



ugr | Universidad
de **Granada**

TRABAJO FIN DE GRADO
INGENIERÍA EN INFORMÁTICA

Integración de Drone para aplicaciones de videovigilancia

Autor

José Javier Alonso Ramos

Directores

Eduardo Ros Vidal

Francisco Barranco Expósito



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y DE
TELECOMUNICACIÓN

—
Granada, Julio de 2020

Integración de Drone para aplicaciones de videovigilancia

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, convolucional, 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 videovigilancia activo para infraestructuras críticas **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 marco de trabajo ROS (Robot Operating System)

El objetivo final del trabajo es el de dotar de autonomía un dron, para que realice tareas de videovigilancia. **Para ello, se aplican algoritmos de visión por computador y de aprendizaje máquina.** Las principales tareas son las de detección y seguimiento de personas, usando la entrada de vídeo proporcionada por la cámara integrada del dron.

Explicaremos las funcionalidades de ROS, el controlador del dron en cuestión (Bebop Autonomy) y el la implementación del comportamiento reactivo. **Finalmente, mostramos los experimentos llevados a cabo y discutimos los resultados obtenidos.**

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.

Drone integration for video surveillance applications

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 an active agent with an onboard camera. The entire project has been implemented in ROS (Robot Operating System).

The final objective of this project is to provide autonomy to a drone, in order to do video surveillance tasks with it. To do this, computer vision and machine learning algorithms are used. The main tasks are people detection and people tracking using the video stream gave by the drone onboard camera.

ROS functionalities will be explained as well as the drone driver (Bebop Autonomy) and the code that implements the reactive behavior. Finally, we show the experiments and discuss the results.

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 hacerme 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 al igual que mi padre, siempre ha estado dispuesto a echar una mano en las pruebas del proyecto, 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 julio de 2020.

Índice

1. Introducción	14
1.1. Motivación	15
1.1.1. Por qué automatizar el sistema	15
1.1.2. Desarrollo de un sistema en estado del arte	16
1.1.3. ROS como marco de trabajo	17
1.2. Estado del Arte	18
1.2.1. Modelos de drones	18
1.2.2. Uso de drones en la industria	20
1.2.3. Sistemas de videovigilancia	22
1.2.4. Algoritmos de Detección y Seguimiento	23
1.3. Objetivos	24
2. Gestión del Proyecto	25
2.1. Metodología de Desarrollo	25
2.2. Planificación	25
2.3. Hitos	27
2.4. Presupuesto	29
2.4.1. Recursos Hardware	29
2.4.2. Recursos Software	29
2.4.3. Recursos Humanos	30
2.4.4. Coste Total	30
3. Entorno Operacional	31
3.1. Ubuntu 16.04	31
3.2. Robotic Operating System	31
3.2.1. Conceptos	32
3.3. Driver <i>bebop-autonomy</i>	35
3.3.1. Comandos ROS	37
3.4. Bebop 2	38
3.4.1. Especificaciones	38
3.5. Ordenador	40
4. Diseño	41
5. Implementación	43
5.1. Visión por Computador	43
5.1.1. Detector - YOLOv3	43
5.1.2. Tracker - DeepSORT	48
5.1.3. Captación de imágenes del dron	49
5.1.4. Captación de imágenes de la webcam	50
5.1.5. Lectura de archivo de vídeo almacenado	50
5.1.6. Escritura de vídeo	52

5.2.	Comportamiento del Dron	53
5.2.1.	Pilotaje Manual	53
5.2.2.	Pilotaje Automático	54
5.2.3.	Opciones Menú Principal	55
6.	Problemas durante el desarrollo	56
6.1.	Restricciones Harware	56
6.2.	Restricciones Software	56
6.3.	Restricciones Sanitarias	56
6.4.	COVID-19	57
6.5.	Problemas Técnicos	57
6.5.1.	Virtualización	57
6.5.2.	Equipo de Trabajo	58
6.5.3.	Límite de Distancia de Conexión	59
6.5.4.	Python y DeepSORT	60
6.5.5.	Instalación de ROS2 junto a ROS Kinetic	60
6.5.6.	Demo práctica del repositorio de <i>Qidian213</i> poco eficiente	61
7.	Experimentación	63
7.1.	Pruebas del Detector	64
7.1.1.	Rendimiento ante distintas resoluciones	65
7.1.2.	Oclusión	66
7.1.3.	Conclusión sobre la Resolución	67
7.2.	Eficiencia del tracker DeepSORT en MOT Challenge	68
7.2.1.	MOT17_12	68
7.3.	<i>Baseline</i> o Ejecución de referencia	69
7.4.	Comparación de Intervalos Superiores a 1	74
7.4.1.	Intervalo: 2 fotogramas	74
7.4.2.	Intervalo: 3 fotogramas	79
7.5.	Consistencia del Tracker	83
7.6.	Comparación: DeepSORT vs UnsupTrack	85
7.7.	Pruebas con el Dron	89
7.7.1.	Ejecución de la aplicación desarrollada	89
7.7.2.	Prueba de aplicación final	91
7.7.3.	Vuelo con app	97
7.7.4.	Ejecución en Equipo Portatil	100
7.7.5.	Otros experimentos para evaluar Características No Funcionales de Calidad	102
8.	Conclusiones y Futuras Mejoras	104

Índice de figuras

1.	Sistema de vigilancia tradicional	17
2.	Sistema de vigilancia moderno	17
3.	Dron de vuelo estático: Octocóptero	18
4.	Aeronaves no tripuladas	18
5.	Cuadricóptero	18
7.	2015 - Uso de drones en la industria	21
9.	Axis Communications - Detección y Seguimiento	22
11.	Comparación de YOLOv3 con otros detectores	23
13.	Captura del tablero de Trello	25
14.	Diagrama de Gantt de Planificación	26
16.	Esquema de flujo de información mediante Tópics en ROS . .	32
18.	Estructura de mensaje Image	33
20.	Estructura de mensaje Twist	34
22.	Esquema de desplazamiento de Bebop 2 con velocidades po- sitivas	35
24.	Organización de directorios de bebop_autonomy	36
26.	Interfaz de conexión Bebop2 - ROS - PC para la obtención de imagen del dron.	37
28.	Parrot Bebop 2	38
30.	Esquema de nodos ROS en la aplicación	41
32.	Esquema de comunicación entre los módulos de la aplicación	42
33.	Rendimiento de YOLOv3 frente a otros algoritmos de detección	43
35.	Estructura de Darknet-53	44
37.	Recorrido de la imagen con dos tipos de <i>ventanas de búsqueda</i>	46
39.	Ejemplo de codificación mediante celdas. Imagen original (608,608,3) codificada a (19,19,5,85)	46
41.	Supresión de NO máximos	47
43.	Comparación de algoritmos de tracking en estado del arte . .	48
45.	Ejemplo de archivo txt generado	52
46.	Menú principal + Menú de pilotaje manual	53
47.	Diálogo de selección de objetivo	54
48.	La imagen muestra la evolución de la cantidad de <i>fps</i> y tracks detectados a lo largo de los frames obtenidos. Como vemos, los <i>fps</i> rondan los 0.7. Analizaremos esta gráfica en mayor profundidad más adelante en el apartado de experimentación.	58
49.	Detección ineficiente en demo.py	61
51.	MOT17_12 imagen:0 426x240	64
52.	MOT17_12 imagen:1 854x480	64
53.	MOT17_12 imagen:2 1920x1080	65
54.	<i>fps</i> frente a la resolución de la imagen	66
56.	Número de tracks detectados por fotograma	69
57.	Tiempo de procesamiento requerido por fotograma	69

58.	Evolución suavizada de trazados en escena	70
59.	Comparación de la evolución de trazados y el tiempo de procesamiento por fotograma	70
60.	Número de tracks detectados por fotograma	71
61.	Tiempo de procesamiento requerido por fotograma	71
62.	Comparativa de evolución de trazados	72
63.	Tiempo de procesamiento junto con evolución de tracks	72
64.	Número de tracks detectados por fotograma	75
65.	Tiempo de procesamiento requerido por fotograma	75
66.	Comparativa de evolución de trazados	75
67.	Tiempo de procesamiento junto con evolución de tracks	75
68.	Número de tracks detectados. Intervalo:2 Periodo: 1	76
69.	Número de tracks detectados. Intervalo:1 Periodo: 2	77
70.	Número de tracks detectados por fotograma	79
71.	Tiempo de procesamiento requerido por fotograma	79
72.	Comparativa de evolución de trazados	79
73.	Tiempo de procesamiento junto con evolución de tracks	79
74.	MOT17_05 secuencia:0	83
75.	MOT17_05 secuencia:1	83
76.	MOT17_05 secuencia:2	83
77.	MOT17_05 secuencia:3	83
78.	MOT17_05 secuencia:4	83
79.	MOT17_06 secuencia:1	85
80.	MOT17_06 secuencia:2	85
81.	MOT17_06 secuencia:3	86
82.	MOT17_06 secuencia:4	86
83.	MOT17_06 secuencia:5	86
84.	MOT17_06 secuencia:6	86
85.	MOT17_06 secuencia:7	86
86.	MOT17_06 secuencia:1	87
87.	MOT17_06 secuencia:2	87
88.	MOT17_06 secuencia:3	87
89.	MOT17_06 secuencia:4	87
90.	MOT17_06 secuencia:5	87
91.	MOT17_06 secuencia:6	87
92.	MOT17_06 secuencia:7	88
93.	Trazados en modo solo visión (pilotaje manual)	90
94.	Tiempos de procesamiento en modo solo visión (pilotaje manual)	90
95.	Trazados en modo pilotaje automático	90
96.	Tiempos de procesamiento en modo pilotaje automático	90
97.	Inicio del sistema	91
98.	Selección de ID en cámara fija	92
99.	Salida de escena	93

100. Despegue automático	93
101. Selección de objetivo en vídeo del dron	94
102. Detección a contra luz - Bebop	94
103. Detección a contra luz - Servidor	95
104. Detección en alto contraste - Bebop	95
105. Detección en alto contraste - Servidor	96
106. Interfaz HUD (Head-Up Display) del modo pilotaje manual de la aplicación Free Flight Pro	97
107. Interfaz HUD (Head-Up Display) del modo pilotaje automáti- co de la aplicación Free Flight Pro	98
108. Dron siguiendo al objetivo de manera activa	99
109. Trazados detectados en el equipo portátil	100
110. Tiempos de procesamiento en el equipo portátil	100
111. Estado del equipo portátil durante la ejecución de la aplicación	101
112. Trazados detectados en el equipo portátil	102
113. Tiempos de procesamiento en el equipo portátil	102

Índice de cuadros

1.	Tabla de costes hardware	29
2.	Tabla de costes software	29
3.	Tabla de costes totales.	30
4.	Tabla sintetizada de problemas, planes de contingencia y soluciones finales	62
5.	FPS frente a Resolución	65
7.	Número de Trazados frente a Resolución	67
9.	Resultados de ejecución en MOT17_12	73
11.	Comparativa Intervalo vs Periodo	76
12.	Resultados de ejecución en MOT17_12	78
14.	Comparativa Intervalo vs Periodo	80
15.	Resultados de ejecución en MOT17_12	81
17.	Comparativa de ejecuciones con distintos intervalos de lectura	82

1. Introducción

Hoy en día, empresas, instituciones públicas y particulares invierten cada vez más en sistemas de videovigilancia para llevar un control riguroso de ciertas zonas, infraestructuras o viviendas y asegurar así su integridad. Muchas de estas áreas son extremadamente sensibles ante posibles “atacantes” y la importancia de su conservación y buen funcionamiento es vital tanto para la entidad contratante como para la sociedad. Nos referiremos a estas áreas sensibles como infraestructuras críticas, término definido por la Directiva Europea el 8 de diciembre de 2008 [1] como:

“El elemento, sistema o parte de este situado en los Estados miembros que es esencial para el mantenimiento de funciones sociales vitales, la salud, la integridad física, la seguridad, y el bienestar social y económico de la población y cuya perturbación o destrucción afectaría gravemente a un Estado miembro al no poder mantener esas funciones”

Ejemplos de infraestructuras críticas serían: Administración (servicios básicos, instalaciones, redes de información, y principales activos y monumentos del patrimonio nacional); Instalaciones del Espacio; Industria Química y Nuclear (producción, almacenamiento y transporte de mercancías peligrosas, materiales químicos, biológicos, radiológicos, etc.); Agua (embalses, almacenamiento, tratamiento y redes); Centrales y Redes de energía (producción y distribución); Tecnologías de la Información y las Comunicaciones (TIC); Salud (sector e infraestructura sanitaria); Transportes (aeropuertos, puertos, instalaciones intermodales, ferrocarriles y redes de transporte público, sistemas de control del tráfico, etc.); Alimentación (producción, almacenamiento y distribución); y Sistema Financiero y Tributario (entidades bancarias, información, valores e inversiones).

La videovigilancia de estas áreas es a lo que va dedicado este proyecto; más concretamente a la videovigilancia activa e inteligente incorporando un elemento móvil (dron) y dotando al sistema de visión por computador.

En los últimos años se han obtenido grandes avances en el campo de visión por computador gracias al desarrollo de avanzadas y complejas técnicas (algoritmos) de Inteligencia Artificial y aprendizaje profundo que permiten una detección precisa de distintas figuras en escena e, incluso, un seguimiento (*tracking*) de múltiples objetos (MOT). Esta situación nos crea un marco contextual ideal para desarrollar sistemas inteligentes de videovigilancia.

Como elemento activo del sistema contamos con un dron Parrot Bebop 2 el cual es la herramienta ideal para desarrollar este papel, al menos, en un ámbito académico y/o de investigación, ya que es una plataforma que nos aporta total libertad para su configuración. Nos permite modificar su

firmware de una manera sencilla (dispone de un puerto USB para conectarlos directamente a la placa controladora) y, además, gracias a bibliotecas disponibles en Python y C++ y a la implementación de los drivers de vuelo en un paquete (*Bebop Autonomy* [2]) del framework ROS (*Robot Operating System* [3]) es relativamente sencillo desarrollar software de visión y pilotaje del aparato.

Dentro del mundo de los drones que podemos obtener de manera particular (sin ser una empresa o institución), se trata de una muy buena opción por todo lo nombrado anteriormente y por su precio, que es sustancialmente menor que otras marcas y modelos.

1.1. Motivación

1.1.1. Por qué automatizar el sistema

Como ya hemos dicho, este trabajo de fin de grado está enfocado al sector de la *videovigilancia*, más específicamente a la videovigilancia 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 frecuentado por personas. Además su seguridad, integridad y buen funcionamiento son de muy alta importancia.

El hecho de que se encuentre aislado y poco accesible es lo que hace muy interesante la incorporación de un sistema inteligente para vigilar el lugar. Esto es porque, actualmente, es personal cualificado que se encuentra físicamente allí, el que se encarga de la vigilancia bien mirando pantallas o monitores 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 (Brown, Carlyle, Salmerón, & Wood, 2006) [4].

En el peor de los casos (Falso Negativo: No dar la alarma ante una emergencia real) 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 sufridos y/o ocasionados.

El caso más leve (Falso Positivo: dar la alarma ante una situación de no peligro) supondrá el despliegue de un equipo policial hasta la zona y acarrear dichos costes. Si se dan varias falsas alarmas, el sistema de vigilancia se considerará poco efectivo y perderá credibilidad de manera que las autoridades pueden no considerar relevante futuras emergencias.

Todo esto junto con el elevado coste periódico que supone mantener un personal especializado para la vigilancia, 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 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.

Dicho esto sobre los sistemas tradicionales, hay que decir que los sistemas de vigilancia más modernos dotados de reconocimiento facial también fallan. Pueden confundir a dos personas de su base de datos si estas son muy parecidas (por ejemplo gemelos) o, como sucede en un caso reportado recientemente sobre el sistema de detección desarrollado por Amazon [5], que el algoritmo de detección no esté suficientemente entrenado para distinguir determinadas características (en el caso de Amazon, distintas tonalidades oscuras de piel). En nuestro caso, esto no supondrá un problema ya que supondremos que todo el que aparezca en escena será un potencial intruso y no identificaremos a la persona detectada más que con un número o *ID*.

1.1.2. Desarrollo de un sistema en estado del arte

El sistema de videovigilancia inteligente contará con funcionalidades propias de sistemas modernos que utilizan técnicas y funcionalidades punteras de visión por computador como son la detección y seguimiento de personas en tiempo real. Estas técnicas aportan mucha más información sobre la situación y permiten al sistema funcionar incluso sin supervisión humana si se utilizan algoritmos suficientemente eficaces. En nuestro caso utilizaremos algoritmos en estado del arte: como detector, usaremos la red neuronal convolucional (CNN) YOLOv3 [6] y, como algoritmo de seguimiento, Deep-SORT [7].

Esto marca una gran diferencia respecto a sistemas de videovigilancia tradicionales donde el vídeo entregado a los vigilantes de la infraestructura esas tan solo la imagen en bruto extraída de las cámaras. Ahora podemos indicar cuántas personas hay en escena, marcar su posición en el plano 2D, asignarles un identificador y seguir su recorrido mientras se mantengan en escena. Además, el objetivo de este trabajo es ampliar aún más estas funcionalidades, incorporando un dron como elemento activo del sistema que permita realizar la persecución de una determinada persona identificada en escena para no perder su posición en caso de que se salga del plano.

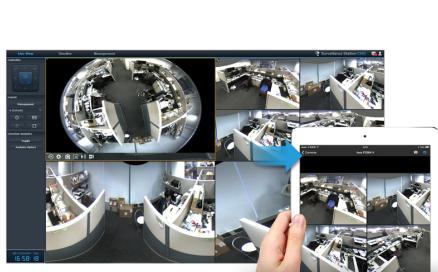


Figura 1: Sistema de vigilancia tradicional

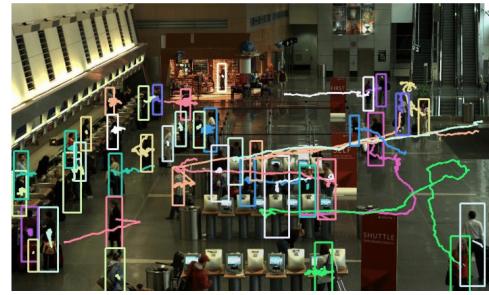


Figura 2: Sistema de vigilancia moderno

En la Figura 1 se aprecia un sistema de videovigilancia que muestra la imagen en bruto de una serie de cámaras fijas mientras que en la imagen de la derecha, se está aplicando detección de personas que salen marcadas dentro de un recuadro o *Bounding Box* y a las que además se realiza un seguimiento del recorrido que hacen por la escena dibujando un trazo allí por donde pasan.

1.1.3. ROS como marco de trabajo

Se pretende crear un sistema modular y escalable que permita la incorporación de múltiples elementos heterogéneos (cámaras fijas, drones, alarmas, mecanismos de cierre de puertas, ...), hacer un uso sincronizado de ellos y que permita al usuario final utilizar el sistema de manera sencilla. Estos requisitos son los que definen el marco de trabajo ROS (Robot Operating System [3]) y es por esto que se ha elegido como plataforma sobre la que desarrollar el proyecto. De otro modo, se tendrían que haber desarrollado estas funcionalidades desde cero u optar por diseñar un sistema no tan completo.

1.2. Estado del Arte

1.2.1. Modelos de drones

En la actualidad encontramos modelos muy avanzados de UAVs (Vehículos Aéreos no Tripulados) los cuales podemos clasificar según el modo de desplazamiento para el que estén diseñados. Principalmente se separan en tres grupos:

- **Drones de vuelo estático:** son aquellos que cuentan con más de 4 rotores (suelen tener seis u ocho). Son los más usados en filmografía por la estabilidad de imagen que proporcionan. En contrapunto a esta estabilidad, poseen una velocidad de movimiento limitada.
- **Aeronaves no tripuladas:** son drones de gran envergadura que cuentan con un equipo de grabación muy avanzado, de gran resolución y con un potente zoom. Son utilizados para cubrir grandes áreas de terreno en poco tiempo ya que vuelan a gran velocidad.
- **Cuadricópteros:** son un término intermedio entre los dos anteriores, y también los más comunes. Poseen estabilidad y velocidad de movimiento sin llegar a ser tan buenos en ninguno de los casos como los drones específicos anteriores.



Figura 3: Dron de vuelo
estático: Octocóptero



Figura 4: Aeronaves no
tripuladas



Figura 5: Cuadricópte-
ro

Imágenes obtenidas de Google Images

Los tipos de drones de las figuras 3 y 5 son también conocidos como VTOL (*Vertical Take-Off and Landing* o Despegue y Aterrizaje en Vertical), por ser capaces de despegar y aterrizar sin la necesidad de recorrer distancias horizontales.

A parte de cómo sea su modo de vuelo, hay características más determinantes a la hora de clasificar a un dron como tecnología puntera o no:

- **Posee GPS:** El dron dispone de un chip GPS que le permite geolocalizar su posición, realizar planes de vuelo indicándole coordenadas a alcanzar y volver a “casa”, es decir, el punto de partida del vuelo.
- **Detección de obstáculos y tecnología anti-colisión:** disponen de sensores de cercanía, radares 3D o software de visión por computador que les permite percibir objetos de su entorno y evitar colisionar con ellos.
- **Giroscopio:** provee al dron de información sobre la estabilidad de vuelo. Indica que inclinación tiene en sus ejes para poder ajustar la potencia de cada motor de manera individual para tratar de mantener una posición lo más estable posible.
- **Transmisión de datos de vuelo:** La comunicación entre el controlador y el dron puede ser unidireccional o bidireccional. En el primer caso, el dron tan sólo recibirá los comandos de movimiento del controlador y los ejecutará. En el segundo, a parte de esto, mandará información de la situación actual: estado de la batería, estado de motores, altura, velocidad, conexión, avisos por tiempo inestable, etc.
- **Retransmisión de vídeo en tiempo real:** el dron transmite en tiempo real las imágenes que capta por la cámara de abordo. Estas imágenes podemos usarlas para realizar un posprocesamiento de las mismas y obtener información de ellas y/o para volar el dron en modo FPV (*First Person View*), es decir, valernos de la imagen proporcionada para controlar el dron desde su perspectiva de modo que no es necesario que tengamos visión directa del aparato para controlarlo.
- **Posibilidad de actualización de firmware:** el dron posee una interfaz de conexión a su tarjeta controladora para poder actualizar o modificar su modo de funcionamiento.
- **Cámara de abordo de gran calidad:** la cámara de abordo del dron y el software que la controla pueden tener varias características. **Cuántas más de ellas posea, o mejor sea más avanzado consideraremos al dron en este aspecto.**
 - a Resolución. Cuanto mayor, mejor.
 - b Estabilizador óptico y/o digital
 - c Zoom óptico y/o digital
 - d Frecuencia de muestreo. Cuanto mayor, mejor.
 - e Visión térmica
 - f Visión nocturna

- **Seguridad en la conexión:** se valora la seguridad de conexión entre el dron y el controlador de manera que un tercero no pueda interrumpir la comunicación o espiar los datos que se transmiten.

El dron utilizado en el proyecto es el Parrot Bebop 2, al cual no podemos considerar una herramienta puntera ya que carece de algunas características nombradas (como sistema anti-collision) y puede mejorar otras (resolución, frecuencia de muestreo,...). Aun así, supone una herramienta más que válida para realizar las pruebas del proyecto.

Drones que sí pueden ser considerados como estado del arte son los modelos de *DJI* que son utilizados tanto por particulares como por empresas dado a que son relativamente asequibles y proporcionan una muy buena calidad de vídeo y sensación de vuelo.

1.2.2. Uso de drones en la industria

Uno de los campos en el que es más fácil ver su amplio uso es en la industria del cine o filmografía más amateur [8] donde son muy utilizados para tomar amplios planos de un paisaje, grandes planos picados o para grabar persecuciones.

Actualmente se está intentando implantar su uso en empresas de mensajería y paquetería como UPS [9], que ha desarrollado un sistema que aprovecha la sinergia del reparto simultáneo con camiones para paquetes pesados y con drones para paquetes ligeros. El gigante de comercio electrónico, Amazon, también está invirtiendo en su propia flota de drones y, de hecho, ya cuenta con varios modelos de prueba [10]. Hasta Correos, en España, ha realizado ya las primeras pruebas de reparto en zonas de difícil acceso mediante drones [11]. La transición hacia este nuevo método de reparto será lenta debido a la falta de legislación y a los grandes cambios de infraestructura que requiere.

También se está estandarizando el uso de drones en el control de planes de urbanismo (como nos explica este artículo de la Universidad de Córdoba [12]), en la vigilancia del tráfico (informe de la DGT [13]), como elemento de monitorización y optimización de plantaciones agrícolas (TFG presentado en la Universidad Politécnica de Valencia en 2017 [14]) y, recientemente, como sistemas de videovigilancia (artículo publicado por IEEE [15]), 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 se pueden encontrar en la actualidad. Para ver hasta 38 situaciones en las que usamos hoy en día los drones se recomienda echar un vistazo a este artículo de *CB INSIGHTS* [16] que ofrece una visión general del tema sin entrar mucho en detalle.

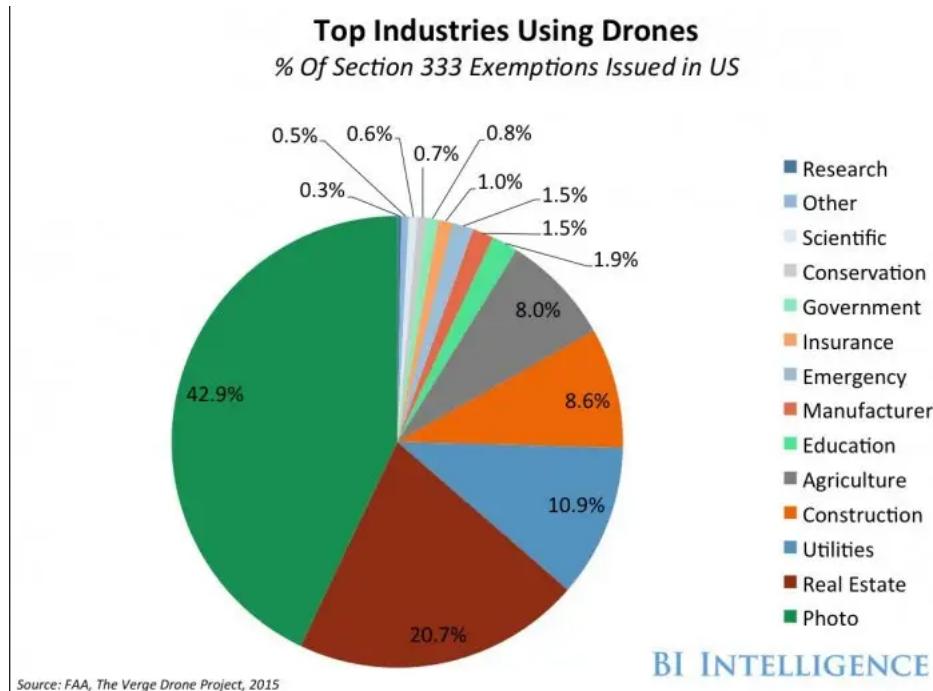


Figura 7: 2015 - Uso de drones en la industria

Gráfico extraído de Business Insider [17]

La figura 7 expuesta en el artículo [17] de *Business Insider* recoge los datos de la *Federal Aviation Administration - FAA* [18] de 2015 sobre el uso de los drones en las distintas industrias en Estados Unidos.

Como se puede ver, el mayor uso de drones (42.9 %) se da en la industria de la fotografía como hemos hablado antes. En segunda posición con aproximadamente 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, aproximadamente, la mitad de uso respecto al segundo puesto (10.9 %). En el cuarto y quinto puesto aparecen los sectores de construcción y agricultura respectivamente (8.6 % y 8.0 %), siendo el uso en el sector de construcción ligeramente superior (0.6 % mayor). La participación de los drones en el resto de sectores vemos que es muy escasa en comparación a los mencionados.

1.2.3. Sistemas de videovigilancia

Retomando el sector al que está orientado este proyecto, se debe mencionar la escasez de empresas que han actualizado su equipo e infraestructuras para adaptarse a las nuevas tecnologías de videovigilancia.

Para considerar que un sistema de videovigilancia se encuentra en estado del arte, debe poseer, a parte de un equipo de cámaras fijas tradicional, un equipo móvil dotado con cámaras y un software que procese en tiempo real las imágenes tomadas por ambos tipos de cámaras para reconocer y seguir a presuntos intrusos.

Una empresa puntera que muestra un ejemplo perfecto del estado del arte en sistemas de videovigilancia es Ascent Video Technologies (AVT) [19], una empresa estadounidense que ha incorporado algoritmos de detección y seguimiento (*tracking*) a todo su sistema compuesto por vehículos terrestres, aéreos y marítimos dotados con cámaras y radares de gran precisión. Otra empresa, además especializada en el uso de drones, es Microdrones [20], que además de ofrecer drones como sistema de vigilancia, lo hace también como método de inspección de zonas catastróficas, como elemento de ayuda a la investigación científica o para realizar topografías del terreno para minas a cielo abierto o construcciones.

Si focalizamos aún más en el objetivo de este proyecto, encontramos la empresa Axis Communications [21], una empresa de videovigilancia especializada en infraestructuras críticas, que realiza detección, seguimiento y posicionamiento 3D de intrusos detectados.



Figura 9: Axis Communications - Detección y Seguimiento

Imagen obtenida de Axis Communication [21]

1.2.4. Algoritmos de Detección y Seguimiento

La aparición de estos algoritmos y de la visión por computador en general, ha revolucionado la manera de procesar y analizar las imágenes de cualquier sistema. Los detectores nos permiten reconocer figuras que aparecen en la imagen, extraer sus características y clasificarlas según estas. Los algoritmos de seguimiento o *trackers* se valen de estas detecciones para identificar cada figura en escena e individualizar así cada detección. **Esto es muy útil para diferenciar entre detecciones de una misma clase (persona),** además de para estudiar el recorrido de la detección por la escena.

En cuanto a algoritmos del estado del arte en el ámbito de seguimiento de múltiples personas, se estudió desde un primer momento utilizar DeepSORT por ser su funcionamiento fácil de comprender, **darnos la libertad de escoger el algoritmo de detección que deseemos** y, lo más importante, funcionar en tiempo real. En cuanto al algoritmo de detección se encontraron tres candidatos: SSD, Mask-RCNN y YOLOv3. Los tres constituyen algoritmos de detección en tiempo real pero SSD y YOLOv3 son especialmente veloces en el procesamiento de las imágenes y Mask-RCNN destaca por su precisión en la detección. Tras realizar un estudio de los tres algoritmos descartamos SSD ya que en el artículo de YOLOv3 [22] encontramos que en esta tercera versión del detector, supera en precisión y velocidad al algoritmo **“Single Shot Detection”**.

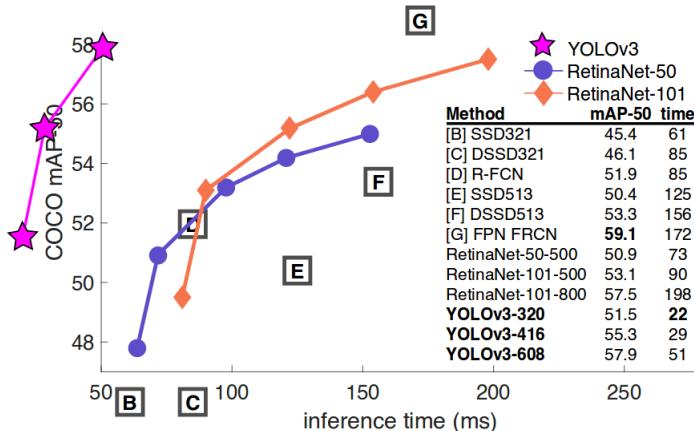


Figura 11: Comparación de YOLOv3 con otros detectores

Imagen obtenida del artículo de YOLOv3 [22]

En la imagen 33 vemos que YOLOv3, aceptando cualquiera de esos tres tamaños de imagen, es más rápido y preciso que las ejecuciones [B] y [E] de SSD.

Por otro lado, Mask-RCNN promete más precisión pero es hasta 100 veces más lento, de nuevo, según la página oficial de YOLOv3 [6]. Por esto, decidimos implementar YOLO en un primer momento y, si proporcionaba

fps de sobra, se implementaría Mask-RCNN para comparar los resultados. Como los *fps* obtenidos no resultaron ser muy elevados (alrededor de 5 *fps*), y la precisión obtenida era más que aceptable, se decidió establecer YOLO como detector del sistema sin llegar a probar el rendimiento de cualquier otra opción, pues según los estudios presentados en los artículos de YOLO, las pruebas iban a concluir con peores resultados.

Para observar una comparación entre Mask-RCNN y YOLO implementados en SORT, la versión sin aprendizaje profundo de DeepSORT, se puede consultar este enlace [23].

1.3. Objetivos

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

- Estudiar el estado del arte de los drones y los sistemas de videovigilancia.
- Aprender las funcionalidades y herramientas de las que dispone el framework ROS que nos permiten abstraernos de la interconexión y comunicación de nodos heterogéneos.
- Integrar algoritmos basados en Inteligencia Artificial utilizando un detector y módulo de seguimiento (*tracker*) basados en *deep learning*: YOLOv3 [22] y DeepSORT [7] respectivamente.
- Implementar un sistema de control manual para pilotar el dron.
- Implementar un comportamiento reactivo que permita al dron perseguir un determinado objetivo.
- Desarrollar software libre en forma de un paquete ROS alojado en GitHub.
- Aprender nuevos programas de planificación y comunicación como *Trello* y *Slack*.

2. Gestión del Proyecto

2.1. Metodología de Desarrollo

Se ha utilizado la metodología de *Desarrollo de Prototipos* [24] 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.

2.2. Planificación

Para organizar el proyecto se ha hecho uso de la plataforma *Trello* [25] conectada con la herramienta de comunicación en equipo *Slack* [26] de manera que tanto yo, como mis tutores hemos tenido acceso a la planificación y avances del trabajo. Además, también conectado al canal de Slack, se encuentra el repositorio del proyecto alojado en GitHub [27].

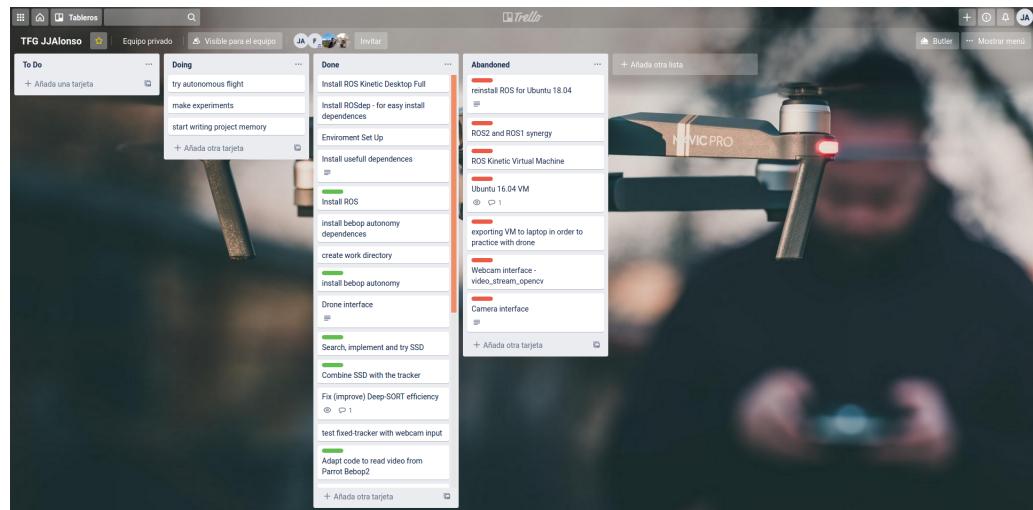


Figura 13: Captura del tablero de Trello

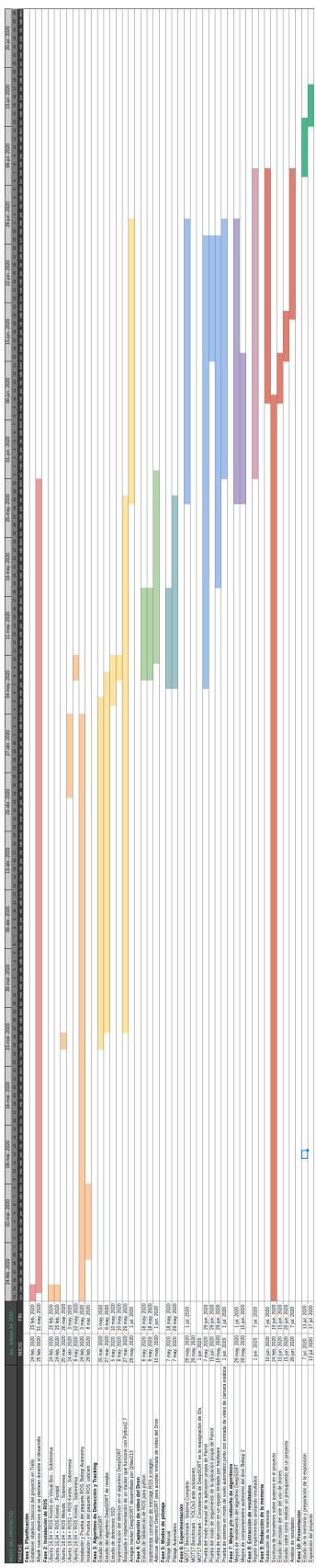


Figura 14: Diagrama de Gantt de Planificación

Ver con más detalle en carpeta MEGA [28]

El diagrama de Gantt representa el tiempo real dedicado a cada tarea.

En general, el desempeño de cada tarea ha sobrepasado la duración estimada al inicio del proyecto. En especial, la instalación de la base sobre la que empezar a trabajar, es decir, la instalación del sistema operativo Ubuntu junto con el marco de desarrollo (*framework*) ROS. Esta tardanza en tener operativa la base del proyecto, se debe a que necesitamos Python2.7 para hacer funcionar nuestro código y debido a la ausencia de soporte para este lenguaje, surgieron distintas incompatibilidades (comentaremos esto más adelante en el apartado de problemas 6.4).

Otra tarea que se extendió en el tiempo mucho más de lo esperado, fue hallar un algoritmo de seguimiento (*tracking*), y fue debido al mismo motivo que antes. Estas dos situaciones hicieron que el proyecto avanzase muy lentamente, y generase un retraso notable en el cumplimiento de hitos 2.3.

El resto de tareas, aunque también tomaron algo más de tiempo que el previsto, fue algo que manteníamos dentro de nuestros márgenes.

2.3. Hitos

Los objetivos específicos 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 Instalación de Ubuntu 16.04

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

2 Instalación ROS Kinetic

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

3 Instalación Bebop Autonomy

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

4 Selección del algoritmo de seguimiento (*tracking*) de personas

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

5 Creación de 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 Desarrollo 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 Desarrollo 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 Implementación 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 Creación de 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

2.4. Presupuesto

2.4.1. 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.

2.4.2. 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

2.4.3. 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€.

2.4.4. 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€

3. 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 esta sección aprovecharé para hablar de los problemas surgidos durante el proyecto ya que la gran mayoría de ellos se han dado durante la preparación de este escenario base sobre el que desarrollar el proyecto.

3.1. 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 a 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.

3.2. Robotic Operating System

Este framework libre, más conocido por su acrónimo ROS [3] [29], 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.

3.2.1. 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, sensorización o algún tipo de acción. 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(Topics):** Los *topics* son canales de comunicación 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. 16

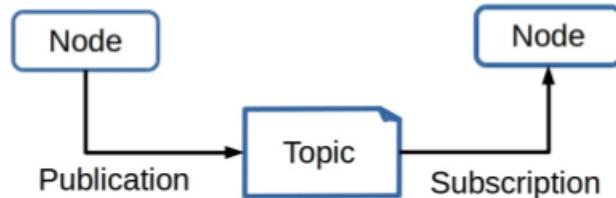


Figura 16: Esquema de flujo de información mediante Tópicos en ROS

Diagrama extraído del TFG de Valero Lavid [30]

- **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.
- **Launchfiles:** Son archivos de ejecución propios de ROS. Permiten ejecutar simultáneamente distintos nodos del sistema, así como establecer parámetros en el Servidor de Parámetros. Permite ejecutar también, otros *launchfiles* así como reiniciar un nodo en caso de que haya finalizado su ejecución de manera inesperada. Son programados en XML.
- **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* 18 y *geometry_msgs/Twist.msg* 20 que presentan la siguiente estructura:

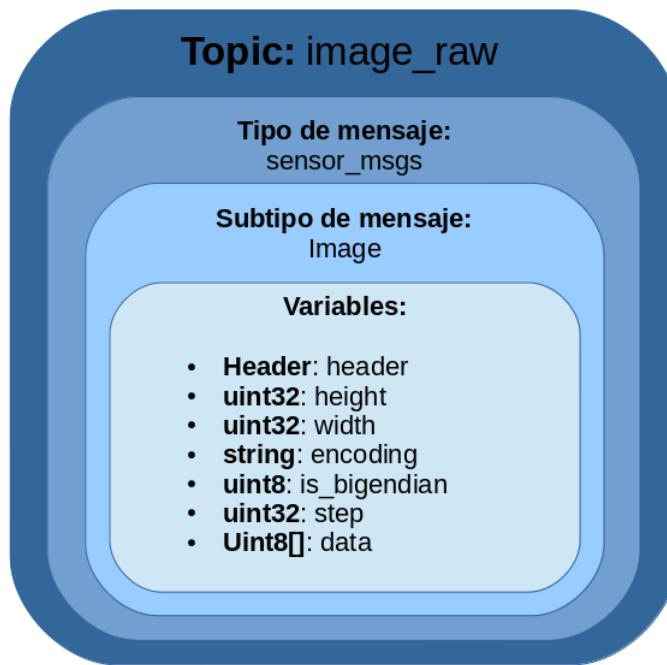


Figura 18: Estructura de mensaje Image

Información obtenida de la documentación de ROS [31]

Idea de diseño obtenida del TFG de Valero Lavid [30]

- a) **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.
- b) **height** y **width** hacen referencia a la resolución de la imagen (W x H píxeles)
- c) **encoding** es el formato en el que se representa internamente la imagen. Número de canales, su orden y significado de cada uno.
- d) **is_bigendian** es suficientemente autodescriptivo.
- e) **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.
- f) Por último **data** contiene la matriz que representa la imagen.



Figura 20: Estructura de mensaje Twist

Información obtenida de la documentación de ROS [32]

Idea de diseño obtenida del TFG de Valero Lavid [30]

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 22: Esquema de desplazamiento de Bebop 2 con velocidades positivas

Imagen extraída del TFG de Valero Lavid [30]

- **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.

3.3. 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 [33].

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_iv.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:

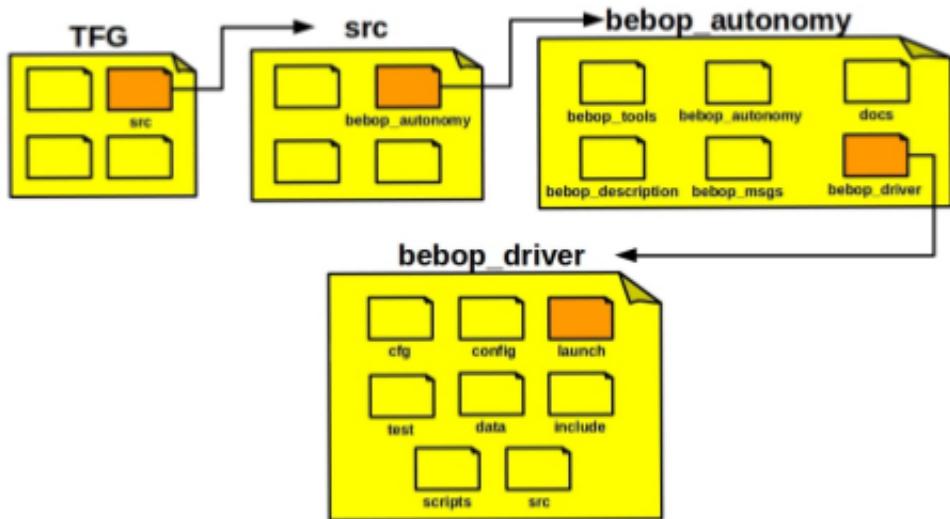


Figura 24: Organización de directorios de bebop-autonomy

Diagrama extraído del TFG de Valero Lavid [30]

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.

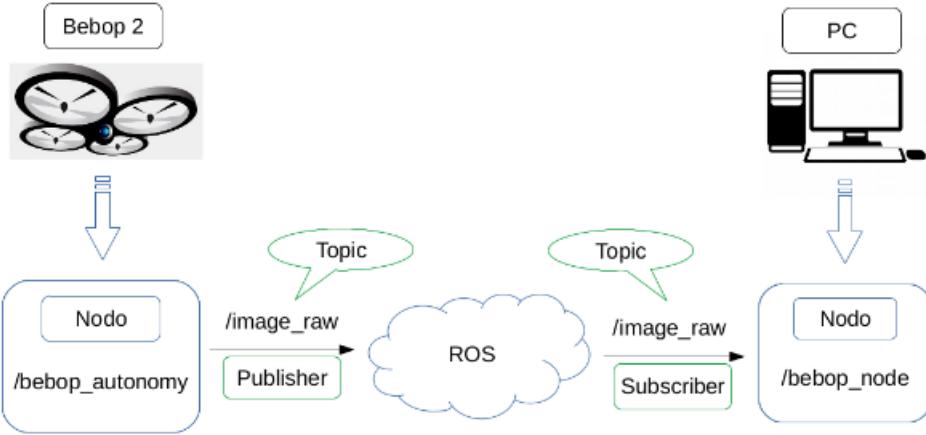


Figura 26: Interfaz de conexión Bebop2 - ROS - PC para la obtención de imagen del dron.

Diagrama extraído del TFG de Valero Lavid [30]

3.3.1. 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 desee.
- **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.

3.4. Bebop 2



Figura 28: Parrot Bebop 2

Imagen de la página oficial de Parrot [34]

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 [34].

3.4.1. 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 ya que el ancho de banda de conexión entre el dron y el sistema es limitado 7.7.5.

- **Conectividad:** Es un quadrocoptero 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 conectarnos 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 seguimiento (**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** (*First Person View* o Visión en Primera Persona en un modo de pilotaje con visión remota [35]) 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 planes 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

3.5. 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 trazados (*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 *13 fps* (*frames per second* o imágenes por segundo) durante la ejecución teniendo una media de *12 fps*. 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 [6] 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 %.

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.

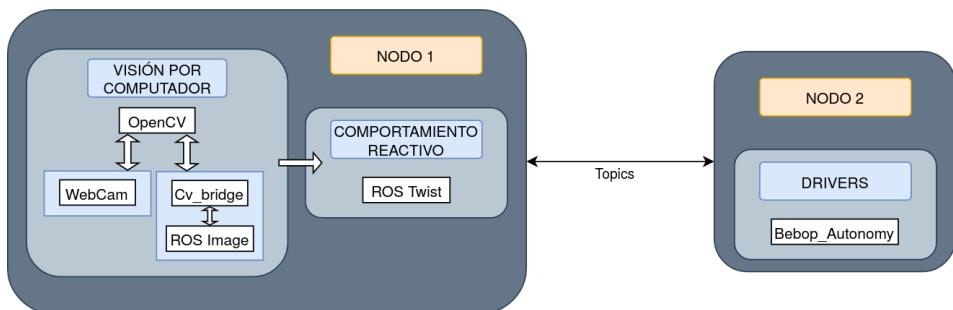


Figura 30: Esquema de nodos ROS en la aplicación

El *NODO 1* se corresponde con el PC y el *NODO 2* con el dron Bebop 2

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*. Está 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.

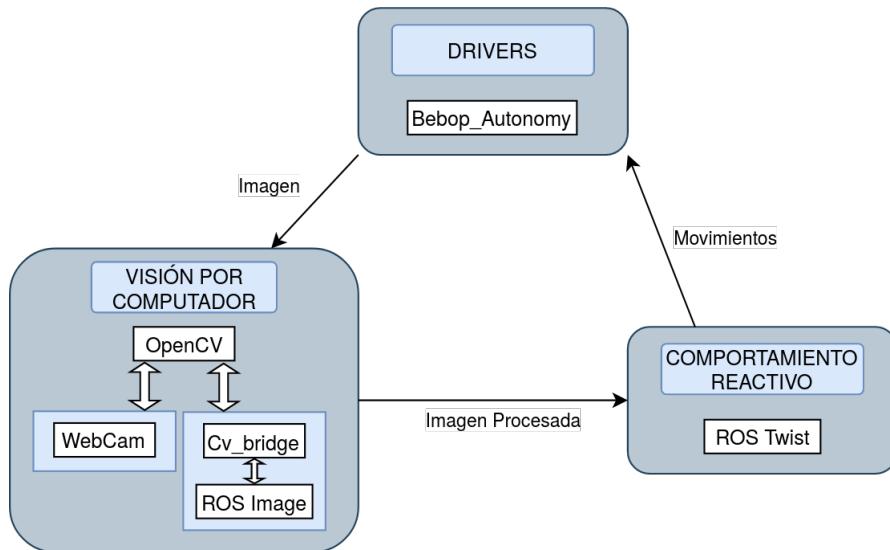


Figura 32: Esquema de comunicación entre los módulos de la aplicación

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 30 ya que es el que se ha desarrollado a lo largo del proyecto y es el encargado de detectar personas en escena y realizar el seguimiento activo de un objetivo indicado.

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, pero antes, vamos a hablar sobre los algoritmos implementados:

5.1.1. Detector - YOLOv3

Como algoritmo de detección se utiliza la red neuronal convolucional (CNN) YOLOv3, un red neuronal profunda en la que se concatenan capas con convoluciones que usan distintos tamaños de máscaras (filtros) con el objetivo de encontrar la combinación óptima de estos para realizar la detección de figuras (en este caso personas) y obtener su posición (x , y , ancho, alto) sobre la imagen inicial. Es un sistema de detección en tiempo real de última generación, que aporta gran precisión y rapidez a la hora de detectar y clasificar objetos. Se ha seleccionado este detector por ser el que mejor relación $\frac{\text{precision}}{\text{tiempo_ejecucion}}$ proporciona de entre los disponibles en la literatura. Aporta una precisión en la detección decente en un tiempo suficientemente pequeño como para poder ejecutar el algoritmo en tiempo real lo cual es indispensable en el proceso de navegación y/o pilotaje del dron.

En la figura 33 podemos ver la comparativa de YOLOv3 frente a otros detectores como SSD [36] o RetinaNet [37]. Se puede observar que el tiempo de ejecución es mucho menor en YOLO que en sus contrincantes mientras que la precisión es prácticamente igual e incluso superior en algunos casos.

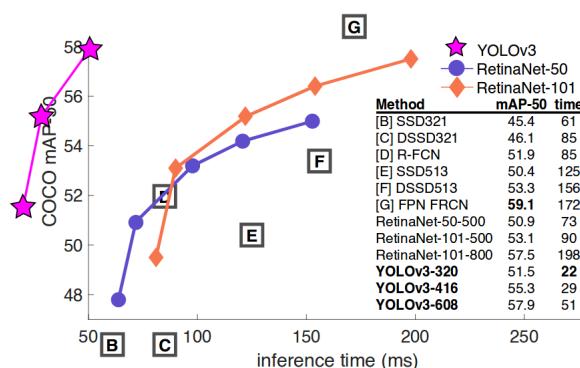


Figura 33: Rendimiento de YOLOv3 frente a otros algoritmos de detección

Paper: [22]

Fucionamiento

Como ya se ha mencionado, YOLOv3 se basa en una red neuronal convolucional, exáctamente en la *CNN Darknet-53* llamada así porque consta de 53 capas convolucionales cada una seguida de una capa de normalización (*Batch Normalization*) y otra de activación (*Leaky RELU*) (Ver figura 35). Las capas de normalización transforman todos los valores resultantes de la convolución al dominio [0,1] (a cada valor se le resta la media de los valores de los píxeles y se divide por la desviación estándar) para asegurar que todas las características de la red (píxeles de la imagen) contribuyen de la misma manera (proporcionalmente hablando) al resultado final. Mediante la normalización también se evita el *overfitting*, es decir, evitar que al entrenar la red esta se ajuste demasiado a los datos de entrenamiento y al introducir una nueva imagen obtengamos malos resultados. Por otro lado, la capa de activación o rectificador, decide si la salida de una determinada neurona es relevante para la ejecución de la red. La resultado de cada neurona se pasa por una función de activación como *ReLU*, *Sigmoide* o *Softmax* [38] que decide si el valor continúa por la red o se “neutraliza” (se elimina o se le da un valor nulo).

Type	Filters	Size	Output
Convolutional	32	3×3	256×256
Convolutional	64	$3 \times 3 / 2$	128×128
1x	Convolutional	32	1×1
1x	Convolutional	64	3×3
	Residual		128×128
	Convolutional	128	$3 \times 3 / 2$
			64×64
2x	Convolutional	64	1×1
2x	Convolutional	128	3×3
	Residual		64×64
	Convolutional	256	$3 \times 3 / 2$
			32×32
8x	Convolutional	128	1×1
8x	Convolutional	256	3×3
	Residual		32×32
	Convolutional	512	$3 \times 3 / 2$
			16×16
8x	Convolutional	256	1×1
8x	Convolutional	512	3×3
	Residual		16×16
	Convolutional	1024	$3 \times 3 / 2$
			8×8
4x	Convolutional	512	1×1
4x	Convolutional	1024	3×3
	Residual		8×8
	Avgpool		Global
	Connected		1000
	Softmax		

Darknet-53 model

Figura 35: Estructura de Darknet-53

Extraída del paper de YOLO: [22]

El algoritmo acepta cualquier resolución de imagen pero, en caso que queramos procesar varias imágenes a la vez (por *batches*), sí será necesario introducir todas las imágenes a la misma resolución para que la GPU pueda realizar los cálculos en paralelo. Como nosotros procesamos una secuencia de vídeo y pretendemos que sea a tiempo real, tenemos que procesar los fotogramas a la mayor velocidad posible y esto lo conseguimos realizando un procesamiento secuencial, llegando incluso a saltarnos algún fotograma de la secuencia (tratamiento asíncrono de la lectura de fotogramas). Si esperásemos a tener n fotogramas para formar un *batch* y analizarlo en paralelo perderíamos la cualidad de que el proceso sea en tiempo real y, por tanto, también reactividad. Aún así, aunque en nuestro caso podríamos usar distintas resoluciones de imagen cada vez al proceder todos los fotogramas de la misma fuente de vídeo, tendrán todos la misma resolución, de manera que si se da la situación en la que se necesite realizar un procesamiento en paralelo mediante *batches* se podría llevar a cabo.

La red reescalía la imagen tres veces por un factor llamado *stride* o *paso de la red*. Lo que hace es dividir la imagen entre el *stride* de manera que agrupa los píxeles de la imagen en grupos llamados *celdas*. En la implementación utilizada, este paso de la red es 32 para una iteración, 16 para la siguiente y 8 para la última, de manera que si recibe como entrada una imagen de resolución 416 x 416 píxeles, obtendríamos una imagen de salida de 13 x 13 *celdas* en la primera iteración, 26 x 26 *celdas* en la segunda, y 52 x 52 *celdas* en la tercera. Será en esta interpretación mediante *celdas* en la que se realizarán las detecciones en la imagen.

Cada subdivisión de la imagen (13x13, 26x26 y 52x52 *celdas*) se recorre con distintos tamaños de ventanas de búsqueda (*Anchor Boxes*) para tratar de identificar las distintas figuras que aparezcan en escena como vemos en la figura 37. Estas *ventanas de búsqueda* son recuadros predefinidos que suelen delimitar la forma de una determinada figura. Si la red está entrenada para detectar animales cuadrúpedos vistos desde el plano horizontal, sería inteligente contar con una *ventanas de búsqueda* rectangular más ancha que alta para lograr detectar estas figuras. Para las personas, por ejemplo, la *ventanas de búsqueda* será más alta que ancha para detectar a personas de pie. La implementación de YOLOv3 escogida utiliza un total de 9 combinaciones diferentes de *ventanas de búsqueda* para detectar las 80 clases que encontramos en el conjunto de datos COCO, que es el conjunto con el que se ha entrenado por defecto a YOLOv3.

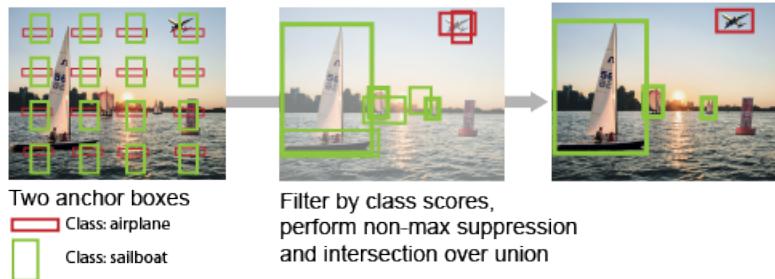


Figura 37: Recorrido de la imagen con dos tipos de *ventanas de búsqueda*

Extraída de [39]

Cada *ventana de búsqueda* se aplica a cada celda de la imagen (si una imagen es 416 x 416 y el paso de la red es 32, obtenemos 13 x 13 celdas) obteniendo resultados de detección para cada una. Para mostrarlo más claramente, si disponemos de 9 *ventanas de búsqueda* e introducimos una imagen 416 x 416 píxeles a color tendríamos el siguiente escenario:

Como entrada tendríamos la imagen original (416,416,3) (3 es el número de canales de color: RGB).

Como salida, tendríamos la imagen representada mediante celdas junto con la precisión de detección y las coordenadas de los *Bounding Boxes* calculados con cada *ventana de búsqueda* para cada celda de imagen, además de un vector *booleano* que indica la clase a la que pertenece la detección: (13,13,9,85). Imagen 13x13, 9 *ventanas de búsqueda*, 85= 5(confianza de la detección, coordenadas X e Y del *Bounding Box* y su ancho y su alto) + (80 clases de COCO). Podemos ver un ejemplo más visual en la figura 39.

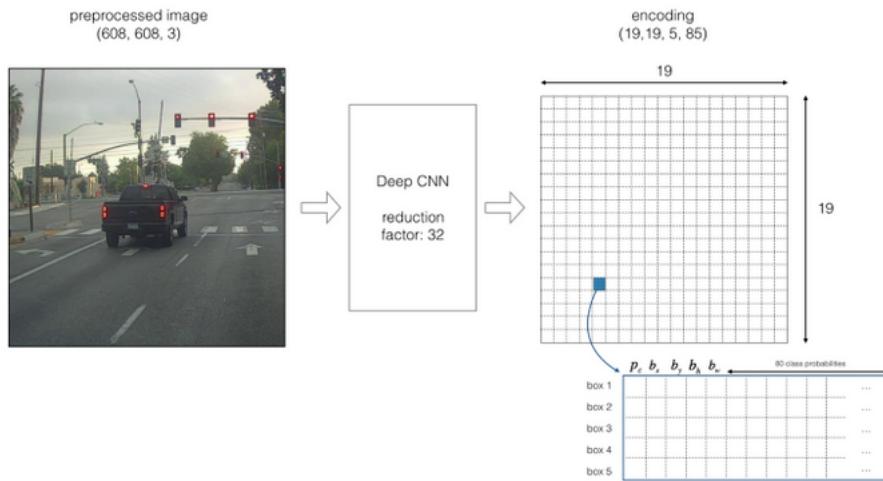


Figura 39: Ejemplo de codificación mediante celdas. Imagen original (608,608,3) codificada a (19,19,5,85)

Extraída de [39]

De todas las detecciones realizadas y, por tanto, de todos los *Bounding Boxes* obtenidos, debemos quedarnos con aquellos que realmente presenten una confianza de detección suficientemente grande. A este proceso se le llama supresión de no máximos. Descartamos todas aquellas detecciones que no superen un umbral de confianza definido arbitrariamente.

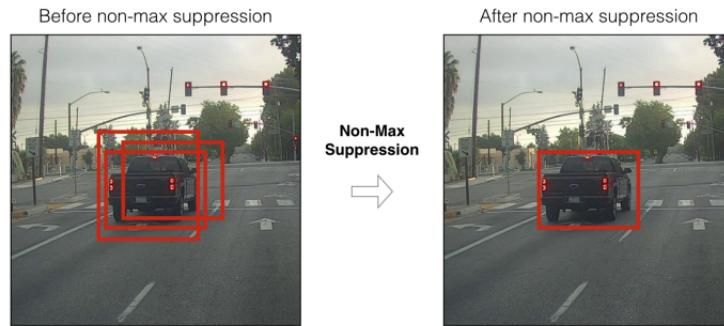


Figura 41: Supresión de NO máximos

Extraída de [39]

Son estos *Bounding Boxes* finales los que proporciona como salida el algoritmo: sus coordenadas, anchura, altura, confianza de detección y clase a la que pertenece la detección. Como en nuestro caso solo obtenemos detecciones de personas, son tan solo los cuatro primeros datos los que recibirá nuestro algoritmo de seguimiento, ya que no necesitamos indicar la clase de la detección, y nos aseguramos de que la confianza sea suficientemente alta ajustando un umbral alto para la misma.

5.1.2. Tracker - DeepSORT

Utilizamos DeepSORT [7] como algoritmo de tracking por pertenecer al estado del arte y ser capaz de proporcionar seguimiento de figuras en tiempo real con una de las mayores frecuencias de procesamiento de entre los algoritmos vistos en la literatura (ver figura 43), además de ser el que menor perdida de trazados (*tracks*) presenta de entre los competidores (métrica *ML*). Además en cuanto a precisión se refiere, se encuentra muy cerca de sus rivales y su mejora respecto a SORT (Simple Online and Real-time Tracker), introduciendo información sobre la apariencia de las detecciones, permite ser menos susceptible a perder trazados debido a occlusiones.

		MOTA ↑	MOTP ↑	MT ↑	ML ↓	ID ↓	FM ↓	FP ↓	FN ↓	Runtime ↑
KDNT [16]*	BATCH	68.2	79.4	41.0%	19.0%	933	1093	11479	45605	0.7 Hz
LMP_p [17]*	BATCH	71.0	80.2	46.9%	21.9%	434	587	7880	44564	0.5 Hz
MCMOT_HDM [18]	BATCH	62.4	78.3	31.5%	24.2%	1394	1318	9855	57257	35 Hz
NOMTwSDP16 [19]	BATCH	62.2	79.6	32.5%	31.1%	406	642	5119	63352	3 Hz
EAMTT [20]	ONLINE	52.5	78.8	19.0%	34.9%	910	1321	4407	81223	12 Hz
POI [16]*	ONLINE	66.1	79.5	34.0%	20.8%	805	3093	5061	55914	10 Hz
SORT [12]*	ONLINE	59.8	79.6	25.4%	22.7%	1423	1835	8698	63245	60 Hz
Deep SORT (Ours)*	ONLINE	61.4	79.1	32.8%	18.2%	781	2008	12852	56668	40 Hz

Figura 43: Comparación de algoritmos de tracking en estado del arte

Extraída del paper de DeepSORT [40]

El algoritmo de seguimiento utiliza varias técnicas para lograr asignar los identificadores a las detecciones y hacerlos perdurar mientras persistan en escena. Estas son:

- **Filtros de Kalman:** En casi cualquier problema que involucre realizar predicciones en el tiempo, utilizar filtros de Kalman es la mejor manera de afrontarlos. Los Filtros de Kalman utilizan las detecciones actuales y las predicciones pasadas para calcular las futuras a la vez que tiene en cuenta la información acerca de los errores que se producen en el proceso.

Al realizar la detección de figuras debemos tener en cuenta que habrá fallos ya que no existe un algoritmo de detección 100 % eficaz. Además, para estas detecciones, se asume que las figuras se mueven según un modelo de velocidad constante para poder predecir su próxima posición en escena. Son en estas dos situaciones donde se encuentran los errores:

Se puede obtener ruido asociado a la velocidad de una figura ya que rara vez se mantendrá constante. A esto se le conoce como *Ruido de Proceso*.

También se puede obtener ruido durante la detección de figuras en sí misma debido a la falta de precisión del detector. Este ruido será *Ruido en la Medición*.

- **Problema de asignación** Una vez se han calculado las nuevas predicciones, surge el problema de asignarlas a las antiguas, es decir, correlacionar dos detecciones bajo un mismo trazo (*track*). Para realizar esta correlación se utiliza el algoritmo Húngaro [41] al que se le proporciona información de movimiento de la figura y su apariencia.
- **Distancia de Mahalanobis. Correlación de movimiento.** Para incorporar la información de movimiento a la hora de asignar una nueva predicción a un *track* utiliza la distancia (cuadrada) de Mahalanobis entre las predicciones de los filtros de Kalman y las nuevas detecciones.

$$d^{(1)}(i, j) = (d_j - y_i)^T S_i^{-1} (d_j - y_i)$$

y_i es el vector resultante de aplicar los filtros de Kalman a las detecciones anteriores y hace referencia al i-ésimo *track*. S_i es la matriz de covarianza de y_i y d_j es un vector que denota la detección j-ésima.

- **Distancia de coseno entre espacio de apariencias. Actualización a SORT.** Este es el cambio más interesante realizado a SORT y por lo que este algoritmo recibe el “prefijo” *Deep*. Se calcula un descriptor para cada track y para cada nueva detección de manera que podamos enfrentarlos entre sí calculando su distancia de coseno. Dicho descriptor se obtiene de la última capa de la CNN que supone el algoritmo detector.

$$c_{i,j} = \lambda d^{(1)}(i, j) + (1 - \lambda)d^{(2)}(i, j)$$

Finalmente obtenemos esta función de distancia donde $d^{(1)}$ es la distancia de Mahalanobis y $d^{(2)}$ es la distancia del coseno, pudiendo dar más peso a una o a otra cambiando el valor del hiperparámetro λ . La importancia de la incorporación del descriptor de apariencias es tan alta que los autores del algoritmo afirman en su paper [40] que son capaces de alcanzar resultados propios del estado del arte dando un valor $\lambda = 0$ siempre y cuando tengamos un alto nivel de *fps*. Si tenemos un bajo ratio de fotogramas (5*fps*), la distancia de Mahalanobis nos será útil para discernir entre figuras por su distancia espacial en escena.

5.1.3. 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 quisiéramos 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.

5.1.4. 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.

5.1.5. 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 reali-

zamos 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.

5.1.6. 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, el total de trazos (*tracks*) calculados, los *fps* de salida obtenidos (se llaman *Real fps* porque es una medida más exacta que el dividir el total de frames entre el tiempo de ejecución), la desviación estándar en los tiempos de detección y seguimiento, el total de frames procesados, el tiempo total que ha tomado el procesamiento del vídeo, y el intervalo y periodo aplicados.

```
Ejecución: 7
Título: ../videos/MOT17/MOT17-12-SDP-raw
res: (426, 240)
Max time: 0.11066198349
Min time: 0.00157904624939
Mean time: 0.0548581803866
Max track: 736
Real FPS: 16.9344600149
Desviación Estandar en Detección: 0.00524717023431
Desviación Estandar en Tracking: 0.000636790949613
Frames Totales: 899
Tiempo total: 53.0870189667
Intervalo: 1
Periodo: 2
*7*
Ejecución: 8
Título: ../videos/MOT17/MOT17-12-SDP-raw
res: (426, 240)
Max time: 0.108312129974
Min time: 0.00146913528442
Mean time: 0.0393322092275
Max track: 489
Real FPS: 23.0309291618
Desviación Estandar en Detección: 0.00516994800831
Desviación Estandar en Tracking: 0.000575805498142
Frames Totales: 899
Tiempo total: 39.0344650745
Intervalo: 1
Periodo: 3
*8*
Ejecución: 9
Título: ../videos/MOT17/MOT17-12-SDP-raw
res: (426, 240)
Max time: 0.104532003403
Min time: 0.00140905380249
Mean time: 0.0317615759916
Max track: 837
Real FPS: 27.9869056456
Desviación Estandar en Detección: 0.00534294228627
Desviación Estandar en Tracking: 0.000648469599825
Frames Totales: 899
Tiempo total: 32.1221649647
Intervalo: 1
Periodo: 4
*9*
```

Figura 45: Ejemplo de archivo txt generado

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 46: 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 46 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 47: 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. Problemas durante el desarrollo

6.1. Restricciones Hardware

- 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 seguimiento (*tracking*) sean adecuados, debido a la apertura focal de la cámara del dron.
- 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.

6.2. 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.

6.3. Restricciones Sanitarias

Debido a la particular situación sanitaria que estamos viviendo en este año 2020 a causa del COVID-19, me he visto limitado en ciertos aspectos a la hora de desarrollar este trabajo de fin de grado 6.4.

6.4. COVID-19

Antes de comenzar con problemas puramente técnicos he de mencionar los ocasionados por la crisis sanitaria del COVID-19. Debido a la emergencia sanitaria, y las restricciones sociales derivadas de ella, el proyecto se ha visto afectado en los siguientes puntos:

- No he podido recurrir a las instalaciones del CITIC para realizar los vuelos del dron en la jaula de la que disponen allí específicamente orientada a ello.
- Tampoco he considerado la opción de buscar un equipo portátil con suficiente potencia para ejecutar el proyecto, ya que no hubiese podido salir a realizar vuelos al aire libre debido al confinamiento.
- No han sido posibles las reuniones presenciales con mis tutores para resolver dudas de una manera más clara y sencilla.
- Por todo lo anterior el proceso de experimentación ha sido mucho más limitado.

Durante esta situación nos hemos centrado mucho más en la investigación de algoritmos más eficientes, en cómo podíamos ajustar más los algoritmos de detección y seguimiento a nuestro caso particular, en el desarrollo de un comportamiento autónomo más eficiente... Aunque esto último, se hizo de manera “intuitiva” ya que no pudimos probar los resultados hasta prácticamente un mes antes de la entrega.

6.5. Problemas Técnicos

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.

6.5.1. Virtualización

En un primer momento si intentó desarrollar el proyecto en una máquina virtual con *VirtualBoX* [42] 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* [43] 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.

6.5.2. Equipo de Trabajo

Como se ha sugerido en 3.5 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 seguimiento (*tracking*). Ejecutando el proyecto final en este ordenador hemos llegado a obtener picos de 0.8 *fps* pero la media ronda los 0.7 *fps*. 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 48: 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 cálculo 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.

6.5.3. 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.

Se intentó crear una red de nodos que funcionasen como repetidores de la señal Wi-Fi del dron Bebop 2 configurando una serie de *routers* en modo puente (*bridge*). Esta idea no tuvo éxito y, debido al corto plazo de tiempo del que disponíamos, decidimos no seguir ahondando en esta dirección. Finalmente decidimos conformarnos con la distancia que nos proporciona el Wi-Fi a 2.4Gh.

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 por el COVID-19. El desarrollo de las pruebas debería haberse realizado en el CITIC [44], que cuenta con una jaula específica para el vuelo de drones.

6.5.4. 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 seguimiento (*tracking*) desarrollados para Python2.7 tienen dependencias de paquetes que ya no es posible descargar. Finalmente encontramos un algoritmo de detección más seguimiento (*tracking*) casi totalmente funcional con python2: una implementación del algoritmo Deep SORT [7] por parte del usuario *Qidian213* en GitHub [45]. El algoritmo nos da un error por la falta de una biblioteca pero esto está resuelto en uno de los *issues* del repositorio [46].

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

6.5.5. 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 brindara 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.

6.5.6. Demo práctica del repositorio de *Qidian213* poco eficiente

La demo proporcionada en el repositorio tiene tres principales problemas:

- Requiere de una biblioteca de lectura asíncrona que no se encuentra en el repositorio. Está solucionado en uno de los *issues* [46].
- Ejecuta dos veces el algoritmo de detección por frame. Esto es innecesario y muy poco eficiente.

```
71      boxes = yolo.detect_image(image)[0]
72      confidence = yolo.detect_image(image)[1]
```

Figura 49: Detección ineficiente en demo.py

Imagen obtenida del repositorio de Qidian213 [45]

- Realiza detección de la escena en cada fotograma, cuando la idea en un algoritmo de seguimiento (*tracking*), es realizar la detección cada X fotogramas de manera que en las imágenes intermedias sea el tracker el que estime la futura posición de la figura. Esto también conlleva unos resultados poco eficientes.

En nuestra implementación en particular, se ha escogido un periodo de detección de 3 frames. Un periodo inferior de 2 frames no proporciona una mejora notable y un periodo de 1 frame es igual que la implementación de *Qidian213*. Con periodos de detección mayores, empezamos a notar la pérdida de consistencia en los trazados (*tracks*).

El periodo de 3 frames, consigue una mejora de rendimiento más que notable que se traduce en una mayor fluidez del vídeo procesado y mayor reactividad en el comportamiento del dron.

Retomaremos este tema en el apartado de experimentación y pruebas.

Problema	Plan de contingencia. Primera Idea	Solución Final
Retrasos + Ba-jo rendimiento del algoritmo de se-guimiento	Dedicación al desarrollo conceptual del proyecto e investigación de algo ritmos	Hallazgo del algoritmo de seguimiento (<i>tracker</i>) funcional en python2.7. Arreglar su código y adaptarlo a nuestro proyecto.
Requiere versión de sistema opera-tivo antigua	Virtualización	Instalación limpia de Ubuntu16.04
Recursos hardwa-re limitados en equipo portátil	Cómputo pesado en la nube	Traslado del desarrollo y ejecución del proyec-to al ordenador de sobremesa. Supuso adqui-rir nuevos módulos de RAM y una tarjeta Wi-Fi
Límite de distan-cia de conexión y, por tanto, límite de la zona de vue-lo	Diseño de una red de repetidores de la señal Wi-Fi del dron Bebop 2	Seguir trabajando con la cobertura dada por la señal Wi-Fi original del dron
Python2.7 obsole-to	Búsqueda de algoritmos que funcionen con Pyt-hon3. Instalar versiones de ROS que trabajen con Python3	Hallazgo de algortimo funcional con Pyt-hon2.7. Instalación de ROS Kinetic
Driver del dron no funciona en versiones de ROS distintas a Kinetic	Instalación conjunta de ROS2 y ROS Kinetic en Ubuntu 18.04 o 20.04	Utilizar ROS Kinetic

Cuadro 4: Tabla sintetizada de problemas, planes de contingencia y solucio-nes finales

7. Experimentación

Métricas

Las comparaciones se realizaran enfrentando los resultados obtenidos en:

- Número total de fotogramas procesados
- Tiempo total de procesamiento (*segundos*)
- Tiempo medio de procesamiento por fotograma (*segundos*)
- FPS (*fotogramas por segundo* procesados. La inversa del dato anterior nos da una aproximación de este)
- Número total de *tracks* detectados: total de identificadores diferentes que asigna el algoritmo de seguimiento a lo largo de la secuencia de vídeo.

Hacemos especial hincapié en las dos últimas ya que los *fps* son, en definitiva, una composición de las anteriores y el número de tracks totales detectados es totalmente independiente de los demás.

Procedimiento

Durante las pruebas alteraremos los valores del intervalo de lectura de fotogramas y del periodo de realización de detección en los fotogramas. Por ello, en primer lugar, pondremos a prueba el funcionamiento del detector YOLOv3 [6] alterando la resolución de la imagen y manteniendo un valor de intervalo y periodo igual a 1, ya que son estos valores los que nos permiten obtener los mejores resultados (leemos todos los fotogramas de la secuencia de vídeo y realizamos detección de figuras en todos ellos. Esta es la configuración que más precisión aporta en la detección). La prueba la realizaremos sobre el benchmark MOT17_12 y con ella pretendemos decidirnos por una resolución con la que trabajar en el resto de experimentos.

En segundo lugar, compararemos el rendimiento del algoritmo haciendo pruebas sobre el benchmark MOT17_05 con distintas configuraciones de valores para los parámetros del sistema. Esto lo haremos para tratar de encontrar la configuración ideal con la que poner a prueba la consistencia del tracker. Los parámetros del sistema son:

- Periodo de detección (cada cuántos fotogramas realizamos detección de personas en la escena).
- Intervalo de procesamiento de fotogramas (No procesar un determinado número de fotogramas de forma periódica. Un intervalo de 1 fotograma implica una lectura completamente secuencial. Un intervalo de 3 fotogramas implica leer solo los fotogramas que ocupan una posición múltiplo de 3 en la secuencia de vídeo: 0, 3, 6, 9, 12, ...).

Después vamos a poner en cuestión la eficacia del algoritmo de seguimiento (*tracking*) ejecutando la aplicación con los parámetros que consideremos más adecuados según lo que hayamos concluido en el apartado anterior. Observaremos en las imágenes de salida de la aplicación el dibujado de los *Bounding Boxes* y sus correspondientes IDs para valorar si se trata de un buen resultado o no.

Por último realizaremos la prueba de vuelo del dron analizando también el procesamiento del vídeo obtenido.

Resultados

Todos los resultados se encuentran disponibles en la carpeta de MEGA [28]. Encontraremos resultados antes y después de la corrección del código de *Qidian213* 6.5.6.

7.1. Pruebas del Detector

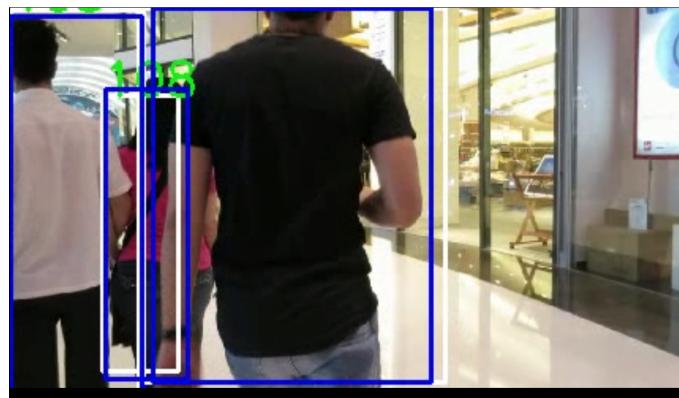


Figura 51: MOT17_12 imagen:0 426x240

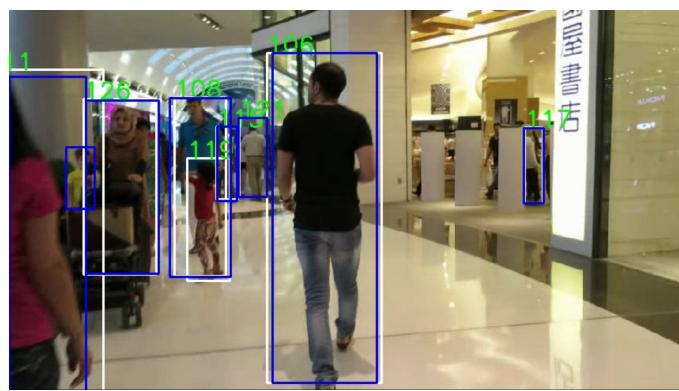


Figura 52: MOT17_12 imagen:1 854x480

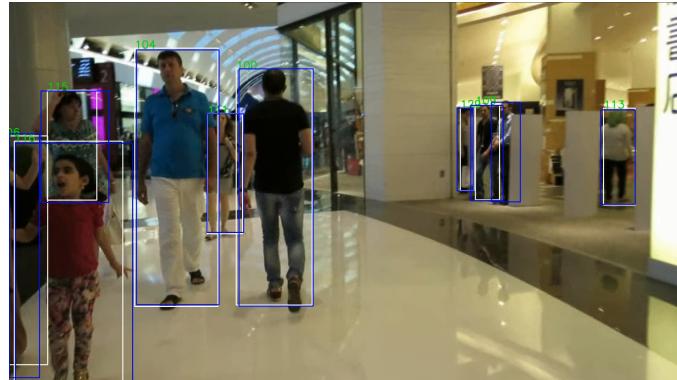


Figura 53: MOT17_12 imagen:2 1920x1080

7.1.1. Rendimiento ante distintas resoluciones

Durante la ejecución de la aplicación sobre este benchmark a distintas resoluciones nos hemos percatado de cómo influye el tamaño de la imagen que va a ser procesada en la velocidad de procesamiento. Cuanto más pequeña es la imagen (menor resolución), menos tiempo toma su procesamiento y, por tanto, mayor es el número de *fps* que obtenemos. Esto tiene sentido, si recordamos que el *modus operandi* del detector es recorrer las zonas de interés de la imagen (celdas de interés) para buscar en ellas figuras a clasificar. Cuanto menor sea la imagen, más pequeñas serán estas áreas y menos tardará el proceso de detección.

Como vemos en el cuadro 5 a menor resolución obtenemos mejores resultados de *fps*. Se pueden obtener conclusiones más rápidamente si observamos estos datos representados gráficamente como ocurre en la figura 54.

Resolución	FPS
426x240	9.4264
854x480	8.3094
1920x1080	5.6543

Cuadro 5: FPS frente a Resolución

Parámetros: periodo=1; intervalo=1

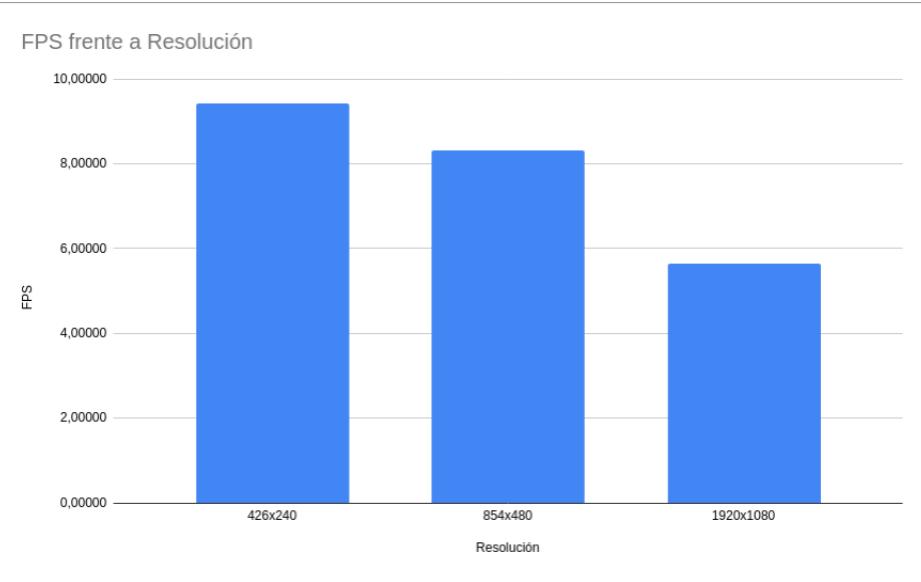


Figura 54: *fps* frente a la resolución de la imagen

Parámetros: periodo=1; intervalo=1

Podemos ver en la figura 54 que los *fps* mejoran (aumentan) a medida que disminuimos la resolución pero no de una manera proporcional, es decir, si disminuimos a la mitad la resolución no obtenemos el doble de *fps*. Es más, ni si quiera se mantiene el ratio de aumento. Los *fps* aumentan, aproximadamente, 1.5 veces al disminuir la resolución de 1080p a 480p que, más o menos, es la mitad la resolución y, sin embargo, solo aumentan 1.13 veces al volver a dividir por la mitad el ratio de aspecto de la imagen (de 480p a 240p). Esto nos indica que a parte de la resolución hay otro u otros factores que afectan a la cantidad de *fps* que podemos obtener; como veremos más adelante, estos son: el intervalo de lectura y el periodo de detección.

7.1.2. Oclusión

A la hora de medir la eficacia de la detección de personas en situaciones con oclusión entre figuras, solamente interviene en el resultado la resolución de la imagen ya que, mediante los otros dos parámetros (intervalo y periodo), lo único que puede variar es que se realice o no la detección de personas en ese fotograma concreto. Nosotros queremos evaluar su eficacia por lo que queremos que sí o sí se realice la detección en el fotograma. Para ello ajustamos un periodo de detección igual a 1 (realizamos detección en todos los fotogramas), y un intervalo de lectura también igual a 1 (realizamos una lectura completamente secuencial del vídeo, procesamos todos sus fotogramas).

El algoritmo de detección se desenvuelve bastante bien ante la oclusión de figuras. Al probar su eficacia con distintas resoluciones (muy dispares

entre ellas), en todas se han obtenido buenos resultados (imágenes 51 52 53).

En la primera imagen se ve cómo detecta a la mujer marcada con el identificador 108 que se encuentra entre dos personas que tapan parte de su cuerpo.

La segunda fotografía muestra la detección del hombre etiquetado con 108 a pesar de que una niña tapa la mitad inferior de su cuerpo. También se contempla la detección de un niño que todavía no ha sido “*traqueado*” (no se ha calculado su trazado o *track*) que se encuentra detrás del carro de bebés.

Por último, en el tercer fotograma, se aprecia una persona identificada con 111 tapada por el hombre 104, y a la mujer 115 tapada por la niña 116.

7.1.3. Conclusión sobre la Resolución

Ya que a menor resolución obtenemos mayor cantidad de fotogramas por segundo y no altera la eficacia ante situaciones de oclusión (o, al menos, no de una manera notable), usaremos 426x240 como resolución. A lo que sí debemos prestar atención es que los identificadores calculados por el algoritmo no son los mismos en los tres casos. Es más, a medida que disminuimos la resolución aumenta el número de IDs asignados a trazados como vemos en el cuadro 7.

Resolución	Nº Tracks
426x240	183
854x480	163
1920x1080	157

Cuadro 7: Número de Trazados frente a Resolución

Parámetros: periodo=1; intervalo=1

Este incremento en los trazos calculados puede deberse a dos distintas situaciones:

- Al reducir la resolución de la imagen, el algoritmo de detección considera como áreas de interés zonas proporcionalmente mayores que en resoluciones más altas. Al ser (proporcionalmente) mayor la superficie de la imagen a explorar en busca de personas, somos capaces de detectar un mayor número de figuras en escena.
- El hecho de redimensionar la imagen a una resolución menor (*down-sampling*), causa que la imagen se difumine y pierda nitidez (aplicamos un filtro de suavizado: Gaussian Blur [47]). Esto puede provocar que la extracción de características de las figuras no sea muy precisa y provoque la pérdida de trazados *tracks* que tengan que volverse a calcular. También es posible que debido a la falta de información de la imagen (falta de detalle) el algoritmo confunda otros tipos de figuras con personas.

7.2. Eficiencia del tracker DeepSORT en MOT Challenge

MOT Challenge o *Multiple Object Tracking Benchmark* [48] 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. Como hemos mencionado en el apartado anterior 7.1, vamos a utilizar una resolución de 426x240 para realizar las pruebas.

7.2.1. MOT17_12

MOT12 es una secuencia de vídeo de 30 segundos de duración a 30 fps.

Al analizar las distintas pruebas realizadas, demostraremos que la fluidez de la ejecución depende mucho de los parámetros nombrados durante la sección (intervalo y periodo) y, también, del número de detecciones realizadas, que se verán reflejadas en el número de trazados (*tracks*) obtenidos. Esta última característica influye en el rendimiento debido a los costes computacionales que conlleva la detección de figuras. Cuantas más detecciones se realicen en un mismo fotograma, más cálculos se han de hacer y más tiempo se tarda en procesar el fotograma.

El intervalo de lectura del vídeo afecta al rendimiento ya que aumenta o disminuye el número de fotogramas a procesar. De un vídeo de 60 fotogramas, procesaríamos todos ellos si aplicamos un intervalo de 1 fotograma. Pero si aplicamos un intervalo de 3 fotogramas, el número de las imágenes procesadas disminuye a 20, lo que se traduce en un incremento de la reactividad del sistema ya que procesamos el mismo fragmento de vídeo en menos tiempo.

El periodo de detección nos indica cada cuántos fotogramas recibidos, realizamos detección de personas en la imagen. Lo que conseguimos es reducir el número de veces que analizamos los fotogramas en busca de figuras, es decir, reducimos el coste de cómputo del detector tantas veces como indiquemos en el intervalo. Si designamos un periodo de 2 fotogramas, estaremos realizando detección en uno de cada dos fotogramas y, por lo tanto, estamos reduciendo el tiempo de procesamiento del detector a la mitad. El tiempo de procesamiento de reasignación de los trazados no varía, por eso con un periodo de 2 fotogramas no conseguimos procesar el vídeo por completo en exactamente la mitad de tiempo que con un periodo de 1 fotograma.

Al valorar las pruebas nos centraremos en los *fps* obtenidos 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 la reactividad de nuestro dron sea la mayor posible.

7.3. Baseline o Ejecución de referencia

Esta es la ejecución que utilizaremos como *baseline*, es decir, como valores de referencia para determinar si ejecuciones futuras son más o menos precisas. Esto lo hacemos porque su configuración de parámetros es la que ofrece una mayor precisión en la ejecución. Con intervalo igual a uno nos aseguramos de leer todos los fotogramas del vídeo y con un periodo de detección de un fotograma realizamos la detección de personas en cada imagen por lo que analizamos el vídeo al completo.

Si las siguientes ejecuciones se alejan de los resultados de esta que consideramos *baseline*, diremos que pierden precisión y/o rendimiento.

Prueba 1

- **Mean time:** 0.100563321681
- **Max track:** 183
- **FPS:** 9.42643260313
- **fotogramas Totales:** 899
- **Tiempo total:** 95.3701190948
- **Intervalo:** 1
- **Periodo:** 1

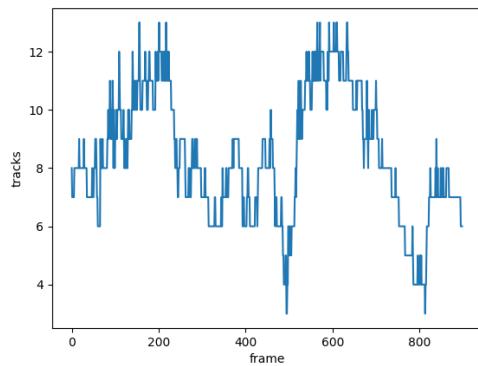


Figura 56: Número de tracks detectados por fotograma

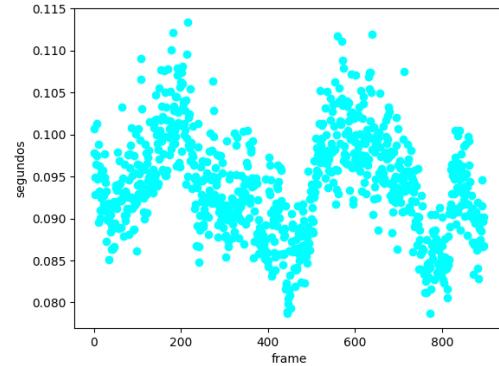


Figura 57: Tiempo de procesamiento requerido por fotograma

La relación entre ambas imágenes (56 y 57) es innegable. En la primera se muestra el número de identificadores (trazados) diferentes que se encuentran activos en un determinado fotograma. La segunda ilustra el tiempo de procesamiento que ha tomado cada fotograma en específico.

Como hablábamos al principio de la sección 7.2.1, la relación entre número de detecciones por fotograma y el tiempo de procesamiento de dicho fotograma es muy estrecha. Vemos como en ambas gráficas los valores toman la misma forma sobre los ejes. Todos los valles y crestas que nos encontramos en la gráfica de número de *tracks* por fotograma, se reflejan a la perfección en la gráfica que muestra los tiempos de procesamiento.

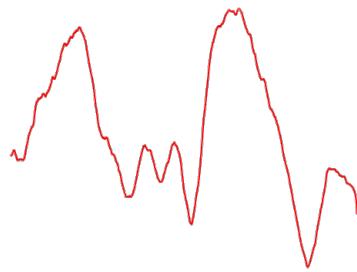


Figura 58: Evolución suavizada de trazados en escena

La gráfica mostrada en la figura 59 muestra un suavizado de la gráfica 56. Esto lo usaremos para realizar las comparaciones entre esta y las próximas ejecuciones ya que nos muestra una forma más genérica con la que comparar la evolución de los trazados y los tiempos de procesamiento.

Para ver la similitud de las figuras 56 y 57 podemos superponer la gráfica suavizada a la gráfica de tiempos de procesamiento para ver que siguen el mismo patrón.

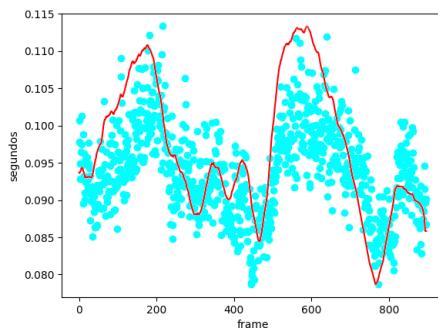


Figura 59: Comparación de la evolución de trazados y el tiempo de procesamiento por fotograma

Prueba 2:

- **Mean time:** 0.0548581803866
- **Max track:** 736
- **FPS:** 16.9344600149
- **fotogramas Totales:** 899
- **Tiempo total:** 53.0870189667
- **Intervalo:** 1
- **Periodo:** 2

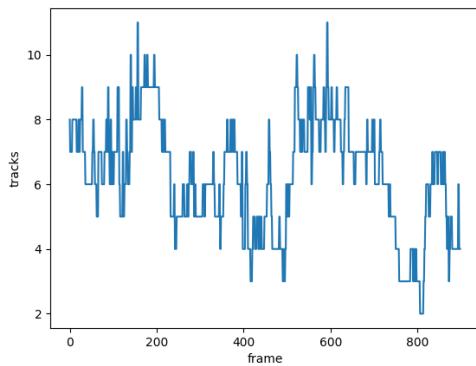


Figura 60: Número de tracks detectados por fotograma

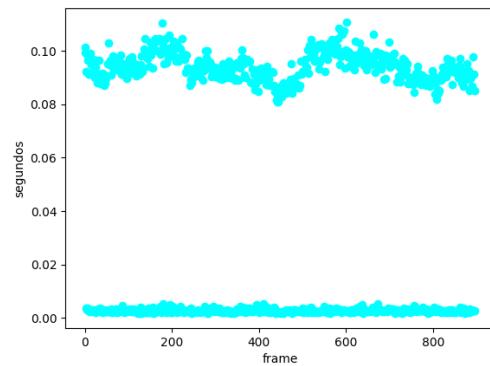


Figura 61: Tiempo de procesamiento requerido por fotograma

Vamos a empezar hablando sobre el número de tracks detectados en esta prueba. Por una parte la figura 60 nos indica que el pico de trazos (*tracks*) por fotograma ha disminuido, sin embargo, el recuento total de *tracks* calculados ha ascendido hasta cuatro veces el número de *tracks* de la primera ejecución. Esto es debido al periodo de detección cada dos fotogramas. La inconsistencia de las detecciones a lo largo de la secuencia de vídeo propician cálculos fallidos de tracks, lo que hace que el número de estos aumente.

La otra diferencia notable la encontramos en la figura 61. Ahora vemos que los puntos se dividen en dos grupos: un grupo cuyo tiempo de procesamiento es muy cercano a 0.0 segundos, y otro formado por fotogramas con un tiempo de cómputo mayor. Si nos fijamos bien, la distribución de los puntos de este último grupo se vuelve a asemejar a la gráfica de número de *tracks* por fotograma (Figura: 60). Esto es porque son los puntos de este grupo los correspondientes a los fotogramas en los que se ha realizado detección de personas, mientras que, en los otros, tan sólo hemos dejado predecir

al algoritmo de seguimiento (*tracker*) las futuras posiciones de los trazados. Con esto vemos claramente que las predicciones requieren un cómputo mucho menor que las detecciones y que el tiempo de cómputo de estas últimas depende mucho del número de personas en escena.

Debido al periodo de detección cada dos fotogramas, hemos conseguido, prácticamente, 9 *fps* más que en la Prueba 1, pero hemos cuadruplicado el número de tracks calculados. A lo largo de las pruebas queremos encontrar una combinación de parámetros que nos aporte un número razonable *fps* a la vez que evitamos que el número de *tracks* calculados ascienda demasiado.

Si comparamos estas nuevas gráficas con la ejecución de referencia obtenemos lo siguiente.

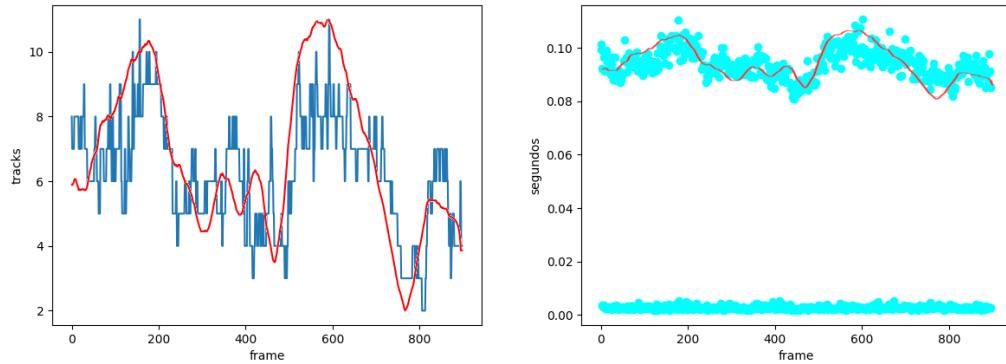


Figura 62: Comparativa de evolución de trazados

Figura 63: Tiempo de procesamiento junto con evolución de tracks

Vemos en la figura 62, como decíamos, que aunque ha cambiado el número de trazados activos por fotogramas, la evolución de la gráfica es prácticamente igual, y esto se puede apreciar también en la forma que toman los fotogramas en los que se realiza detección de personas en la figura 63.

Resto de pruebas con Intervalo 1

Dado que el resto de gráficas son muy parecidas a las de la Prueba 2 y lo único a destacar en ellas es que los puntos del grupo de fotogramas en los que se aplica detección cada vez son más distantes entre sí a medida que aumentamos el periodo de detección, vamos a realizar el resto de pruebas sin mostrarlas (se pueden consultar en la carpeta de MEGA [28]), obtendremos los resultados y los compararemos entre sí para decantarnos por una combinación de parámetros “ganadora”.

Al tomar como referentes los datos de la primera prueba por ser la más precisa, se considerará el aumento de *fps* como una medida de mejora y, el aumento del número de trazados (*tracks*) calculados, como una medida de error. También se indicará cuánto supone el error respecto a la mejora obtenida para valorar más fácilmente que opción merece más la pena:

Periodo	fps	Nº Tracks	Mejora	Error	Error / Mejora
1	9.4264	183	1	1	1
2	16.9344	736	1.8	4.02	2.23
3	23.0309	489	2.44	2.67	1.095
4	27.9869	837	2.97	4.57	1.54
5	20.7414	730	2.2	3.99	1.81

Cuadro 9: Resultados de ejecución en MOT17_12

Constantes: intervalo=1, resolución:426x240

El periodo es el único valor de la tabla que es escogido por nosotros. Para cada configuración de periodo obtenemos un resultado de *fps* y número de trazados (*tracks*). Los *fps* es el número de fotogramas por segundo que es capaz de procesar el algoritmo y el número de trazados es la cantidad de identificadores diferentes asignados a las detecciones a lo largo del vídeo. La columna de la mejora indica cuántas veces es mayor el número de *fps* obtenidos respecto a la ejecución de referencia. La columna de error indica cuántas veces es mayor el número de trazados diferentes calculados respecto a la ejecución de referencia. La columna de error respecto a la mejora es una medida orientativa para ayudarnos a decidir qué configuración de periodo es la más adecuada. Es orientativa ya que debemos prestar atención a los valores obtenidos y no sólo a su relación: no merece la pena tener cientos de fotogramas por segundo si tenemos miles de identificadores diferentes. Hay un momento en el que la mejora no es tan importante como para justificar cierta cantidad de error.

Tras analizar los resultados obtenidos, vemos que una muy buena opción es la ejecución de la aplicación con un periodo de detección de 3 fotogramas, ya que nos aporta 2.44 veces más fotogramas que con un periodo de 1 fotograma y, aunque el error es 2.67 veces mayor que el error de referencia y esto, a priori, nos podría parecer mucho, comparado con el error obtenido en el resto de configuraciones no lo es tanto. Además, la proporción del error respecto a la mejora es tan sólo 1.1, lo que implica que obtenemos casi la misma cantidad de mejora que de error.

7.4. Comparación de Intervalos Superiores a 1

La lectura de imágenes del dron la haremos de manera asíncrona para asegurarnos de que siempre estamos analizando la imagen más reciente y poder así reaccionar correctamente a la situación. Nuestro dron en cuestión (Bebop 2) tiene una frecuencia de muestreo y envío de imágenes de 30 *fps* y, como hemos visto antes, con un periodo de detección cada 3 fotogramas y una lectura síncrona del vídeo conseguimos procesar 23 imágenes por segundo. Con esta pequeña diferencia de *fps*, estimamos que el máximo de fotogramas que perderemos al realizar la lectura asíncrona será 2 o 3. Por ello los siguientes experimentos muestran los resultados de ejecución con intervalos de lectura de 2 y 3 fotogramas.

7.4.1. Intervalo: 2 fotogramas

Prueba 1

- **Mean time:** 0.109478187455
- **Max track:** 160
- **FPS:** 8.56456529636
- **fotogramas Totales:** 449
- **Tiempo total:** 52.4253110886
- **Intervalo:** 2
- **Periodo:** 1

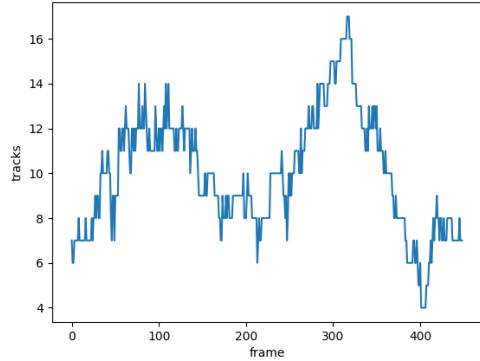


Figura 64: Número de tracks detectados por fotograma

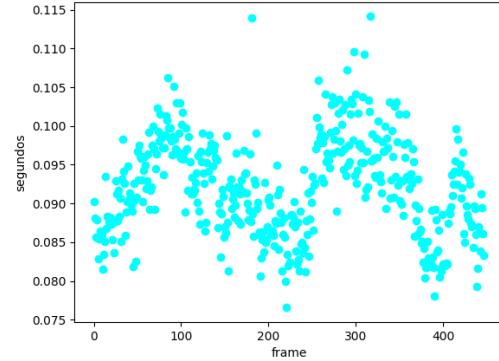


Figura 65: Tiempo de procesamiento requerido por fotograma

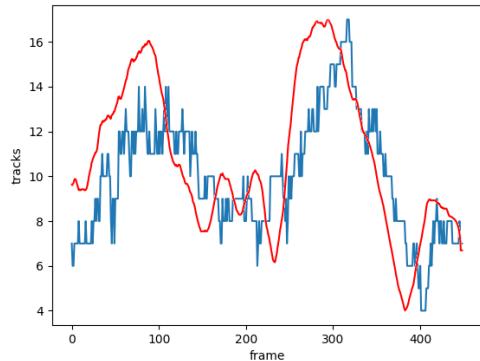


Figura 66: Comparativa de evolución de trazados

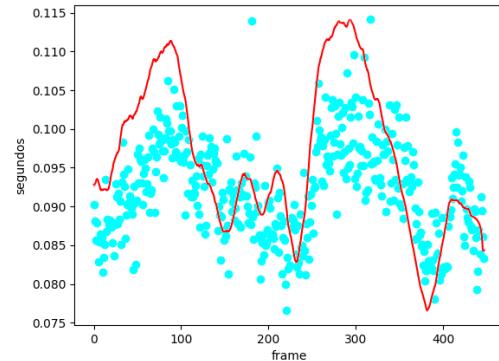


Figura 67: Tiempo de procesamiento junto con evolución de tracks

Lo primero a mencionar es la consecuencia más obvia que tiene aplicar un intervalo de lectura superior a 1 y es que el número de fotogramas procesados ha disminuido. Como el intervalo que hemos aplicado es igual a 2, hemos procesado exactamente la mitad de fotogramas (449 frente a los 899 totales). Aún así, vemos en las figuras 66 y 67 que la forma de las gráficas sigue siendo similar a las obtenidas en la ejecución de referencia. La distribución de los trazados (y por tanto detecciones) a lo largo del vídeo es muy similar aunque omitimos la lectura de la mitad de los fotogramas como es el caso, y el tiempo de procesamiento, como era de esperar, no se ve afectado simplemente, no se realiza detección en los fotogramas no leídos.

Otra característica interesante es el número de *fps* conseguidos. Conseguimos incluso menos imágenes por segundo que con un periodo e intervalo de 1 fotograma pero procesamos el vídeo al doble de velocidad ya que nos

saltamos el procesamiento de uno de cada dos fotogramas. Los valores del intervalo y el periodo están relacionados:

Intervalo	Periodo	FPS	Tiempo de procesamiento medio por fotograma	Tiempo Procesamiento
1	2	16.93	0.1094	53.0870
2	1	8.56	0.0548	52.4253

Cuadro 11: Comparativa Intervalo vs Periodo

Aunque en el primer caso obtenemos el doble de imágenes por segundo, vemos que el tiempo total de ejecución en ambos casos es muy parecido (53.087 segundos para el primer caso y 52.425 segundos para el segundo). Esto es porque aunque procesamos los fotogramas a la mitad de velocidad cuando tenemos un periodo de 1 fotograma, leemos el vídeo dos veces más rápido al tener un intervalo de dos. En definitiva, se aplica detección a la misma cantidad de fotogramas en ambos casos.

El máximo de trazados por imagen es el más alto hasta ahora, superando los 16 trazados en su pico más alto. Esto es debido a que algunos trazados se mantienen activos más tiempo del normal porque no han seguido una transición fluida de su objetivo al saltarnos fotogramas. A pesar de esto, que haya mayor número de trazados por fotograma es también debido a que utilizamos un periodo detección igual a 1, y esto nos permite llevar un seguimiento de muchas más personas en escena.



Figura 68: Número de tracks detectados. Intervalo:2 Periodo: 1

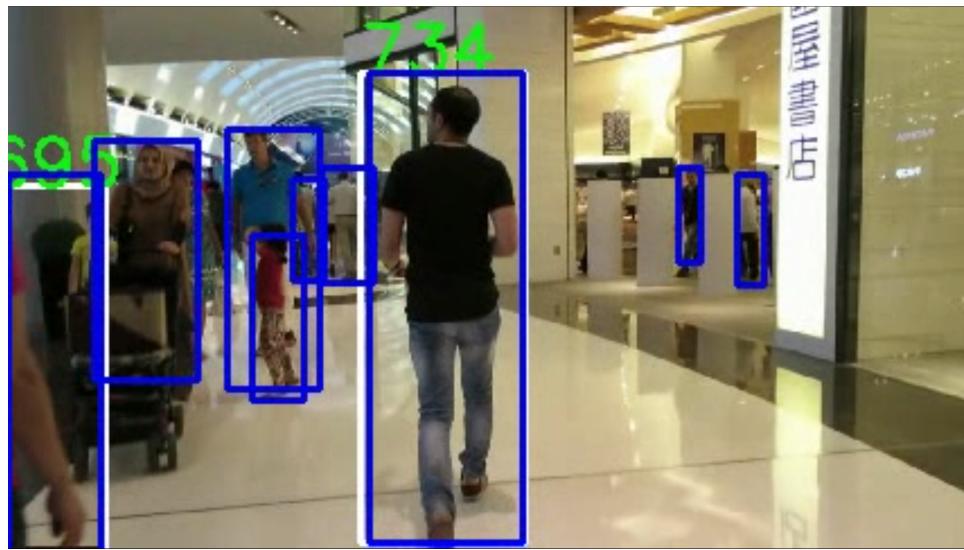


Figura 69: Número de tracks detectados. Intervalo:1 Periodo: 2

Vemos que aunque el número de detecciones es el mismo (*Bounding Boxes* azules) el número de *tracks* (*Bounding Boxes* blancos) es mucho mayor cuando tenemos un periodo de detección de 1 fotograma. Esto ocurre para todos los intervalos, como decimos, el único factor influyente es aumentar el periodo de detección.

Comparativa de pruebas Intervalo: 2

Como en la sección anterior, vamos a comparar los resultados obtenidos esta vez, con un intervalo de lectura igual a 2, junto a distintas configuraciones de periodos de detección, para ver qué combinación es la mejor. Como referencia tendremos la ejecución realizada con periodo de detección igual a 1 e intervalo 2 ya que, como hemos visto en las figuras 68 y 69, es la ejecución que mejor resultado muestra a la hora de asignar trazados a personas en escena. Cuando tengamos un candidato ganador lo compararemos con la ejecución de referencia de intervalo y periodo igual a 1 fotograma.

Las columnas de la tabla muestran la misma información que en la mostrada anteriormente pero, como decimos, esta vez teniendo como referencia la ejecución de la primera fila.

Periodo	FPS	Nº Tracks	Mejora	Error)	Error / Mejora
1	8.5645	160	1	1	1
2	15.0613	797	1.76	5	2.84
3	19.3285	499	2.26	3.12	1.4
4	22.3785	385	2.61	2.4	0.92
5	18.1146	205	2.11	1.3	0.62
6	19.5321	566	2.28	3.53	1.55

Cuadro 12: Resultados de ejecución en MOT17_12

Constantes: Intervalo=2 Resolución:426x240

Nos encontramos dos resultados cuya proporción de Error/Mejora está en el intervalo [0-1). Esto implica que nos aportan mayor mejora que error. Estas ejecuciones no llevan una gran cantidad de error, de hecho, tan sólo supera por 45 trazados a la ejecución de referencia mientras que casi triplica la cantidad de *fps* aportados. Así pues, podemos decir que el periodo de detección más adecuado para un intervalo de 2 fotogramas es 5, ya que, a partir de él, la proporción error/mejora vuelve a incrementar.

7.4.2. Intervalo: 3 fotogramas

Prueba 1

- **Mean time:** 0.130514128152
- **Max track:** 140
- **FPS:** 7.15106310074
- **fotogramas Totales:** 299
- **Tiempo total:** 41.8119649887
- **Intervalo:** 3
- **Periodo:** 1

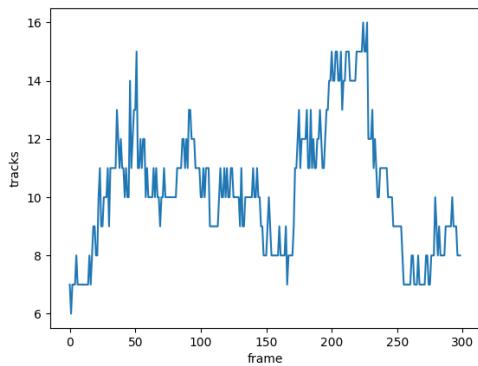


Figura 70: Número de tracks detectados por fotograma

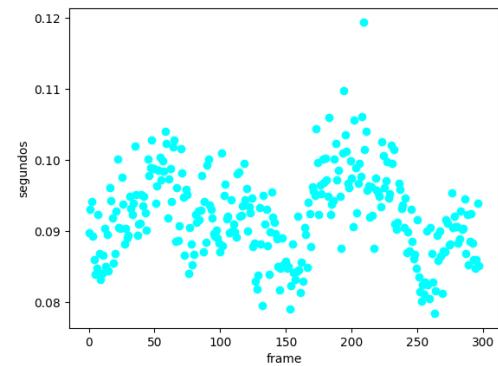


Figura 71: Tiempo de procesamiento requerido por fotograma

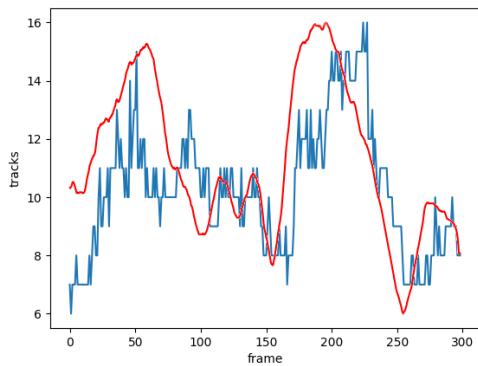


Figura 72: Comparativa de evolución de trazados

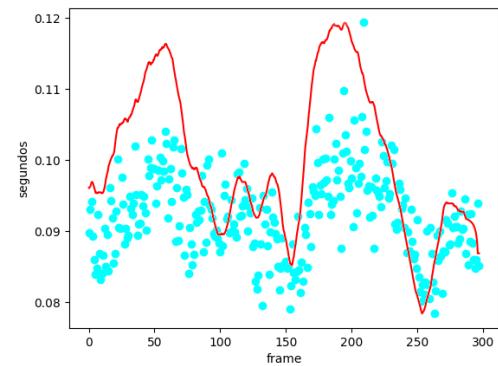


Figura 73: Tiempo de procesamiento junto con evolución de tracks

Como hicimos con el intervalo de 2 fotogramas, lo primero que mencionaremos es que el número de fotogramas procesados ha vuelto a disminuir al aplicar un intervalo mayor. Esta vez hemos procesado un tercio de fotogramas (299 frente a los 899 totales).

Al realizar la comparativa con la ejecución de referencia (intervalo = periodo = 1) se comienza a apreciar la falta de información al saltarnos un tercio de fotogramas pero, de todos modos, se puede seguir apreciando que se mantiene la forma de las gráficas en las figuras 72 y 73.

Como antes, podemos volver a ver la relación entre intervalo y periodo comparando las siguientes ejecuciones:

Intervalo	Periodo	FPS	Tiempo de procesamiento medio por fotograma	Tiempo Procesamiento
1	3	23.40	0.04478	38.4049
3	1	7.22	0.12906	41.3844

Cuadro 14: Comparativa Intervalo vs Periodo

La relación entre los *fps* es ahora de 3 a 1 debido al periodo de detección cada 3 fotogramas de la ejecución de la primera fila. Pero, aún siendo el procesamiento medio por fotograma cerca de 3 veces más rápido, el tiempo total es prácticamente igual debido a la lectura en intervalos de 3 fotogramas por parte de la ejecución de la segunda fila.

El pico de trazados también se muestra más alto de lo que lo hace en la ejecución de referencia de intervalo y periodo de 1 fotograma y es por los mismos motivos explicados anteriormente: debido a que la secuencia de fotogramas no es fluida puede haber trazados o *tracks* que perduren en el tiempo buscando una figura que ya no está en escena.

Comparativa de pruebas Intervalo: 3

En las dos comparativas anteriores mostraremos los resultados de las ejecuciones en una tabla donde la primera fila será la ejecución de referencia por tener el periodo de detección de 1 fotograma y ser por tanto la más precisa en cuanto a los trazados calculados.

Periodo	FPS	Nº Tracks	Mejora (FPS / 7.22)	Error (Nº Tracks / 140)	Error / Mejora
1	7.2249	140	1	1	1
2	11.3784	494	1.58	3.5	2.23
3	14.5387	353	2.01	2.52	1.25
4	18.81	268	2.6	1.91	0.74
5	15.6862	124	2.17	0.9	0.41
6	16.8406	353	2.32	2.52	1.1

Cuadro 15: Resultados de ejecución en MOT17_12

Constantes: Intervalo=3 Resolución:426x240

Como antes, nos encontramos dos resultados cuya proporción de Error / Mejora está en el intervalo [0-1) y por lo tanto nos aportan mayor mejora que error. De estas dos, hay una ejecución en específico que nos aporta menos tracks que la ejecución de referencia por lo que podríamos decir que no sufre errores, de hecho diríamos que mejora la primera ejecución, pero en este caso no sería tan acertada esta afirmación. La primera ejecución ya presenta menos tracks que la ejecución de referencia de la Prueba 1 (intervalo y periodo de 1 fotograma) lo que supone menos precisión respecto al seguimiento de personas. Esta ejecución con periodo de detección cada 5 fotogramas presenta menos trazados todavía, por lo que sería menos precisa aún. Sin embargo, como todo este estudio va dirigido a la ejecución del sistema de videovigilancia en una infraestructura crítica, donde el tránsito de personas es muy escaso o nulo, el hecho de poder aplicar seguimiento hasta a 124 figuras es más que suficiente. Por ello, podemos declarar la configuración de periodo de detección igual a 5 fotogramas como la más adecuada para un intervalo de 3 fotogramas.

Comparativa de Intervalos

Intervalo	Periodo	FPS	Nº Tracks	Tiempo de Ejecución
1	3	23.0309	489	39.0345
2	5	18.1146	205	24.7866
3	5	15.6862	124	19.0612

Cuadro 17: Comparativa de ejecuciones con distintos intervalos de lectura

Como hemos dicho al inicio de la comparativa con intervalos superiores a 1 fotograma, el dron transmite 30 imágenes por segundo. Con la primera opción de parámetros, procesamos 23 imágenes por segundo lo cual implica perder 7 fotogramas de los 30 mandados por el dron (si es que no hubiese problemas de conexión y los recibiéramos todos). Con cualquier otra opción perderíamos muchos más de 7 fotogramas. Perderíamos la información de 15 fotogramas en el caso de usar un intervalo de 2 fotogramas y 20 si el intervalo fuese de 3. Por esto, decidiremos aplicar un intervalo dinámico o asíncrono, con un periodo de detección cada 3 fotogramas ya que, aunque sea asíncrono, conseguiremos leer la mayoría de ellos secuencialmente (intervalo de lectura=1).

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 apenas notable.
- **Intervalo:** Influye en la cantidad de tracks detectados debido a que, al saltarnos fotogramas, perdemos información. Y reduce el tiempo de procesamiento total por la misma razón.
- **Periodo:** Influye mucho en la consistencia de los trazos (*tracks*) calculados ya que funciona como una realimentación para los atributos del identificador. También influye en el tiempo de procesamiento al evitar realizar detección de figuras en escena en todos los fotogramas.

Recordamos que todas las pruebas junto con los resultados, vídeos y gráficas obtenidas se encuentran en la carpeta de MEGA [28] para cualquier consulta y/o aclaración.

7.5. Consistencia del Tracker

Como hemos visto en el apartado de la oclusión, el detector se desenuelve muy bien incluso a una resolución tan baja como es 426x240. Será esta resolución la que usaremos para realizar las siguientes pruebas además de un intervalo de lectura de 1 fotograma y un periodo de detección cada 3 fotogramas como hemos concluido en las experimentaciones anteriores. Vamos a evaluar la consistencia del algoritmo de seguimiento (*tracker*) con el benchmark MOT17_05:

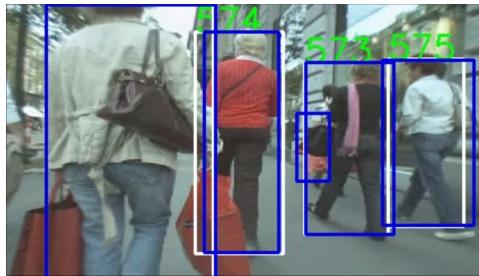


Figura 74: MOT17_05 secuencia:0



Figura 75: MOT17_05 secuencia:1



Figura 76: MOT17_05 secuencia:2



Figura 77: MOT17_05 secuencia:3

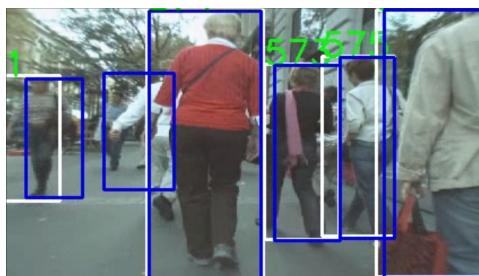


Figura 78: MOT17_05 secuencia:4

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 tres personas con identificadores 574, 573 y 575.

En las tres siguientes, otra persona se cruza delante de la cámara y las tapa por completo perdiendo así su detección, pero los tracks siguen activos buscando figuras que encajen con su descripción.

En las figuras 76 y 78, las tres personas vuelven a aparecer en escena y el algoritmo es capaz de identificar que se tratan de las mismas figuras que antes y les reasigna el mismo ID.

Esta situación grabada desde un dron sería, casi con total seguridad, más sencilla para el algoritmo de seguimiento. La escena mostraría un plano picado de la zona y, por lo tanto, las personas se mostrarían más separadas entre sí. No tendríamos tantas situaciones de oclusión como sucede en la secuencia superior y obtendríamos un rango de visión mucho más amplio. La red neuronal tendría que ser entrenada para detectar personas desde planos picados, ya que el conjunto de datos COCO entrena la red para reconocer personas que se encuentren de pie y desde un punto de vista que permita ver el cuerpo entero pero no vistas desde arriba. Como en nuestro caso, por limitaciones de distancia de conexión no podemos elevar el dron hasta grandes alturas, las imágenes tomadas desde él muestran personas en planos reconocibles por el detector y no ha sido necesario volver a entrenar el detector YOLOv3 [6].

7.6. Comparación: DeepSORT vs UnsupTrack

Vamos a comparar los resultados obtenidos por la implementación utilizada de DeepSORT y un tracker más preciso como es UnsupTrack (*Simple Unsupervised Multi-Object Tracker*) [49] en el vídeo MOT17_06. Realizamos esta comparación debido a que UnsupTrack es un algoritmo más preciso ya que realiza cálculos más complejos y no es a tiempo real. Con esta comparativa queremos ver si el funcionamiento de la implementación escogida se acerca lo suficiente a lo que consideraríamos un buen resultado. La implementación escogida de UnsupTrack utiliza el detector DPM (Deformable Part Models) [50].

Sobre el papel vemos la diferencia de precisión si nos fijamos en el número total de identificadores diferentes asignados. Mientras que UnsupTrack utiliza tan sólo 97 IDs, DeepSORT ha usado 529. Por otro lado no debemos dejar de mencionar que DeepSORT permite realizar una ejecución en tiempo real mientras que UnsupTrack no. En el primero hemos obtenido alrededor de 8 *fps* mientras que en el segundo tan sólo 2 *fps* y esto teniendo en cuenta que contamos con un equipo hardware limitado, al contrario que ocurre con UnsupTrack, que contaron con hardware de última generación para realizar las pruebas. Aún así, conseguimos cuaduplicar el número de fotogramas por segundo.

A continuación compararemos los resultados atendiendo a las imágenes obtenidas. Primero mostraremos toda la secuencia de imágenes para cada algoritmo y luego buscaremos las semejanzas y diferencias entre ambas ejecuciones:

UnsupTrack

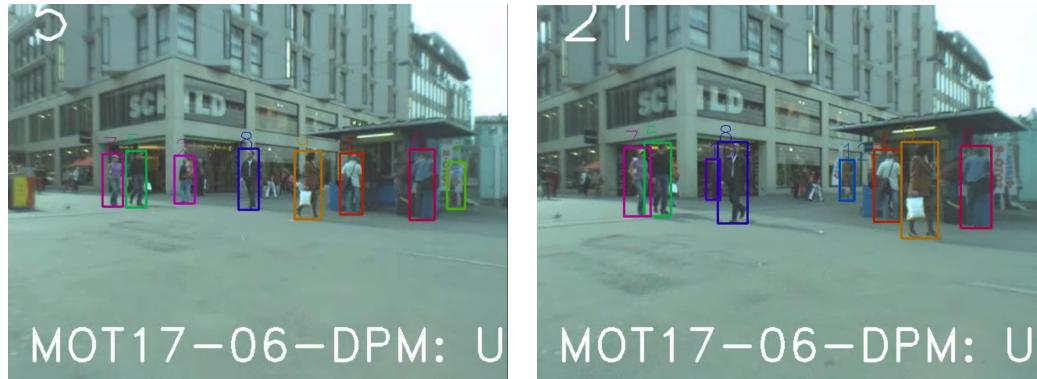


Figura 79: MOT17_06 secuencia:1

Figura 80: MOT17_06 secuencia:2

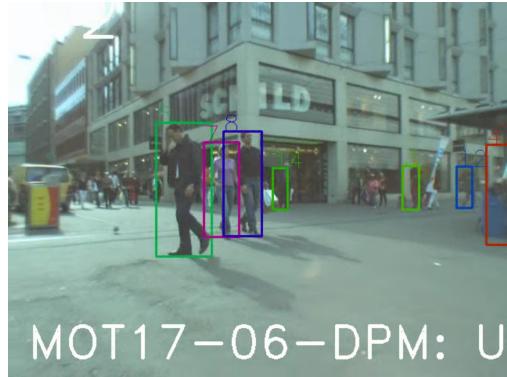


Figura 81: MOT17_06 secuencia:3



Figura 82: MOT17_06 secuencia:4



Figura 83: MOT17_06 secuencia:5



Figura 84: MOT17_06 secuencia:6



Figura 85: MOT17_06 secuencia:7

DeepSORT

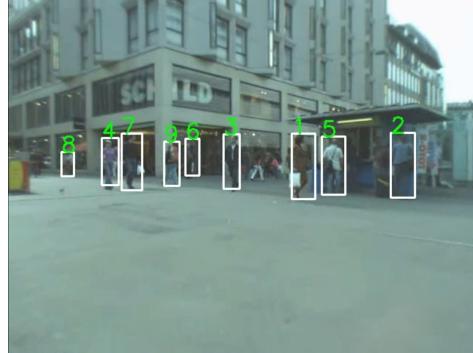


Figura 86: MOT17_06 secuencia:1

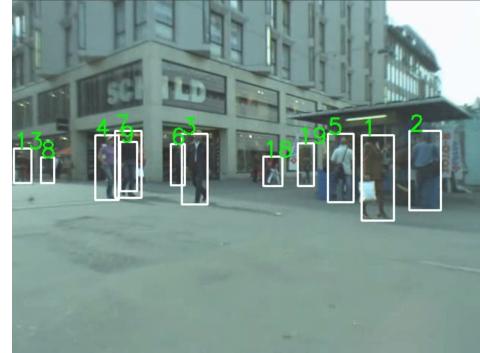


Figura 87: MOT17_06 secuencia:2

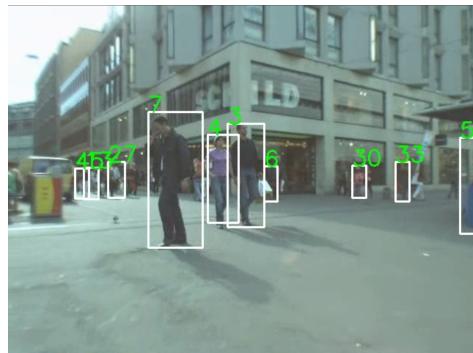


Figura 88: MOT17_06 secuencia:3

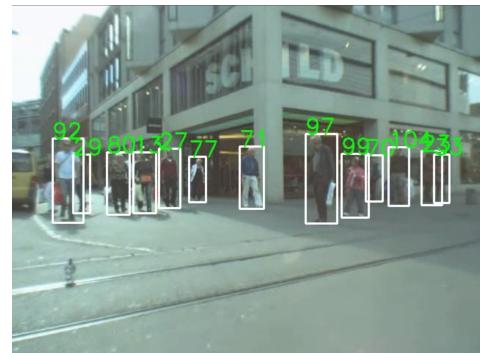


Figura 89: MOT17_06 secuencia:4



Figura 90: MOT17_06 secuencia:5

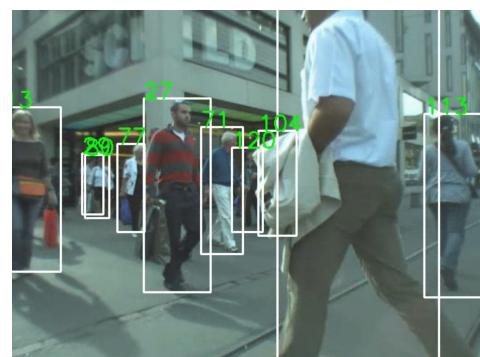


Figura 91: MOT17_06 secuencia:6



Figura 92: MOT17_06 secuencia:7

En las parejas de figuras (79, 80) y (86, 87) mostramos un ejemplo sencillo de consistencia para ver si el ID asignado al hombre que viste de blanco en el quiosco cambia o no al pasar la mujer por delante. En ambos casos el ID se mantiene inalterable.

Entre las secuencias (80, 81) y (87, 88) vemos qué ocurre cuando el hombre marcado con 8 en UnsupTrack y con 7 en DeepSORT pasa por delante de la pareja de personas marcada con 5 y 7 en el primer caso y con 4 y 7 en el segundo. En ambos casos, el hombre que se cruza intercambia su ID con una de las personas de la pareja. Sufren el mismo error ambas ejecuciones.

En las últimas secuencias (4,5,6,7) debemos fijarnos en las personas marcadas en UnsupTrack con los IDs 2, 23, 17, 19 y 49 y en DeepSORT con 80, 13, 27, 71 y 113. En UnsupTrack todos los identificadores se mantienen constantes e inalterables (excepto el ID:5) a pesar de las sucesivas oclusiones debidas a los cruces entre personas. En cambio, en la implementación de DeepSORT, la persona marcada con ID:80 en la figura 89 cambia a ser el ID:90 en la siguiente figura 90. Por otro lado, la persona marcada con ID:5 en la figura 82 pierde este identificador y lo cambia por ID:44 mientras que en la implementación de DeepSORT aparece con ID:77 en la figura 89 y se mantiene así hasta que desaparece la persona de escena.

Como vemos, la consistencia de la implementación escogida del tracker DeepSORT, es muy parecida a la de un tracker en estado del arte. La mayor diferencia la encontramos a la hora de asignar identificadores a las personas en escena. El tracker que no es en tiempo real es mucho más preciso a la hora de “contar” personas y asignarles identificadores pero en la característica que más nos importa, la consistencia ante oclusiones y salidas de escena, obtenemos un buen comportamiento. Por lo tanto, podemos concluir que la implementación escogida de DeepSORT es suficientemente buena para obtener resultados favorables.

7.7. Pruebas con el Dron

7.7.1. Ejecución de la aplicación desarrollada

La aplicación cuenta con dos modos de vuelo: manual y automático. El modo manual no activa el módulo de pilotaje o comportamiento reactivo, solo funciona el módulo de visión por computador de manera que la detección y seguimiento de personas se encuentra activa así como la visualización de la cámara del dron en el equipo donde ejecutamos la aplicación. En cambio, el modo de vuelo automático inicia una nueva hebra para llevar a cabo el cómputo requerido por el módulo de comportamiento reactivo. El hecho de iniciar o no el vuelo automático, repercute en la fluidez de la ejecución ya que al crear una nueva hebra sobrecarga el equipo. En el equipo de sobremesa no supone ninguna limitación pero, como veremos más adelante, en un equipo que no posea tarjeta gráfica y cuya CPU sea limitada, supone un gran sobre esfuerzo. Esta es la comparación de los resultados en vuelo manual y automático en el ordenador de sobremesa.

Solo visión por computador

- **Mean time:** 0.0341253438511
- **Max track:** 16
- **FPS:** 25.8549240759
- **fotogramas Totales:** 1480
- **Tiempo total:** 57.2424809933
- **Intervalo:** Asíncrono
- **Periodo:** 3

Visión por computador + Movimiento automático

- **Mean time:** 0.039121880077
- **Max track:** 13
- **Real FPS:** 23.7507723727
- **fotogramas Totales:** 1291
- **Tiempo total:** 54.3561270237
- **Intervalo:** Asíncrono
- **Periodo:** 3

Vemos que el activar el modo de vuelo automático solo repercute ligeramente en el tiempo de procesamiento de cada fotograma y por tanto en los *fps*.

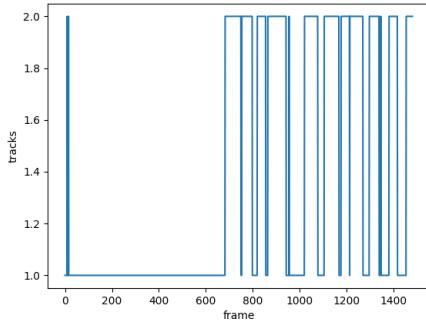


Figura 93: Trazados en modo solo visión (pilotaje manual)

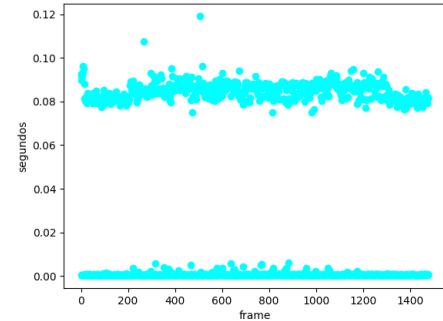


Figura 94: Tiempos de procesamiento en modo solo visión (pilotaje manual)

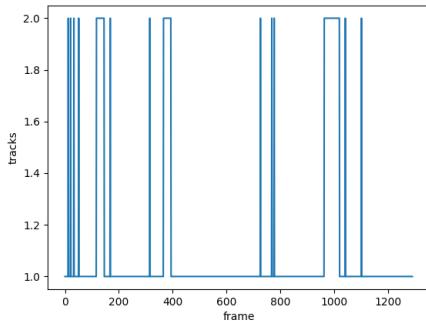


Figura 95: Trazados en modo pilotaje automático

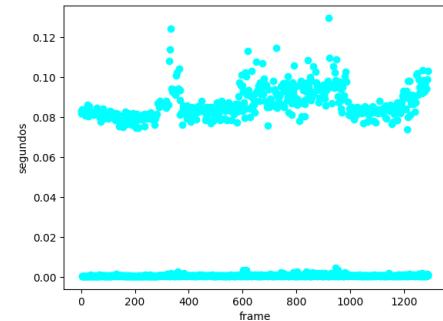


Figura 96: Tiempos de procesamiento en modo pilotaje automático

Vemos en el gráfico 94 que el tiempo de procesamiento se mantiene más o menos constante durante toda la ejecución. En cambio, en el gráfico de procesamiento del modo de vuelo automático 96, vemos que los tiempos de procesamiento aumentan en los fotogramas en los que se está siguiendo a una figura (entre los fotogramas [250,400], [600,1000] y [1200,fin] aproximadamente) y bajan cuando el trazado del objetivo se desactiva o se pierde (entre los fotogramas [0,250], [400,600] y [1000,1200]). Sin embargo, esta subida en el tiempo de procesamiento no es apenas notable ya que en el peor caso supone, aproximadamente, 0.13 segundos

7.7.2. Prueba de aplicación final

Para ponernos en contexto, esta prueba la realizamos para dar un ejemplo aproximado del fin para el que se ha desarrollado este proyecto. Suponemos una infraestructura crítica que cuenta con cámaras de video vigilancia fijas o estáticas (que en nuestra prueba será la webcam), y con una cámara móvil incorporada en un dron (en la prueba será el dron Bebop 2). Este sistema vigilará mediante las cámaras fijas una determinada zona y si entra un sospechoso se procederá a seguirlo por escena. Si el individuo desaparece de la imagen proporcionada por las cámaras estáticas, se mandará la cámara móvil (dron) al escenario para que siga realizando el seguimiento del intruso pero esta vez pudiendo perseguirlo activamente en el mundo real. Una vez puestos en situación, vamos a ver los resultados:

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 trazado 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.

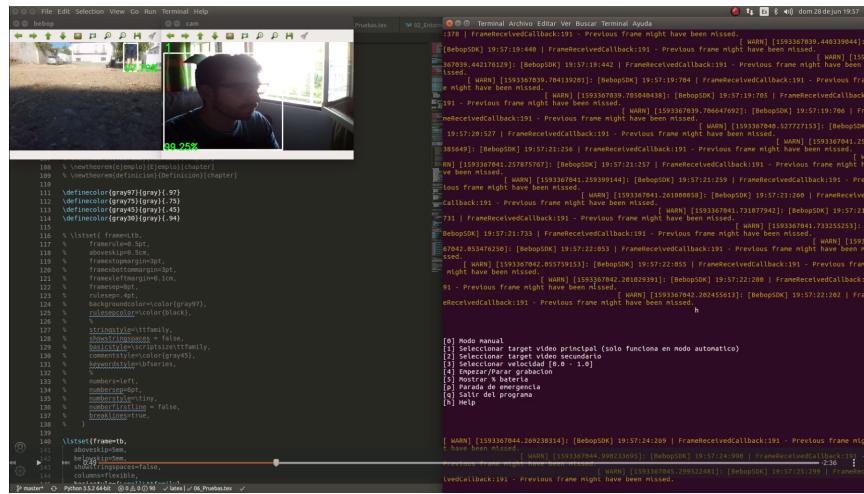


Figura 97: Inicio del sistema

En esta captura vemos que al iniciar el sistema contamos con la visión de la cámara fija en la que aparece una figura con un trazado asignado. 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 alrededor 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
[6] Parada de emergencia
[7] 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 98: Selección de ID en cámara fija

Si ampliamos la captura de pantalla superior 98, 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 (opción 2). Seleccionamos el identificador 1 que es el único que aparece en pantalla.

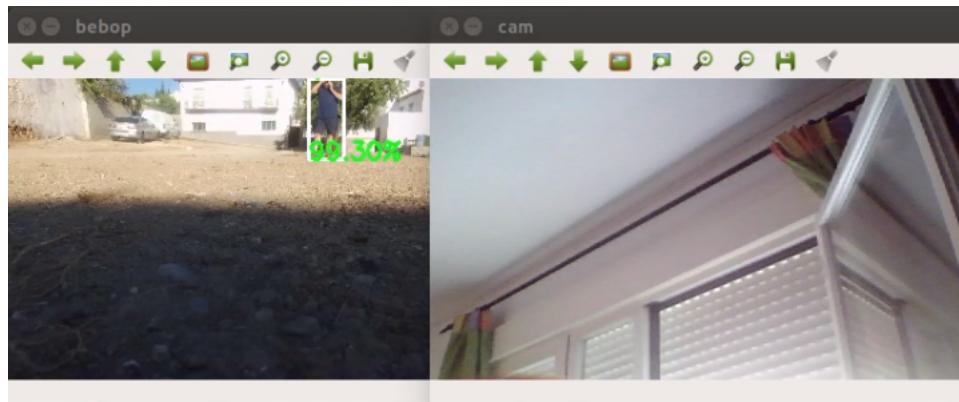


Figura 99: Salida de escena

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 trazado se desactiva y se manda un comando al dron para que despegue.

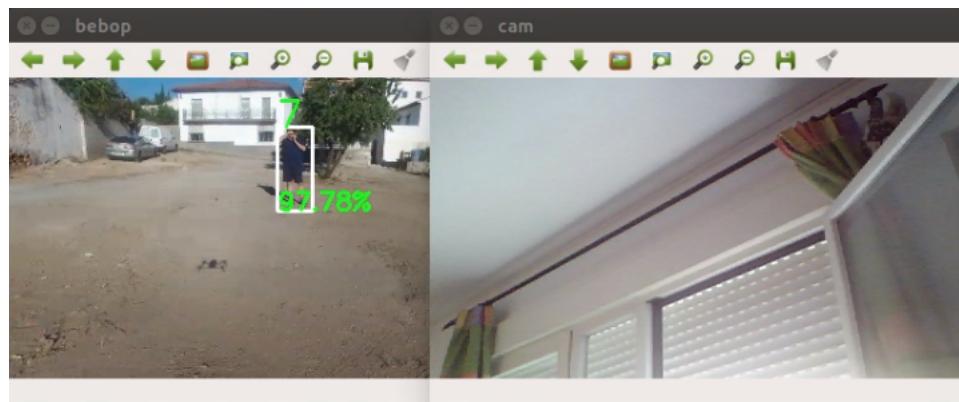


Figura 100: Despegue automático

Aquí podemos ver al dron ya en el aire pero aún estático. A continuación indicamos el ID del trazado 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 101: Selección de objetivo en vídeo del dron

Seleccionamos la opción del menú para indicar el trazado que aparece en el vídeo del dron (opción 1) e introducimos el trazado 7. En este momento se actualiza el trazado 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 seguimiento situando el dron en situaciones adversas.



Figura 102: 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*'.

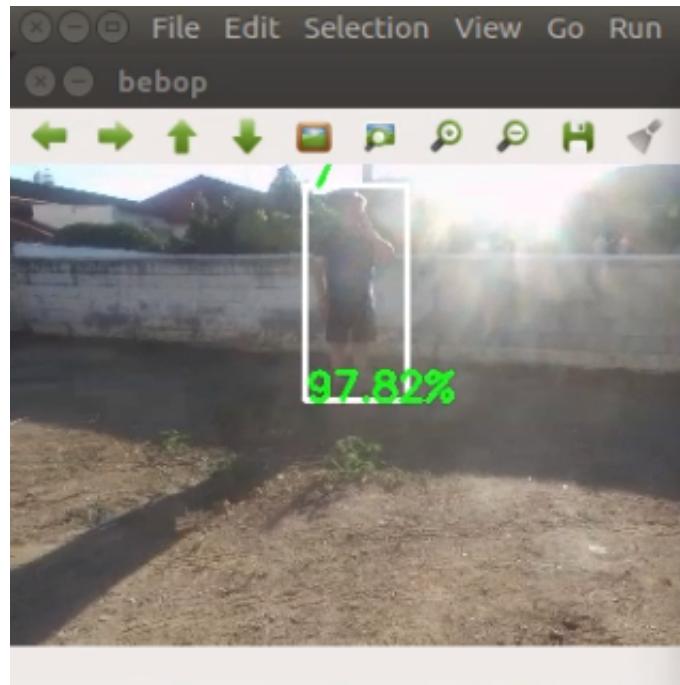


Figura 103: Detección a contra luz - Servidor

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 104: Detección en alto contraste - Bebop

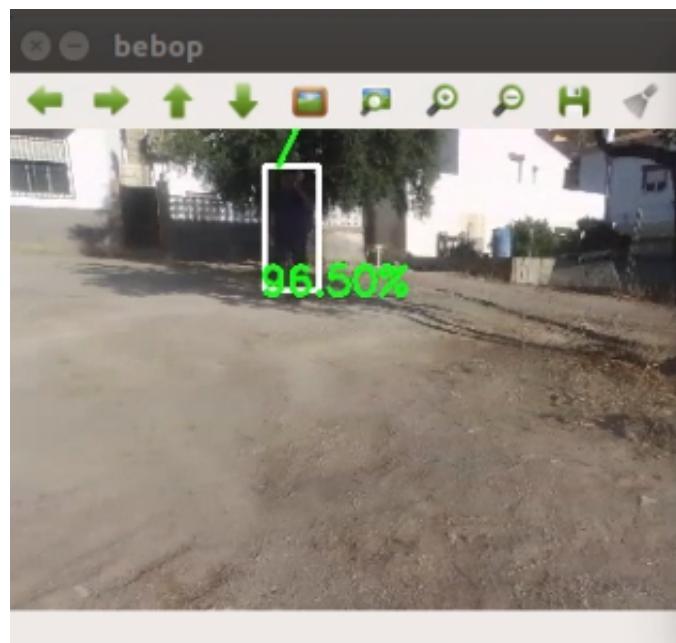


Figura 105: Detección en alto contraste - Servidor

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 [28] unos vídeos que muestran toda la consecución de hechos de principio a fin.

7.7.3. Vuelo con app

- **Modo Manual:**

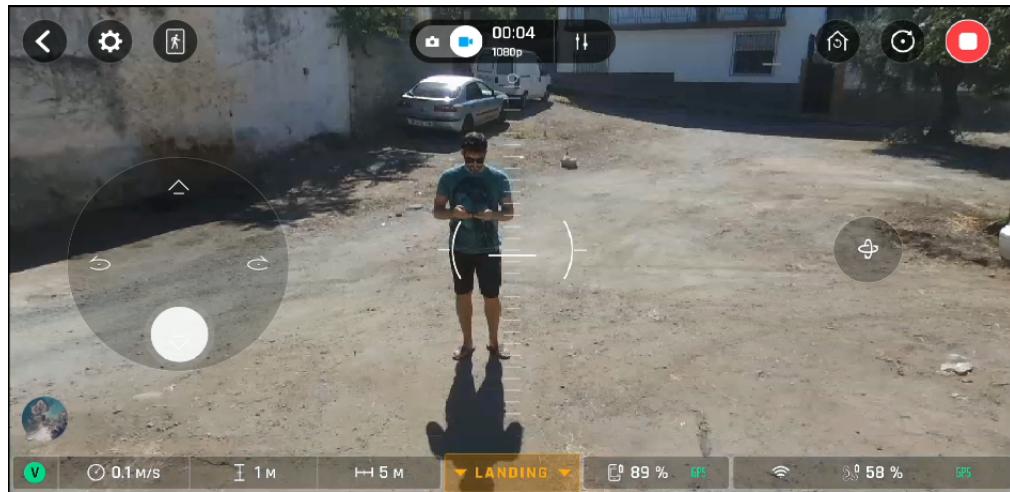


Figura 106: Interfaz HUD (Head-Up Display) del modo pilotaje manual de la aplicación Free Flight Pro

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 107: 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.

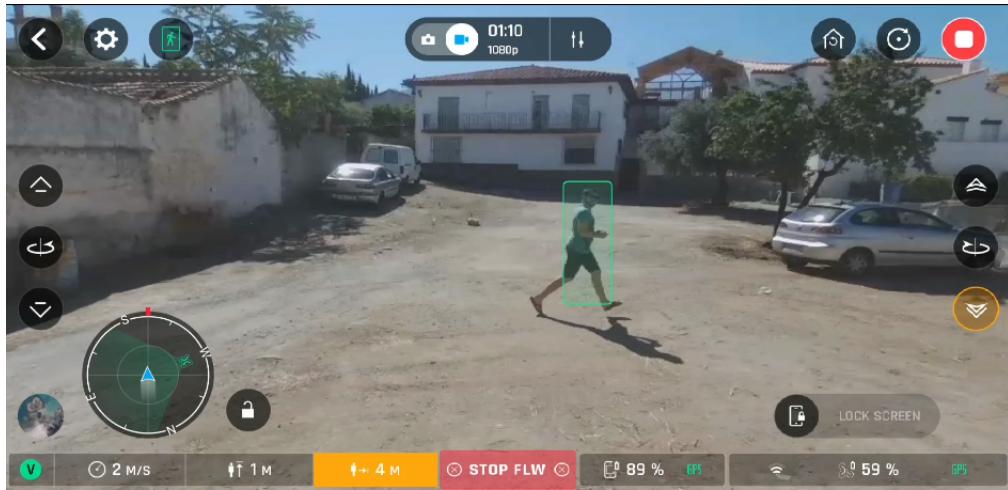


Figura 108: Dron siguiendo al objetivo de manera activa

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.

7.7.4. Ejecución en Equipo Portátil

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

- **Mean time:** 0.435020330448
- **Max track:** 2
- **FPS:** 2.28239927663
- **fotogramas Totales:** 207
- **Tiempo total:** 90.6940350533
- **Intervalo:** Asíncrono
- **Periodo:** 3

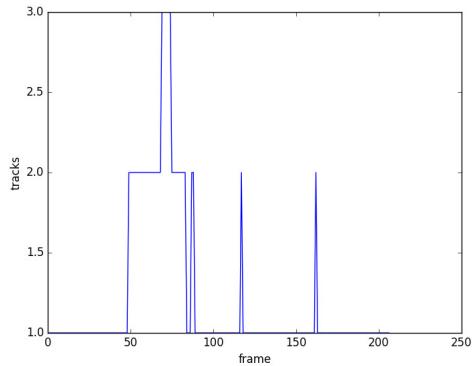


Figura 109: Trazados detectados en el equipo portátil

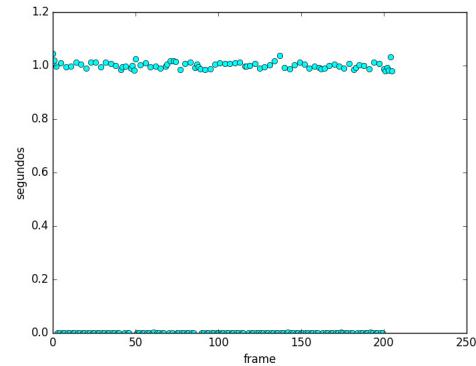


Figura 110: Tiempos de procesamiento en el equipo portátil

Tardamos prácticamente lo mismo en procesar 207 fotogramas en el equipo portátil que 899 con el equipo de sobremesa. Como tiempo de procesamiento medio obtenemos 0.5 segundos, pero si observamos la gráfica de tiempos de procesamientos de fotogramas vemos que en aquellos donde aplicamos detección tardamos en procesarlos 1 segundo cuando en el ordenador de sobremesa no hemos llegado a sobrepasar los 0.115 segundos. Con esta velocidad de procesamiento no conseguimos una cantidad aceptable de *fps* ni si quiera siendo una lectura asíncrona y con un periodo de detección cada 3 fotogramas.

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 necesaria para realizar el procesamiento

requerido por el sistema.

La siguiente imagen 111 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 aún sólo cuando es el módulo de visión por computador el único que está activo.

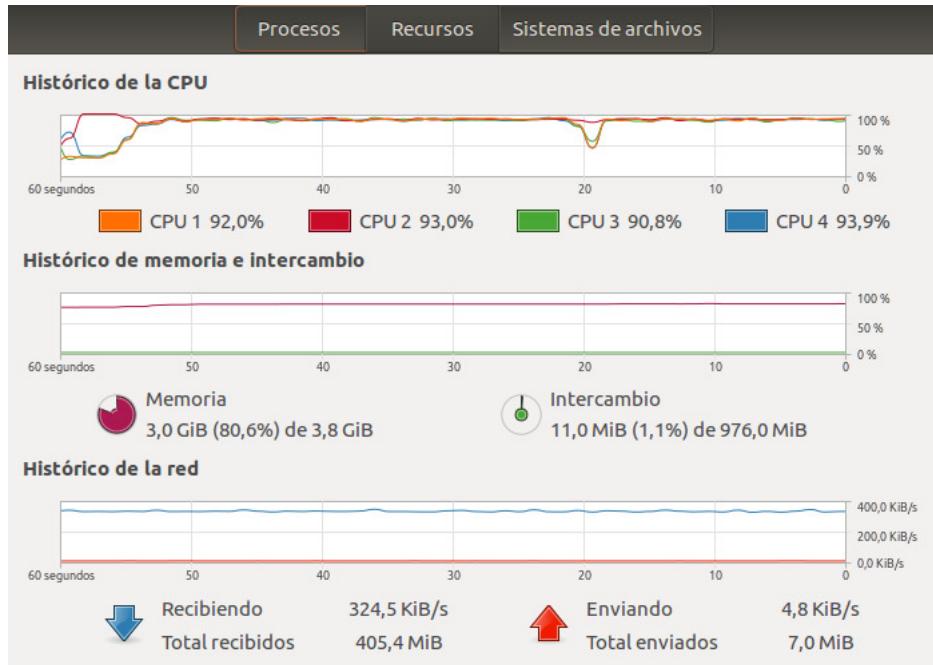


Figura 111: Estado del equipo portátil durante la ejecución de la aplicación

Si iniciamos el módulo de movimiento autónomo, que supone la creación de una nueva hebra, los resultados son incluso peores:

- **Mean time:** 0.833248885666
- **Max track:** 6
- **FPS:** 1.19439321945
- **fotogramas Totales:** 82
- **Tiempo total:** 68.6541070938
- **Intervalo:** Asíncrono
- **Periodo:** 3

Vemos cómo han empeorado los tiempos de procesamiento drásticamente. Obtenemos 1.2 *fps* cuando antes teníamos 2.3, casi el doble. En la gráfica

113, podemos ver exactamente cuándo activamos el modo de pilotaje automático ya que los tiempos de procesamiento ascienden debido a la creación de la nueva hebra (aproximadamente en el fotograma 25).

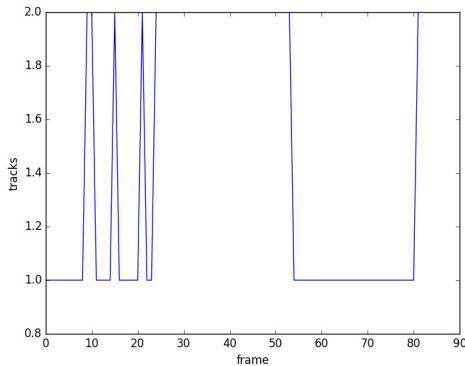


Figura 112: Trazados detectados en el equipo portátil

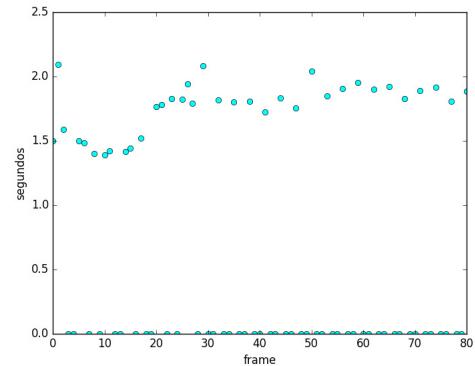


Figura 113: Tiempos de procesamiento en el equipo portátil

Es por esto que el desarrollo y ejecución del proyecto en el equipo portátil del que disponíamos ha sido inviable pero, sin duda, si este equipo hubiese contado con las características que dispone el ordenador de sobremesa, habría sido la opción más interesante dado que nos aporta la libertad de ejecutar la aplicación en cualquier lugar y podríamos haber hecho vuelos en zonas más despejadas o preparadas para vuelos de drones sin tener que preocuparnos por problemas de conexión. Todo esto, claro, en la medida que nos hubiese sido posible debido al COVID-19.

7.7.5. Otros experimentos para evaluar Características No Funcionales de Calidad

- **Ancho de banda:** Hemos medido el ancho de banda que consume comunicarnos con el dron desde el fotogramawok ROS. El *topic image_raw* por el que mandamos los mensajes tipo *Image* que contienen los fotogramas 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 repercute 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 al rededor de 20 minutos reales con unas baterías nuevas.

8. 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 seguimiento o *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 parece importante, 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 [27].

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] Directiva europea: 2008/114/CE - Infraestructuras Críticas. <https://eur-lex.europa.eu/legal-content/ES/TXT/?uri=celex:32008L0114>.
- [2] Documentación Bebop Autonomy. <https://bebop-autonomy.readthedocs.io/en/latest/index.html>.
- [3] ROS. <https://www.ros.org/>.
- [4] G. G. Brown, W. M. Carlyle, J. Salmerón, and R. K. Wood, “Defending critical infrastructure,” *Interfaces*, vol. 36, no. 6, pp. 530–544, 2006. <https://pubsonline.informs.org/doi/abs/10.1287/inte.1060.0252>.
- [5] Amazon prohíbe a la policía usar su reconocimiento facial - Urban Tecno. <https://urbantecno.com/tecnologia/amazon-reconocimiento-facial-policia>.
- [6] Joseph Redmon - YOLOv3. <https://pjreddie.com/darknet/yolo/>.
- [7] DeepSORT. <https://nanonets.com/blog/object-tracking-deepsort/>.
- [8] Drones en el Cine. <https://dronelife.com/2018/08/07/drones-are-becoming-a-filmmakers-tool/>.
- [9] Servicio de Paquetería con Drones - UPS. <https://www.ups.com/us/es/services/knowledge-center/article.page?kid=cd18bdc2>.
- [10] Servicio de Paquetería con Drones - Amazon. <https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011>.
- [11] Servicio de Paquetería mediante el uso de Drones - Correos. <http://www.fesofi.es/noticias/correos-utilizara-drones-para-repartir-en-zonas-dificiles/>.
- [12] Aplicaciones de Drones Urbanísticas - Universidad de Córdoba. <http://drones.uv.es/aplicaciones-de-drones-urbanisticas/>.
- [13] Drones DGT. <http://revista.dgt.es/es/multimedia/infografia/2019/04ABRIL/0417-Drones.shtml#.XvtoMpZS9hF>.
- [14] Monitorización y optimización de tierras con drones y fotogrametría aérea para aplicaciones de precisión en agricultura - Universidad Politécnica de Valencia. <https://riunet.upv.es/handle/10251/86353>.

- [15] Z. Zaheer, A. Usmani, E. Khan, and M. A. Qadeer, “Aerial surveillance system using UAV,” in *Thirteenth International Conference on Wireless and Optical Communications Networks, WOCN 2016, Hyderabad, Telangana State, India, July 21-23, 2016*, pp. 1–7, IEEE, 2016. <https://doi.org/10.1109/WOCN.2016.7759885>.
- [16] CB-Insights. <https://www.cbinsights.com/research/drone-impact-society-uav/>.
- [17] Divya Joshi - Business Insider. <https://www.businessinsider.com/drone-technology-uses-applications?IR=T>.
- [18] Federal Aviation Administration. <https://www.faa.gov/>.
- [19] Ascent Vision Technologies. <https://ascentvision.com/>.
- [20] Microdrones. <https://www.microdrones.com/en/>.
- [21] Axis Communications. <https://www.axis.com/solutions-by-industry/critical-infrastructure>.
- [22] J. Redmon and A. Farhadi, “Yolov3: An incremental improvement,” *CoRR*, vol. abs/1804.02767, 2018. <https://pjreddie.com/media/files/papers/YOL0v3.pdf>.
- [23] YOLO vs Mask-RCNN. SORT tracker. <https://medium.com/neuromation-blog/tracking-cows-with-mask-r-cnn-and-sort-fcd4ad68ec4f>.
- [24] Modelo de Prototipos. https://es.wikipedia.org/wiki/Modelo_de_prototipos.
- [25] Trello. <https://trello.com/es>.
- [26] Slack. <https://slack.com/intl/es-es/>.
- [27] Repositorio del proyecto - TFG-Bebop-YOLO. <https://github.com/JJavier98/TFG-Bebop-YOLO>.
- [28] Carpeta MEGA con resultados de los experimentos. https://mega.nz/folder/YQpWEAiY#Xq4C890boVrrkhrA-68_AQ.
- [29] ROS Wiki. <http://wiki.ros.org/ROS/Introduction>.
- [30] Desarrollo de aplicaciones basadas en visión con entorno ROS para el dron Bebop 2 - Valero Lavid - Universidad Politécnica de Alcalá. <https://ebuah.uah.es/dspace/handle/10017/30383>.
- [31] Estructura de mensaje Image de ROS. http://docs.ros.org/melodic/api/sensor_msgs/html/msg/Image.html.

- [32] Estructura de mensaje Twist de ROS. http://docs.ros.org/melodic/api/geometry_msgs/html/msg/Twist.html.
- [33] Mani Monajjemi y Colaboradores. <http://bebop-autonomy.readthedocs.io/en/latest/contribute.html#sec-contribs>.
- [34] Bebop2. <https://www.parrot.com/es/drones/parrot-bebop-2>.
- [35] Estructura de mensaje Twist de ROS. https://es.wikipedia.org/wiki/Pilotaje_con_visi%C3%B3n_remota.
- [36] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, “SSD: single shot multibox detector,” *CoRR*, vol. abs/1512.02325, 2015. <https://arxiv.org/abs/1512.02325>.
- [37] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, “Focal loss for dense object detection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 2, pp. 318–327, 2020. <https://ieeexplore.ieee.org/document/8417976>.
- [38] Funciones de Activación. <https://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/>.
- [39] Anchor Box. <https://es.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html>.
- [40] DeepSORT Paper. <https://arxiv.org/pdf/1703.07402.pdf>.
- [41] Hungarian Algorithm. <https://brilliant.org/wiki/hungarian-matching/>.
- [42] VirtualBox. <https://www.virtualbox.org/>.
- [43] NVIDIA CUDA. <https://developer.nvidia.com/cuda-zone>.
- [44] CITIC. <http://citic.ugr.es/>.
- [45] Qidian213 - DeepSORT. https://github.com/Qidian213/deep_sort_yolov3.
- [46] DeepSORT videocaptureasync module. https://github.com/Qidian213/deep_sort_yolov3/issues/154.
- [47] Gaussian Blur. https://en.wikipedia.org/wiki/Gaussian_blur.
- [48] MOT Challenge. <https://motchallenge.net/>.
- [49] S. Karthik, A. Prabhu, and V. Gandhi, “Simple unsupervised multi-object tracking,” *CoRR*, vol. abs/2006.02609, 2020. <https://arxiv.org/abs/2006.02609>.

- [50] R. B. Girshick, F. N. Iandola, T. Darrell, and J. Malik, “Deformable part models are convolutional neural networks,” pp. 437–446, 2015.
<https://arxiv.org/abs/1409.5403>.

*