea la hebra que ejecuta el comportamiento reactivo.

Durante la ejecución ponemos a prueba la consistencia en la detección y el tracking situando el dron en situaciones adversas.



Figura 29: Detección a contra luz - Bebop

La captura ha sido tomada directamente del vídeo grabado a bordo del dron por eso no aparece el Bounding Box pero si observamos el vídeo capturado en la ejecución en servidor vemos que el objetivo sigue siendo detectado y '*trakeado*'.

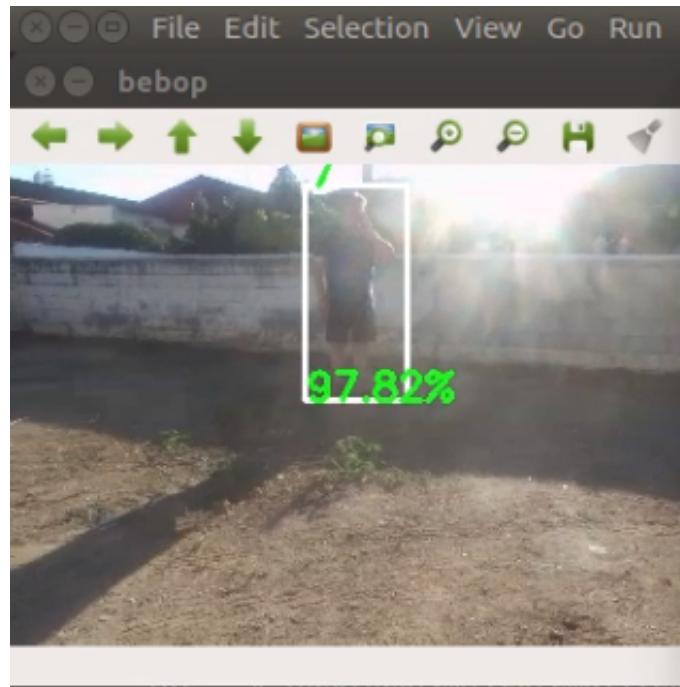


Figura 30: Detección a contra luz - Servidor

Otra situación compleja es llevar al objetivo a una zona oscura rodeada de mucha luz. Lo que hacemos es situar al dron en una escena con altos contrastes. Al igual que antes, la primera imagen corresponde con el vídeo a bordo del dron y la segunda a la captada en servidor.



Figura 31: Detección en alto contraste - Bebop

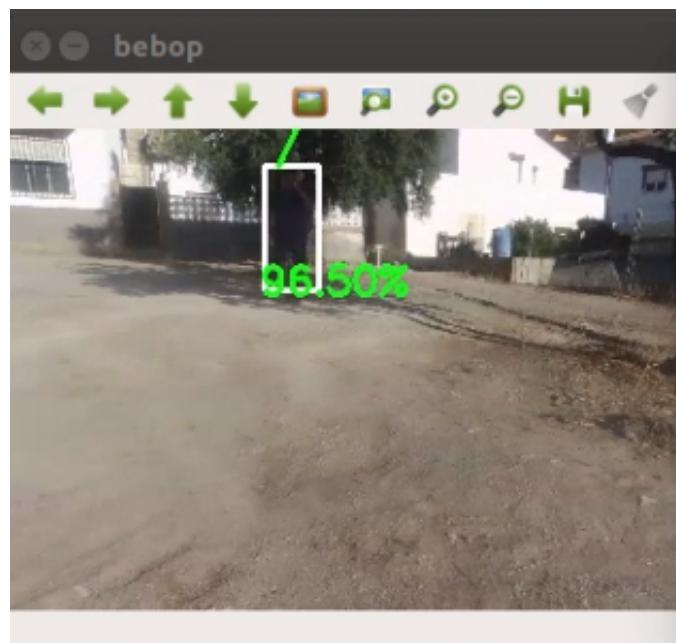


Figura 32: Detección en alto contraste - Servidor

En esta situación también se desenvuelve bien la aplicación.

Para ver realmente cómo funciona el sistema y ver todo el procedimiento de la ejecución realizada, hemos dejado en la carpeta de MEGA [23] unos vídeos que muestran toda la consecución de hechos de principio a fin.

Para finalizar vamos a mostrar los gráficos de la evolución de los FPS y el número de track que aparecen en escena durante la ejecución.

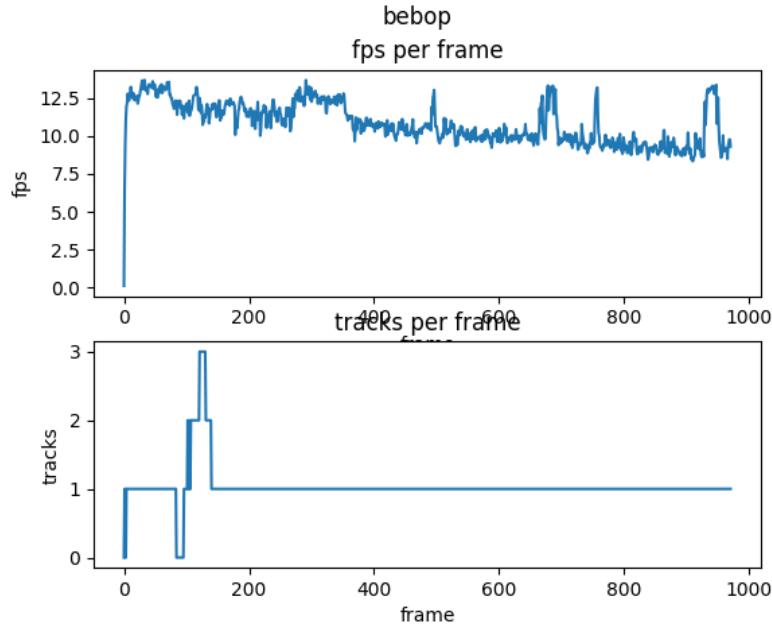


Figura 33: Evolución en vídeo Bebop

Al principio de la figura vemos que el número de tracks es inestable y es porque se está transportando el dron hacia la zona de vuelo y, durante el trayecto, se realizan más detecciones pero durante el resto de la ejecución el número de tracks se mantiene constante en 1.

El decremento de FPS a lo largo de la ejecución se debe a la pérdida de frames durante la transmisión como hemos comentado tras la figura 24. Por eso se producen picos de subida, que coinciden con los momentos en los que obtenemos una mejora de conexión y evitamos dicha pérdida de datos.

Aún con la pérdida de fotogramas, la media de FPS se mantiene cerca de los números dados al inicio de la sección obteniendo 10.85 FPS de media y rondando los 12.5 al principio de la ejecución, de hecho, alcanzando un pico de 13.72 FPS.

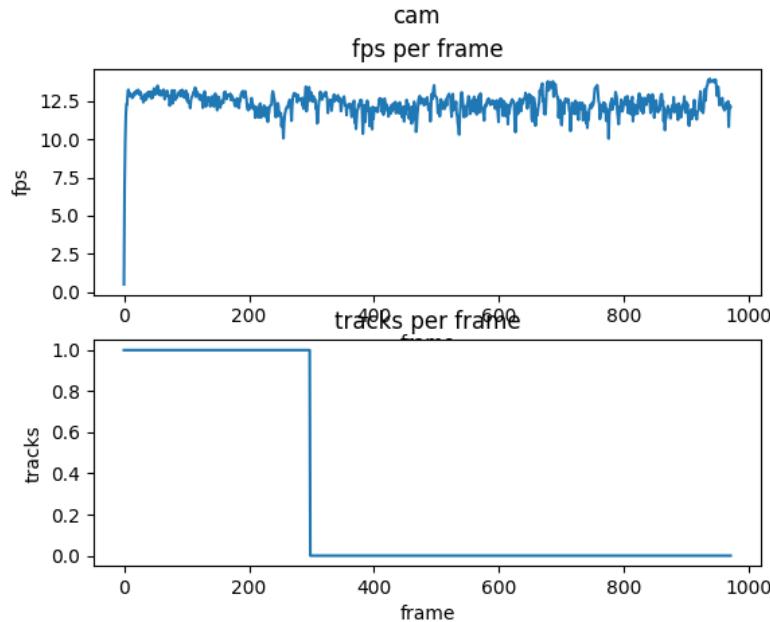


Figura 34: Evolución en vídeo Cámara fija

Respecto a la cámara fija, el número de FPS es relativamente constante y se mantiene en todo momento en torno a 12.5. En el subgrafo que marca el número de tracks, se aprecia que iniciamos la ejecución con una persona en escena y que, sobre el frame número 300, desaparece de ella. Es aquí cuando se manda al dron la orden de despegar porque hemos perdido el track de la persona que habíamos marcado como objetivo.

6.3.4. Experiments Varios

- **Ancho de banda:** Hemos medido el ancho de banda que consume comunicarnos con el dron desde el framework ROS. El *topic image_raw* por el que mandamos los mensajes tipo *Image* que contienen los frames del vídeo que capta el dron, ocupa un ancho de banda de 37.3MB/s. Y el *topic cmd_vel* por el que mandamos los comandos de movimiento, suponen 1.57KB/s de consumo de ancho de banda.
- **Consumo de batería:** Hemos comparado el consumo de batería al usar la aplicación móvil propietaria de *Parrot* y al usar la aplicación desarrollada en este proyecto. A la hora de medir el consumo de batería lo hemos hecho con el dron en reposo tan sólo haciendo uso del módulo de visión por computador *trackeando* a una persona. Hemos quitado de la ecuación el hecho de volar el dron, ya que en los dos sistemas supone el mismo esfuerzo y, de hecho, puede haber variaciones debidas a ráfagas de viento ante las que el dron se tenga que estabilizar. Por

ello, tan sólo hemos medido el consumo ejecutando la característica que realmente diferencia a los dos sistemas: su método de *tracking*.

Para hacer la medición, marcamos una figura a seguir y mantenemos la ejecución durante 15 minutos. Tras este tiempo los resultados fueron:

- **Sistema desarrollado durante el proyecto:** Empieza en un 98 % de batería y finaliza con 93 %.
- **Sistema propietario de Parrot:** Empieza con 91 % y acaba con 85 %.

La diferencia no es, ni mucho menos, significativa (5 % vs 6 %) aunque puede que si alargásemos la prueba, las distinciones fueran más notables. Aún si fuera el caso, no repercutiría mucho a la hora de decidir qué sistema usar, ya que las baterías del dron permiten vuelos de 25 minutos según el fabricante, lo cual nos dice que serán alrededor de 20 minutos reales con unas baterías nuevas.

7. Conclusiones y Futuras Mejoras

En este proyecto hemos estudiado y desarrollado un sistema de video vigilancia activo mediante la incorporación de drones que supone una solución a los problemas que sufren los sistemas tradicionales y que va especialmente enfocado a la supervisión de zonas o infraestructuras críticas debido a su carácter automático.

A continuación se definen los objetivos planteados inicialmente y el estado en el que se encuentran tras finalizar el proyecto.

En primer lugar, se estableció el objetivo de estudiar el estado del arte de los sistemas de video vigilancia y del uso dado a los drones. Tras este estudio, pusimos de manifiesto la amplia gama de sectores e industrias en las que se utilizan los drones, y las diferentes tareas a las que están destinados en cada una de ellas. Respecto a los sistemas de vigilancia actuales, se presentan los problemas socioeconómicos que conlleva actualmente la video vigilancia con personal humano y más aún en infraestructuras críticas.

Se propuso, en segundo lugar, aprender las funcionalidades y herramientas que concede el framework ROS y que nos permiten integrar módulos de naturaleza y comportamiento heterogéneo en un mismo sistema dirigido a trabajar con robots. Durante la realización del proyecto hemos aprendido a incorporar paquetes de terceros en el sistema, hemos creado un paquete propio, aprendido a trabajar con su sistema de suscripciones y publicaciones en *topics*, hemos aprendido cómo es su estructura de directorios, los comandos más comunes con los que trabajar en el framework, a crear ficheros de ejecución complejos y, debido a los diversos problemas a la hora crear una instalación sólida, hemos estudiado distintas versiones y distribuciones del producto como ROS Melodic o ROS2.

En tercer lugar, nos propusimos estudiar el estado del arte de algoritmos de detección y tracking de personas y valorar su capacidad de funcionamiento en un sistema de video vigilancia. Las mejores opciones encontradas pertenecían al ámbito de la Inteligencia Artificial y así, no sólo estudiamos los métodos más actuales en la labor de identificación y seguimiento de personas, sino que, además, los implementamos en nuestro proyecto dotando al mismo de características igual de potentes como llamativas. Se ha implementado el algoritmo DeepSORT junto a la CNN (red neuronal convolucional) YOLOv3 y se ha mostrado su eficacia junto con los costes computacionales que supone y las limitaciones que conlleva. Estas limitaciones nos obligaron a ejecutar la aplicación en un equipo que contase con una tarjeta gráfica NVIDIA dedicada para trabajar con los drivers CUDA.

Nos propusimos un cuarto objetivo poco ambicioso, pero a su vez muy necesario, útil y realista: la elaboración de una funcionalidad que nos permitiese controlar el dron de manera manual. Este objetivo, que comparado con la meta final del proyecto, no ~~causa gran alboroto~~, es un buen paso intermedio que nos ha permitido conocer la integración de ROS con python mediante bibliotecas, y crear un mecanismo de seguridad en el sistema por el que podemos tomar el control total del dron si fuera necesario.

Como quinto objetivo se fijó intentar dotar el dron de un comportamiento reactivo por el cual fuese capaz de perseguir a un objetivo indicado. A través de las características obtenidas por los algoritmos de detección y tracking somos capaces de focalizar el seguimiento en una única persona de entre una multitud, y adaptar los movimientos del dron según aparezca esa persona representada en escena. Esta funcionalidad se ve altamente afectada por el número de FPS que consiga procesar nuestro algoritmo de tracking, por lo que para obtener buenos resultados, debemos disponer de un buen equipo hardware.

El sexto objetivo trata de dar un enfoque más genérico al proyecto y hacerlo multipropósito. Se pretende realizar un sistema configurable de manera que podamos introducirlo en la mayor cantidad de sectores e industrias diferentes posibles. Esta capacidad la hemos obtenido gracias a incorporar YOLOv3 como detector del sistema que además ha sido entrenado con la base de datos COCO. De esta manera, no es sólo capaz de identificar personas, también gran cantidad de animales, vehículos y objetos varios. Sin retocar demasiado el sistema actual, también podríamos orientarlo al mundo de cine para grabar persecuciones o tomas aéreas de paisajes. Si decidimos, por ejemplo realizar detección y seguimiento de animales de granja, podríamos usarlo, además de para vigilar los animales, para realizar la función de pastor o guía. Si reentrenamos la red neuronal para detectar ciertos tipos de señales arbitrariamente seleccionadas, podríamos inferir en el sector de la agricultura y recorrer grandes campos de cultivos señalizados con estas balizas para plantar semillas, echar pesticida o, de nuevo, llevar un control de la zona. Son muchas las aplicaciones que podemos dar al sistema ya que, además de la libertad que nos confiere el detector, el módulo de comportamiento es fácilmente desacoplable y podríamos diseñar otro nuevo o modificar el existente para adaptarlo a nuevas funcionalidades.

El séptimo objetivo es contribuir a la comunidad de software libre, más específicamente, a la comunidad orientada a la robótica, liberando el código desarrollado y alojarlo en GitHub. Esto se ha hecho tal cual se prometió y podemos encontrar el proyecto subido en el siguiente enlace [13].

Por último, gracias a mis tutores del trabajo de fin de grado y a los miembros del grupo de investigación de FITOPTIVIS que me han ayudado con el proyecto, he aprendido las herramientas de planificación y comunicación *Trello* y *Slack*, las cuales consiguen hacer estas tareas más sencillas y además permiten ver mi planificación y avances a mis supervisores del proyecto.

Como futuras mejoras se propone mejorar el equipo hardware sobre el que se ejecuta el algoritmo de *tracking*, ya sea adquiriendo mejor hardware local o implementar la ejecución del código en un servidor externo.

También podemos incorporar nuevas características al sistema de visión como un software que extraiga un mapa de profundidad a partir de una imagen 2D para así poder evitar o sortear obstáculos con el dron. También podríamos implementar un sistema de posicionamiento 3D con la ayuda de varias cámaras estáticas de manera que cuando el sujeto objetivo abandonase la escena tendríamos unas coordenadas espaciales a las que poder enviar el dron para seguir buscando el objetivo.

Por supuesto debemos estar atentos a los avances en el estado del arte de cualquier aspecto de rendimiento y eficiencia considerados en el proyecto. Dado que se ha desarrollado de manera modular, no es difícil actualizar cualquiera de los módulos del proyecto sin afectar al funcionamiento del resto del sistema.

Referencias

- [1] Joseph Redmon - YOLOv3 - Paper <https://pjreddie.com/media/files/papers/YOLOv3.pdf>.
- [2] CB-Insights <https://www.cbinsights.com/research/drone-impact-society-uav/>.
- [3] Divya Joshi - Business Insider <https://www.businessinsider.com/drone-technology-uses-applications?IR=T>.
- [4] Federal Aviation Administration. <https://www.faa.gov/>.
- [5] ROS. <https://www.ros.org/>.
- [6] ROS Wiki. <http://wiki.ros.org/ROS/Introduction>.
- [7] Mani Monajjemi y Colaboradores. <http://bebop-autonomy.readthedocs.io/en/latest/contribute.html#sec-contribs>.
- [8] Bebop2. <https://www.parrot.com/es/drones/parrot-bebop-2>.
- [9] Joseph Redmon - YOLOv3. <https://pjreddie.com/darknet/yolo/>.
- [10] Modelo de Prototipos. https://es.wikipedia.org/wiki/Modelo_de_prototipos.
- [11] Trello. <https://trello.com/es>.
- [12] Slack. <https://slack.com/intl/es-es/>.
- [13] Repositorio del proyecto - TFG-Bebop-YOLO. <https://github.com/JJavier98/TFG-Bebop-YOLO>.
- [14] VirtualBox. <https://www.virtualbox.org/>.
- [15] NVIDIA CUDA. <https://developer.nvidia.com/cuda-zone>.
- [16] CITIC. <http://citic.ugr.es/>.
- [17] DeepSORT. <https://nanonets.com/blog/object-tracking-deepsort/>.
- [18] Qidian213 - DeepSORT. https://github.com/Qidian213/deep_sort_yolov3.
- [19] DeepSORT videocaptureasync module. https://github.com/Qidian213/deep_sort_yolov3/issues/154.
- [20] Distancia de Mahalanobis. https://es.wikipedia.org/wiki/Distancia_de_Mahalanobis.

- [21] Hungarian Algorithm. https://en.wikipedia.org/wiki/Hungarian_algorithm, https://es.wikipedia.org/wiki/Algoritmo_h%C3%BAngaro.
- [22] MOT Challenge. <https://motchallenge.net/>.
- [23] Carpeta MEGA con resultados de los experimentos. https://mega.nz/folder/YQpWEAiY#Xq4C890boVrrkhrA-68_AQ.

*