TRABAJO FIN DE GRADO

INGENIERÍA EN INFORMÁTICA

Seguimiento Activo de Personas con dron Parrot Bebop2

Integración en ROS

Autor
José Javier Alonso Ramos

Directores
Francisco 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 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.

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 an active agent with an onboard camera. The entire proyect 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 funcionalities 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 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 15 de julio de 2020.

Índice

Índice de figuras

Índice de cuadros

1. Implementación

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

1.1. 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:

1.1.1. Detector - YOLOv3

Como algoritmo de detección se utiliza la red neuronal convolucional (CNN) YOLOv3, un sistema de detección en tiempo real de última generación, que aporta gran precisión y rapidez a la hora de detectar y clasificar objetos. No es el detector más preciso que existe pero sí uno de los que mejor relación $\frac{precision}{tiempo\_ejecucion}$ tienen. Por esto mismo, por proporcionar una precisión en la detección decente en un tiempo suficientemente pequeño como para poder ejecutar el algoritmo en tiempo real, se utiliza YOLOv3 en la elaboración del proyecto.

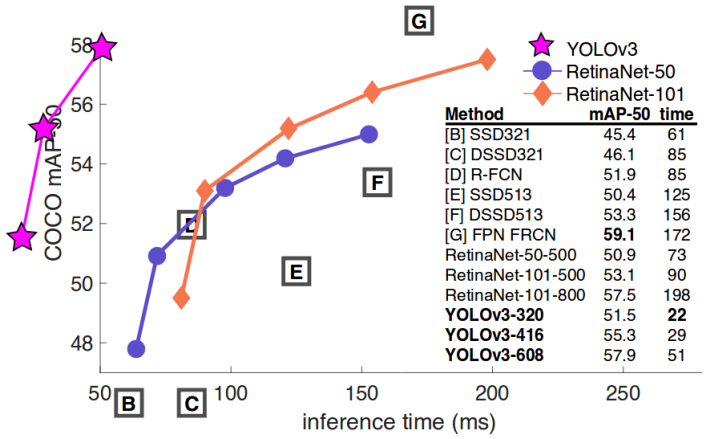

Figura 1: Rendimiento de YOLOv3 frente a otros algoritmos de detección

Paper: [1]

Funcionamiento

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*). Las capas de normalización transforman todos los valores resultantes de la convolución al dominio [0,1] para que sea más fácil y eficiente trabajar con ellos mientras que, la capa de activación o rectificador, decide si la salida de una determinada neurona es relevante para la ejecución de la red.

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

Darknet-53 model

Figura 3: Estructura de Darknet-53

Extraída del paper de YOLO: [1]

El algoritmo acepta cualquier resolución de imagen pero, en caso que queramos procesar varias imágenes a la vez (por *batchs*), sí será necesario introducir todas las imágenes a la misma resolución para que la GPU pueda realizar los cálculos paralelamente. Como nosotros procesamos una secuencia de vídeo y pretendemos que sea a tiempo real, no da lugar a realizar este procesamiento de manera concurrente. Aún así, al proceder todas las imágenes de la misma fuente de vídeo, tendrán todas la misma resolución.

La red reescala la imagen tres veces por un factor llamado *paso de la red*. En nuestro, este paso de 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 para la reescalado con paso igual a 32. En esta imagen resultante es sobre la que se realizan las detecciones.

La imagen resultante tras el reescalado (*downsampling*) se recorre con distintos tamaños de máscaras o recuadros llamados *Anchor Boxes* para tratar de identificar las distintas figuras que aparezcan en escena. Estos *anchor boxes* 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 un *anchor box* rectangular más ancho que alto para lograr detectar estas figuras. Para las personas, por ejemplo, el *anchor box* será más alto que ancho para detectar a personas de pie. En nuestro caso, contamos con un total de 9 combinaciones diferentes de máscaras para detectar las 80 clases diferentes que encontramos en el conjunto de datos COCO, que es el conjunto con el que se ha entrenado por defecto a YOLOv3.

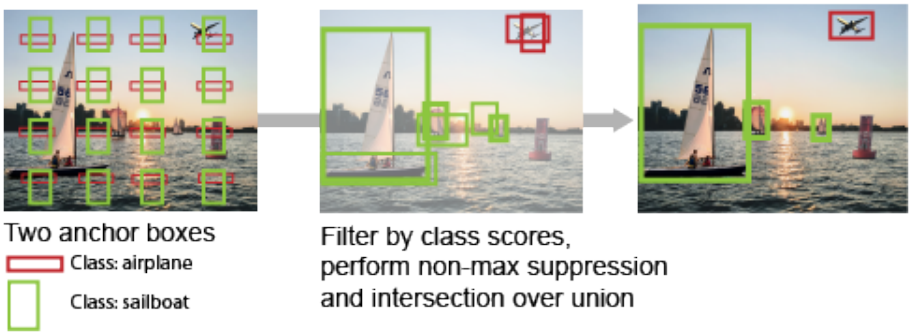

Figura 5: Recorrido de la imagen con dos tipos de *anchor box*

Extraída de [2]

Cada *anchor box* se aplica a cada célula de la imagen (si una imagen es 416 x 416 y el paso de la red es 32, obtenemos 13 x 13 células) obteniendo resultados de detección para cada una. Para mostrarlo más claramente, si disponemos de 9 *anchor boxes* 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 reescalada junto con la precisión de detección y las coordenadas de los *Bounding Boxes* calculados con cada *Anchor Box* para cada célula de imagen, además de un vector *booleano* que indica la clase a la que pertenece la detección: (13,13,9,85). Imagen 13x13, 9 *anchor boxes*, 80= 5(confianza de la detección, coordenadas X e Y del

Bounding Box y su ancho y su alto) + (80 clases de COCO).

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 lo llamamos supresión de no máximos. Descartamos todas aquellas detecciones que no superen un umbral de confianza definido arbitrariamente. Aún así, podemos obtener varios *Bounding Boxes* solapándose alrededor de una figura y debemos decantarnos por uno de ellos para que sea el delimitador de la figura. Para ello utilizamos IoU (Intersección sobre Unión), es decir, cuál de todos los *Bounding Boxes* presenta una mayor proporción al dividir el área de su intersección con los demás entre el área de la unión de todos los *Bounding Boxes*.

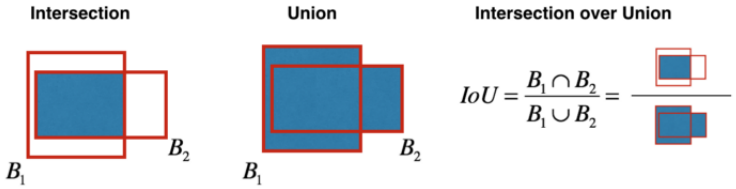

Figura 7: Intersección sobre Unión

Extraída de [2]

Figura 9: Intersección sobre Unión

Extraída de [2]

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.

1.1.2. Tracker - DeepSORT

Utilizamos DeepSORT [3] como algoritmo de tracking por ser puntero en la tarea que ocupa. Se trata de una actualización, una mejora del algoritmo SORT (Simple Online and Real-time Tracker) en la que aplicamos *deep learning*. Para realizar la asignación de trazados o *tracks* a cada detección y hacerlos perdurar en el tiempo, o lo que es lo mismo, realizar un seguimiento de estos, hace uso de distintas técnicas. 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 posibilidad de obtener errores en el proceso.

  En nuestro, caso debemos tener en cuenta que nuestro detector no es 100 % preciso y, en el hipotético caso de que lo fuese, también debemos tener en cuenta el posible ruido en la imagen o pérdidas de fotogramas que nos hagan fallar en la detección de alguna figura. Al realizar estas detecciones, asumimos 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 encontramos los errores:

  Podemos obtener ruido asociado a la velocidad de una figura ya que rara vez se mantendrá constante. Llamaremos a esto *Ruido de Proceso*.

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

- **Problema de asignación** Una vez tenemos 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 [4] al que proporcioneamos información de movimiento de la figura y su apariencia.

  - **Distancia de Malhalanobis. 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 Malhalanobis entre las predicciones de los filtros de Kalman y las nuevas detecciones.

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

    Denotamos la proyección de la distribución del *track* i-ésimo en el espacio de medición por (y_i, S_i) y el j-ésimo *Bounding Box* detectado por d_j.

Además podemos descartar asociaciones improbables estableciendo un umbral en la distancia obtenida.

- **Distancia de coseno entre espacio de apariencias. Actualización a SORT.** Este es el cambio más interesante realizado a SORT y por que este algoritmo recibe el "prefijo" *Deep*. Se calucla 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 Malhalanobis y $d^{(2)}$ es la distancia del coseno, pudiendo dar más peso a una o a otra cambiando el valor λ. La importancia de la incorporación del descriptor de apariencias es tan alta que los autores del algoritmo afirman en su paper [5] que son capaces de alcanzar resultados propios del estado del arte dando un valor $\lambda = 0$.

Referencias

[1] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018. https://pjreddie.com/media/files/papers/YOLOv3.pdf.

[2] Anchor Box. https://es.mathworks.com/help/vision/ug/anchor-boxes-for-object-detection.html.

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

[4] Hungarian Algorithm. https://brilliant.org/wiki/hungarian-matching/.

[5] DeepSORT Paper. https://arxiv.org/pdf/1703.07402.pdf.

*