Universidad Autónoma de Baja California

Práctica 5 Procesamiento de imagen y color

MATERIA: Lenguaje de Programación Python

Carrera: Ingeniería en Computación

Maestro: Adolfo Heriberto Ruelas Puente

Alumno: David Salazar Sánchez

Matricula: 1141849

Web: https://davidsz.web.app

1. Realiza un programa en el cual muestre en tiempo real, la imagen captada por su webcam, en donde se mostrara en blanco y negro la imagen, resaltando en blanco los bordes.

Para poder captutar la imagen de vídeo de nuestra Webcam lo podemos hacer usando la siguiente función.

```
cap = cv2.VideoCapture(0)
```

Con está instrucción le indicamos de que canal queremos tomar la señal de vídeo y la guardamos en la variable *cap*.

Para hacer una lectura constante y que podamos ver la imagen en tiempo real hacemos la lectura de la imagen dentro de un ciclo *while* infinito. Para leer la imagen usamos el método *read()*, este método nos devuelve una dupla. El primero sera un booleano que nos indicara si la instrucción se ejecutó con éxito y el segundo es el *frame* o la imagen de la Webcam.

Para evitar que se nos abra una ventana muy grande, ajustamos su tamaño con la siguiente línea.

```
    while(1):

        _, frame = cap.read()
        webCam = cv2.resize(frame, (0, 0), fx=0.9,
fy=0.9)
```

Para poder pasar la imagen que nos devuelve la Webcam que es a color, a blanco y negro, cambiamos el espectro de color usando *cv2. COLOR_BGR2GRAY*. Con esta instrucción le decimos al método *cv2.cvtColor()* que cambié de color a blanco y negro o escala de grises.

```
# Pasando la imagen de webCam a Blanco y Negro

        webCamGray = cv2.cvtColor(webCam,
cv2.COLOR_BGR2GRAY)
```

La salida de este método lo guardamos en una nueva variable *webCamGray*.

Con openCV tenemos un método que ya detecta los bordes de una imagen. El método *canny()* recibe tres parámetros, el primero sera la imagen para trabajar, el segundo un umbral mejor y el tercero también sera el mismo umbral pero mayor. Es decir, este umbral le indica al metodo dentro de que rango quieremos dectar un borde.

```
        # Detectar bordes usando Canny

        umbral = 100
        cny = cv2.Canny(webCamGray, umbral, umbral * 2)
```

En éste caso declaramos una varibale *umbral = 100* y usamos esta misma al doblé para el umbral mayor. Éste número se seleccionó aleatoria mente hasta dar con el número que mejor se adecue al resultado que estamos buscando.

Para mostrar el resultado obtenido usamos el método *cv2.imshow()*, este método recibe com parámetros la imagen a mostrar y en este caso le pasamos un string con el nombre de la ventana.

Además para poder cerrar la ventana y el ciclo infinito definido anteriormente utilizamos un condicional que sale del ciclo cuando se preciona la tecla "*q*" de nuestro teclado.

```
        cv2.imshow('webCam', cny)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cv2.destroyAllWindows()
```

Para finalizar cerramos todas las ventanas que abrimos.

2. Realizar un programa en tiempo real en donde muestre la captura de la imagen en blanco, con excepción de el color rojo, el color rojo debe mostrarse como es.

En este segundo ejercicio utilizamos el mismo método para vizualizar la imagen de nuestra webcam.

```
def run():
    cap = cv2.VideoCapture(0)

    while(1):
        _, frame = cap.read()
        webCam = cv2.resize(frame, (0, 0), fx=0.9,
fy=0.9)
```

Para poder hacer el filtrado de un color en específico, es necesario primero definir un rango de este color. Ya que si especificamos un color en concreto será muy difícil que el programa detecte solo ese color de una webcam. Esto debido a los tonos del objeto, iluminación y calidad de nuestra cámara.

Para hacer este rango de colores es necesario importar las librerias *cv2* y *numpy*.
Definimos un método que reciba como parámetro la imagen de nuestra webcam. Realizamos una conversión del espectro de color, ya que por default la camara nos devuelve una imagen en BGR y es más sencillo hacer el filtrado en el espectro de HSV. Declaramos dos arreglos, uno con los colores mas claros del rojo y otro con los colores más oscuros del rojo. Al final usamos el método *inRange()* para pasarle la imagen en el espectro HSV y los dos arreglos con los rangos de color.

```
import cv2
import numpy as np

def getMask(img):
    # Convert BGR to HSV
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    # define range of red color in HSV (hue,saturation,
value)
    lower_red = np.array([135, 100, 20], np.uint8)
    upper_red = np.array([179, 255, 255], np.uint8)
    mask = cv2.inRange(hsv, lower_red, upper_red)
    return mask
```

Como vemos, este método nos devuelve una máscara. Está máscara será la que nos sirva para filtrar el rango de color especificado.

Usamos esa máscara y hacemos una multiplicación binaria de nuestra webcam. El resultado sera la imagen de nuestra webcam pero cambiando a color negro todo lo que el programa haya detectado como color rojo, dentro del rango que indicamos.

```
mask = getMask(webCam)
        res = cv2.bitwise_and(webCam, webCam,
mask=mask)
```

El resultado obtenido lo utilizamos para realizarle una inversión binaria. Es decir, invertimos cada bit de la imagen. Si era negro lo cambiamos a blanco.

```
resInv = cv2.bitwise_not(res)
```

El resultado de la operación anterior lo guardamos en una nueva variable. Este nuevo resultado lo utilizamos para realizarle una multiplicación binaria nuevamente, pero como ya dijimos usando *resInv* como parámetro para el método.

```
res2 = cv2.bitwise_and(resInv, resInv, mask=mask)
res2 = cv2.bitwise_not(res2)
```

Nuevamente invertimos el resultado y de esta forma podemos obtener el fondo de nuestra imagen quede en color blanco y solo mostrar el color rojo tal cual se ve.

Ahora solo mostramos ambas imágenes y definimos la lógica para cerrar y terminar el programa.

```
        cv2.imshow('Deteccion del color Rojo', res2)
        cv2.imshow('WebCam', webCam)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cv2.destroyAllWindows()
```

3. Realizar un programa en la cual muestre en tiempo real la imagen captada por su webcam, en la cual salgan ustedes con una camiseta de un color especifico, y esta tiene que ser sustituida por un fondo de su elección (otra imagen).

Para este ejercicio definimos un de forma un poco distinta la manera en la que tomamos la imagen de nuestra webcam y la forma en la que realizamos el filtro del color rojo.

```
import cv2
import numpy as np

cap = cv2.VideoCapture(0)

bg = None

rojoBajo1 = np.array([0, 150, 40], np.uint8)
rojoAlto1 = np.array([8, 255, 255], np.uint8)
rojoBajo2 = np.array([170, 150, 40], np.uint8)
rojoAlto2 = np.array([180, 255, 255], np.uint8)

while True:

    ret, frame = cap.read()
    if ret == False:
        break

    if bg is None:
        bg = frame

    frameHSV = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    maskRojo1 = cv2.inRange(frameHSV, rojoBajo1,
rojoAlto1)
    maskRojo2 = cv2.inRange(frameHSV, rojoBajo2,
rojoAlto2)
    mask = cv2.add(maskRojo1, maskRojo2)
    mask = cv2.medianBlur(mask, 13)
```

Utilizamos la máscara obtenida y la dilatamos para mejorar la detección usando un kernel que sera un arreglo de unos.

```
    kernel = np.ones((5, 5), np.uint8)
    mask = cv2.dilate(mask, kernel, iterations=2)
```

Después definimos un área con el color a detectar combinando lo que tenemos en la variable *bg* y nuestra máscara. El *bg* es una toma del fondo que se obtiene al inicio de la toma de la imagen de nuestra webcam.
Posteriormente invertimos la máscara y repetimos el proceso pero ahora usando *frame* y *maskInv* para detectar el área sin color.

```
areaColor = cv2.bitwise_and(bg, bg, mask=mask)
    maskInv = cv2.bitwise_not(mask)
    sinAreaColor = cv2.bitwise_and(frame, frame,
mask=maskInv)
```

Para finializar sumamos ambas áreas *areaColor* y *sinAreaColor*. Esto nos devolverá una imagen en la que podemos ver una imagen en lugar del color especificado.

```
    finalFrame = cv2.addWeighted(areaColor, 1,
sinAreaColor, 1, 0)
    cv2.imshow('Frame', frame)
    cv2.imshow('mask', mask)
    cv2.imshow('finalFrame', finalFrame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

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

4. Realizar un programa el cual muestre en tiempo real la imagen captada por su webcam, en la cual salgan ustedes con una camiseta de un color especifico, y esta tiene que cambiar de color de su elección.

En este ejercicio en especial usamos el código que se uso en los primeros ejercicios, ya que requerimos de exactamente los mismos pasos. Tomamos la imagen de la webcam y obtenemos una máscara para combinarlas y hacer un filtrado del color rojo.

```
def run():
    cap = cv2.VideoCapture(0)

    while(1):
        _, frame = cap.read()
        webCam = cv2.resize(frame, (0, 0), fx=0.9,
fy=0.9)

        mask = getMask(webCam)
        mask = cv2.medianBlur(mask, 13)
        mask = cv2.bitwise_not(mask)
        res = cv2.bitwise_and(webCam, webCam,
mask=mask)
```

Utilizamos la imagen obtenida en *res* y le aplicamos un ajuste al arreglo. Le indicamos que todos los bits de color negro de la imagen los pase a un color azul. Con esto realizamos un cambio de color y se coloca una especie de filtro para el color rojo.

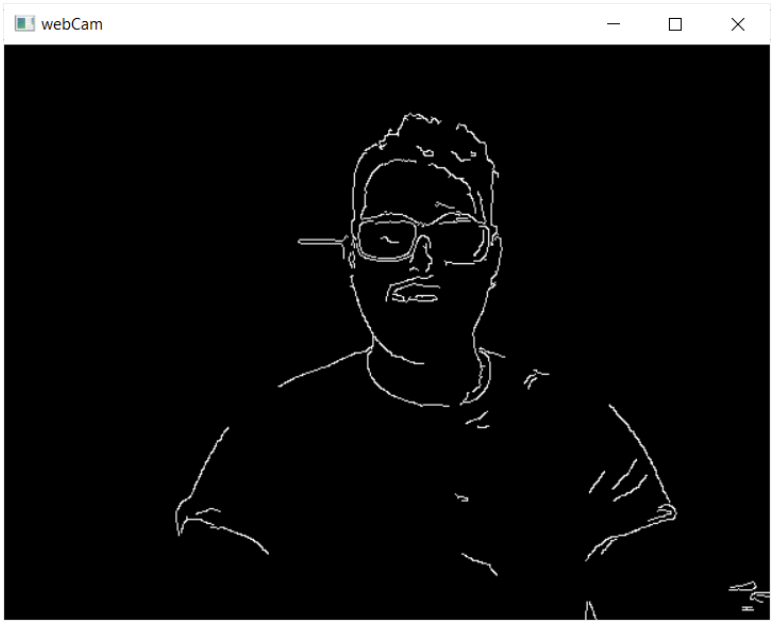
```
res[np.where((res == [0, 0, 0]).all(axis=2))] = [255,
50, 0]
```

Ahora solo combinamos la salida original de nuestra webcam y el resultado obtenido en *res* y como imagen final tenemos un cambio de color rojo a color azul.

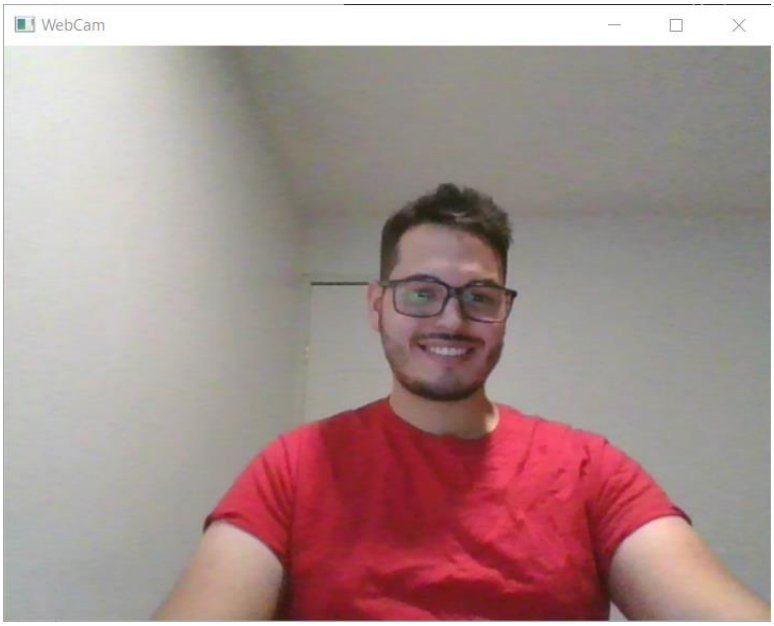
```
        res = cv2.bitwise_and(webCam, res)


        cv2.imshow('Cambio del color rojo', res)
        cv2.imshow("Mask", mask)
        cv2.imshow('WebCam', webCam)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cv2.destroyAllWindows()
```

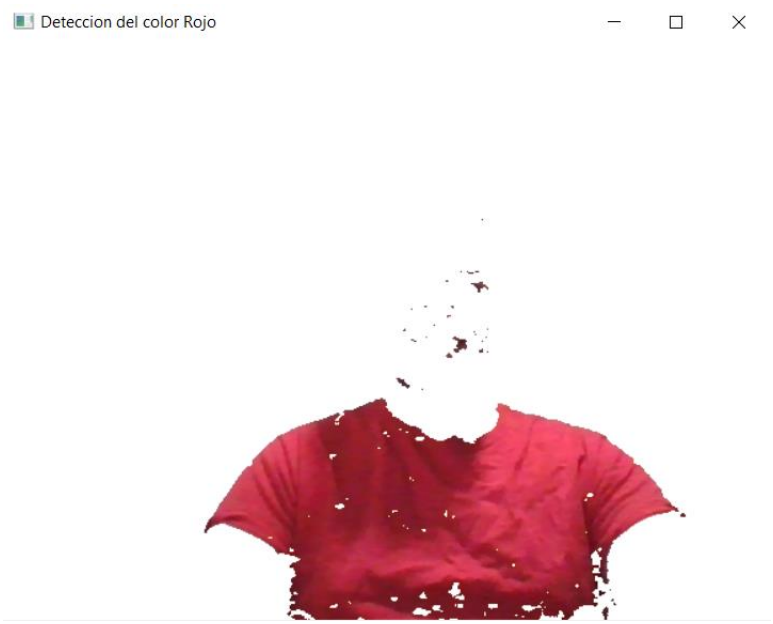
RESULTADOS

1.

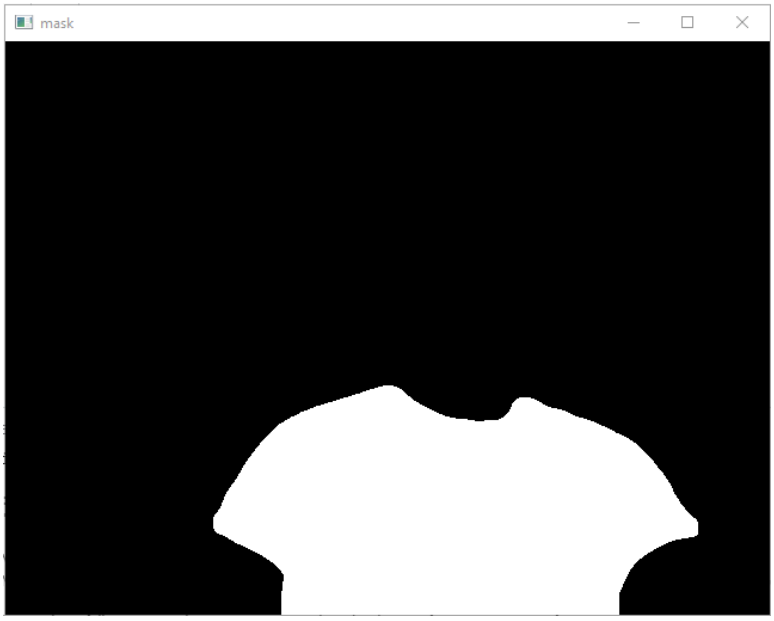


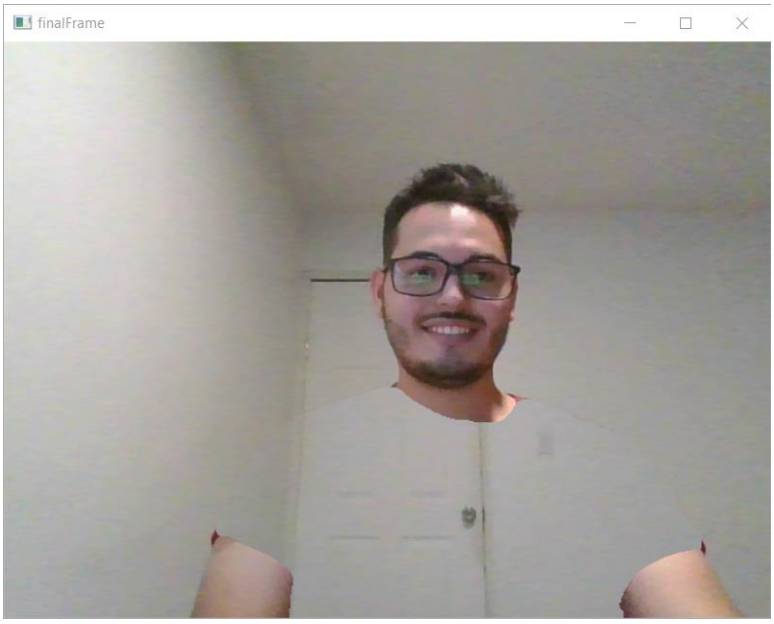
2.



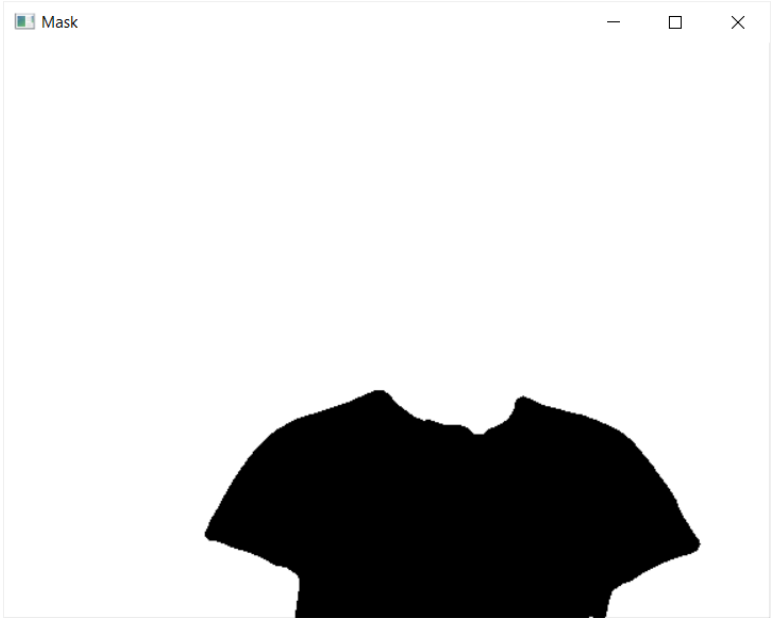


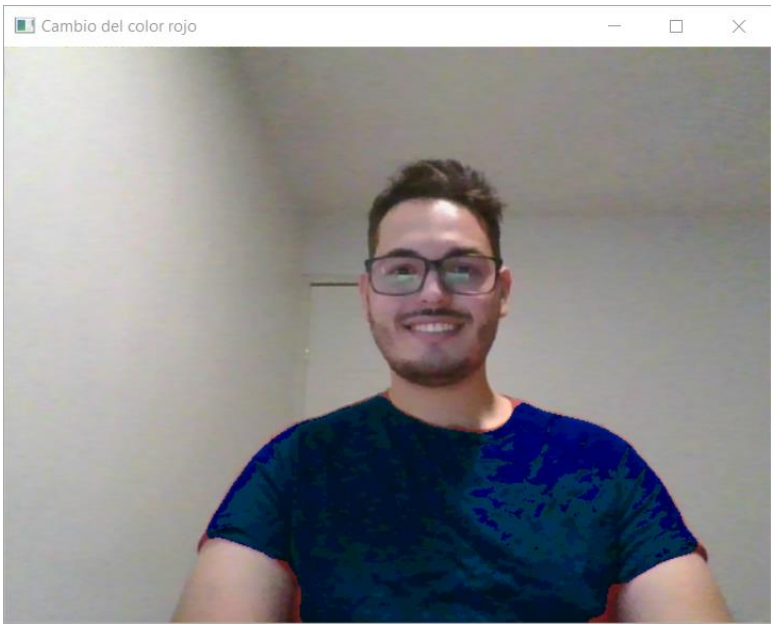
3.





4.




Cambio del color rojo

CONCLUSIÓN

Como podemos ver, los resultados obtenidos en todos los ejercicios se puede con muy pocas líneas de código gracias a todos los métodos que tienen las librerías de openCV y en algunos caso la de numpy. Es soprendente lo que se puede lograr y lo relativamente fácil que es hacer aplicaciones prácticas si se quiere desarrollar más a fondo.

En el desarrollo de esta práctica solo se tuvieron algunos inconvenientes con la lógica de que imagen combinar, invertir, excluir primero. Pero se solucionó haciendo una pequeña depuración paso a paso para identificar cada una de las salidas y el resultado que se deseaba.