SEGMENTACIÓN DE IMAGEN BASADO EN ESPACIOS DE COLORES

Ceballos Barrios Martin Camilo - Quintana Fuentes Jose Daniel
Procesamiento de Contenido Multimedia
Universidad del Magdalena
Santa Marta, Magdalena
martinceballoscb@unimagdalena.edu.co
josequintanadf@unimagdalena.edu.co

***Resumen*—En esta práctica de laboratorio se realizó por medio de la herramienta Colab un código propuesto por el docente, el cuál fue modificado para nuestra conveniencia hacia el filtrado por medio de colores, usando como ejemplo 6 pelotas de diferente color.**

Palabras Clave: filtrado, colores.

***Abstract*— In this laboratory practice, a code proposed by the teacher was carried out using the Colab tool, which was modified for our convenience towards filtering through colors, using as an example 6 balls of different colors.**

Keywords: filtering, colors.

OBJETIVOS:

- → Desarrollo del filtrado de imágenes por medio de colores.
- → Ver el funcionamiento de cada uno de los umbrales de colores disponibles, para así realizar el filtrado.

I. MODELO TEORICO

Un espacio de color es un sistema de interpretación del color, es decir, una organización específica de los colores en una imagen o video. Depende del modelo de color en combinación con los dispositivos físicos que permiten las representaciones reproducibles de color, por ejemplo, las que se aplican en señales analógicas (televisión a color) o representaciones digitales. Un espacio de color puede ser arbitrario, con colores particulares asignados según el sistema y estructurados matemáticamente.

Un modelo de color es un modelo matemático abstracto que describe la forma en la que los colores pueden representarse como tuplas de números, normalmente como tres o cuatro valores o componentes de color (por ejemplo, RGB y HSV son modelos de color). Sin embargo, un modelo de color que no tiene asociada una función de mapeo a un espacio de color absoluto es más o menos un sistema de color arbitrario sin conexión a un sistema de interpretación de color.

Se puede crear un amplio rango de colores mediante pigmentos de colores primarios (cian (C), magenta (M), amarillo (Y), y negro (K)). Otra manera de crear los mismos colores es usando su matiz (eje X), su saturación (eje Y), y su brillo (eje Z). A esto se le llama modelo de color HSV.

El modelo de color RGB está implementado de formas diferentes, dependiendo de las capacidades del sistema utilizado. De lejos, la implementación general más utilizada es la de 24 bits, con 8 bits, o 256 niveles de color discretos por canal. Cualquier espacio de color basado en ese modelo RGB de 24 bits está limitado a un rango de 256×256×256 ≈ 16,7 millones de colores. Algunas implementaciones usan 16 bits por componente para un total de 48 bits, resultando en la misma gama con mayor número de colores. Esto es importante cuando se trabaja con espacios de color de gama amplia (donde la mayoría de los colores se localizan relativamente juntos), o cuando se usan consecutivamente un amplio número de algoritmos de filtrado digital. El mismo principio se aplica en cualquier espacio de color basado en el mismo modelo de color, pero implementado en diferentes profundidades de color.

En cuanto a la conversión del espacio de color es la traducción de la representación de un color de una base a otra. Esto ocurre normalmente en el contexto de convertir una imagen representada en un espacio de color a otro espacio de color, teniendo como objetivo que la imagen convertida se parezca lo más posible a la original [1]. Además, en Python y openCV se puede hacer un cambio de espacio de color como de RGB a gris, BGRA, YCrCb, XYZ, HSV, Lab, Luv, HLS, YUV usando la función cvtColor, esta es una función de openCV, con más de 150 espacios disponibles, que convierte imágenes de un espacio de color a otro [2].

La segmentación es uno de los problemas generales del campo de la visión artificial y consiste en dividir una imagen digital en varias regiones (grupos de píxeles) denominadas segmentos. Más concretamente, la segmentación es un proceso de clasificación por píxel que asigna una categoría a cada píxel de la imagen analizada. Este problema general se divide en problemas especializados, dando lugar por ejemplo a:

- segmentación por color
- segmentación por texturas
- superpíxel
- segmentación semántica

Además, cada problema especializado le otorga un significado propio a las categorías que se usan en la clasificación de los píxeles. Uno de los casos más elementales de segmentación es la umbralización, un tipo particular de segmentación por color con solo dos categorías: claro y oscuro. Cada píxel se clasifica como claro u oscuro comparando su intensidad con una intensidad de referencia dada denominada umbral. El objetivo de la segmentación es localizar regiones con significado. La segmentación se usa tanto para localizar objetos como para encontrar sus bordes dentro de una imagen. El resultado de la segmentación de una imagen es un conjunto de segmentos que cubren toda la imagen sin superponerse. Se puede representar como una imagen de etiquetas (una etiqueta para cada píxel) o como un conjunto de contornos [3].

II. DESARROLLO

Importando librerias

```
from PIL import Image
from matplotlib import image
from matplotlib import pyplot
import matplotlib.pyplot as plt
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
```

imread, leer imagen RGB
img2: variable usada para almacenar la conversión de espacio de color de img.

- cvtColor tiene dos argumentos img y COLOR_RGB2GRAY
- img2: imagen original sin cambios en el espacio de color
- COLOR_BGR2RGB código de la conversión del espacio de color en este caso la conversión es de BGR a RGB

```
image2=cv2.imread('PELOTASLISAS-COLORES.jpg')
rgb2=cv2.cvtColor(image2,cv2.COLOR_BGR2RGB)
```

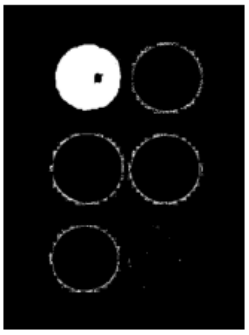
Pasando al paso de colores de hsv para facilitar la segmentación de la imagen.

```
hsv=cv2.cvtColor(image2,  cv2.COLOR_BGR2HSV)

plt.axis('off')
plt.imshow(rgb2)
plt.show()
```

```
#Blanco
```

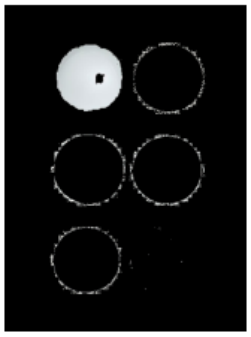
Ajuste de rango de matiz, saturación y brillo para Blanco, dado que se trabaja en el espacio de color HSV.

```
lower_B = np.array([0,5,170])
upper_B = np.array([179,20,255])
mask_B=cv2.inRange(hsv,lower_B,upper_B)

RGB_B=cv2.cvtColor(mask_B,cv2.COLOR_BGR2RGB)


plt.axis('off')
plt.imshow(RGB_B)
plt.show()
```



Volviendo al espacio de color convencional RGB.

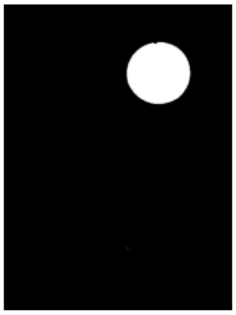
```
white=cv2.bitwise_and(image2,image2,mask=mask_B)
RGB2_B=cv2.cvtColor(white,cv2.COLOR_BGR2RGB)
plt.axis('off')
plt.imshow(RGB2_B)
plt.show()
```



```
#Amarillo
```

Ajuste de rango de matiz, saturación y brillo para Amarillo, dado que se trabaja en el espacio de color HSV.

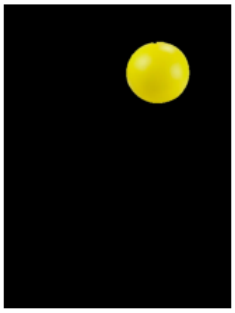
```
lower_A = np.array([22,100,20])
upper_A = np.array([35,255,255])
mask_A=cv2.inRange(hsv,lower_A,upper_A)
RGB_A=cv2.cvtColor(mask_A,cv2.COLOR_BGR2RGB
)

plt.axis('off')
plt.imshow(RGB_A)
plt.show()
```



Volviendo al espacio de color convencional RGB.

```
yellow=cv2.bitwise_and(image2,image2,mask=mask_A)
RGB2_A=cv2.cvtColor(yellow,cv2.COLOR_BGR2RGB)
```

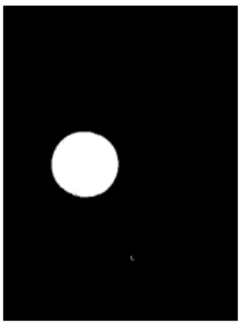
```
plt.axis('off')
plt.imshow(RGB2_A)
plt.show()
```



#Naranja

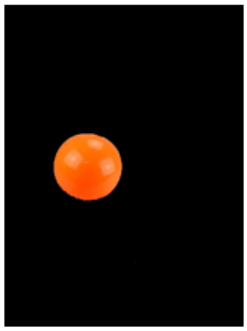
Ajuste de rango de matiz, saturación y brillo para Naranja, dado que se trabaja en el espacio de color HSV.

```
lower_N = np.array([7,100,20])
upper_N = np.array([15,255,255])
mask_N=cv2.inRange(hsv,lower_N,upper_N)
RGB_N=cv2.cvtColor(mask_N,cv2.COLOR_BGR2RGB)

plt.axis('off')
plt.imshow(RGB_N)
plt.show()
```



Volviendo al espacio de color convencional RGB.

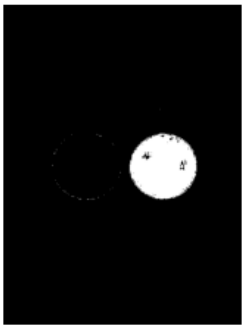
```
orange=cv2.bitwise_and(image2,image2,mask=mask_N)
RGB2_N=cv2.cvtColor(orange,cv2.COLOR_BGR2RGB)

plt.axis('off')
plt.imshow(RGB2_N)
plt.show()
```



#Rojo

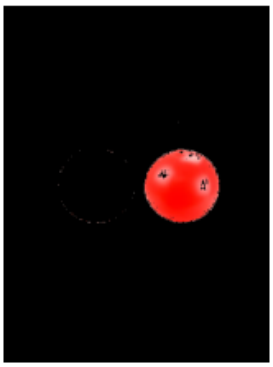
Ajuste de rango de matiz, saturación y brillo para Rojo, dado que se trabaja en el espacio de color HSV.

```
lower_R = np.array([0,10,240])
upper_R = np.array([4,255,255])
mask_R=cv2.inRange(hsv,lower_R,upper_R)
RGB_R=cv2.cvtColor(mask_R,cv2.COLOR_BGR2RGB)

plt.axis('off')
plt.imshow(RGB_R)
plt.show()
```



Volviendo al espacio de color convencional RGB.

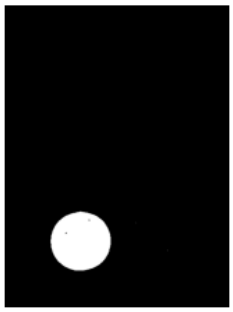
```
red=cv2.bitwise_and(image2,image2,mask=mask_R)
RGB2_R=cv2.cvtColor(red,cv2.COLOR_BGR2RGB)

plt.axis('off')
plt.imshow(RGB2_R)
plt.show()
```



#Azul

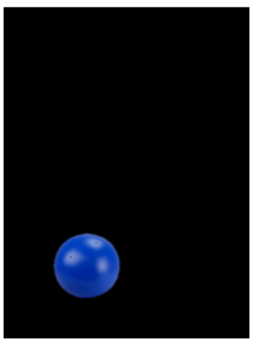
Ajuste de rango de matiz, saturación y brillo para Azul, dado que se trabaja en el espacio de color HSV.

```
lower_B = np.array([100,100,20])
upper_B = np.array([130,255,255])
mask_B=cv2.inRange(hsv,lower_B,upper_B)
RGB_B=cv2.cvtColor(mask_B,cv2.COLOR_BGR2RGB)


plt.axis('off')
plt.imshow(RGB_B)
plt.show()
```



Volviendo al espacio de color convencional RGB.

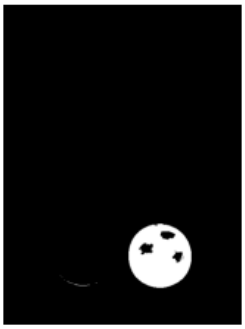
```
blue=cv2.bitwise_and(image2,image2,mask=mask_B)
RGB2_B=cv2.cvtColor(blue,cv2.COLOR_BGR2RGB)
plt.axis('off')
plt.imshow(RGB2_B)
plt.show()
```



#Negro

Ajuste de rango de matiz, saturación y brillo para Negro, dado que se trabaja en el espacio de color HSV.

```
lower_BL = np.array([0,0,0])
upper_BL = np.array([255,255,110])
mask_BL=cv2.inRange(hsv,lower_BL,upper_BL)
RGB_BL=cv2.cvtColor(mask_BL,cv2.COLOR_BGR2RGB)


plt.axis('off')
plt.imshow(RGB_BL)
plt.show()
```



Volviendo al espacio de color convencional RGB.

```
black=cv2.bitwise_and(image2,image2,mask=mask_BL)
RGB2_BL=cv2.cvtColor(black,cv2.COLOR_BGR2RGB)

plt.axis('off')
plt.imshow(RGB2_BL)
plt.show()
```

III. CONCLUSIONES

Finalmente se pudo evidenciar el funcionamieto mediante la aplicación de los rangos de colores que permiten segmentar la imagen y de esta forma filtrar el color deseado. Además, se aprendio a diferenciar los tipos de espacio de colores y a convertir de uno a otro, debido a esto se realizaron diversas

pruebas para seleccionar el espacio de color adecuado en cada segmentación para así generar un buen filtrado de las pelotas lisas de colores.

Referencias

[1] WikiPedia, «Espacio de color,» [En línea]. Available: https://es.wikipedia.org/wiki/Espacio_de_color.

[2] kipunaEc, «Cambio de espacios de color openCV - python,» [En línea]. Available: https://noemioocc.github.io/posts/Cambio-de-espacio-de-color-openCV-python/.

[3] WikiPedia, «Segmentación (procesamiento de imágenes),» [En línea]. Available: https://es.wikipedia.org/wiki/Segmentación_(procesamiento_de_imágenes).