DETECCION DE COLORES CON OPENCV Y PYTHON

Antes de comenzar con la explicación de la programación del detector de colores algo que es importante y debe de quedar claro es el tipo de modelo HSV (Hue, Saturation, Value) o en español (Matiz, Saturación, Valor) o también conocido como HSB (Matiz, Saturación, Brillo) que se utiliza en este tipo de detecciones, en este modelo cada canal tiene un valor con el cual se logra obtener cada color.

- El matiz tiene un valor de entre 0 a 179
- La saturación tiene un valor de 0 a 255
- El valor o brillo un valor de 0 a 255

El matiz se encarga de modelar el tipo de color, la saturación lo podríamos entender como la fuerza con la que se muestra el color y por último el valor que es el brillo o intensidad del color, en la figura 1 se muestra la relación de esto.

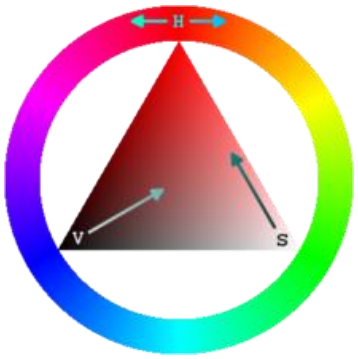


Figura 1: Espacio de color HSV. wikipedia

Una vez entendido esto pasaremos a la programación, usaremos la librería de Open cv, Numpy e Imutils para este proyecto.

Empezaremos importando estas librerías en el archivo e inicializando nuestro video captura, en este proyecto se usó la extensión .ipynb pero igual se puede usar .py.

```
import cv2
import numpy as np
import imutils

#START THE VIDEOCAPTURE
Cap=cv2.VideoCapture(0)
```

Ahora pondremos en práctica lo visto en el inicio acerca del modelo HSV, primero tenemos que tener en cuenta que para detectar correctamente y de mejor forma los colores necesitamos de los valores HSV del color tanto oscuros como en su nivel más brillante esto para tener un margen de entre que valores está el color a detectar, en este caso se puede observar en el código mostrado más adelante que el valor para la variable "Dark_red " y "Light_red" aparece dos veces con el numero 1 y 2 esto es porque en HSV el color rojo aparece dos veces, en la siguiente grafica HSV mostrada en la figura 2 se puede apreciar como el color rojo se encuentra tanto en la parte derecha como en la izquierda de la imagen.

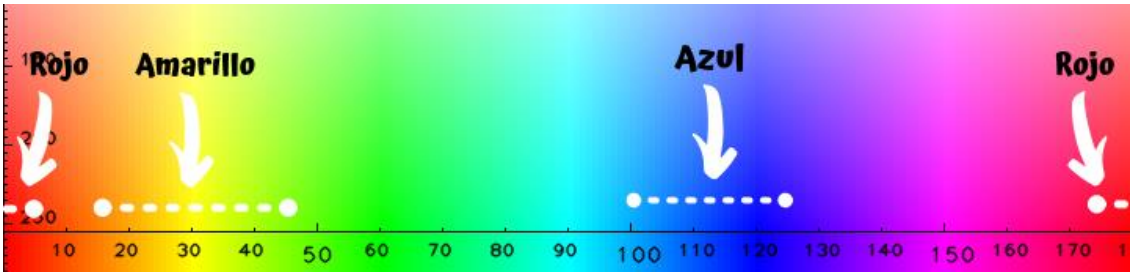


Figura 2: Grafica de los valores de matiz para los colores en modelo HSV. OMES

Aquí podemos ver que los valores mostrados en la parte de abajo son los del matiz(H), estos son valores que veremos en el siguiente código de programación.

En esta parte del código lo que se hace es asignar los valores oscuros y brillantes de cada color a detectar en un numpy array, veras que también se agrega "uint8" esto es el tipo de dato habitual para las imágenes.

```
#VALUES TO DETECT COLORS IN HSV SCALE
Dark_red1= np.array([0,50,120], np.uint8)
Light_red1= np.array([10,255,255], np.uint8)

Dark_red2= np.array([170,100,100], np.uint8)
Light_red2= np.array([179,255,255], np.uint8)

Dark_green= np.array([40,70,80], np.uint8)
Light_green= np.array([70,255,255], np.uint8)

Dark_blue= np.array([90,60,0], np.uint8)
Light_blue= np.array([121,255,255], np.uint8)

Dark_yellow= np.array([25,70,100], np.uint8)
Light_yellow= np.array([30,255,255], np.uint8)
```

Para que estos valores sean óptimos para su detección se debe probar con distintos valores hasta obtener el adecuado, use como guía la gráfica HSV mostrada en la figura 2.

Siguiendo con el código crearemos nuestro ciclo while en el cual verificaremos si la variable "Ret" es verdadera y nuestra variable "Frame" que leerá el contenido de la video captura, seguido de esto tenemos que convertir nuestros frames de BGR (formato que Open cv usa por defecto) a HSV que es el formato que usaremos para la detección.

```
while True:
    #READ THE FRAMES AND CHECK THE RET
    Ret, Frame= Cap.read()
    if Ret==True:
    #TRANSORM THE ORIGINALS BGR FRAMES TO HSV TO DETECT COLORS
        Transform_hsv= cv2.cvtColor(Frame,cv2.COLOR_BGR2HSV)
```

Ahora crearemos la máscara para cada color con la función .inRange() la cual usa tres parámetros los cuales son la matriz principal en este caso seria "Transform_hsv" seguido de la matriz con el valor oscuro del color y por último la matriz brillante, la función de estas mascaras es especificar que valores queremos conservar.

Para el caso de la máscara del color rojo al haber tenido dos valores bajos y altos, se tendrán que sumar ambas mascaras para crear una sola como se puede apreciar en la variable "Mask_red_end".

```
#CREATE A MASK TO EVERY COLOR WITH THE DARK AND LIGHT VALUE
        Mask_red1=cv2.inRange(Transform_hsv, Dark_red1, Light_red1)
        Mask_red2=cv2.inRange(Transform_hsv, Dark_red2, Light_red2)
        Mask_red_end=Mask_red1 + Mask_red2

        Mask_green=cv2.inRange(Transform_hsv, Dark_green, Light_green)
        Mask_blue=cv2.inRange(Transform_hsv, Dark_blue, Light_blue)
        Mask_yellow=cv2.inRange(Transform_hsv, Dark_yellow, Light_yellow)
```

A continuación, procederemos a crear una umbralización para cada mascara ya que esto nos ayudara a la hora de dibujar los contornos de los objetos de mejor forma más adelante.

El valor para la variable "Thresh" fue de 100 pero se recomienda probar con otros valores para encontrar el que más se adecue a las necesidades.

En esta función de .threshold() le daremos cuatro parámetros los cuales son:

- La variable para aplicar la umbralización
- El valor del umbral (en este caso declarado en la variable "Thresh")
- El valor máximo (generalmente 255)
- El método de umbralización

```
#THRESHOLD
Thresh=100
Ret, Thresh_red= cv2.threshold(Mask_red_end,Thresh,255,cv2.THRESH_BINARY)
Ret, Thresh_green= cv2.threshold(Mask_green,Thresh,255,cv2.THRESH_BINARY)
Ret, Thresh_blue= cv2.threshold(Mask_blue,Thresh,255,cv2.THRESH_BINARY)
Ret, Thresh_yellow= cv2.threshold(Mask_yellow,Thresh,255,cv2.THRESH_BINARY)
```

Para utilizar esta umbralización que acabamos de hacer ocuparemos de dos funciones, la primera .findContours() a la cual le daremos tres parámetros, el primero será la variable binaria de Thresh asignada a cada color, como segundo parámetro usaremos el modo de detección y como ultimo parámetro el método de detección, para conocer acerca de los distintos modos y métodos puedes leer el siguiente documento https://goo.su/QBb8Ms2.

La segunda función será .grab_contours() de la librería imutils a la cual le entregaremos el parámetro de la variable creada para encontrar los contornos anteriormente en .findContours().

```
#FIND THE CONTOUR OF THE OBJECTS TO DETECT
        Countour1= cv2.findContours(Thresh_red, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
        Countour1= imutils.grab_contours(Countour1)

        Countour2= cv2.findContours(Thresh_green, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
        Countour2= imutils.grab_contours(Countour2)

        Countour3= cv2.findContours(Thresh_blue, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
        Countour3= imutils.grab_contours(Countour3)

        Countour4= cv2.findContours(Thresh_yellow, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
        Countour4= imutils.grab_contours(Countour4)
```

Ya teniendo encontrados los contornos pasaremos a crear un ciclo for para cada contorno en el cual vamos a definir el área y a partir de esto dibujar el contorno encontrado junto con un círculo y un texto.

Primero vamos a crear la variable “c”, después encontraremos el área del contorno con la función .contourArea(), pasaremos a un condicional if en el cual si el valor del área es mayor a 4000 dibujará los contornos con la función .drawContours() esta función tendrá como parámetros la imagen en la cual se dibujará el contorno, seguido de los elementos de la tupla de la variable “c” [c]. Después el contourIdx seguido del color del contorno y por último el ancho de la línea del contorno.

Para dibujar el circulo y poner el texto necesitamos de las coordenadas, para esto usamos la función .moments() la cual nos dará los valores en el eje x,y. Para ver más sobre la función moments visita la página https://goo.su/JYiT.

Teniendo los valores de x,y pasaremos a dibujar el circulo con la función .circle() a la cual le daremos como parámetros:

- Imagen en la cual se dibujará el circulo
- Coordenadas en x,y
- El radio del circulo
- El color
- El grosor del circulo

Y por último mostraremos el texto en la imagen con los parámetros de:

- Imagen en la cual se mostrará el texto
- Texto a mostrar
- Coordenadas en x,y
- Tipo de fuente
- Tamaño de la fuente
- Color del texto
- Grosor del texto

```
#DRAW THE CONTOURS, A CIRCLE AND PUT TEXT FOR EVERY CONTOUR IN THE FRAMES
        for c in Countour1:
            Area1=cv2.contourArea(c)
            if Area1 > 4000:
                cv2.drawContours(Frame,[c],-1,(0,255,0),2)
                M=cv2.moments(c)
                x=int(M["m10"]/M["m00"])
                y=int(M["m01"]/M["m00"])
                cv2.circle(Frame,(x,y),5,(255,255,255),-1)
                cv2.putText(Frame,"Red",(x-20,y),cv2.FONT_HERSHEY_SIMPLEX,2,(255,255,255),2)
```

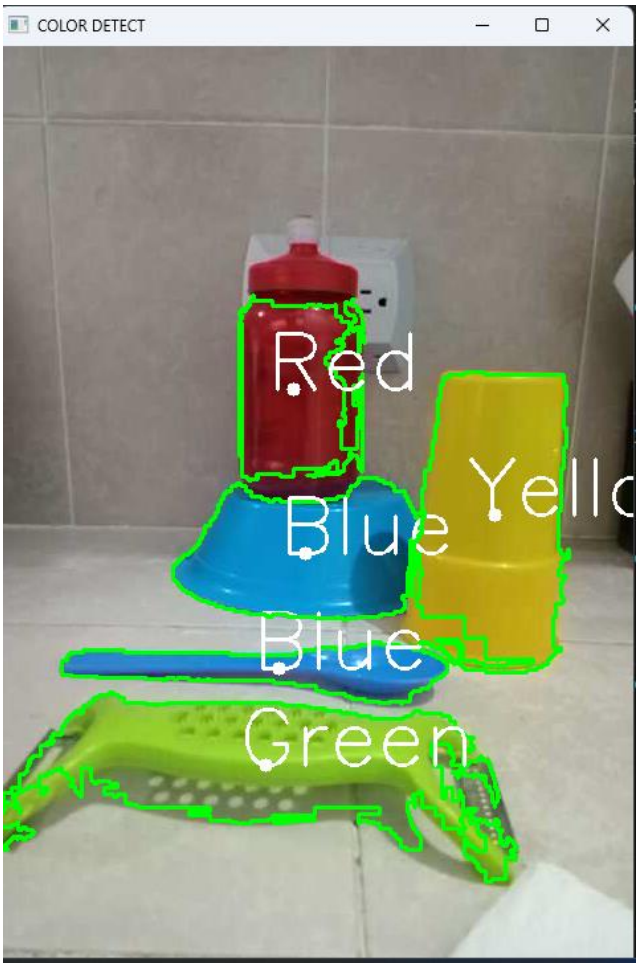
Todos los ciclos for para cada contorno tiene la misma sintaxis solo debes cambiar algunas variables como se muestra en el código siguiente.

```
#DRAW THE CONTOURS, A CIRCLE AND PUT TEXT FOR EVERY CONTOUR IN THE FRAMES
        for c in Countour1:
            Area1=cv2.contourArea(c)
            if Area1 > 4000:
                cv2.drawContours(Frame,[c],-1,(0,255,0),2)
                M=cv2.moments(c)
                x=int(M["m10"]/M["m00"])
                y=int(M["m01"]/M["m00"])
                cv2.circle(Frame,(x,y),5,(255,255,255),-1)
                cv2.putText(Frame,"Red",(x-20,y),cv2.FONT_HERSHEY_SIMPLEX,2,(255,255,255),2)

        for c in Countour2:
            Area2=cv2.contourArea(c)
            if Area2 > 4000:
                cv2.drawContours(Frame,[c],-1,(0,255,0),2)
                M=cv2.moments(c)
                x=int(M["m10"]/M["m00"])
                y=int(M["m01"]/M["m00"])
                cv2.circle(Frame,(x,y),5,(255,255,255),-1)
                cv2.putText(Frame,"Green",(x-20,y),cv2.FONT_HERSHEY_SIMPLEX,2,(255,255,255),2)

        for c in Countour3:
            Area3=cv2.contourArea(c)
            if Area3 > 4000:
                cv2.drawContours(Frame,[c],-1,(0,255,0),2)
                M=cv2.moments(c)
                x=int(M["m10"]/M["m00"])
                y=int(M["m01"]/M["m00"])
                cv2.circle(Frame,(x,y),5,(255,255,255),-1)
                cv2.putText(Frame,"Blue",(x-20,y),cv2.FONT_HERSHEY_SIMPLEX,2,(255,255,255),2)

        for c in Countour4:
            Area4=cv2.contourArea(c)
            if Area4 > 4000:
                cv2.drawContours(Frame,[c],-1,(0,255,0),2)
                M=cv2.moments(c)
                x=int(M["m10"]/M["m00"])
                y=int(M["m01"]/M["m00"])
                cv2.circle(Frame,(x,y),5,(255,255,255),-1)
                cv2.putText(Frame,"Yellow",(x-20,y),cv2.FONT_HERSHEY_SIMPLEX,2,(255,255,255),2)
```

Por último, mostraremos el video en pantalla con la función .imshow() y como parámetros el nombre que llevara la ventana y la imagen que se mostrara en este caso la variable "Frame" que contiene la video captura y crearemos una condicional para cerrar la pantalla de video con la tecla ESC del teclado, la tecla ESC en código ASCII corresponde al número 27.

```
        cv2.imshow("COLOR DETECT", Frame)
        t=cv2.waitKey(1)
        if t==27:break

Cap.release()
cv2.destroyAllWindows()
```

A continuación se verán los resultados obtenidos con este código, se puede observar cómo detecta los colores correctamente y el contorno de las imágenes se detecta también, algo que se debe de tener muy en cuenta al hacer detección de colores es el tema de la iluminación en algunas ocasiones algunos colores no se detectan debido a una falta de iluminación al igual que los contornos se pueden llegar a dibujar en sombras como lo es el caso del color verde que el reflejo de la sombra se está tomando como contorno del objeto.



Si llegaste hasta esta parte te agradezco el tiempo que me regalaste y espero te haya ayudado con tus dudas o inquietudes de aprender más, esto solo es un pequeño fragmento de explicación si quieres seguir aprendiendo más puedes visitar las siguientes páginas que te ayudaran a seguir aprendiendo y mejorando en el tema de la visión artificial y Python.

1.- https://omes-va.com/

2.- https://www.pythonpool.com/

3.- https://hetpro-store.com/TUTORIALES/

4.- https://learnopencv.com/

Lo principal es la sabiduría; adquiere sabiduría, Y con todo lo que obtengas adquiere inteligencia. Proverbios 4:7