SEP
14
2019

# Detección de colores en OpenCV – Python (En 4 pasos)

*Por* ADMINISTRADOR

**CONTENIDO**

- HSV (Hue, Saturation, Value) en OpenCV
- Detección de colores con HSV en OpenCV
    - Paso 1: Imagen a procesar
    - Paso 2: Transformar de BGR a HSV
    - Paso 3: Determinar los rangos en donde se encuentra el color a detectar
    - Paso 4: Visualización
- Si has pensado en usar la detección de colores, ten en cuenta que..

El tema de este post es la detección de colores usando OpenCV en Python, para ello se han usado las versiones de 3.4.4 y 3.6 respectivamente. Trataremos brevemente el espacio de color HSV, para luego explicar los 4 pasos para detectar colores. ¡Empecemos!

## HSV (Hue, Saturation, Value) en OpenCV

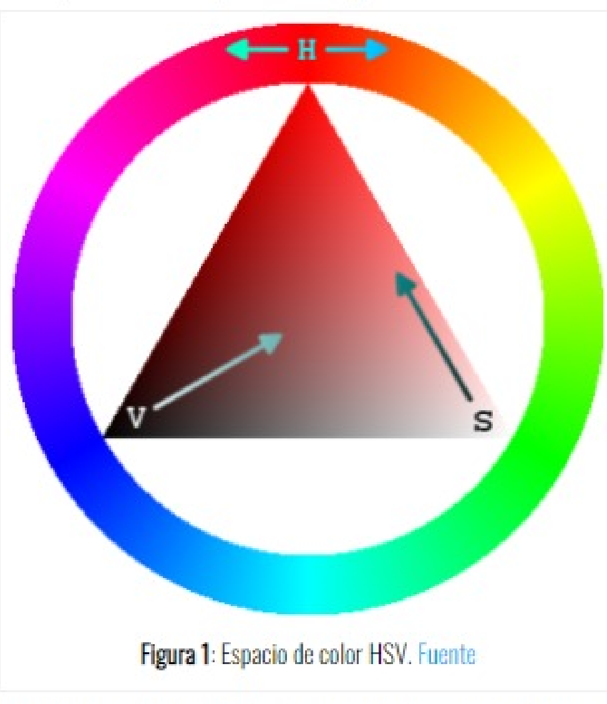


**Figura 1**: Espacio de color HSV. Fuente

El espacio de color HSV (Hue, Saturation, Value / Matiz, Saturación, Brillo), posee 3 componentes, similar al espacio de color RGB que tratamos anteriormente en un post. Vamos a utilizar este espacio de color debido a que podremos determinar de forma más sencilla los rangos de los colores que deseamos detectar.

Para determinar un color nos vamos a centrar principalmente en el componente H que corresponde al matiz. En la figura 1 podemos ver como este componente va cambiando de rojo, amarillo, verde, violeta hasta llegar nuevamente a rojo.

Es necesario conocer que valores pueden tomar cada uno de los canales de HSV en OpenCV, veamos:

- H: 0 a 179
- S: 0 a 255
- V: 0 a 255

Esta breve información acerca de este espacio de color te va a ayudar a comprender especialmente el tercer paso para la detección de colores.

## Detección de colores con HSV en OpenCV

Para detectar un color en OpenCV usando el espacio de color HSV, he determinado 4 pasos, vamos a desglosarlos uno por uno como se ha hecho en el video.

### Paso 1: Imagen a procesar

Lo primero que necesitamos es una imagen o fotograma para sobre este detectar los colores. Podemos conseguir nuestra "materia prima" de dos formas, leyendo una imagen o a través de un video. En esta ocasión lo haremos a través de un video streaming:

```
1.  import cv2
2.  import numpy as np
3.
4.  cap = cv2.VideoCapture(0)
5.
6.  while True:
7.    ret,frame=cap.read()
8.    if ret==True:
9.      cv2.imshow('frame', frame)
10.     if cv2.waitKey(1) & 0xFF == ord('s'):
11.       break
12. cap.release()
13. cv2.destroyAllWindows()
```

En la **línea 1 y 2** importamos OpenCV y numpy, luego inicializamos el proceso de captura del video streaming. Luego en la **línea 7** obtenemos los fotogramas con los que vamos a trabajar. Estoy pasando rápidamente esta explicación debido a que esto ya lo he explicado de forma más detallada anteriormente (Capturar, guardar y leer un video en OpenCV y Python).

### Paso 2: Transformar de BGR a HSV

Por defecto OpenCV lee a las imágenes o fotogramas en BGR, por ello es necesario transformarlas al espacio de color HSV. Para ello nos ayudaremos de la función `cv2.cvtColor`, como primer argumento le daremos la imagen a transformar, y luego `cv2.COLOR_BGR2HSV`, para indicar que transformaremos de BGR a HSV. Veamos como quedaría nuestro código hasta aquí:

```
1.  import cv2
2.  import numpy as np
3.
4.  cap = cv2.VideoCapture(0)
5.
6.  while True:
7.    ret,frame = cap.read()
8.    if ret==True:
9.      frameHSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
10.     cv2.imshow('frame', frame)
11.     if cv2.waitKey(1) & 0xFF == ord('s'):
12.       break
13. cap.release()
14. cv2.destroyAllWindows()
```

En la **línea 9** podemos ver como se ha aplicado la función antes mencionada, entonces esta transformación queda guardada en la variable `frameHSV`.

### Paso 3: Determinar los rangos en donde se encuentra el color a detectar

En este post vamos a detectar el color rojo, que se encuentra al principio y al final del componente H:

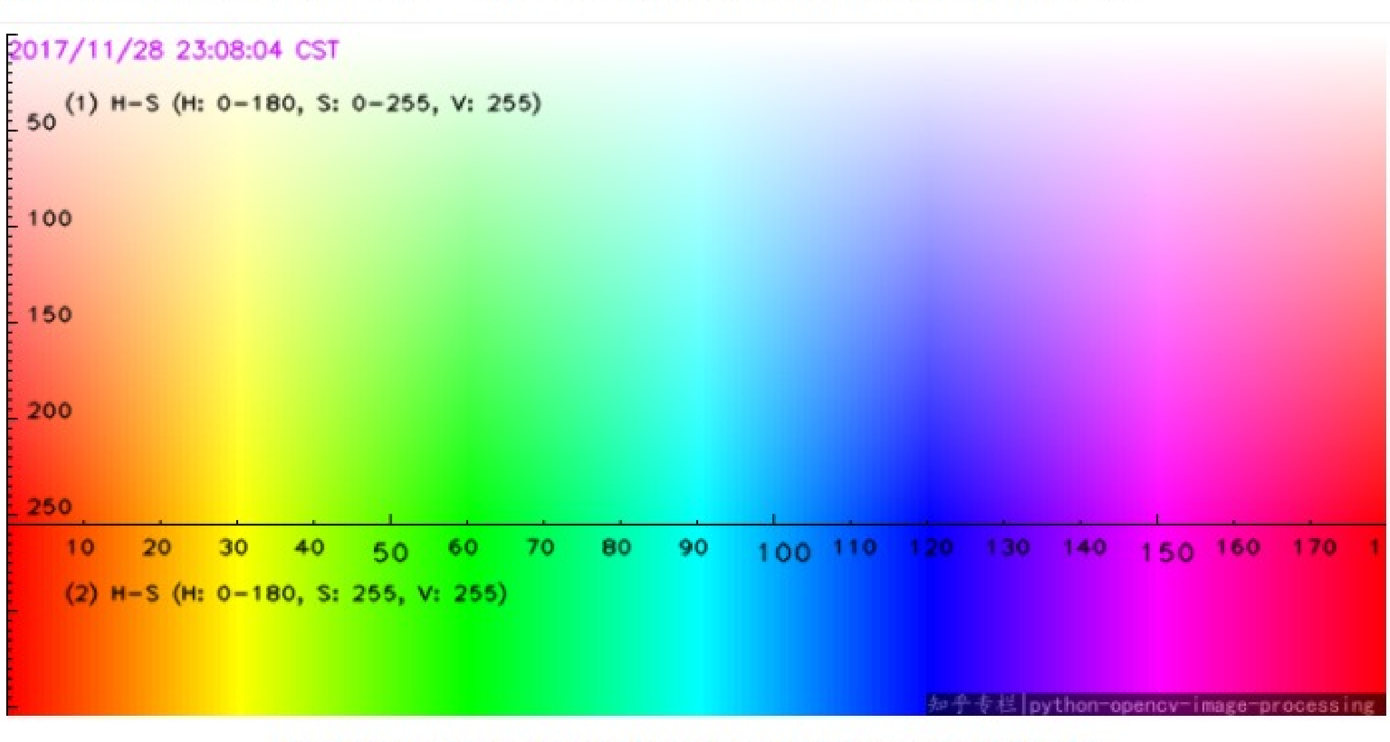


**Figura 2:** Vista de los componentes HSV. Imagen obtenida de una respuesta en stackoverflow.

Como vemos en la figura 2 el componente en H va de 0 a179, y en este pasan colores de rojo, naranja, amarillo, verde, azul, violeta y nuevamente rojo, entonces vamos a determinar 2 rangos para el color rojo que está presente al principio y al final, el primer rango lo determinaremos de 0 a 8, y el segundo de 175 a 179 en H, mientras que para los componente S de 100 a 255, y V de 20 a 255, para ambos rangos.

```
1.  import cv2
2.  import numpy as np
3.
4.  cap = cv2.VideoCapture(0)
5.
6.  redBajo1 = np.array([0, 100, 20], np.uint8)
7.  redAlto1 = np.array([8, 255, 255], np.uint8)
8.
9.  redBajo2=np.array([175, 100, 20], np.uint8)
10. redAlto2=np.array([179, 255, 255], np.uint8)
11.
12. while True:
13.   ret,frame = cap.read()
14.   if ret==True:
15.     frameHSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
16.     maskRed1 = cv2.inRange(frameHSV, redBajo1, redAlto1)
17.     maskRed2 = cv2.inRange(frameHSV, redBajo2, redAlto2)
18.     maskRed = cv2.add(maskRed1, maskRed2)
19.
20.     cv2.imshow('frame', frame)
21.     if cv2.waitKey(1) & 0xFF == ord('s'):
22.       break
23. cap.release()
24. cv2.destroyAllWindows()
```

En la **línea 6 y 7** se especifica el primer rango, para ello debemos crear un array con numpy, allí se especificarán los componentes en H, S y V, seguido de `np.uint8.` El mismo procedimiento se realiza en las **líneas 9 y 10**, ahora para el segundo rango. Estas están ubicadas previo al while, debido a que solo será necesario declararlas una vez.

¿Te fijaste en las líneas 16, 17 y 18? Estas nuevas líneas son necesarias para buscar los rangos, en este caso para detectar el color rojo, por ello en la **línea 16** se usa la función `cv2.inRange`, el primer argumento es la imagen en la cual se buscará el rango, en este caso `frameHSV`, luego el límite inicial y final del primer rango, el mismo proceso se realizará en la **línea 17**, para el segundo rango.

De aquí se obtendrán las imágenes `maskRed1` y `maskRed2`, que son binarias en donde el color blanco refleja el lugar en donde está presente el primer y segundo rango respectivamente, mientras que el color negro mostrará el lugar donde no están presentes estos rangos.

Como deseamos detectar el color rojo, y tenemos 2 rangos establecidos, debemos unirlos, para ello usaremos la **línea 18.** Aquí estamos sumando ambas imágenes para mostrar en una sola (`maskRed`), la presencia del color rojo. `maskRed` por lo tanto también será binaria.

### Paso 4: Visualización

Entonces tenemos una imagen binaria, en donde el color blanco representa el color rojo detectado según los rangos que hemos dado en el paso 3. Es hora de visualizar este resultado.

```
1.  import cv2
2.  import numpy as np
3.
4.  cap = cv2.VideoCapture(0)
5.
6.  redBajo1 = np.array([0, 100, 20], np.uint8)
7.  redAlto1 = np.array([8, 255, 255], np.uint8)
8.
9.  redBajo2=np.array([175, 100, 20], np.uint8)
10. redAlto2=np.array([179, 255, 255], np.uint8)
11.
12. while True:
13.   ret,frame = cap.read()
14.   if ret==True:
15.     frameHSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
16.     maskRed1 = cv2.inRange(frameHSV, redBajo1, redAlto1)
17.     maskRed2 = cv2.inRange(frameHSV, redBajo2, redAlto2)
18.     maskRed = cv2.add(maskRed1, maskRed2)
19.     maskRedvis = cv2.bitwise_and(frame, frame, mask= maskRed)
20.     cv2.imshow('frame', frame)
21.     cv2.imshow('maskRed', maskRed)
22.     cv2.imshow('maskRedvis', maskRedvis)
23.     if cv2.waitKey(1) & 0xFF == ord('s'):
24.       break
25. cap.release()
26. cv2.destroyAllWindows()
```

En la **línea 20** visualizamos el video streming, en mi caso se vería algo así:

**Figura 3:** Captura de video streaming.

Ahora bien, vamos a visualizar la imagen binaria que tenemos en `maskRed`, para ello usamos `cv2.imshow`, como se aprecia en la **línea 21**, tal y como lo habíamos hecho para `frame`.

**Figura 4:** Imagen binaria obtenida luego de aplicar la detección del color rojo.

Como vemos en la figura 4, la región de color blanco corresponde a la ubicación de la pelota roja. También podemos ver unas pequeñas regiones de color blanco, esto vendría a ser ruido en nuestra detección. En el siguiente post veremos como mejorar la detección (O puedes ver este video, que contiene la información que vendrá en el siguiente post). Sin embargo a pesar de ello ya hemos detectado el color rojo en una imagen. ¡Felicitaciones!

Tal vez te estés preguntando, sobre la **línea 19,** pues bien, esta es una visualización adicional, en donde podremos ver el color rojo detectado, y en las regiones de la imagen donde no se presente este color se visualiza en negro. Veamos:

**Figura 5:** Visualización adicional.

Para poder realizar esta visualización necesitamos de la **línea 19.** Aquí usamos la función `cv2.bitwise_and`, a esta le entregamos como primer y segundo argumento la imagen o fotograma en BGR (`frame`), luego, `mask = maskRed`. Esto lo que hace es que la región blanca de `maskRed` permita visualizar los colores de `frame`, mientras que la región en negro se mantenga.

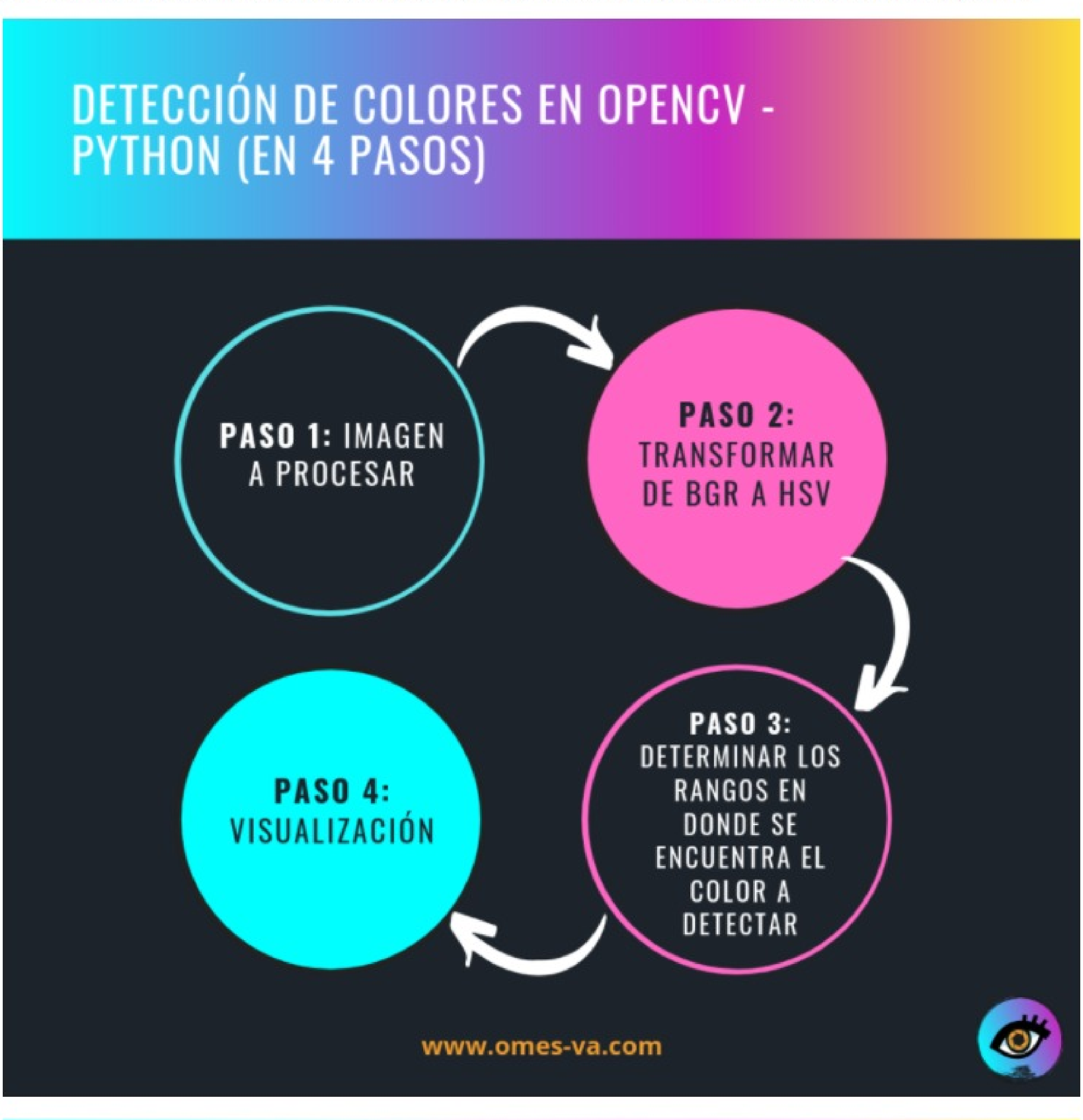
## Si has pensado en usar la detección de colores, ten en cuenta que..

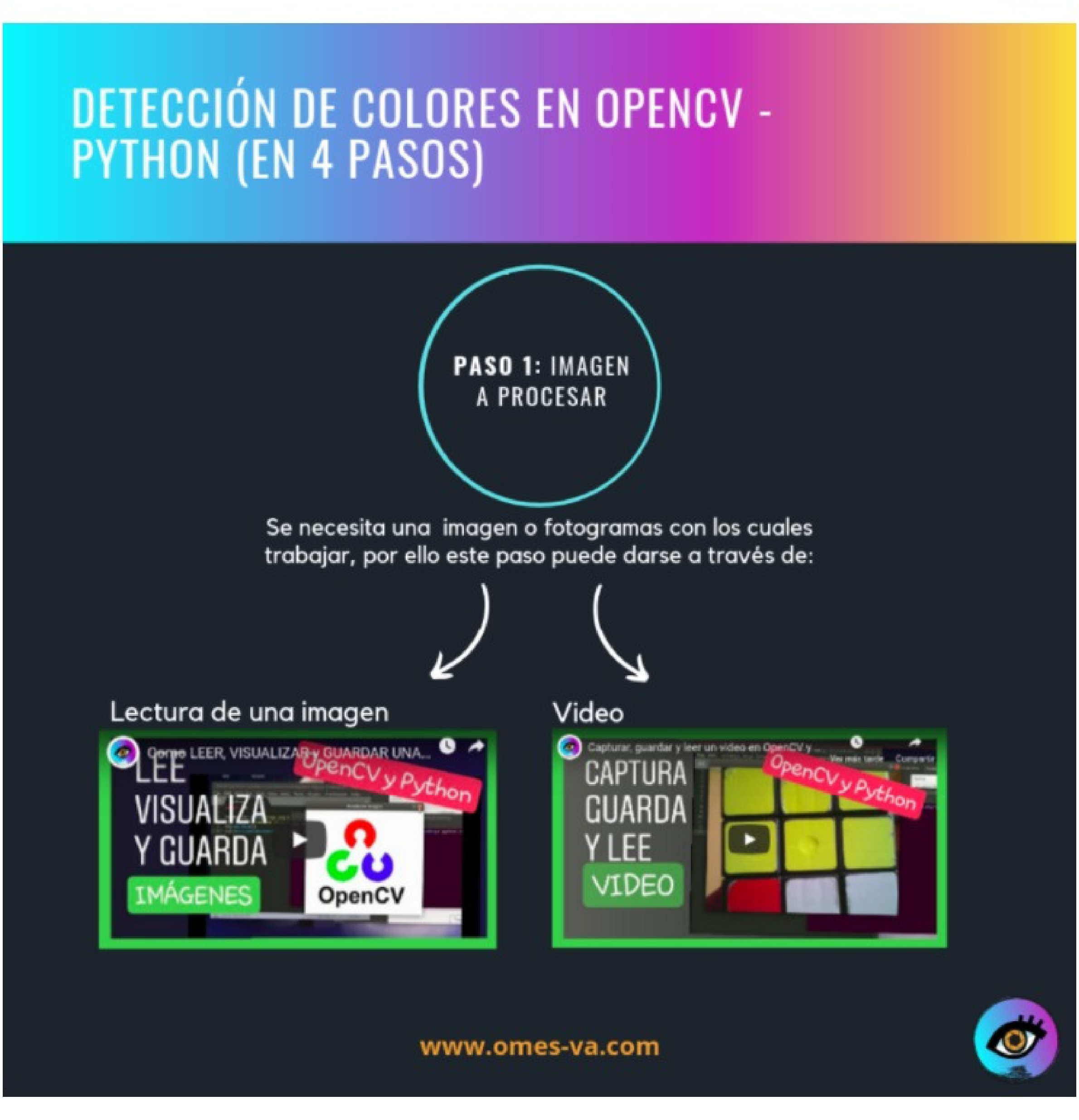
La detección de colores puede ayudarte a realizar un montón de aplicaciones, sin embargo hay dos aspectos importantes a considerar:

- **Iluminación.** Es de importancia tener controlado este aspecto, ya que a más o menos iluminación sobre el objeto o la región que deseamos detectar cierto color, puede alterar el resultado de dicha detección.
- **Fondo.** El fondo de la imagen también podría irrumpir en el proceso de detección, ya que si en este están presentes colores que deseamos detectar, no solo se detectaría el objeto de interés, sino también parte del fondo.

Y bien, por ahora hemos acabado. Espero te haya servido este pequeño tutorial de como detectar el color rojo, recuerda que puedes practicar detectando distintos colores. En el próximo post veremos como mejorar la detección, nos vemos pronto.

Por cierto, no te vayas sin ver las hojitas del contenido, u hojas de resumen, te podrían ayudar a entender mejor el proceso.





Se necesita una imagen o fotogramas con los cuales trabajar, por ello este paso puede darse a través de: