CENTRO DE ENSEÑANZA TÉCNICA INDUSTRIAL

INGENIERÍA EN MECATRÓNICA

Visión artificial

Práctica 6:
Filtros de color HSV – RGB – YUV

Alumna:
Vanessa Aguirre Diaz

Registro:
22310274

Fecha:
26 de abril del 2025

Objetivo:

Encontrar colores específicos o remover colores específicos. (Green Screen).Hacer el filtro de rojo, verde y azul.

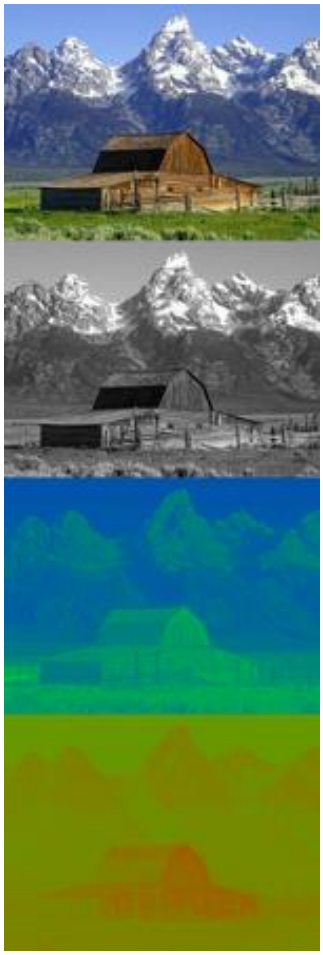
Desarrollo teórico:

Una "imagen" se refiere específicamente a una representación digital de datos visuales, a menudo en forma de una cuadrícula de píxeles. Se trata de un formato estructurado donde cada píxel tiene un valor numérico que representa su color o intensidad. Las imágenes son un tipo fundamental de datos que se utiliza en diversas tareas de aprendizaje automático, en particular en visión artificial.

Una imagen es una matriz bidimensional de valores de píxeles donde cada píxel representa la información de color en un punto específico.

Diferentes formatos de imágenes

1. Formato YUV



El espacio de color YUV se utiliza a menudo en la compresión de imágenes y vídeos, donde la imagen está representada por tres componentes:Y (luminancia o brillo), U (crominancia o luminancia azul) y V (crominancia o luminancia roja).Este formato separa la imagen en componentes que representan la información de brillo y color por separado, lo que puede ser útil para los algoritmos de compresión que aprovechan la mayor sensibilidad del ojo humano a los cambios de brillo que de color.

Componente Y (luminancia) :

- El componente Y representa el brillo o luminancia de la imagen.Determina la intensidad de cada píxel en escala de grises.
- El rango de valores normalmente varía entre 0 y 255, donde 0 representa el negro y 255 representa el blanco.

Componentes U y V (Crominancia) :

- Los componentes U y V representan la información de color de la imagen.
- U representa la diferencia entre el azul y la luminancia (Y).
- V representa la diferencia entre el rojo y la luminancia (Y).

- El rango de valores para U y V también suele variar entre 0 y 255.

Almacenamiento en memoria :

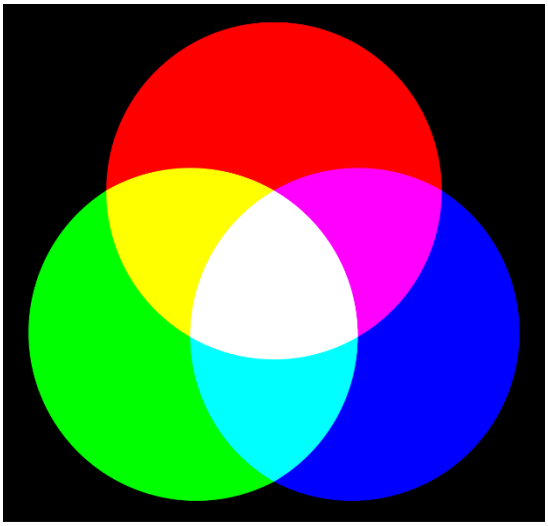
- En la memoria, una imagen en formato YUV se almacena como tres matrices o conjuntos separados:
- Una matriz para el componente Y (brillo), que normalmente se almacena como una imagen en escala de grises de 8 bits.
- Dos matrices para los componentes U y V (crominancia), cada una almacenada normalmente como canales de color de 8 bits.

El formato YUV es especialmente útil en estándares de compresión de vídeo como MPEG (Moving Picture Experts Group). Al almacenar el componente de luminancia (Y) a resolución completa y submuestrear los componentes de crominancia (U y V) a resoluciones más bajas (como 4:2:0 o 4:2:2), se pueden lograr mejoras significativas en la compresión sin una pérdida apreciable de calidad de imagen, ya que el sistema visual humano es menos sensible a los cambios en el detalle del color que al brillo.

En resumen, el formato YUV separa una imagen en sus componentes de luminancia y crominancia, ofreciendo almacenamiento y procesamiento eficientes para aplicaciones de compresión de imágenes y videos.

2. Formato RGB

El formato RGB es un modelo de color aditivo utilizado en imágenes y pantallas digitales, donde cada píxel de una imagen se representa mediante tres canales de color: rojo (R), verde (G) y azul (B). Este modelo se basa en la teoría del color aditivo, donde las diferentes intensidades de estos tres colores primarios se combinan para producir una amplia gama de colores visibles para el ojo humano.



Canales de color :

- **Rojo (R)** : Representa la cantidad de luz roja en el píxel. Un valor de 0 indica ausencia de rojo, mientras que 255 indica intensidad máxima.
- **Verde (G)** : Representa la cantidad de luz verde en el píxel. De forma similar, 0 significa ausencia de verde y 255 significa intensidad máxima.
- **Azul (B)** : Representa la cantidad de luz azul en el píxel. Los valores van de 0 (sin azul) a 255 (intensidad máxima).

El RGB utiliza la mezcla aditiva de colores, donde la combinación de diferentes intensidades de luz roja, verde y azul puede crear un amplio espectro de colores.

Representación de píxeles :

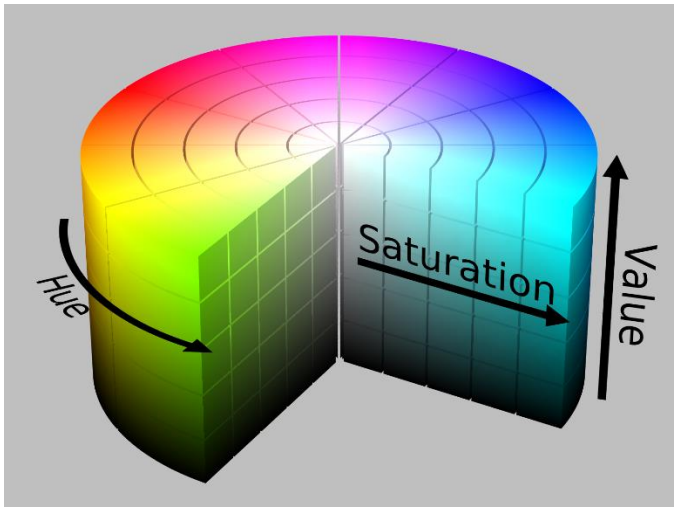
- Cada píxel de una imagen RGB se representa típicamente como un triplete de valores (R, G, B). Por ejemplo, (255, 0, 0) representa el rojo puro, (0, 255, 0) representa el verde puro y (0, 0, 255) representa el azul puro.
- Las combinaciones de estos colores primarios en diferentes intensidades producen todos los demás colores visibles al ojo humano.

3. Formato HSV

HSV (Tono, Saturación, Valor) es un modelo de color similar a HSL (Tono, Saturación, Luminosidad) pero con algunas diferencias en cómo representa y manipula los colores.Se utiliza a menudo en software de gráficos y aplicaciones de visión artificial por su simplicidad a la hora de especificar y ajustar los atributos de color.

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

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



Tono (H):

El tono representa el tipo de color en una rueda cromática. Representa el atributo de un color (rojo, azul, amarillo, etc.). Varía de 0° a 360°, donde 0° (o 360°) corresponde al rojo, 120° al verde, 240° al azul y los valores intermedios a los colores intermedios correspondientes. Por ejemplo, los tonos entre 0° y 60° representan matices de rojo, entre 60° y 120° representan matices de amarillo, y así sucesivamente en toda la rueda cromática.

Saturación (S):

La saturación se refiere a la intensidad o pureza del color. Varía del 0 al 100 %, donde el 0 % representa un tono de gris (sin color) y el 100 % representa la intensidad total del color. Los valores bajos de saturación resultan en colores más apagados o similares a la escala de grises, mientras que los valores altos resultan en colores vivos y vibrantes.

Valor (V):

El valor representa el brillo o la luminosidad (luminancia) del color. Varía del 0 al 100 %, donde el 0 % corresponde al negro y el 100 % al brillo máximo. Este componente controla la claridad u oscuridad del color. Ajustar el valor manteniendo constantes el tono y la saturación puede crear matices del mismo color.

Aplicaciones:

- Gráficos y diseño: HSV se utiliza comúnmente en software de gráficos para seleccionar y manipular colores según parámetros intuitivos.
- Visión artificial: En aplicaciones como el procesamiento de imágenes y la detección de objetos, HSV puede ser útil para tareas de segmentación de color donde es necesario aislar colores específicos de una imagen.
- Selección de color: Permite a los usuarios seleccionar colores según atributos familiares (tono, saturación y brillo), lo que puede ser más intuitivo que el RGB para ciertas tareas.

Detección de colores en Python

La detección de color es un aspecto crucial del procesamiento de imágenes y la visión artificial. Ya sea que esté desarrollando un robot que pueda identificar objetos, creando una herramienta para el análisis de imágenes basado en el color o simplemente experimentando con la manipulación de imágenes, detectar e identificar colores puede ser muy útil.

Los colores en las imágenes digitales suelen representarse mediante el espacio de color RGB, donde cada píxel se define por sus componentes rojo, verde y azul. Sin embargo, para muchas tareas de procesamiento de imágenes, el espacio de color HSV (Tono, Saturación, Valor) es más útil, ya que separa la intensidad de la imagen (brillo) de la información de color.

Pasos para la detección de color

1. Lee la imagen.
2. Convertir la imagen al espacio de color HSV.
3. Definir el rango de colores para la detección.
4. Crea una máscara para el color definido.
5. Creación de contornos, aplicación y seguimiento de máscaras y adición de texto.
6. Ejecutando el programa.

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.

Desarrollo práctico

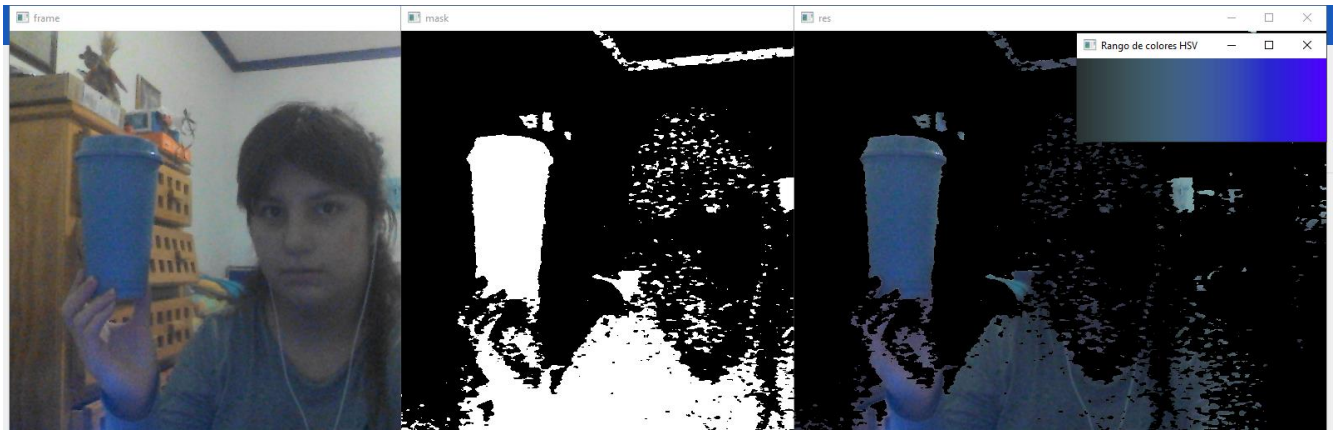
Filtros HSV

Filtro azul:

```
# Practica 6: Filtros de color HSV (AZUL)

import cv2
import numpy as np

# Inicia captura de video desde la cámara web (índice 0)
cap = cv2.VideoCapture(0)

# Define los valores HSV mínimo y máximo
# Rango mas abierto:
#min_HSV = np.array([90, 50, 50])
#max_HSV = np.array([130, 255, 255])

# Mas cerrado (MEJOR)
min_HSV = np.array([100, 100, 100])
max_HSV = np.array([115, 255, 255])

# ---------- Visualización del rango de colores ----------
height, width = 100, 300
img = np.zeros((height, width, 3), dtype=np.uint8)

for x in range(width):
    ratio = x / (width - 1)
    hsv_color = min_HSV + (max_HSV - min_HSV) * ratio
    hsv_color = np.uint8([[hsv_color]])
    bgr_color = cv2.cvtColor(hsv_color, cv2.COLOR_HSV2BGR)[0][0]
    img[:, x] = bgr_color

# Muestra la barra de colores solo una vez al inicio
```

```
cv2.imshow('Rango de colores HSV', img)

# ---------- Bucle principal para video en vivo ----------
while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Convierte el frame a HSV
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Crea una máscara con los valores del color definido
    mask = cv2.inRange(hsv, min_HSV, max_HSV)

    # Aplica la máscara al frame original
    res = cv2.bitwise_and(frame, frame, mask=mask)

    # Muestra las ventanas
    cv2.imshow('frame', frame)    # Imagen original
    cv2.imshow('mask', mask)      # Máscara binaria
    cv2.imshow('res', res)        # Resultado filtrado

    # Presiona ESC (27) para salir
    if cv2.waitKey(5) & 0xFF == 27:
        break

# Libera recursos
cap.release()
cv2.destroyAllWindows()
```

Para este filtro tenemos 2 opciones de rango: uno mas abierto y otro mas cerrado. A continuación vemos primero el que tiene un rango mas abierto:



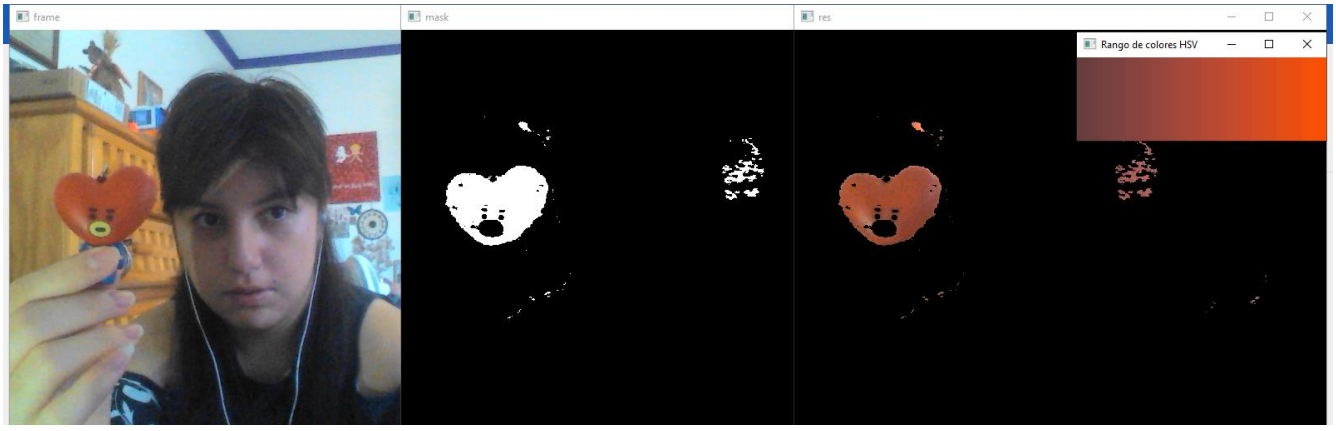
Como podemos ver, nos detecta el vaso color azul pero tambien otros elementos del entorno que son azules, aunque no del tono deseado. Para solucionar este problema y que solo detecte el objeto de interes (el vaso) ajustamos el rango de colores que admite nuestro filtro y obtenemos el siguiente resultado:

El rango de colores que capta el programa se ha limitado mas y asi solo detecta nuestro vaso. Cabe aclarar que si la iluminacion cambia, el color del vaso que capta la camara puede cambiar levemente y ya no ser detectado por el filtro, por lo que hay que cuidar ese aspecto.

Filtro rojo:

```
# Practica 6: Filtros de color HSV (ROJO)

import cv2
import numpy as np

# Inicia captura de video desde la cámara web (índice 0)
cap = cv2.VideoCapture(0)

# Define los valores HSV mínimo y máximo
min_HSV = np.array([0, 100, 100])     # Rojo puro (zona baja)
max_HSV = np.array([10, 255, 255])

# ---------- Visualización del rango de colores ----------
height, width = 100, 300
img = np.zeros((height, width, 3), dtype=np.uint8)

for x in range(width):
    ratio = x / (width - 1)
    hsv_color = min_HSV + (max_HSV - min_HSV) * ratio
    hsv_color = np.uint8([[hsv_color]])
    bgr_color = cv2.cvtColor(hsv_color, cv2.COLOR_HSV2BGR)[0][0]
    img[:, x] = bgr_color
```

```
# Muestra la barra de colores solo una vez al inicio
cv2.imshow('Rango de colores HSV', img)

# ---------- Bucle principal para video en vivo ----------
while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Convierte el frame a HSV
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Crea una máscara con los valores del color definido
    mask = cv2.inRange(hsv, min_HSV, max_HSV)

    # Aplica la máscara al frame original
    res = cv2.bitwise_and(frame, frame, mask=mask)

    # Muestra las ventanas
    cv2.imshow('frame', frame)    # Imagen original
    cv2.imshow('mask', mask)      # Máscara binaria
    cv2.imshow('res', res)        # Resultado filtrado

    # Presiona ESC (27) para salir
    if cv2.waitKey(5) & 0xFF == 27:
        break

# Libera recursos
cap.release()
cv2.destroyAllWindows()
```



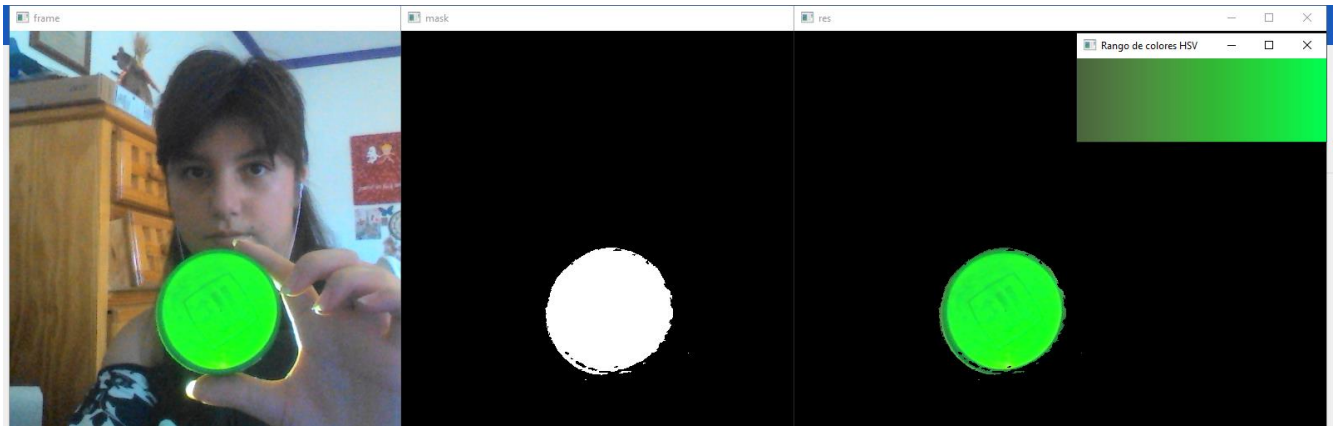
Aquí podemos ver que el filtro solo capta el color rojo del corazón y el cuadro del fondo. En este caso, ambos elementos son del mismo tono, así que aunque se ajuste mas el rango, ambos elementos seguirán siendo captados por el filtro, por lo que en una aplicación

práctica, si deseamos que solo sea captado el objeto que mostramos ante la cámara debemos cuidar que no haya elementos en el fondo con tonos similares que puedan colarse en el video.

Filtro verde:

```
# Practica 6: Filtros de color HSV (VERDE)

import cv2
import numpy as np

# Inicia captura de video desde la cámara web (índice 0)
cap = cv2.VideoCapture(0)

# Define los valores HSV mínimo y máximo
# Rango mas abierto:
#min_HSV = np.array([35, 50, 50])
#max_HSV = np.array([85, 255, 255])

# Mas cerrado
min_HSV = np.array([50, 100, 100])
max_HSV = np.array([70, 255, 255])

# ---------- Visualización del rango de colores ----------
height, width = 100, 300
img = np.zeros((height, width, 3), dtype=np.uint8)

for x in range(width):
    ratio = x / (width - 1)
    hsv_color = min_HSV + (max_HSV - min_HSV) * ratio
    hsv_color = np.uint8([[hsv_color]])
    bgr_color = cv2.cvtColor(hsv_color, cv2.COLOR_HSV2BGR)[0][0]
    img[:, x] = bgr_color

# Muestra la barra de colores solo una vez al inicio
cv2.imshow('Rango de colores HSV', img)

# ---------- Bucle principal para video en vivo ----------
while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Convierte el frame a HSV
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Crea una máscara con los valores del color definido
    mask = cv2.inRange(hsv, min_HSV, max_HSV)
```

```
    # Aplica la máscara al frame original
    res = cv2.bitwise_and(frame, frame, mask=mask)

    # Muestra las ventanas
    cv2.imshow('frame', frame)    # Imagen original
    cv2.imshow('mask', mask)      # Máscara binaria
    cv2.imshow('res', res)        # Resultado filtrado

    # Presiona ESC (27) para salir
    if cv2.waitKey(5) & 0xFF == 27:
        break

# Libera recursos
cap.release()
cv2.destroyAllWindows()
```



En este caso, fue necesario controlar la iluminación del objeto, ya que el objeto por si solo era color verde pero un verde muy oscuro, por lo que al acercarlo a la lampara y ser plástico, el tono ya se aclaró y ya lo capto el filtro.

Como nota, todos los programas cuentan con una barra que indica el rango de colores que abarca el filtro ubicada en la parte superior derecha, y como podemos ver en este caso, capta verdes muy claros. Esto se puede cambiar en la parte de valor mínimo y maximo que abarca el filtro para que capte valores mas oscuros, pero al tratarse de un modelo de color con el que uno no está muy familiarizado, es complicado realizar este ajuste, así que se dejará así por el momento.

Filtros RGB

Filtro azul:

```
# Practica 6: Filtro de color RGB (azul)

import cv2                  # Importa la librería OpenCV para visión por computadora
import numpy as np          # Importa NumPy para trabajar con arreglos numéricos

# Inicia la captura de video desde la cámara (0 es la cámara principal)
cap = cv2.VideoCapture(0)

# Define el rango mínimo y máximo del color azul en RGB
# Rango más abierto
min_RGB = np.array([0, 0, 150])       # B: alto, G: bajo, R: bajo (azul)
max_RGB = np.array([150, 150, 255])   # Azul claro sin llegar a celeste brillante

# OpenCV usa formato BGR en lugar de RGB, por lo que invertimos el orden
bgr_min = min_RGB[::-1]          # Invierte los valores a BGR (Blue, Green, Red)
bgr_max = max_RGB[::-1]

# Crear una imagen que muestre el degradado del rango BGR detectado
def create_color_bar(min_color, max_color, width=300, height=50):
    # Crea una barra horizontal con un gradiente entre los colores min y max
    bar = np.zeros((height, width, 3), dtype=np.uint8)
    for i in range(width):
        alpha = i / width
        color = (1 - alpha) * min_color + alpha * max_color
        bar[:, i] = color
    return bar

# Crear la barra de color visualmente del rango
color_bar = create_color_bar(bgr_min, bgr_max)


# ---------- Bucle principal para video en vivo -------------------------
while True:
    ret, frame = cap.read()          # Lee un frame de la cámara
    if not ret:                  # Si no se pudo leer, rompe el bucle
        break

    # Crea una máscara con los píxeles dentro del rango BGR definido (color azul)
    mask = cv2.inRange(frame, bgr_min, bgr_max)

    # Aplica la máscara al frame original: muestra solo los píxeles que estén dentro del
rango
    res = cv2.bitwise_and(frame, frame, mask=mask)
```

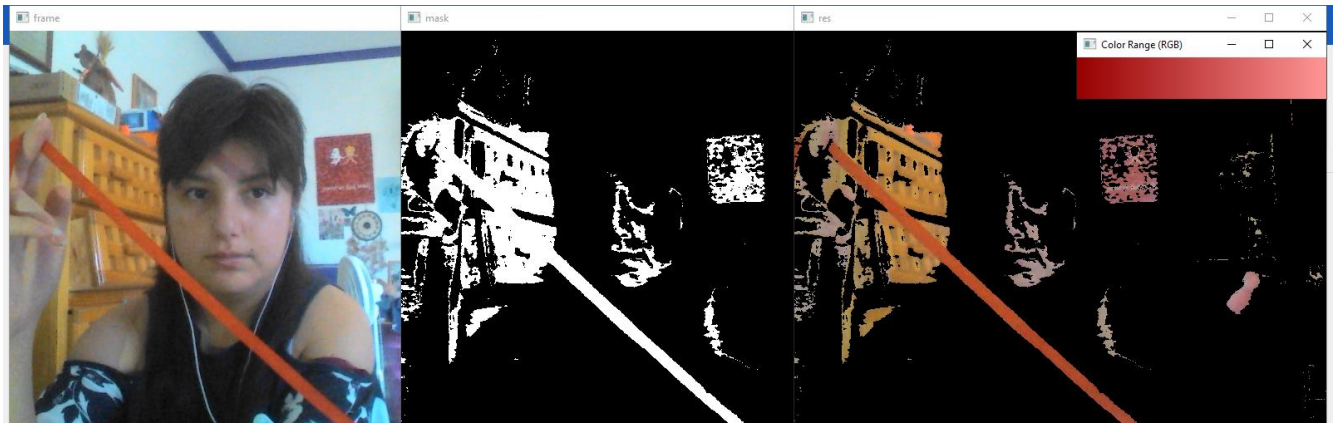
```
    # Muestra la imagen original
    cv2.imshow('frame', frame)

    # Muestra la máscara en blanco y negro (blanco = azul detectado)
    cv2.imshow('mask', mask)

    # Muestra la imagen resultante con solo los tonos azules visibles
    cv2.imshow('res', res)

    # Mostrar la barra del rango de color detectado
    cv2.imshow('Color Range (RGB)', color_bar)

    # Espera 5 ms por una tecla; si se presiona ESC (código 27), se rompe el bucle
    if cv2.waitKey(5) & 0xFF == 27:
        break

# Libera la cámara
cap.release()

# Cierra todas las ventanas abiertas
cv2.destroyAllWindows()
```

Filtro rojo:

```
#Practica 6: Filtro de color RGB (rojo)

import cv2                  # Importa la librería OpenCV para visión por computadora
import numpy as np          # Importa NumPy para trabajar con arreglos numéricos

# Inicia la captura de video desde la cámara (0 es la cámara principal)
cap = cv2.VideoCapture(0)

# Define el rango mínimo y máximo del color rojo en RGB
# Rango mas abierto
#min_RGB = np.array([150, 0, 0])       # R: alto, G: bajo, B: bajo (rojo)
#max_RGB = np.array([255, 150, 150])   # Hasta rojo más claro, sin llegar a rosado

# Rango mas cerrado
min_RGB = np.array([150, 0, 0])       # R: alto, G: bajo, B: bajo (rojo)
max_RGB = np.array([255, 100, 100])   # Hasta rojo más claro, sin llegar a rosado

# OpenCV usa formato BGR en lugar de RGB, por lo que invertimos el orden
bgr_min = min_RGB[::-1]          # Invierte los valores a BGR (Blue, Green, Red)
bgr_max = max_RGB[::-1]

# Crear una imagen que muestre el degradado del rango BGR detectado
def create_color_bar(min_color, max_color, width=300, height=50):
    # Crea una barra horizontal con un gradiente entre los colores min y max
    bar = np.zeros((height, width, 3), dtype=np.uint8)
    for i in range(width):
        alpha = i / width
        color = (1 - alpha) * min_color + alpha * max_color
        bar[:, i] = color
    return bar

# Crear la barra de color visualmente del rango
color_bar = create_color_bar(bgr_min, bgr_max)

# ---------- Bucle principal para video en vivo ----------------------------------------------
while True:
    ret, frame = cap.read()          # Lee un frame de la cámara
    if not ret:                      # Si no se pudo leer, rompe el bucle
        break

    # Crea una máscara con los píxeles dentro del rango BGR definido (color rojo)
    mask = cv2.inRange(frame, bgr_min, bgr_max)

    # Aplica la máscara al frame original: muestra solo los píxeles que estén dentro del
rango
```

```
    res = cv2.bitwise_and(frame, frame, mask=mask)

    # Muestra la imagen original
    cv2.imshow('frame', frame)

    # Muestra la máscara en blanco y negro (blanco = rojo detectado)
    cv2.imshow('mask', mask)

    # Muestra la imagen resultante con solo los tonos rojos visibles
    cv2.imshow('res', res)

    # Mostrar la barra del rango de color detectado
    cv2.imshow('Color Range (RGB)', color_bar)

    # Espera 5 ms por una tecla; si se presiona ESC (código 27), se rompe el bucle
    if cv2.waitKey(5) & 0xFF == 27:
        break

# Libera la cámara
cap.release()

# Cierra todas las ventanas abiertas
cv2.destroyAllWindows()
```

Con este filtro probamos modificar el rango de rojos del filtro como en el caso del filtro HSV azul anterior. Primero con un rango mas abierto, lo que nos dio como resultado esto:



Debido a que en el ambiente existen elementos con tonos rojizos como el mueble de madera del fondo y el tono de mi piel, se optó por cerrar el rango de colores que capta a uno mas cercano al color real del liston que se desea captar:

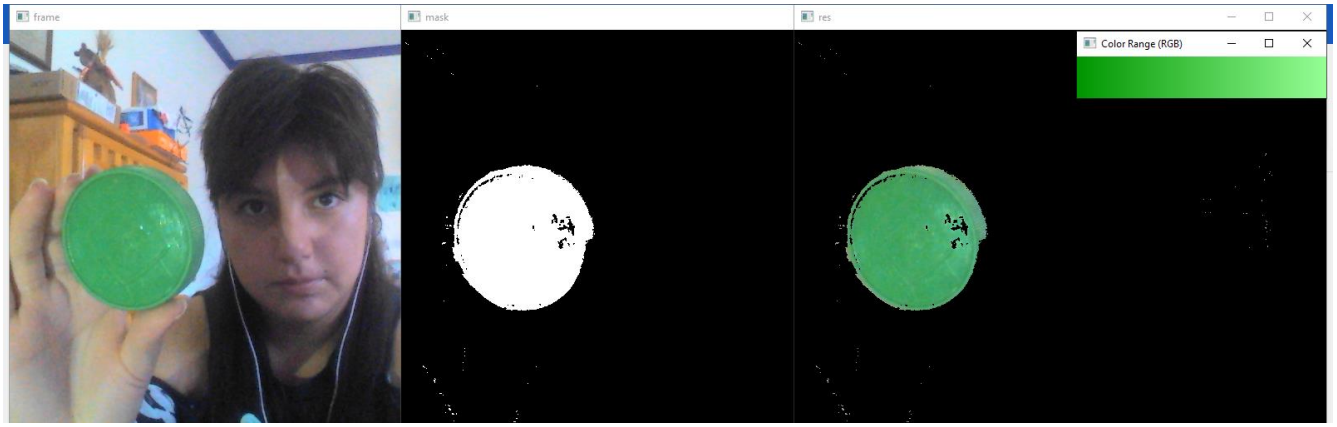
Con esto deja de captar tonos mas bajos de rojos y solo capta el listón rojo que es la zona de interés. El cuadro del fondo sigue presente, mismo caso del filtro rojo HSV, ya que son ambos de un tono similar. Para solucionarlo podríamos cerrar mas el rango de colores a uno mas cercano al del listón, pero un ligero cambio en la iluminación puede hacer que este ya no se vea, por eso no conviene cerrar tanto el rango del filtro en este caso donde no tenemos una iluminación fija y buena.

Filtro verde:

```
# Practica 6: Filtro de color RGB (verde)
import cv2                  # Importa la librería OpenCV para visión por computadora
import numpy as np          # Importa NumPy para trabajar con arreglos numéricos

# Inicia la captura de video desde la cámara (0 es la cámara principal)
cap = cv2.VideoCapture(0)

# Define el rango mínimo y máximo del color verde en RGB
min_RGB = np.array([0, 150, 0])       # G: medio-alto, R y B bajos
max_RGB = np.array([150, 255, 150])   # Verde claro, permite mezcla con un poco de azul o rojo

# OpenCV usa formato BGR en lugar de RGB, por lo que invertimos el orden
bgr_min = min_RGB[::-1]         # Invierte los valores a BGR (Blue, Green, Red)
bgr_max = max_RGB[::-1]

# Crear una imagen que muestre el degradado del rango BGR detectado
def create_color_bar(min_color, max_color, width=300, height=50):
    # Crea una barra horizontal con un gradiente entre los colores min y max
    bar = np.zeros((height, width, 3), dtype=np.uint8)
    for i in range(width):
        alpha = i / width
        color = (1 - alpha) * min_color + alpha * max_color
        bar[:, i] = color
    return bar
```

```
# Crear la barra de color visualmente del rango
color_bar = create_color_bar(bgr_min, bgr_max)

# ---------- Bucle principal para video en vivo -------------------------
while True:
    ret, frame = cap.read()          # Lee un frame de la cámara
    if not ret:                      # Si no se pudo leer, rompe el bucle
        break

    # Crea una máscara con los píxeles dentro del rango BGR definido (color verde)
    mask = cv2.inRange(frame, bgr_min, bgr_max)

    # Aplica la máscara al frame original: muestra solo los píxeles que estén dentro del
rango
    res = cv2.bitwise_and(frame, frame, mask=mask)

    # Muestra la imagen original
    cv2.imshow('frame', frame)

    # Muestra la máscara en blanco y negro (blanco = verde detectado)
    cv2.imshow('mask', mask)

    # Muestra la imagen resultante con solo los tonos verdes visibles
    cv2.imshow('res', res)

    # Mostrar la barra del rango de color detectado
    cv2.imshow('Color Range (RGB)', color_bar)

    # Espera 5 ms por una tecla; si se presiona ESC (código 27), se rompe el bucle
    if cv2.waitKey(5) & 0xFF == 27:
        break

# Libera la cámara
cap.release()

# Cierra todas las ventanas abiertas
cv2.destroyAllWindows()
```

En este caso, a comparación del filtro verde HSV, no fue necesario cuidar tanto el tomo de la tapa, ya que abarca una mayor cantidad de tonos de verde que van desde uno claro a uno mas oscuro. Fue mas fácil manejar estos tonos de verde debido a que el formato RGB es uno con el que estoy mas familiarizada y en general es más fácil de ajustar debido a que se basa en la combinación de colores y su proporción, no en su tono o saturación que son parámetros que uno no esta acostumbrado a manipular y combinar.

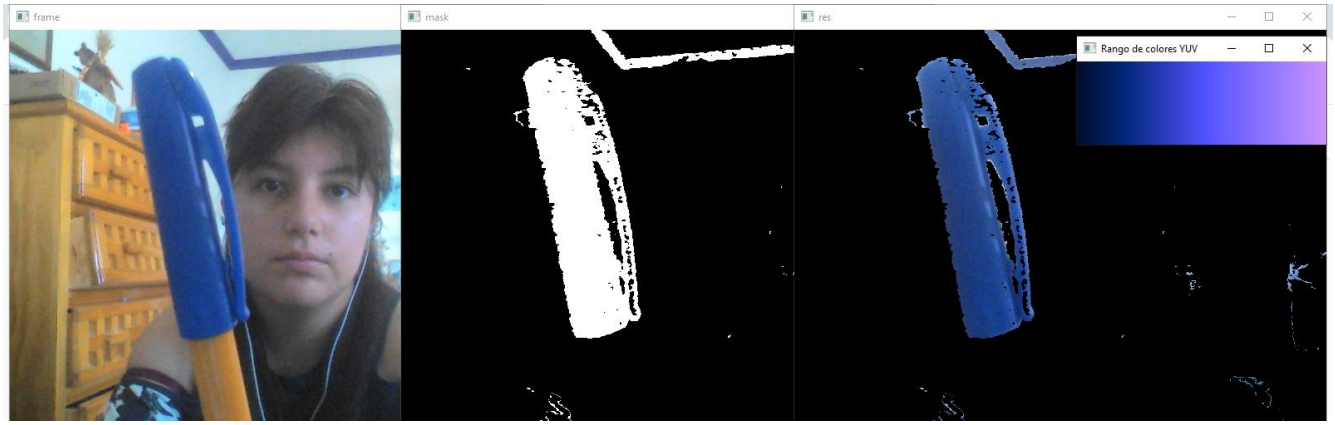


Filtros YUV

Filtro azul:

```
# Practica 6: Filtro de color YUV (AZUL)

import cv2
import numpy as np

# Inicia captura de video desde la cámara web
cap = cv2.VideoCapture(0)

# Define los valores YUV mínimo y máximo para detectar azul
# El canal U es el que representa los tonos azulados
min_YUV = np.array([0, 150, 90])      # Y bajo, U alto (azul), V medio
max_YUV = np.array([200, 255, 130])    # Y alto, U máximo, V bajo

# ---------- Visualización del rango de colores YUV ----------
height, width = 100, 300
img = np.zeros((height, width, 3), dtype=np.uint8)

for x in range(width):
    ratio = x / (width - 1)
    yuv_color = min_YUV + (max_YUV - min_YUV) * ratio
    yuv_color = np.clip(yuv_color, 0, 255).astype(np.uint8)
    bgr_color = cv2.cvtColor(np.uint8([[yuv_color]]), cv2.COLOR_YUV2BGR)[0][0]
    img[:, x] = bgr_color

# Muestra la barra de colores solo una vez al inicio
cv2.imshow('Rango de colores YUV', img)

# ---------- Bucle principal para video en vivo ----------
while True:
    ret, frame = cap.read()
```

```
    if not ret:
        break

    # Convierte el frame a YUV
    yuv = cv2.cvtColor(frame, cv2.COLOR_BGR2YUV)

    # Crea una máscara con los valores del color azul en YUV
    mask = cv2.inRange(yuv, min_YUV, max_YUV)

    # Aplica la máscara al frame original
    res = cv2.bitwise_and(frame, frame, mask=mask)

    # Muestra las ventanas
    cv2.imshow('frame', frame)
    cv2.imshow('mask', mask)
    cv2.imshow('res', res)

    # Presiona ESC para salir
    if cv2.waitKey(5) & 0xFF == 27:
        break

# Libera recursos
cap.release()
cv2.destroyAllWindows()
```



Filtro rojo:

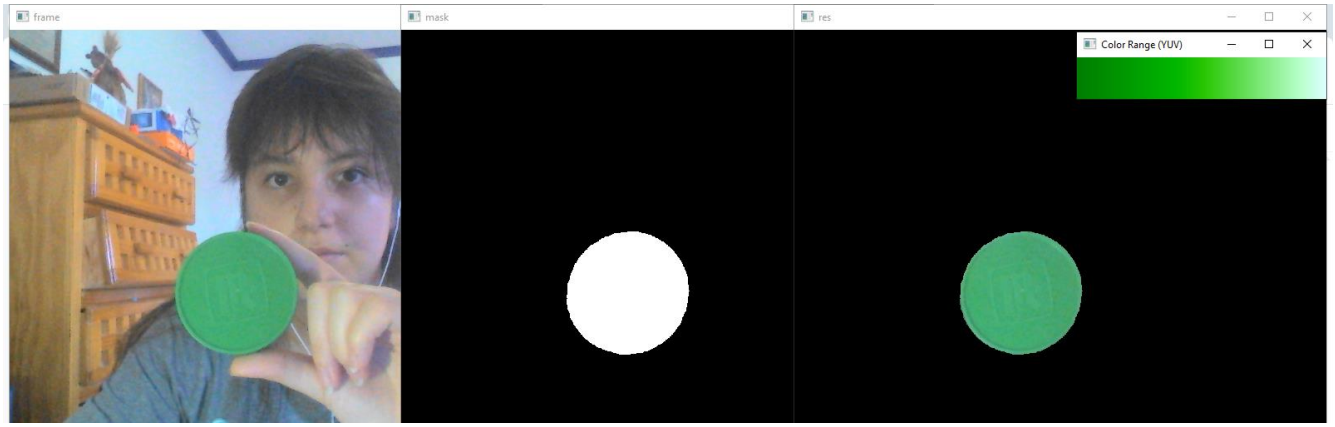
```
# Practica 6: Filtro de color YUV (rojo)

import cv2
import numpy as np

# Inicia la captura de video desde la cámara principal
cap = cv2.VideoCapture(0)

# Define el rango mínimo y máximo del color rojo en YUV
# El canal V (rojo) debe ser alto para detectar tonos rojos
min_YUV = np.array([0, 90, 170])     # Y bajo, U medio, V alto
max_YUV = np.array([200, 140, 255])   # Y alto, U medio, V máximo

# Crear una barra de gradiente visual basada en el rango YUV
def create_color_bar_yuv(min_color, max_color, width=300, height=50):
    bar = np.zeros((height, width, 3), dtype=np.uint8)
    for i in range(width):
        alpha = i / width
        yuv_color = (1 - alpha) * min_color + alpha * max_color
        yuv_color = np.clip(yuv_color, 0, 255).astype(np.uint8)
        bgr_color = cv2.cvtColor(np.uint8([[yuv_color]]), cv2.COLOR_YUV2BGR)[0][0]
        bar[:, i] = bgr_color
    return bar

# Crear la barra de color
color_bar = create_color_bar_yuv(min_YUV, max_YUV)

# ---------- Bucle principal para video en vivo -------------------------
while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Convertir el frame de BGR a YUV
    yuv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2YUV)

    # Crear una máscara con los píxeles dentro del rango YUV definido (color rojo)
    mask = cv2.inRange(yuv_frame, min_YUV, max_YUV)

    # Aplicar la máscara al frame original
    res = cv2.bitwise_and(frame, frame, mask=mask)

    # Mostrar la imagen original
    cv2.imshow('frame', frame)
```

```
    # Mostrar la máscara en blanco y negro
    cv2.imshow('mask', mask)

    # Mostrar la imagen resultante con los tonos verdes detectados
    cv2.imshow('res', res)

    # Mostrar la barra de color
    cv2.imshow('Color Range (YUV)', color_bar)

    # Salir si se presiona ESC
    if cv2.waitKey(5) & 0xFF == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

Filtro verde:

```
# Practica 6: Filtro de color YUV (verde)

import cv2                  # Importa la librería OpenCV para visión por computadora
import numpy as np          # Importa NumPy para trabajar con arreglos numéricos

# Inicia la captura de video desde la cámara (0 es la cámara principal)
cap = cv2.VideoCapture(0)

# Define el rango mínimo y máximo del color verde en YUV
# Estos valores pueden ajustarse según el tono de verde que se quiera detectar
min_YUV = np.array([0, 0, 0])        # Límite inferior (Y, U, V)
max_YUV = np.array([255, 130, 100])   # Límite superior

# Crear una imagen que muestre el degradado del rango YUV detectado (no muy
representativo, pero se mantiene visual)
```

```
def create_color_bar_yuv(min_color, max_color, width=300, height=50):
    bar = np.zeros((height, width, 3), dtype=np.uint8)
    for i in range(width):
        alpha = i / width
        yuv_color = (1 - alpha) * min_color + alpha * max_color
        bgr_color = cv2.cvtColor(np.uint8([[yuv_color]]), cv2.COLOR_YUV2BGR)[0][0]
        bar[:, i] = bgr_color
    return bar

color_bar = create_color_bar_yuv(min_YUV, max_YUV)

# ---------- Bucle principal para video en vivo -------------------------
while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Convertir el frame de BGR a YUV
    yuv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2YUV)

    # Crear máscara con los píxeles dentro del rango YUV definido
    mask = cv2.inRange(yuv_frame, min_YUV, max_YUV)

    # Aplicar la máscara al frame original
    res = cv2.bitwise_and(frame, frame, mask=mask)

    # Mostrar la imagen original
    cv2.imshow('frame', frame)

    # Mostrar la máscara en blanco y negro
    cv2.imshow('mask', mask)

    # Mostrar la imagen resultante con los tonos verdes detectados
    cv2.imshow('res', res)

    # Mostrar la barra de color
    cv2.imshow('Color Range (YUV)', color_bar)

    # Salir si se presiona ESC
    if cv2.waitKey(5) & 0xFF == 27:
        break

cap.release()
cv2.destroyAllWindows()
```



Fuentes:

https://medium.com/@abhishekjainindore24/all-about-images-and-their-formats-1bcba5c854e7

https://agneya.medium.com/color-detection-using-python-and-opencv-8305c29d4a42

https://pythonprogramming.net/color-filter-python-opencv-tutorial/