TRABAJO FIN DE GRADO

INGENIERÍA EN INFORMÁTICA

Seguimiento Activo de Personas con dron Parrot Bebop2

Integración en ROS

Autor
José Javier Alonso Ramos

Directores
Fran Barranco Expósito
Eduardo Ros Vidal

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Julio de 2020

*

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

José Javier Alonso Ramos

Palabras clave: parrot, bebop, bebop2, autonomy, autonomía, dron, drone, UAV, track, tracking, YOLO, YOLOv3, python, python2, python2.7, Deep, SORT, Deep-SORT, CNN, 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 video vigilancia activo y un estudio de la eficacia y la eficiencia del mismo. Como elemento activo se utiliza el dron Parrot Bebop 2 que hace la función de cámara móvil. Todo el trabajo se implementa sobre el framework ROS (Robot Operating System)

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

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

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

Active Person Tracking using Parrot Bebop2 drone: Running in ROS

José Javier Alonso Ramos

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

Abstract

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

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

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

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

Poner aquí agradecimientos...

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 25 de junio de 2020.

Índice

Índice de figuras

Índice de cuadros

1. Introducción

1.1. Objetivos

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

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

1.2. Estado del Arte

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

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

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

Otro sector del que no puede faltar mención, es sin duda el más avanzado en el tema: el sector militar. Es aquí donde se producen las mayores inversiones en temas de investigación y desarrollo y donde podemos encontrar la

tecnología más puntera. Desde drones de tamaño minúsculo, hasta drones gigantescos del tamaño de una avioneta.

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

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

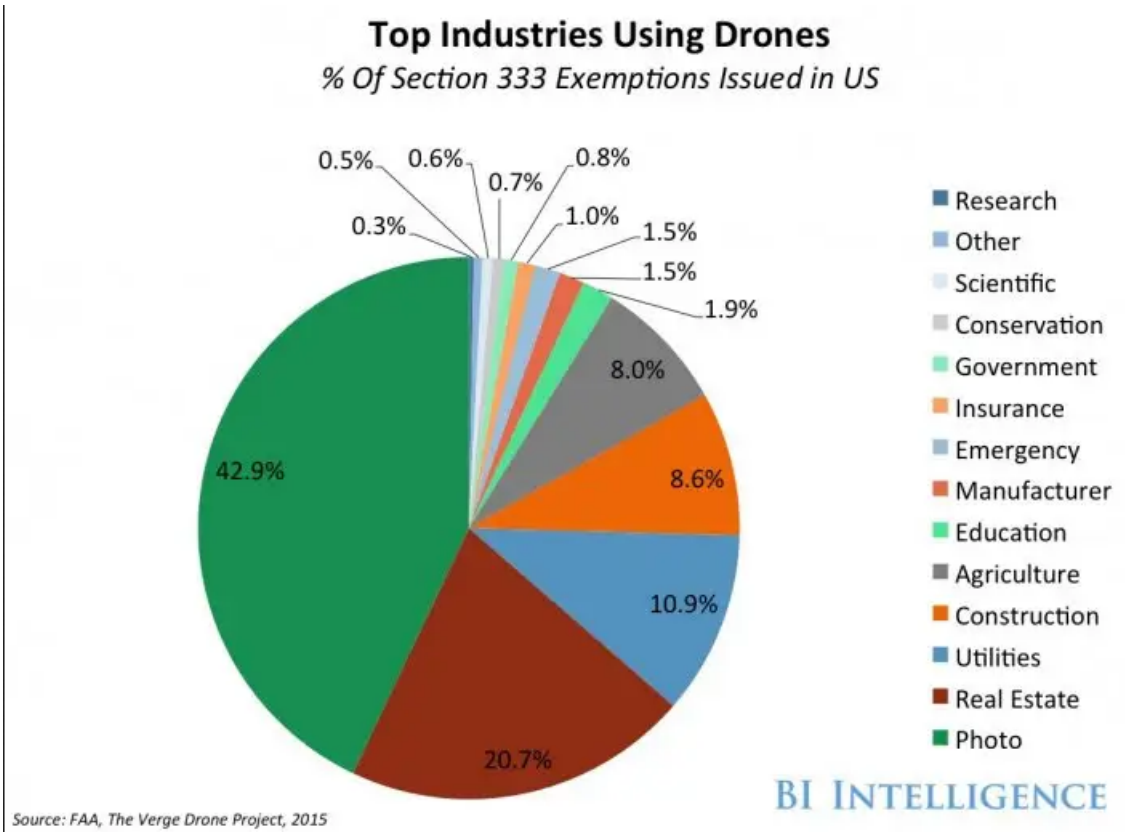


Figura 1: 2015 - Uso de drones en la industria

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

Como podemos ver, el mayor uso de drones (42.9 %) se da en la industria

de la fotografía como hemos hablado antes. En segunta 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 la mitad de uso respecto al segundo puesto (10.9 %). En el cuarto y quinto puesto aparecen los sectores de construcción y agricultura respectivamente, siendo el uso en el sector de construcción mínimamente superior (0.6 % mayor). El uso en el resto de sectores es mínimo lo cual es una pena ya que entre ellos se encuentran investigación y ciencia (0.3 % y 0.6 % respectivamente), sectores que impulsarían y mejorarían esta tecnología con mayor rapidez.

1.3. Motivación

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

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

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

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

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

2. Entorno Operacional

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

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

2.1. Ubuntu 16.04

El proyecto se ha desarrollado en esta distribución de Linux por dos decisivos motivos. El primero es que ROS ofrece una de sus versiones más robustas para esta distribución: ROS Kinetic. Esta versión de ROS tiene una gran comunidad de soporte y por tanto podemos encontrar gran cantidad de paquetes, bibliotecas y ayuda para casi cualquier problema que tengamos. El segundo motivo ha sido el que realmente 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.

2.2. Robotic Operating System

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

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

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

2.2.1. Conceptos

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

- **Paquetes:** Son la unidad básica de organización de ROS. Un paquete puede contener nodos, una biblioteca independiente, un conjunto de datos, archivos de configuración, un software de terceros o cualquier otra cosa que constituya un módulo útil. El objetivo de estos paquetes es proporcionar esta funcionalidad útil de una manera fácil de consumir para que el software pueda reutilizarse fácilmente. La definición de un paquete de ROS perfecto es aquel que aporta suficiente funcionalidad para resultar útil pero que no pesa demasiado para resultar difícil de usar en otro proyecto.

- **Nodos:** Son los procesos de ROS que realizan computación. Los nodos pueden comunicarse entre sí por medio de *stream topics* o tópicos de transmisión, servidores RPC (Remote Procedural Call) o servidores de parámetros.

- **Temas(Tópics):** Los *tópics* son buses etiquetados sobre los que los nodos pueden escribir y leer mensajes. Las publicaciones y suscripciones en los buffers son anónimas de manera que un nodo con el rol de *"publisher"* no sabe con qué otro nodo se está comunicando. Puede haber varios suscriptores y publicadores en un mismo buffer pero la comunicación siempre es unilateral. 2

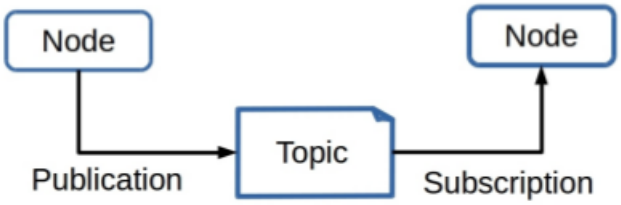


Figura 2: Esquema de flujo de información mediante Tópics en ROS

- **Servidores RPC:** El paradigma de los *tópics* es muy flexible pero la comunicación unidireccional puede llegar a ser un problema. Para las comunicaciones bidireccionales utilizamos los servidores RPC en los que un nodo proveedor ofrece un servicio al que un nodo cliente llama enviando un mensaje y esperando una respuesta.

- **Servidores de Parámetros:** Un servidor de parámetros es un diccionario compartido al que se puede acceder a través de las API de red. Los nodos usan este servidor para almacenar y recuperar parámetros

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

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

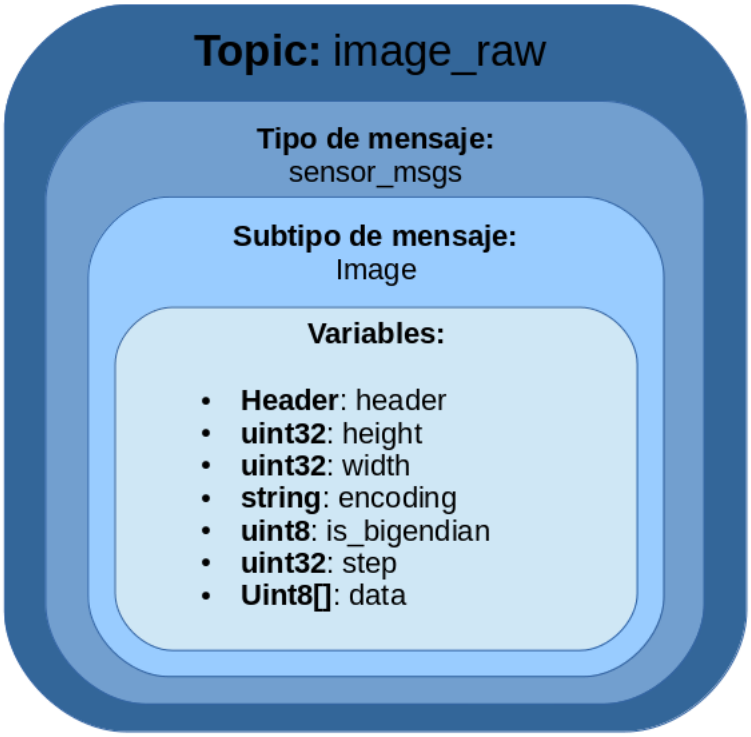


Figura 3: Estructura de mensaje Image

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

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

Figura 4: Estructura de mensaje Twist

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

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

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

2.3. Driver *bebop_autonomy*

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

Al ejecutar el driver bebop_autonomy se crean una serie de Topics a los que nos podemos subscribir 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:

```
$ roslaunch bebop_driver bebop_nodelet.launch
```

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

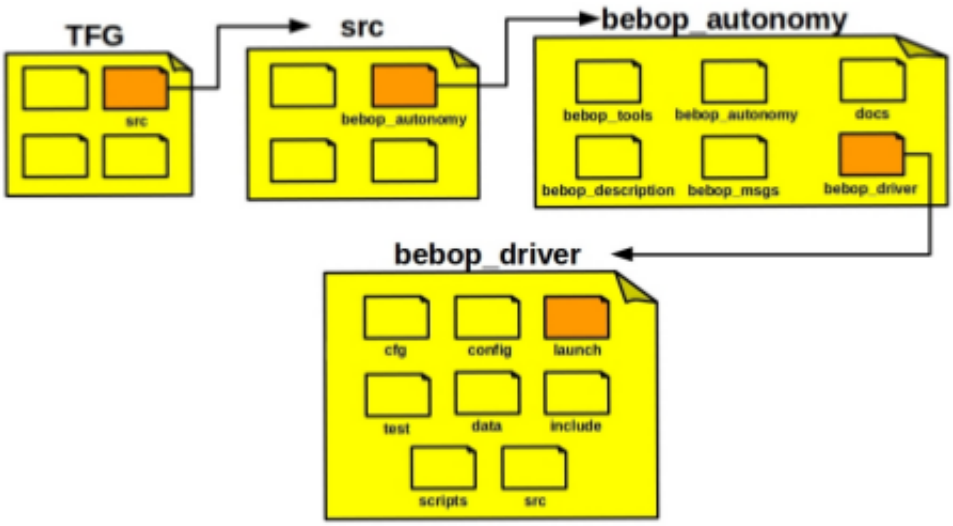


Figura 6: Organización de directorios de bebop_autonomy

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

```
$ roslaunch bebop_tools bebop_nodelet_iv.launch
```

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

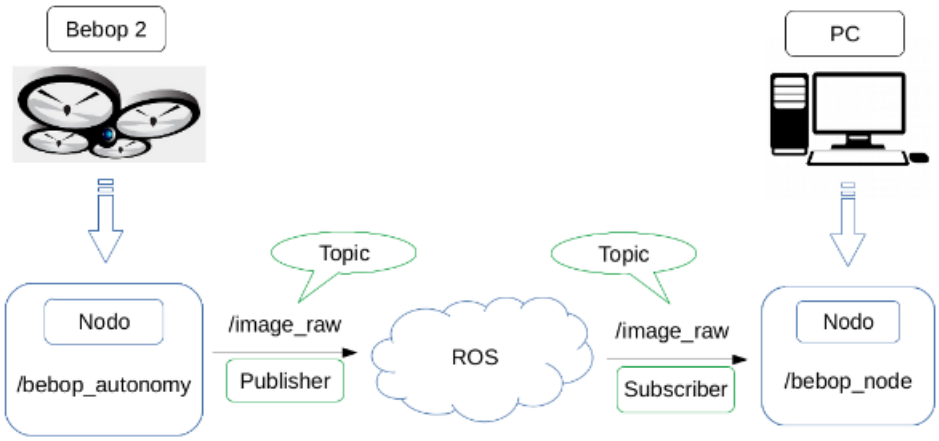


Figura 7: Interfaz de conexión Bebop2 - ROS - PC para la obtención de imagen del dron.

2.3.1. Comandos ROS

- **roscore:** Inicializa el núcleo de ROS y permite la comunicación entre nodos, servicios, parámetros, etc.
- **roscreate-pkg**: Con esta herramienta podemos crear paquetes ROS, pudiendo además especificar las dependencias de los mismos.
- **rospack:** Muestra información de un paquete.
- **rosstack:** Información de la "pila" que se 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:** ObtSiene la información de un tipo de mensaje.
- **rosrun:** Ejecuta un archivo ejecutable de un paquete específico.
- **roslaunch:** ROS permite ejecutar varios nodos a la vez. Mediante esta herramienta se puede ejecutar archivos con formato .launch, en estos archivos se puede configurar varios nodos para poder ejecutarlos a la vez facilitando al usuario el uso del sistema global de ROS.

2.4. Bebop 2

Figura 8: Parrot Bebop 2

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

2.4.1. Especificaciones

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

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

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

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

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

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

- **Hardware y características físicas:**

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

2.5. Ordenador

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

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

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

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

3. Gestión del Proyecto

3.1. Metodología de Desarrollo

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

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

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

3.2. Planificación

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

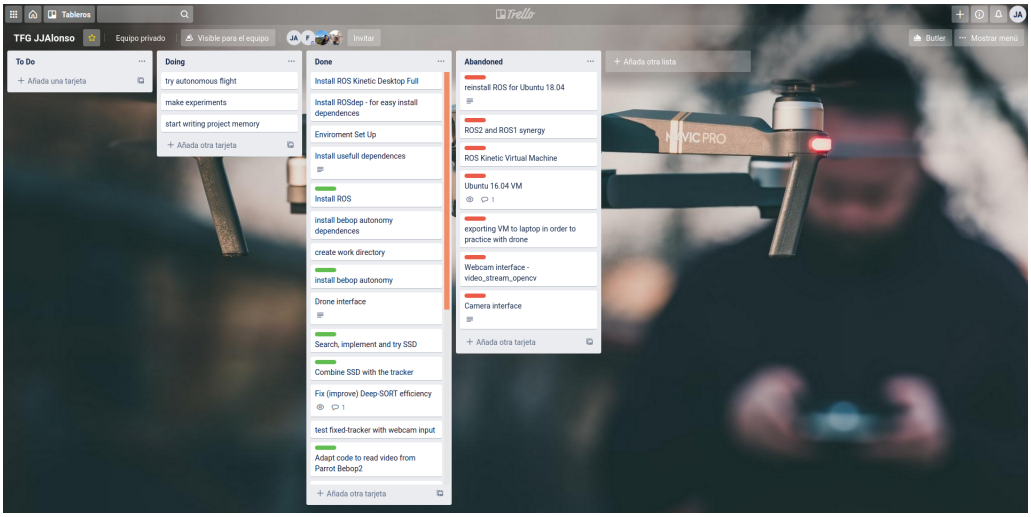


Figura 9: Captura del tablero de Trello

3.3. Restricciones del Sistema

3.3.1. Restricciones Harware

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

3.3.2. Restricciones Software

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

3.4. Hitos

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

1 Instalar Ubuntu 16.04

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

2 Instalar ROS Kinetic

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

3 Instalar Bebop Autonomy

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

4 Escoger algoritmo de tracking de personas

Buscar un algoritmo de tracking que funcione en nuestra instalación.

5 Crear una interfaz para leer imágenes del dron

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

6 Desarrollar un control manual para el dron

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

7 Desarrollar un control autónomo para el dron

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

8 Implementar paquete ROS con el proyecto

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

9 Crear launchfile para ejecutar el proyecto desde ROS

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

Estos objetivos podemos sintetizarlos en dos grandes grupos:

- Conseguir un sistema estable en el que llevar a cabo el desarrollo

  Objetivos: 1,2,3

- Establecer una conexión dron-servidor, y programar el comportamiento del dron

  Objetivos: 4,5,6,7,8,9

3.5. Problemas

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

3.5.1. Virtualización

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

3.5.2. Equipo de Trabajo

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

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

El gran problema, como dije antes, es que carece de una tarjeta gráfica que realice los cómputos de detección y tracking. Ejecutando el proyecto final en este ordenador hemos llegado a obtener picos de 0.6fps pero la media ronda los 0.5fps. Claramente esto es insostenible y es imposible realizar

pruebas y experimentos si recibimos 1 frame cada 2 segundos. La reactividad del dron es inexistente.

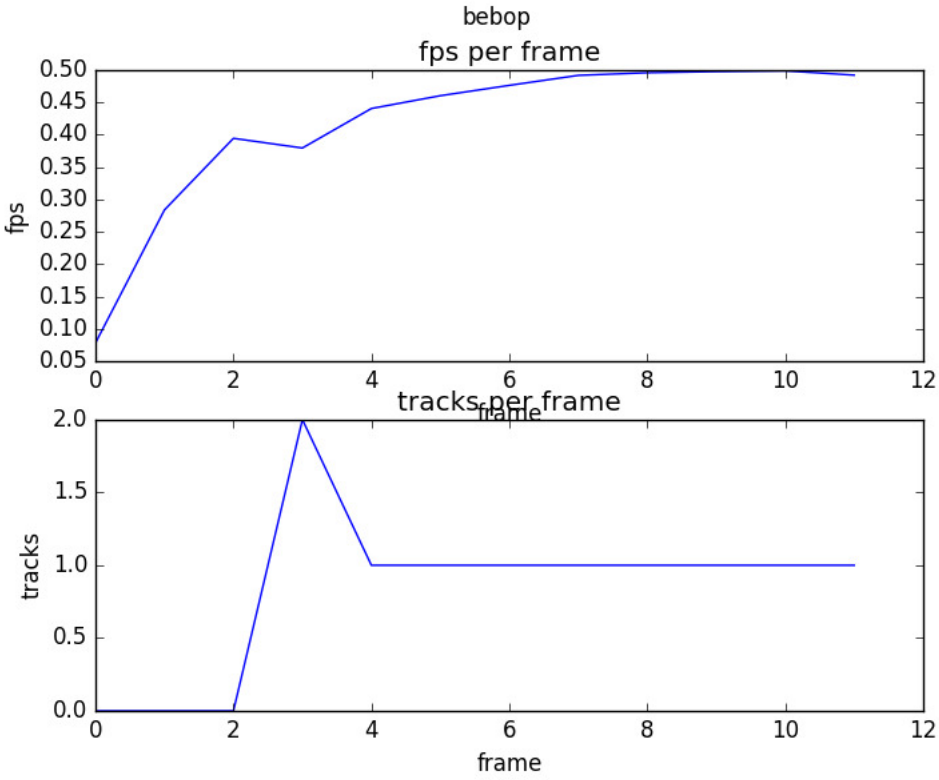


Figura 10: La imagen muestra la evolución de la cantidad de FPS y tracks detectados a lo largo de los frames obtenidos. Como vemos, los FPS se estabilizan en 0.5. Analizaremos esta gráfica en mayor profundidad más adelante.

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

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

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

3.5.3. Límite de Distancia de Conexión

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

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

3.5.4. Python y DeepSORT

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

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

3.5.5. Instalación de ROS2 junto a ROS Kinetic

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

Por esto tratamos de construir el proyecto sobre un entorno que nos 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.

3.6. Presupuesto

3.6.1. Recursos Hardware

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

$$(D/V) * C$$

Donde:

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

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

Cuadro 1: Tabla de costes harware.

3.6.2. Recursos Software

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

Cuadro 2: Tabla de costes software

3.6.3. Recursos Humanos

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

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

3.6.4. Coste Total

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

Cuadro 3: Tabla de costes totales.

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

PRECIO TOTAL: 5608.13€

4. Diseño

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

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

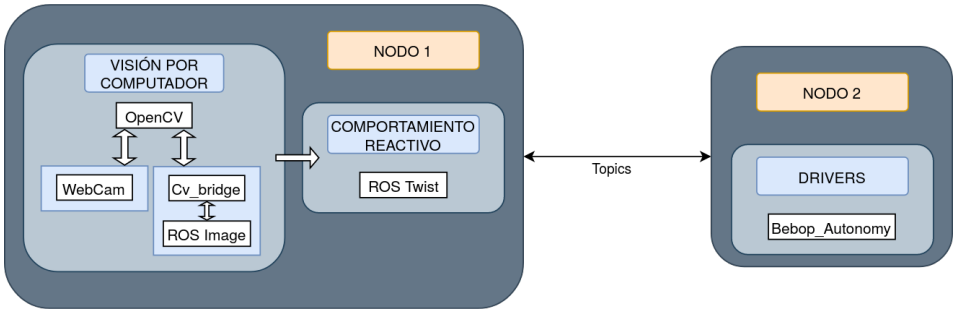


Figura 11: Esquema de nodos ROS en la aplicación

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

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

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

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

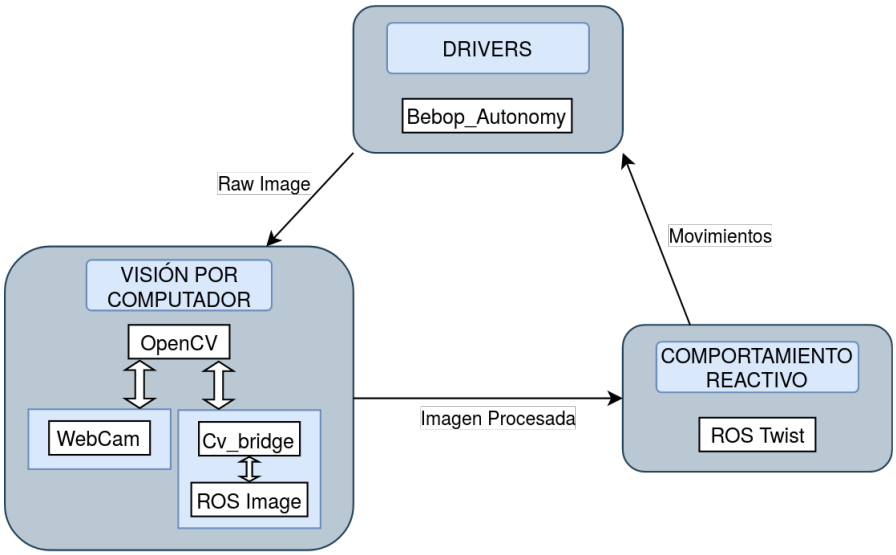


Figura 12: Esquema de comunicación entre los módulos de la aplicación

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

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

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

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

5. Implementación

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

5.1. Visión por Computador

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

5.1.1. Captación de imágenes del dron

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

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

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

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

De esta manera la imagen *img* se actualiza cada vez que hay una nueva publicación en el topic por parte del dron. Si 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.

5.1.2. Captación de imágenes de la webcam

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

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

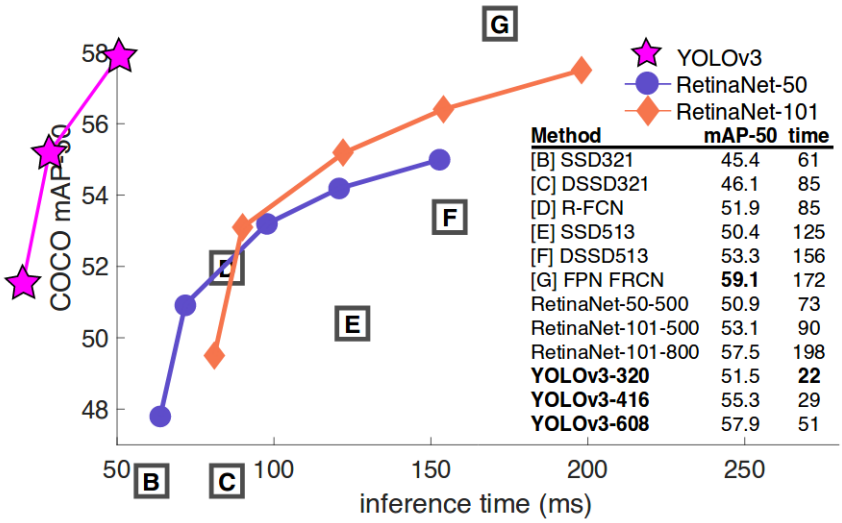
5.1.3. Lectura de archivo de vídeo almacenado

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

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

5.1.4. Detección - YOLOv3

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



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

Figura 13: Rendimiento de YOLOv3 frente a otros algoritmos de detección. Paper: [1]

Mediante YOLO obtenemos los Bounding Boxes de los objetos (personas) detectados junto con la probabilidad de que el objeto detectado pertenezca a X clase; en nuestro caso, a la clase *persona*.

Referencias

[1] Joseph Redmon - YOLOv3 - Paper https://pjreddie.com/media/files/papers/YOLOv3.pdf.

[2] CB-Insights https://www.cbinsights.com/research/drone-impact-society-uav/.

[3] Divya Joshi - Business Insider https://www.businessinsider.com/drone-technology-uses-applications?IR=T.

[4] Federal Aviation Administration. https://www.faa.gov/.

[5] ROS. https://www.ros.org/.

[6] ROS Wiki. http://wiki.ros.org/ROS/Introduction.

[7] Mani Monajjemi y Colaboradores. http://bebop-autonomy.readthedocs.io/en/latest/contribute.html#sec-contribs.

[8] Bebop2. https://www.parrot.com/es/drones/parrot-bebop-2.

[9] Joseph Redmon - YOLOv3. https://pjreddie.com/darknet/yolo/.

[10] Modelo de Prototipos. https://es.wikipedia.org/wiki/Modelo_de_prototipos.

[11] Trello. https://trello.com/es.

[12] Slack. https://slack.com/intl/es-es/.

[13] Repositorio del proyecto - TFG-Bebop-YOLO. https://github.com/JJavier98/TFG-Bebop-YOLO.

[14] VirtualBox. https://www.virtualbox.org/.

[15] NVIDIA CUDA. https://developer.nvidia.com/cuda-zone.

[16] CITIC. http://citic.ugr.es/.

[17] DeepSORT. https://nanonets.com/blog/object-tracking-deepsort/.

[18] Qidian213 - DeepSORT. https://github.com/Qidian213/deep_sort_yolov3.

[19] DeepSORT videocaptureasync module. https://github.com/Qidian213/deep_sort_yolov3/issues/154.

*