Proyecto OpenCV

Autor: Carlos Martín Muelas

Índice

Planteamiento 3
Manual de usuario 3
Código 5
crear_filtros.py 5
Librerías 5
Variables globales: 5
Funciones 6
usar_filtros.py 10
Librerías 10
Función principal 10
Problemas encontrados y posibles mejoras 13

Planteamiento

El proyecto trata sobre un programa que permite al usuario poder dibujar filtros de modo que puedan ser utilizados con la cámara de su ordenador en tiempo real.

Uno de los enfoques que se le podría dar sería convertirlo en una aplicación para móviles de forma que sería mucho más accesible, por lo que su uso quizás se vería incrementado.

La interacción con el programa es simple e intuitiva, pues hasta un niño podría manejarlo sin problemas.

De hecho, lo que llevó a desarrollar este proyecto era la idea de poder crear algo que pudiera ser interactivo y atractivo para los más pequeños de la casa.

Manual de usuario

El programa consta de dos partes separadas. Por un lado, está la creación de los filtros y por otro, el uso de estos.

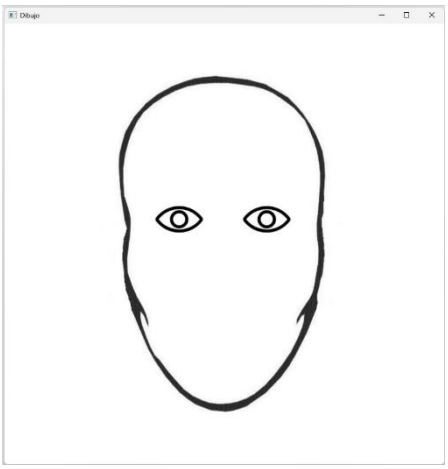
En la parte de la creación de filtros se encuentran las siguientes ventanas.

Ventana Controles

Sirve para poder elegir entre los distintos colores a utilizar en la plantilla, así como para seleccionar el nivel de grosor del lápiz.

Ventana Dibujo

En ella se muestra una plantilla de una cara y permite dibujar en ella.



En caso de que el usuario se equivoque dibujando, podrá borrar todo y empezar de cero pulsando la tecla ***b***.

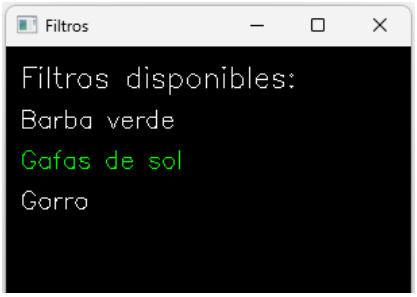
Para guardarlo habría que pulsar la tecla ***g*** e introducir un nombre. Finalmente, para salir del modo creación simplemente habría que presionar ***q***.

Por otro lado, como ya se ha mencionado, está la parte donde el usuario puede utilizar estos filtros.

Al igual que antes, también está dividido en dos ventanas.

Ventana Filtros

Es una ventana de apoyo que sirve para visualizar una lista con todos los filtros que hay almacenados y que por tanto se encuentran disponibles para su uso. Además, resalta en un color llamativo el filtro que se esté utilizando en el momento.



Ventana Webcam

En esta pestaña se abre la cámara del ordenador y al detectar una cara coloca el filtro de la lista anterior elegido por el usuario. A su vez, para más ayuda se indica el nombre del filtro que se está utilizando.

Los controles de esta ventana se basan en la navegación entre filtros con las teclas **n** y ***b*** para poder avanzar y retroceder entre ellos respectivamente. Al igual que en la parte anterior se debe presionar ***q*** para salir.

Código

crear_filtros.py

Esta parte del programa permite la creación de nuevos filtros para que sean almacenados posteriormente. Su funcionamiento es bastante intuitivo ya que simplemente hay que seleccionar el color y/o el grosor y comenzar a dibujar sobre la pizarra.

Librerías

OpenCV:

- Carga, muestra y guarda imágenes.
- Detecta y maneja eventos de mouse en la ventana de dibujo, lo cual permite la funcionalidad de dibujo interactivo.
- Permite crear la interfaz.

NumPy:

- Crea una imagen en blanco que representa la paleta de colores y el área de control de grosor.
- Crea una máscara para el dibujo que permite gestionar la transparencia (canal alfa) al guardar la imagen.
- Utiliza operaciones en matrices para manejar la transparencia y la superposición en las operaciones de dibujo.

OS:

- Verifica si existe una carpeta llamada filtros en la que se guardarán las imágenes, y la crea si no está presente.
- Construye la ruta donde se guardará la imagen dibujada con transparencia (el filtro creado).

Variables globales:

- **color:** identifica el color seleccionado.
- **grosor_seleccionado:** identifica el grosor seleccionado.
- **dibujando:** variable booleana que permite habilitar el modo de dibujo.
- **paleta_altura:** altura de la paleta de colores.
- **ancho_ventana:** define el ancho de la ventana con la plantilla.
- **altura_ventana:** define la altura de la ventana con la plantilla.
- **ancho_controles:** define el ancho de la ventana con la paleta.
- **colores_paleta:** lista con todos los colores que hay en la paleta.
- **niveles_grosor:** lista de los grosores disponibles.
- **img:** imagen principal para dibujar.
- **mask:** máscara usada para almacenar las partes del dibujo.

Funciones

pintar(evento, x, y, flags, param)

Esta función controla el evento de dibujo en la ventana principal. Es llamada cada vez que el usuario interactúa con el mouse en la ventana de dibujo.

El parámetro ***event*** indica el tipo de evento del mouse (presionar, mover, soltar), y ***x***, ***y*** son las coordenadas donde ocurre el evento.

Cuando el usuario hace clic izquierdo (***EVENT_LBUTTONDOWN***), activa el modo de dibujo estableciendo ***dibujando = True*** y guarda la posición inicial.

Si el usuario mueve el mouse (***EVENT_MOUSEMOVE***) y está en modo dibujo, se dibuja una línea desde la posición anterior hasta la actual en una copia temporal (***overlay***) de la imagen. Luego, se superpone sobre la imagen principal y se actualiza la máscara.

Al soltar el botón izquierdo (***EVENT_LBUTTONUP***), el modo de dibujo se desactiva.

```
def pintar(event, x, y, flags, param):
    global x_prev, y_prev, color, grosor_seleccionado, dibujando, mask

    if event == cv2.EVENT_LBUTTONDOWN:
        dibujando = True
        x_prev, y_prev = x, y
    elif event == cv2.EVENT_MOUSEMOVE and dibujando:
        overlay = img.copy()
        cv2.line(overlay, (x_prev, y_prev), (x, y), color, grosor_seleccionado)
        cv2.addWeighted(overlay, 1, img, 0, 0, img)
        cv2.line(mask, (x_prev, y_prev), (x, y), (255, 255, 255), grosor_seleccionado)
        x_prev, y_prev = x, y
        cv2.imshow('Dibujo', img)
    elif event == cv2.EVENT_LBUTTONUP:
        dibujando = False
```

seleccionar_color(x)

Selecciona un color de la paleta basado en la posición ***x*** del clic.

Calcula en qué recuadro de color ocurrió el clic y actualiza la variable ***color*** con el color correspondiente.

```
def seleccionar_color(x):
    global color
    ancho_recuadro = ancho_controles // len(colores_paleta)
    indice_color = x // ancho_recuadro
    if 0 <= indice_color < len(colores_paleta):
        color = colores_paleta[indice_color]
```

seleccionar_grosor(x)

Selecciona un grosor de la lista ***niveles_grosor*** basado en la posición ***x*** del clic.

Calcula en qué recuadro de grosor ocurrió el clic y actualiza la variable ***grosor_seleccionado*** con el valor correspondiente.

```
def seleccionar_grosor(x):
    global grosor_seleccionado
    ancho_recuadro = ancho_controles // len(niveles_grosor)
    indice_grosor = x // ancho_recuadro
    if 0 <= indice_grosor < len(niveles_grosor):
        grosor_seleccionado = niveles_grosor[indice_grosor]
```

seleccionar_color_controles(event, x, y, flags, param)

Esta función detecta la selección de color o grosor en la ventana de controles.

De nuevo se reciben parámetros relacionados con los eventos del mouse.

Si el usuario hace clic izquierdo (***EVENT_LBUTTONDOWN***), verifica si el clic fue en la parte superior (para elegir color) o en la inferior (para elegir grosor) y llama a la función correspondiente.

```
def seleccionar_color_controles(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:
        if y >= paleta_altura:
            seleccionar_grosor(x)
        else:
            seleccionar_color(x)
```

borrar_dibujo(contorno)

Restaura la imagen principal y la máscara a su estado inicial (la imagen base de contorno).

El parámetro ***contorno*** es la imagen base que se muestra al inicio, de modo que lo copia en ***img*** para reiniciar la imagen del dibujo. Por último, restaura la máscara a cero (sin transparencia) y actualiza la ventana.

```
def borrar_dibujo(contorno):
    global img, mask
    img[:] = contorno.copy()
    mask = np.zeros_like(img)
    cv2.imshow('Dibujo', img)
```

guardar_dibujo()

Guarda la imagen final en formato ***PNG*** con transparencia basada en la máscara.

El resultado se almacenará en la carpeta **filtros** (si no existe la crea) en formato *PNG* con el nombre introducido por el usuario.

Crea una nueva imagen (***imagen_guardada***) con cuatro canales (RGB y Alfa). Rellena el canal Alfa de ***imagen_guardada*** usando la máscara, de modo que las áreas dibujadas sean visibles y las no dibujadas sean transparentes.

```
def guardar_dibujo():
    if not os.path.exists('filtros'):
        os.makedirs('filtros')

    nombre_archivo = input("Ingresa el nombre del archivo para guardar (sin extensión): ")
    ruta_guardado = os.path.join('filtros', f"{nombre_archivo}.png")
    imagen_guardada = np.zeros((img.shape[0], img.shape[1], 4), dtype=np.uint8)
    imagen_guardada[:, :, :3] = img
    imagen_guardada[:, :, 3] = np.where(mask[:, :, 0] == 255, 255, 0)
    cv2.imwrite(ruta_guardado, imagen_guardada)
    print(f"Imagen guardada en {ruta_guardado}")
```

dibujar_area_controles()

Dibuja la ventana de controles con la paleta de colores y los niveles de grosor.

Crea una imagen en blanco para el área de controles y dibuja rectángulos de colores en la parte superior para cada color de ***colores_paleta***.

Finalmente, en la parte inferior dibuja círculos para representar los distintos niveles de grosor.

```
def dibujar_area_controles():
    img_controles = np.ones((paleta_altura * 2, ancho_controles, 3), dtype=np.uint8) * 255

    # Paleta de colores
    ancho_recuadro = ancho_controles // len(colores_paleta)
    for i, c in enumerate(colores_paleta):
        x_inicio = i * ancho_recuadro
        x_fin = (i + 1) * ancho_recuadro
        cv2.rectangle(img_controles, (x_inicio, 0), (x_fin, paleta_altura), c, -1)

     # Controles de grosor
    ancho_recuadro = ancho_controles // len(niveles_grosor)
    for i, grosor in enumerate(niveles_grosor):
        x_inicio = i * ancho_recuadro
        x_fin = (i + 1) * ancho_recuadro
        cv2.rectangle(img_controles, (x_inicio, paleta_altura),
                      (x_fin, paleta_altura * 2), (200, 200, 200), -1)
        cv2.circle(img_controles, ((x_inicio + x_fin) // 2, paleta_altura
                    + paleta_altura // 2), grosor, (0, 0, 0), -1)

    cv2.imshow('Controles', img_controles)
```

crear_filtro()

Función principal que ejecuta el programa, iniciando la interfaz gráfica para el dibujo interactivo.

Carga y redimensiona la imagen ***contornoCara.jpg*** como base para el dibujo. Si no se encuentra, muestra un mensaje de error.

Muestra las instrucciones en la consola sobre cómo usar las teclas de guardado (***g***), borrado (***b***), y salir (***q***). Obviamente, esta función contiene un bucle que espera a que el usuario pulse alguna de esas teclas.

Finalmente, configura las ventanas de dibujo y controles, asignando ***pintar*** como función de callback para la ventana de dibujo y ***seleccionar_color_controles*** para la de controles.

```
def crear_filtro():
    global img, mask

    contorno = cv2.imread('contornoCara.jpg')
    if contorno is None:
        print("Error: La imagen contornoCara.jpg no se pudo cargar.")
        return

    print("Crea tu propio filtro interactivo.
           Presiona 'g' para guardar, 'b' para borrar y 'q' para salir.")

    contorno = cv2.resize(contorno, (ancho_ventana, altura_ventana))
    img = contorno.copy()
    mask = np.zeros_like(img)

    cv2.imshow('Dibujo', img)
    cv2.setMouseCallback('Dibujo', pintar)

    cv2.namedWindow('Controles')
    dibujar_area_controles()
    cv2.setMouseCallback('Controles', seleccionar_color_controles)

    while True:
        key = cv2.waitKey(1) & 0xFF
        if key == ord('g'):
            guardar_dibujo()
        elif key == ord('b'):
            borrar_dibujo(contorno)
        elif key == ord('q'):
            break

    cv2.destroyAllWindows()
```

usar_filtros.py

Este código permite seleccionar y aplicar un filtro a las caras detectadas en tiempo real a partir de una lista de filtros almacenada en la carpeta ***filtros***. Se manejan eventos de teclado para cambiar el filtro activo y mostrar su nombre en pantalla.

Librerías

OpenCV:

- Se utiliza para la detección de caras y el manejo de la cámara.
- Maneja la transparencia de las imágenes referidas a los filtros.
- Manipula la imagen al sobreponer el filtro en la cara detectada.
- Detecta la entrada de teclado para controlar las distintas funciones.

NumPy:

- Crea una imagen en negro para mostrar la lista de filtros.
- Facilita el acceso y la manipulación de los valores de los píxeles de las imágenes.

OS:

- Permite navegar entre los diferentes filtros almacenados en la carpeta.

Función principal

A diferencia del fichero anterior, este únicamente consta de una función que engloba todo el funcionamiento. Dentro de dicha función se realizan diversas operaciones.

Cargar el clasificador de caras

Se utiliza un clasificador en cascada de Haar, el cual sigue un patrón para poder identificar rostros humanos a través de la evaluación de la imagen.

En este caso se utiliza para detectar la cara del usuario para así poder aplicarle el filtro.

```
    clasificador = cv2.CascadeClassifier('haarcascade/haarcascade_frontalface_default.xml')
```

Obtener la lista de filtros

Tras definir la carpeta donde se encuentran almacenadas las imágenes que se utilizan como filtros, se genera una lista (***filtros_disponibles***) con los nombres de los archivos ***PNG*** en esa carpeta (sin extensión) para que puedan ser usados.

```
    carpeta_filtros = 'filtros'

    filtros_disponibles = [f[:-4] for f in os.listdir(carpeta_filtros)
                           if f.endswith('.png')]
    if not filtros_disponibles:
        print("No se encontraron filtros en la carpeta 'filtros'.")
        exit()
```

Configuración de variables de la cámara junto con elección y escalado del filtro

Inicia la captura de video desde la cámara y ajusta el tamaño de la ventana. Si la cámara no está disponible, el programa se detiene.

Carga el primer filtro en la lista ***filtros_disponibles***. Utiliza la variable ***factor_escala*** para controlar el tamaño del filtro de modo que pueda adaptarse mejor al rostro detectado.

```
# Configuración inicial de variables
indice_filtro = 0  # Índice del filtro actualmente seleccionado
filtro_actual = filtros_disponibles[indice_filtro]
filtro_path = os.path.join(carpeta_filtros, filtro_actual + '.png')
area_min = 10000  # Área mínima para considerar una región como un rostro detectado
camara = cv2.VideoCapture(0)

# Verificar si la cámara está disponible
if not camara.isOpened():
    print("No es posible abrir la cámara")
    exit()

# Obtener las dimensiones del cuadro de video para la colocación del filtro
ancho_ventana = int(camara.get(cv2.CAP_PROP_FRAME_WIDTH))
alto_ventana = int(camara.get(cv2.CAP_PROP_FRAME_HEIGHT))

# Cargar el primer filtro con soporte para canal alfa (transparencia)
filtro = cv2.imread(filtro_path, cv2.IMREAD_UNCHANGED)

# Factor de escala para el tamaño del filtro sobre el rostro detectado
factor_escala = 1.8
```

Bucle principal

Este es el bucle en el que ocurre el procesamiento continuo del video en tiempo real.

Captura cada frame de la cámara y lo convierte a escala de grises. Utiliza el clasificador cargado para detectar rostros en el frame y muestra el nombre del filtro actual en pantalla.

Para cada rostro detectado que supere el área mínima (***area_min***), calcula el tamaño del filtro según sus dimensiones. Ajusta la posición para que el filtro quede centrado en la cara. Si el filtro no cabe dentro de los límites de la ventana, se descarta esa detección y en caso de que tenga un canal alfa (transparencia), se aplica sobre el área seleccionada usando ***alpha_filtro***, que indica la opacidad de cada pixel.

Por un lado, muestra el frame procesado (con filtro aplicado) en una ventana de OpenCV y por el otro crea una imagen en negro en otra ventana para mostrar la lista de filtros donde se resalta en color verde el filtro actual.

A su vez maneja la entrada de teclado para poder avanzar y retroceder entre los diferentes filtros.

```
while True:
        ret, frame = camara.read()
        if not ret:
            print("No es posible obtener la imagen")
            break

        frame_byn = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        caras = clasificador.detectMultiScale(frame_byn)

        cv2.putText(frame, f"Filtro: {filtro_actual}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX,
                    1, (255, 255, 255), 2)

        for (x_cara, y_cara, ancho_cara, alto_cara) in caras:
            if ancho_cara * alto_cara > area_min:
                ancho_filtro = int(ancho_cara * factor_escala)
                proporciones_filtro = filtro.shape[1] / filtro.shape[0]
                alto_filtro = int(ancho_filtro / proporciones_filtro)
                filtro_redimensionado = cv2.resize(filtro, (ancho_filtro, alto_filtro))

                y_filtro = y_cara - int(alto_filtro * 0.22)
                x_filtro = x_cara - int(ancho_filtro * 0.22)

                if (x_filtro >= 0 and y_filtro >= 0 and
                    x_filtro + ancho_filtro <= ancho_ventana and
                    y_filtro + alto_filtro <= alto_ventana):

                    roi = frame[y_filtro:y_filtro + alto_filtro, x_filtro:x_filtro
                        + ancho_filtro]

                    if filtro_redimensionado.shape[2] == 4:
                        alpha_filtro = filtro_redimensionado[:, :, 3] / 255.0
                        for c in range(0, 3):
                            roi[:, :, c] = roi[:, :, c] * (1 - alpha_filtro)
                                        + filtro_redimensionado[:, :, c] * alpha_filtro
                    else:
                        roi[:] = filtro_redimensionado[:]

                    frame[y_filtro:y_filtro + alto_filtro, x_filtro:x_filtro
                          + ancho_filtro] = roi

        cv2.imshow('Webcam', frame)

        filtros_img = np.zeros((400, 300, 3), dtype=np.uint8)
        cv2.putText(filtros_img, "Filtros disponibles:", (10, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 1)
```

```
        for i, nombre_filtro in enumerate(filtros_disponibles):
            color = (0, 255, 0) if i == indice_filtro else (255, 255, 255)
            cv2.putText(filtros_img, nombre_filtro, (10, 60 + i * 30),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.6, color, 1)

        cv2.imshow("Filtros", filtros_img)

        print("Presiona 'q' para salir, 'n' para el siguiente filtro,
              'b' para el filtro anterior.")
        key = cv2.waitKey(10)
        if key == ord('q'):
            break
        elif key == ord('n'):
            indice_filtro = (indice_filtro + 1) % len(filtros_disponibles)
            filtro_actual = filtros_disponibles[indice_filtro]
            filtro_path = os.path.join(carpeta_filtros, filtro_actual + '.png')
            filtro = cv2.imread(filtro_path, cv2.IMREAD_UNCHANGED)
        elif key == ord('b'):
            indice_filtro = (indice_filtro - 1) % len(filtros_disponibles)
            filtro_actual = filtros_disponibles[indice_filtro]
            filtro_path = os.path.join(carpeta_filtros, filtro_actual + '.png')
            filtro = cv2.imread(filtro_path, cv2.IMREAD_UNCHANGED)
```

Liberación de recursos

Tras la finalización del bucle anterior se libera la cámara y se cierran todas las ventanas.

```
    camara.release()
    cv2.destroyAllWindows()
```

Problemas encontrados

Durante el desarrollo del proyecto se intentaron implementar otro tipo de funciones de modo que harían una aplicación más completa.

Una de las funcionalidades cuyo desarrollo no pudo salir a flote consistía en añadir una **goma de borrar**. El problema encontrado fue que no había forma de implementarlo sin que quedaran residuos en la imagen (filtro) una vez esta era guardada, pues no era capaz de "pintar" una línea transparente que dejara a la vista la plantilla con el contorno de la cara.

A raíz de ello, otra posible solución al problema anterior era añadir las opciones de **rehacer** y **deshacer**. En principio parecía que funcionaba, pues consistía en crear un historial de la máscara utilizada para el filtro. Finalmente, dicha funcionalidad tuvo que ser eliminada, pues al cambiar entre las distintas máscaras almacenadas daba problemas con el color, pues al elegir uno se cambiaba todo lo dibujado a dicho color.