

# Polígonos Simples Coloreados como Instrucciones de Vuelo

1<sup>st</sup> Tomás Oelckers Abogabir

Facultad de Ingeniería y Ciencias Aplicadas  
Universidad de los Andes  
Santiago, Chile  
toelckers@miuandes.cl

2<sup>nd</sup> Tarik Sáez Riadi

Facultad de Ingeniería y Ciencias Aplicadas  
Universidad de los Andes  
Santiago, Chile  
tsaez@miuandes.cl

**Abstract**—En este documento se explica cómo detectar polígonos simples de color con un Dron DJI Tello y OpenCV para usarlas como instrucciones de vuelo. Se comprende el funcionamiento del dron, del algoritmo, las pruebas y los resultados del proyecto.

## I. INTRODUCCIÓN

Los humanos tomamos decisiones naturalmente en base a estímulos visuales, y es porque tenemos la capacidad de enfocarnos y entender diferentes partes de una escena (una imagen o una secuencia de ellas) en tiempo real. Sin embargo, implementar esta forma de tomar decisiones en una máquina autónoma es una tarea difícil y compleja. Extendiéndonos al área de la aviación, los pilotos están entrenados para poder volar con VFR (*Visual Flight Rules*) o IFR (*Instrument Flight Rules*). Por defecto, un dron común no controlado por un humano vuela con IFR, pues toma decisiones basados en la información que le entregan sus sensores; sin embargo, por defecto no es capaz de tomar decisiones en base a información visual.

Existen algunos sistemas autónomos, incluyendo drones, como las IFM (*Intelligent Flying Machines*) de Nvidia, que son capaces de hacer VFR, pero utilizan una tarjeta de desarrollo (Jetson) la cual es muy costosa, y a veces son asistidas por procesamiento con redes neuronales.

Este proyecto consistió en el diseño e implementación de un algoritmo rápido y barato que le permite a un dron DJI Tello tomar decisiones de vuelo a través de instrucciones basadas en las imágenes que capta la cámara del dron, usando la librería OpenCV de Python para procesar las imágenes en tiempo real.

Este documento describe el dron utilizado, el algoritmo diseñado, las restricciones del algoritmo y las pruebas y resultados obtenidos usando este algoritmo.

## II. DRON DJI TELLO

DJI Tello es un mini dron con cámara frontal fácil de maniobrar desde un smartphone por conexión wifi. Cuenta con las siguientes especificaciones:

### A. Especificaciones

#### 1) Dron:

- Peso: Aproximadamente 80 gramos (Incluyendo hélices y batería).
- Dimensiones: 98x92.5x41 mm.
- Funciones Integradas: Barómetro, LED, Sistema de Visión, 2.4 GHz 802.11n Wi-Fi, 720p cámara en vivo.
- Puerto: Puerto de Carga Micro USB.

#### 2) Performance de Vuelo:

- Distancia Máxima de Vuelo: 100 mts.
- Velocidad Máxima: 8 m/s.
- Tiempo Máximo de Vuelo: 13 minutos.
- Altura Máxima de Vuelo: 30 mts.

#### 3) Batería:

- Batería Desmontable: 1.1Ah/3.8V.
- Cámara:
- Foto: 5MP (2592x1936).
- FOV: 82.6°.
- Video: HD 720p 30 FPS.
- Formato: JPG(Foto); MP4(Video).
- EIS: Si.

### B. Control del Dron

DJI Tello viene con SDK incluido que permite desarrollar programas en cualquier lenguaje de programación que pueda conectarse al dron por medio de una conexión Wi-Fi UDP. Esto funciona enviando mensajes con ciertos comandos establecidos por el SDK en forma de texto a través de la conexión Wi-Fi UDP. Los principales comando usados para el proyecto permiten hacer streaming de video, despegar, aterrizar, girar en sentido horario y anti-horario, subir y bajar, ir hacia delante, hacia atrás, hacia la derecha y hacia la izquierda.

Para este proyecto el lenguaje de programación usado es python3 con la librería djitellopy 1.5 [1] desarrollada por Damia Fuentes. Esta librería permite, por medio de la clase socket, conectarse con el dron por la IP: 192.168.10.1 y puerto UDP 8889. Esta conexión permite enviar comandos SDK [4] y recibir respuestas. también se conecta al puerto 11111 por medio de la función VideoCapture de open-cv usando un url,

"`udp://@192.168.10.1:11111`", que recibe el stream de video de la cámara.

### III. SEÑALES UTILIZADAS

Para darle instrucciones al dron, se decidió utilizar polígonos simples: triángulos, rectángulos o círculos con cuatro posibles colores: rojo, azul, verde o amarillo. Esto da un total de 12 instrucciones posibles de vuelo. Considerando el rango de instrucciones necesarias para que el dron recorra un circuito, se consideró que 12 instrucciones bastan.

### IV. ALGORITMO

#### A. Detección de Colores

Para hacer la detección de colores se utilizó el espacio de colores HSV (Hue, Saturation & Value) predefinido en la librería *OpenCV*, con el cual se definió un espectro de detección para cada uno de los colores. Hue (Matiz) varía en un rango entre  $0^\circ$  y  $180^\circ$  mientras que Saturation (Saturación) y Value (Valor) varían en un rango entre 0 y 255. En la figura a continuación se muestra una vista amplia del espacio de color que se puede definir en HSV, mostrando el significado de cada parámetro en el espacio de color.

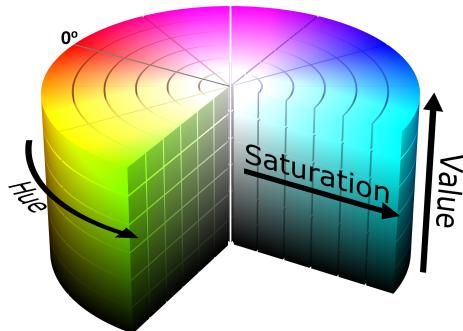


Fig. 1. Espacio de color HSV [2].

Una característica importante de destacar, es que el color rojo queda dividido en dos partes diferentes del Hue: Antes de los  $180^\circ$  y después de los  $0^\circ$ .

Los rangos de color del algoritmo fueron definidos por prueba y error:

	H	S	V
Rojo 1	[160 - 179]	[130 - 255]	[50 - 255]
Rojo 2	[0 - 10]	[170 - 255]	[70 - 255]
Verde	[35 - 90]	[86 - 255]	[37 - 194]
Azul	[91 - 155]	[100 - 255]	[76 - 255]
Amarillo	[19 - 30]	[150 - 255]	[100 - 255]

TABLA I

RANGO DE VALORES HSV PARA CADA COLOR DEFINIDO.

Los rangos de color son una de las partes vitales del algoritmo, ya que marcan la diferencia entre los objetos deseados y no deseados de detectar; adicionalmente, un color puede aparecer ser diferente bajo diferentes fuentes y exposiciones de luz. El ejemplo más claro y recurrente durante el desarrollo

del algoritmo fue evitar la detección del color piel, que bajo ciertas luces y parámetros de HSV, podría ser considerada como rojo por el algoritmo, si no se restringe bien el rango de color. Esto puede conllevar a que el dron confunda las caras o manos de personas dentro del cuadro como instrucciones y falle durante su misión.

#### B. Filtros

Se usaron tres filtros para la imagen: Un filtro Gaussiano con un kernel de  $15 \times 15$  [6], erosión y dilatación; los últimos dos con un kernel de  $1 \times 1$  [5].

#### C. Restricción de Tamaño para la Detección

Naturalmente existen muchos objetos de diversas formas, colores y tamaños que podrían ser fácilmente confundidos con las figuras que se quieren detectar. Es por esta razón que fue necesario restringir el tamaño de los objetos detectados, para evitar que el dron tomara decisiones con figuras no deseadas. Se restringió el área de la detección, es decir, si la cantidad de píxeles que se cuentan como parte de una figura suman más de un cierto límite, se tomará en cuenta la figura. De esta forma, no sólo se evitan conflictos en la detección en el fondo de la imagen, sino que también anomalías producidas por las mismas sombras y luces que podrían formarse dentro de la figura que se quiere detectar.

#### D. Detección de Figuras

La detección de figuras se hizo mediante la función de *OpenCV FindContours*, que funciona encontrando la mejor curva continua para un mismo color. Luego, esta curva es aproximada con la función *approxPolyDP* con un parámetro de *precisión* especificado, que en términos simples determina la cantidad de lados que se puede encontrar en una figura; una precisión muy baja resultará en pocos lados detectados mientras que una precisión alta detecta muchos lados en una figura. Por esta razón, fue necesario encontrar un parámetro suficientemente bueno para distinguir entre las tres figuras deseadas para que no hubiesen conflictos ni errores en la detección [3].

Para detectar cuadrados, se utilizó la función *boundingRect*, la cual entrega cuatro parámetros; dos de ellos sirven para calcular la relación de aspecto. Se determinó que si la relación de aspecto está entre 0.95 y 1.05, es decir, en torno a 1, se encontrará un cuadrado. En otras palabras, si la relación de aspecto está en un valor alrededor del 1, significa que los cuatro lados del cuadrilátero detectado son más o menos iguales entre sí, lo que significa que se detectó un cuadrado.

Para detectar círculos se utilizó la función *Hough Circles* que trabaja sobre la máscara de cada color para encontrar círculos confinados a un cierto rango de tamaños, según parámetros de búsqueda del centro del círculo dentro de la imagen y un filtro *Canny Edge Detector* interno de la misma función. Como la máscara ya tiene una restricción de tamaño (ver sección C), el rango de radios se dejó en un rango

relativamente alto, para no generar falsos negativos.

Tanto *FindContours* como *boundingRect* y *Hough Circles* funcionan para una sola máscara de color a la vez, por lo que fue necesario diseñar el algoritmo para que repitiera este proceso, incluyendo los filtros de la sección B., para detectar figuras de cada uno de los colores definidos en la sección A.

#### E. Ejecución de Instrucciones

Para ejecutar una instrucción el dron debe reconocer una figura (una forma con un color), esto procede de la siguiente forma. El dron capta 30 cuadros por segundo; en cada uno de esos cuadros el algoritmo de reconocimiento detecta todas las figuras que se encuentren presente en los últimos 10 cuadros y toma la figura que mas se repita como la detectada, y según esta figura, procederá a ejecutar la instrucción. Durante la ejecución de la instrucción, el dron suspende la detección hasta que termine de ejecutarla y luego vuelve a detectar.

#### V. PRUEBAS Y RESULTADOS

Se realizaron varias pruebas con distintas iluminaciones y obstáculos (por ejemplo mostrar piel o cubrir parcialmente una figura) mediante las cuales se ajustaron iterativamente los rangos de color y los parámetros de los filtros para minimizar la detección de errores (principalmente falsos positivos).

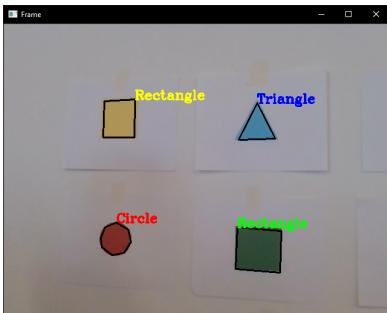


Fig. 2. Captura de pantalla durante una de las pruebas con la webcam.

Posteriormente se procedió a utilizar la cámara del dron para tomar las imágenes de prueba. Se realizó una prueba para comparar los tamaños de detección y minimizar falsos positivos en la misma.

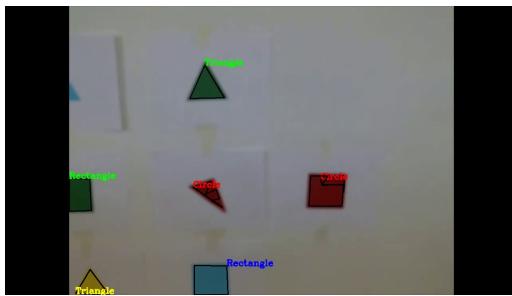


Fig. 3. Prueba de detección con poca restricción de área.

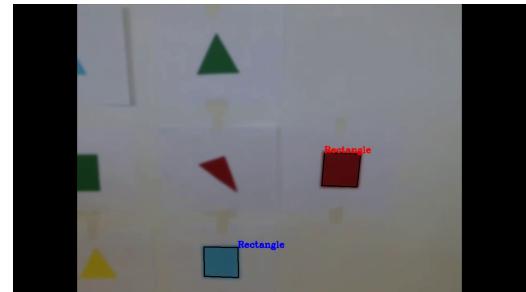


Fig. 4. Prueba de detección con mucha restricción de área.

Como se puede ver en las figuras 3 y 4, cuando se restringe mucho el área, no se detectan figuras que están a la misma distancia que otras. Por otro lado, cuando se restringe poco, se detectan anomalías dentro de las mismas figuras que producen falsos positivos.

Un ejemplo de una de las pruebas realizadas que demostró el buen funcionamiento del algoritmo deja en evidencia que el algoritmo no se confunde con objetos lejanos, ni con piel humana.



Fig. 5. Captura de una de las pruebas utilizando la cámara del dron y la última versión del algoritmo.

#### VI. DESVENTAJAS

Durante el desarrollo de este proyecto, quedó en evidencia que el algoritmo no es infalible, y está directamente relacionado con la detección de formas y colores, en parte debido a la inestabilidad de la imagen, una consecuencia directa de mantenerla en vuelo.

La detección de formas está restringida a poder detectar triángulos, rectángulos y círculos, siendo éste último el más delicado de todos. En estricto rigor computacional, un círculo es un polígono con muchos lados; en particular para el algoritmo desarrollado, seis o más lados. En el caso de querer detectar una mayor variedad de polígonos, la precisión de las funciones explicadas en la sección IV.D. debería ser modificada para minimizar los errores entre figuras. Basados en las pruebas y resultados obtenidos, se decidió no incluir

polígonos de 5 o más lados para mantener una tasa de error baja en la detección.

Cabe mencionar dentro de la detección de figuras que es necesario evitar los movimientos del dron al máximo mientras esté en modo de detección, para evitar que se produzcan artefactos visuales durante la transmisión de imágenes o que se distorsionen las figuras comando.

Por otro lado, la detección de colores está restringida por el espectro definido en la Tabla I. Como consecuencia, si las figuras deseadas reciben mucha o poca luz, el algoritmo podría no detectar los colores en estas condiciones y fallar en la detección, puesto que una imagen saturada (mucha luz) en estricto rigor altera los colores percibidos.

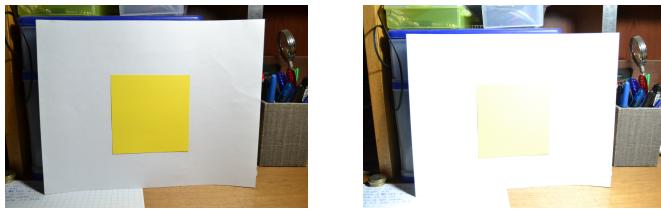


Fig. 6. Imagen con buena iluminación (izquierda) e imagen saturada (derecha).

## VII. CONCLUSIONES

Podemos concluir que es posible desarrollar un programa que permite detectar polígonos coloreados a través de la cámara de un dron y procesadas en tiempo real por un computador conectado a este vía Wi-Fi. Aun así, este programa tiene falencias tales como:

- Falta de precisión de detección de figuras de 6 o más aristas.
- Errores en la detección de colores por exceso o falta de luminosidad.

Estas falencias son debidas a la simpleza del algoritmo que buscaba resolver un problema de manera rápida y sencilla a bajo costo de procesamiento. El próximo desafío, a este proyecto, apuntaría a corregir estas imprecisiones con un red neuronal entrenada en distintos escenarios ambientales.

## REFERENCIAS

- [1] “DJI Tello Py”, *Python Software Foundation*, [Online]. Disponible: <https://pypi.org/project/djitetellopy/>. [Accedido: 20-06-2019].
- [2] “HSL and HSV”, *Wikipedia*, [Online]. Disponible: [https://en.wikipedia.org/wiki/HSL\\_and\\_HSV](https://en.wikipedia.org/wiki/HSL_and_HSV). [Accedido: 20-06-2019].
- [3] “Structural Analysis and Shape Descriptors”, *docs OpenCV*, [Online]. Disponible: [https://docs.opencv.org/2.4/modules/imgproc/doc/structural\\_analysis\\_and\\_shape\\_descriptors.html](https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html). [Accedido: 20-06-2019].
- [4] Ryze Robotics, “Tello SDK”. <https://www.ryzerobotics.com/tello/downloads>
- [5] “Morphological Transformations”, *docs OpenCV*, [Online]. Disponible: [https://docs.opencv.org/3.0-beta/doc/py\\_tutorials/py\\_imgproc/py\\_morphological\\_ops/py\\_morphological\\_ops.html](https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html)
- [6] “Smoothing Images”, *docs OpenCV*, [Online]. Disponible: [https://docs.opencv.org/3.1.0/d4/d13/tutorial\\_py\\_filtering.html](https://docs.opencv.org/3.1.0/d4/d13/tutorial_py_filtering.html)