SEP 20 2019

DETECCIÓN DE COLORES Y Tracking en OpenCV – Parte2

20

Por ADMINISTRADOR

CONTENIDO

- Detección del color Azul con OpenCV
  - Encontrar los contornos correspondientes al color azul con OpenCV
  - Dibujar contornos de acuerdo a su área, buscar su centro y visualizar sus coordenadas con OpenCV y Python
- Detectar varios colores en OpenCV

En el post anterior vimos como detectar colores utilizando OpenCV y Python, y llegamos a obtener una imagen binaria, en donde la región en blanco representaba la presencia del color que se deseaba detectar, mientras que en negro donde no se encontraba dicho color. Pues bien, vamos a trabajar un poquito más con la detección de colores y encerraremos las regiones en donde estén los colores deseados, descartaremos regiones pequeñas que no sean de importancia, además encontraremos el punto central y coordenadas del color detectado. ¡Vamos por ello!

Te había hablado de 4 pasos que sigo para poder detectar un color, en el post anterior, estos consistían de:

Paso 1: Imagen a procesar

Paso 2: Transformar de BGR a HSV

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

Paso 4: Visualización

En ese post tratamos de detectar el color rojo, ahora en cambio detectaremos el color azul, veamos:

Detección del color Azul en una imagen con OpenCV

Tomemos en cuenta la siguiente imagen para determinar los rangos en H donde está pudiera estar presente el color azul:

Figura 1: Sección en H donde está presente el color azul.

Continuemos con la programación:

```
1.  import cv2
2.  import numpy as np
3.
4.  cap = cv2.VideoCapture(0)
5.
6.  azulBajo = np.array([100,100,20], np.uint8)
7.  azulAlto = np.array([125,255,255], np.uint8)
8.
9.  while True:
10.
11.   ret, frame = cap.read()
12.
13.   if ret==True:
14.     frameHSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
15.     mask = cv2.inRange(frameHSV, azulBajo, azulAlto)
16.
17.     cv2.imshow('maskAzul', mask)
18.     cv2.imshow('frame', frame)
19.     if cv2.waitKey(1) & 0xFF == ord('s'):
20.       break
21. cap.release()
22. cv2.destroyAllWindows()
```

En la **líneas 1 y 2** importamos OpenCV y numpy, mientras que en las **líneas 6 y 7** entregamos el rango necesario para detectar el color azul en HSV. Como este procedimiento se lo va a realizar a través de un video streaming, la imagen o fotograma que vamos a usar es `frame`, en la **línea 11**. Luego en la **línea 14** transformamos la imagen de BGR a HSV, en tanto que en la **línea 15** obtenemos una imagen binaria que visualizamos en la **línea 17**, mientras que a `frame` lo visualizamos en la **línea 18**. La visualización la tendríamos de la siguiente manera:

Figura 2: Izq. Imagen de entrada. Der. Imagen binaria en donde el blanco representa el color azul, y el negro la ausencia del mismo.

Encontrar los contornos correspondientes al color azul con OpenCV

Es a partir de aquí que vamos a incursionar en los contornos. Una vez que tenemos una imagen binaria podemos aplicar `cv2.findContours`, en donde se encerrarán las áreas de color blanco, para ello añadiremos esta función, y para dibujar los contornos encontrados usaremos `cv2.drawContours`. Veamos como quedaría el código hasta aquí:

```
1.  import cv2
2.  import numpy as np
3.
4.  cap = cv2.VideoCapture(0)
5.
6.  azulBajo = np.array([100,100,20],np.uint8)
7.  azulAlto = np.array([125,255,255],np.uint8)
8.
9.  while True:
10.
11.   ret,frame = cap.read()
12.
13.   if ret==True:
14.     frameHSV = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
15.     mask = cv2.inRange(frameHSV,azulBajo,azulAlto)
16.     _,contornos,_ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
17.       cv2.CHAIN_APPROX_SIMPLE)
18.     cv2.drawContours(frame, contornos, -1, (255,0,0), 3)
19.
20.     cv2.imshow('maskAzul',mask)
21.     cv2.imshow('frame',frame)
22.     if cv2.waitKey(1) & 0xFF == ord('s'):
23.       break
24. cap.release()
25. cv2.destroyAllWindows()
```

En a **línea 16** se ha especificado la función `cv2.findContours`, a ella le entregamos como primer argumento la imagen binaria, en nuestro caso `mask`, el segundo argumento indica que se tomarán en cuenta únicamente los contornos externos y finalmente con `cv2.CHAIN_APPROX_SIMPLE`, se indica que se guarden solo algunos puntos del contorno total. Si quieres profundizar un poquito más en este tema te dejo este video.

NOTA: Recuerda que para este programa usé OpenCV 3.4.4, por lo tanto se obtienen 3 valores al usar `cv2.findContours`, donde el segundo corresponde a los contornos. Si trabajas con otra versión como OpenCV 4, asegurate de tomar esto en cuenta ya que allí obtendrás dos valores. Puedes revisar esto en la documentación de OpenCV.

En la **línea 18** con `cv2.drawContours` dibujamos todos los contornos encontrados en `frame`, luego especificados los contornos que se dibujaran, con -1 dibujamos todos los contornos encontrados, luego especificamos el color en BGR con el que rodeamos los contornos y finalmente el grosor de línea.

Si se te hace un poquito complicado entender estas dos líneas, recuerda que tengo un video explicándolo, además iré subiendo poco a poco el contenido de mis videos para que puedas acceder a la programación. Bien, tendríamos la siguiente visualización:

Figura 3: Izq. Imagen de entrada sobre la cual se dibujan los contornos encontrados en azul. Der. Imagen binaria

Como podrás ver en la figura 2, los contornos se están dibujando en la imagen de entrada, y lo están haciendo de color azul, esto debido a que en `cv2.drawContours` especificamos que se dibuje en `frame`. Además se puede apreciar que también se han encerrado pequeñas áreas que no precisamente corresponden al objeto que deseamos detectar. ¡Hasta aquí vamos muy bien!.

Dibujar contornos de acuerdo a su área, buscar su centro y visualizar sus coordenadas con OpenCV y Python

En esta sección veremos como dibujar contornos de acuerdo a su área, esto para descartar esas pequeñas porciones en la imagen que se visualizaban como de color azul pero que no corresponden al objeto (en este caso la pelotita de color azul), y que se podría decir que son el ruido de nuestra detección de colores. Una vez que se tengan los contornos con los cuales trabajar se procederá a encontrar su centro para finalmente visualizar sus coordenadas x e y en la imagen.

```
1.  import cv2
2.  import numpy as np
3.
4.  cap = cv2.VideoCapture(0)
5.
6.  azulBajo = np.array([100,100,20],np.uint8)
7.  azulAlto = np.array([125,255,255],np.uint8)
8.  while True:
9.
10.   ret,frame = cap.read()
11.
12.   if ret==True:
13.     frameHSV = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
14.     mask = cv2.inRange(frameHSV,azulBajo,azulAlto)
15.     _,contornos,_ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
16.       cv2.CHAIN_APPROX_SIMPLE)
17.     #cv2.drawContours(frame, contornos, -1, (255,0,0), 3)
18.     for c in contornos:
19.       area = cv2.contourArea(c)
20.       if area > 3000:
21.         M = cv2.moments(c)
22.         if (M["m00"]==0): M["m00"]=1
23.         x = int(M["m10"]/M["m00"])
24.         y = int(M['m01']/M['m00'])
25.         cv2.circle(frame, (x,y), 7, (0,255,0), -1)
26.         font = cv2.FONT_HERSHEY_SIMPLEX
27.         cv2.putText(frame, '{},{}'.format(x,y),(x+10,y), font, 0.75,(0,255,0),1,cv2.LINE_AA)
28.         nuevoContorno = cv2.convexHull(c)
29.         cv2.drawContours(frame, [nuevoContorno], 0, (255,0,0), 3)
30.     #cv2.imshow('maskAzul',mask)
31.     cv2.imshow('frame',frame)
32.     if cv2.waitKey(1) & 0xFF == ord('s'):
33.       break
34. cap.release()
35. cv2.destroyAllWindows()
```

La línea 17 que dibuja todos los contornos encontrados se va a comentar, ya que como te decía antes, necesitamos cernir todos los contornos para eliminar aquellos que no fueran importantes.

Línea 18: Se recorre cada uno de los contornos encontrados con un `for`, ahora se analiza cada contorno (`c`).

Línea 19: Se emplea la función `cv2.contourArea`, para determinar el área en pixeles del contorno.

Línea 20: Del área encontrada se comparará con `3000` pixeles por ejemplo (pero este valor puede variar de acuerdo a tu aplicación), para solo dejar pasar a los contornos que superen dicho valor, por lo tanto los más pequeños serán descartados.

Línea 21: Se encuentran los momentos del contorno, esto se utiliza para poder encontrar los puntos centrales del contorno en x e y como podemos ver en las **líneas 23 y 24**. (Debido a que se realiza una división para determinar las coordenadas se establece la **línea 22**, para que no exista división para cero, por lo que se iguala a 1).

Línea 25: Aquí vamos a dibujar un círculo con `cv2.circle`, este va a ser dibujado en `frame`, en las coordenadas x e y encontradas, con un radio de 7 pixeles, de color verde que en BGR sería `(0,255,0)`, finalmente con `-1` especificamos que sea un círculo y no una circunferencia.

NOTA: Estamos empleando `cv2.circle`, luego usaremos `cv2.putText`, por ello te recomiendo que si tienes un poquito de problemas para entender como funcionan veas el siguiente video:

Línea 26: Como necesitamos visualizar texto, se debe especificar el tipo de funte a utilizar, en este caso `cv2.FONT_HERSHEY_SIMPLEX`.

Línea 27: Ahora se utiliza `cv2.putText` para visualizar el texto, a esta función le entregamos la imagen en donde se va a visualizar, en este caso `frame`, luego el texto que se va a visualizar que serían los valores de x e y, después están las coordenadas en donde se va a ubicar el texto, la fuente que habíamos declarado en la **línea 26**, el tamaño del texto, el color en BGR `(0,255,0)` para verde y finalmente el grosor del texto.

Línea 28: Aquí vamos a usar `cv2.convexHull`, para mejorar la visualización del contorno, te dejo el link de los documentos de openCV por si quieres más información sobre esta función. Recuerda que el uso de esta función dependerá de tu aplicación.

Línea 29: Se dibujan los contornos con la función `cv2.drawContours` en `frame`, el contorno que se va a visualizar es el siguiente argumento, luego como no todos los contornos encontrados por la **línea 15** se van a dibujar usamos 0, después el color en el que se va a dibujar que es azul y finalmente el grosor de línea.

Veamos como se visualiza el resultado de todo este proceso:

Figura 4: Detección de un objeto en color azul en donde se ha determinado su contorno, centro y coordenadas.

Detectar varios colores en OpenCV

Ahora que hemos pasado por todo este proceso de detección, ¿qué te parece si optimizamos un poquito el código y detectamos 2 colores más?

Vamos a detectar el color azul, amarillo y rojo, y te darás cuenta que va a ser un proceso similar.

Figura 5: Sección en H donde está presente el color rojo, amarillo y azul.

En la figura 5 he determinados distintos rangos para cada color, recuerda que **los valores que he establecido NO son una regla,** pueden cambiar acorde a la aplicación, iluminación y color que desees detectar. Veamos la programación:

```
1.  import cv2
2.  import numpy as np
3.
4.  def dibujar(mask,color):
5.    _,contornos,_ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
6.        cv2.CHAIN_APPROX_SIMPLE)
7.    for c in contornos:
8.      area = cv2.contourArea(c)
9.      if area > 3000:
10.       M = cv2.moments(c)
11.       if (M["m00"]==0): M["m00"]=1
12.       x = int(M["m10"]/M["m00"])
13.       y = int(M['m01']/M['m00'])
14.       nuevoContorno = cv2.convexHull(c)
15.       cv2.circle(frame,(x,y),7,(0,255,0),-1)
16.       cv2.putText(frame,'{},{}'.format(x,y),(x+10,y), font, 0.75,(0,255,0),1,cv2.LINE_AA)
17.       cv2.drawContours(frame, [nuevoContorno], 0, color, 3)
18.
19. cap = cv2.VideoCapture(0)
20.
21. azulBajo = np.array([100,100,20],np.uint8)
22. azulAlto = np.array([125,255,255],np.uint8)
23.
24. amarilloBajo = np.array([15,100,20],np.uint8)
25. amarilloAlto = np.array([45,255,255],np.uint8)
26.
27. redBajo1 = np.array([0,100,20],np.uint8)
28. redAlto1 = np.array([5,255,255],np.uint8)
29.
30. redBajo2 = np.array([175,100,20],np.uint8)
31. redAlto2 = np.array([179,255,255],np.uint8)
32.
33. font = cv2.FONT_HERSHEY_SIMPLEX
34. while True:
35.
36.   ret,frame = cap.read()
37.
38.   if ret == True:
39.     frameHSV = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
40.     maskAzul = cv2.inRange(frameHSV,azulBajo,azulAlto)
41.     maskAmarillo = cv2.inRange(frameHSV,amarilloBajo,amarilloAlto)
42.     maskRed1 = cv2.inRange(frameHSV,redBajo1,redAlto1)
43.     maskRed2 = cv2.inRange(frameHSV,redBajo2,redAlto2)
44.     maskRed = cv2.add(maskRed1,maskRed2)
45.     dibujar(maskAzul,(255,0,0))
46.     dibujar(maskAmarillo,(0,255,255))
47.     dibujar(maskRed,(0,0,255))
48.     cv2.imshow('frame',frame)
49.     if cv2.waitKey(1) & 0xFF == ord('s'):
50.       break
51. cap.release()
52. cv2.destroyAllWindows()
```

El proceso que hemos seguido hasta ahora es el mismo, solo que con más colores, es por eso que en un principio importamos los paquetes necesarios. Pasaremos por alto la función `dibujar` por ahora.

Comience con
30$ gratis
Broker multipremiado, multirregulado
Invierta con ejecución rápida y directa
XM 10 YEARS ANNIVERSARY
Abrir una cuenta
Nuestros servicios implican un riesgo significativo y pueden producir la pérdida de su capital invertido. Aplican TyC.

En la **línea 19** iniciamos el video streaming, mientras que de la **línea 21 a la 31** establecemos los rangos en HSV donde está presente el color azul, amarillo y rojo. En la **línea 33** determinamos el tipo de fuente que usaremos para visualizar texto en la imagen.

La imagen que vamos a usar para procesar está presente en la **línea 36** con `frame`, posteriormente en la **línea 39** vamos a transformarla de BGR a HSV, entonces de la **línea 40 a 44** obtenemos imágenes binarias correspondientes a cada color detectado.

Ahora sí, procedemos con la explicación de la función `dibujar`, esta nos pide dos argumentos, el primero es la imagen binaria correspondiente a la detección de cada uno de los colores que estamos detectando, en este caso `maskAzul`, `maskAmarillo` y `maskRojo`. El segundo argumento es un color en BGR, con el cual se van a dibujar los contornos encontrados por cada color. Según se observa en la **línea 45, 46 y 47** se dibujarán los contornos en azul `(255,0,0)`, amarillo `(0,255,255)` y rojo `(0,0,255)`.

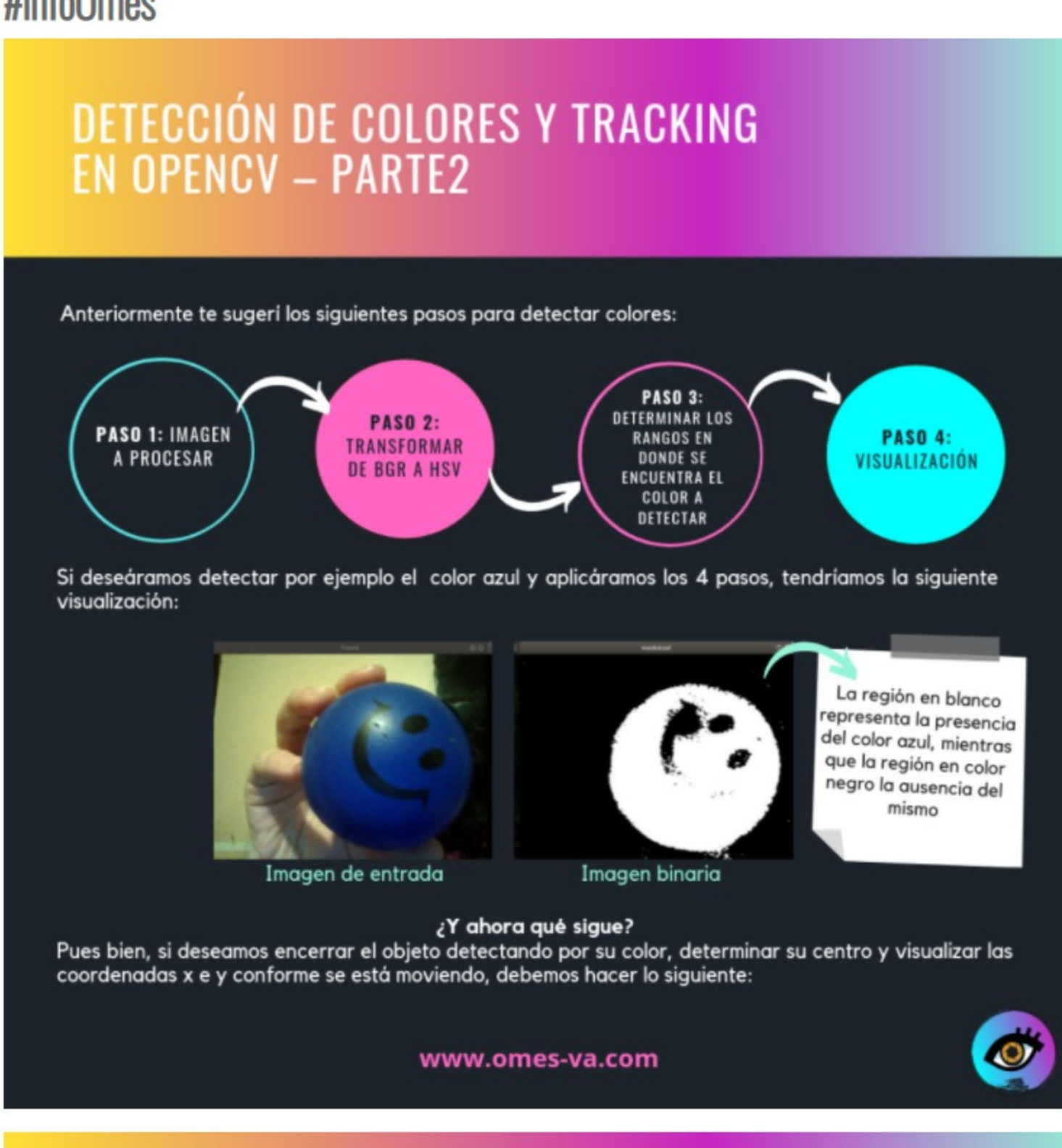
En la **línea 5** se buscan todos los contornos de la imagen binaria que en ese momento haya sido proporcionada, y se estudia cada uno de estos contornos en la **línea 7**, determinando su área en la **línea 8**. Si el área encontrada es mayor a 3000 pixeles entonces se continúa con el proceso, caso contrario esto quiere decir que el área es muy pequeña y se descartará.

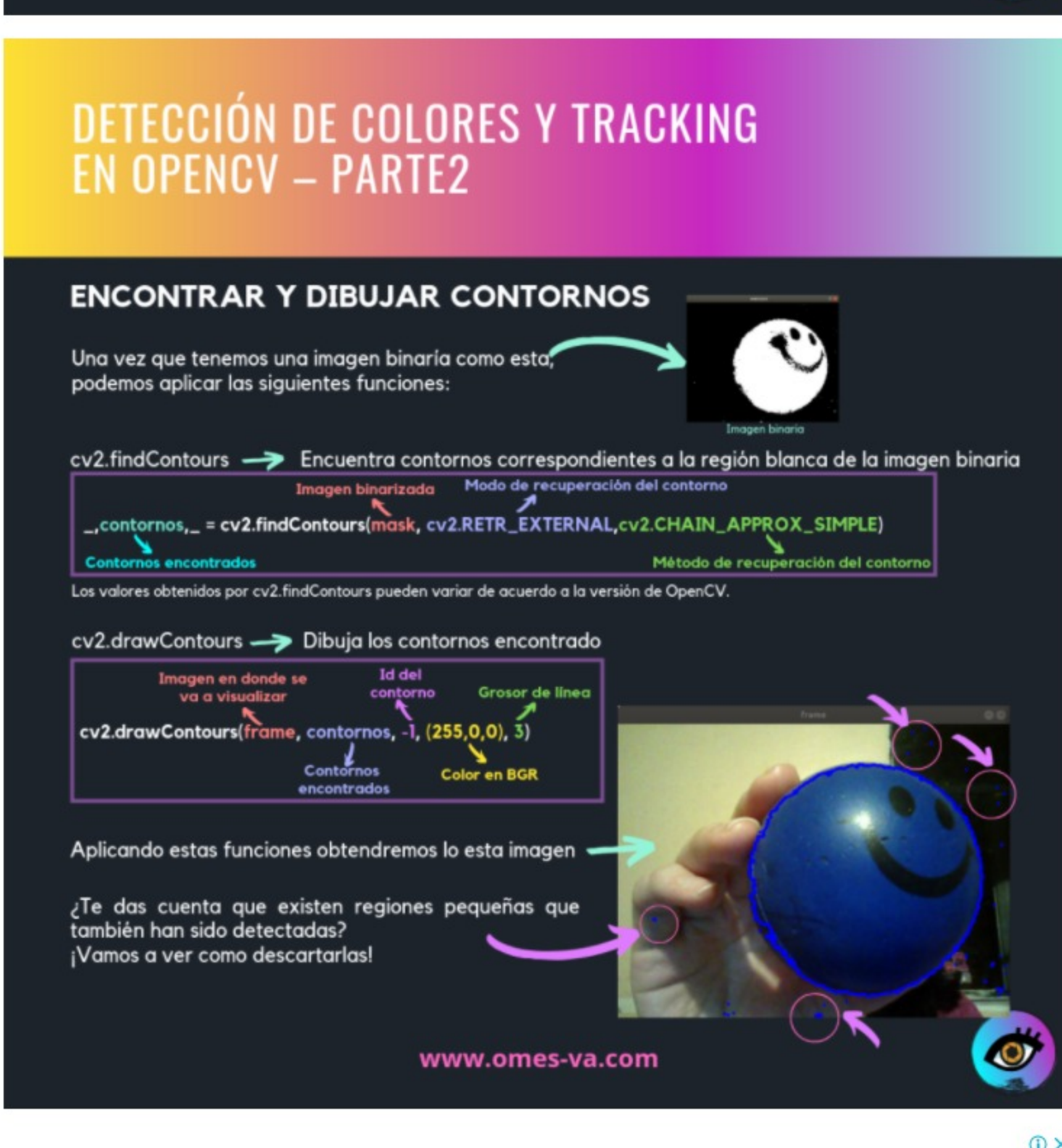
En las **líneas 10 a 13** se está determinando las coordenadas centrales de cada contorno con respecto a la imagen. Finalmente de la **línea 15 a 17** se dibuja un círculo correspondiente al punto central del contorno, luego las coordenadas centrales y finalmente el contorno, todo esto se va a dibujar en `frame`.

Figura 6: Detección de varios colores con OpenCV y Python

Y hemos llegado al final de este post, recuerda que tienes un video en donde también te explico como realizar este procedimiento. Cuéntame si te ayudo la explicación y si lo probaste. No olvides darle un vistazo a las hojitas de resumen. Nos vemos en el siguiente post.

#infoOmes





READY TO USE GRAPHIC ASSETS
envato elements
FREE ITEMS
TEMPLATES
MOCKUPS
ICONS
GRAPHICS
AND MORE!
START NOW