LA UNIVERSIDAD EN INTERNET

APLICACIÓN DE ALGORITMOS DE VISIÓN ARTIFICIAL

Desarrollar un programa que sea capaz de detectar objetos, por ejemplo, caras, en una imagen o en un streaming de vídeo

Profesor: Sergio Mauricio Martínez Monterrubio

Ernesto González Pradas
ernesto.gonzalez023@comunidadunir.net

Índice

Introducción

El objetivo de este laboratorio es el de desarrollar un programa que sea capaz de detectar objetos, por ejemplo, caras, en una imagen o en un streaming de vídeo.

Se trata de utilizar OpenCV para detectar objetos en imágenes y en capturas de vídeo.

Preparación del entorno

Para realizar esta actividad hemos configurado nuestro IDE de desarrollo (IntelliJ IDEA 2021.2.4 Ultimate Edition) instalando el paquete de OpenCV e importando cv2 como librería para nuestro programa.

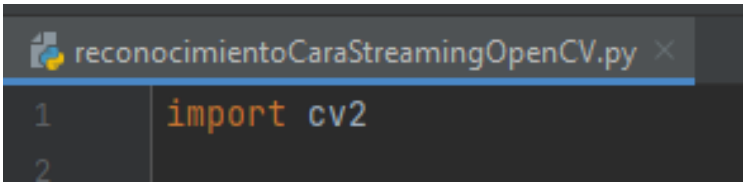

Ilustración 1: Importación del paquete cv2 de OpenCV

Desarrollo del programa y explicación

A continuación, se procede a explicar paso por paso como se ha ido elaborando el programa que reconoce tanto caras como ojos en un streaming de video.

Una vez importada la librería de OpenCV, lo siguiente que realizamos es cargar los clasificadores pre-entrenados de dicha librería para reconocer tanto caras como ojos. Dentro de esta librería existen otros muchos clasificadores para reconocer por ejemplo sonrisas:

```
# Cargamos los clasificadores pre-entrenados
clasificador_cara = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
clasificador_ojo = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_eye.xml')
```

Ilustración 2: Clasificadores pre-entrenados

Una vez cargados los clasificadores pre-entrenados inicializamos la captura de vídeo de nuestra webcam:

```
# Inicializamos la captura de video
captura = cv2.VideoCapture(0)
```

Ilustración 3:Inicializamos captura de vídeo

A continuación, y debido a muchas pruebas, es posible que el proceso de la cámara se quede levantado u otros programas hagan uso de ella por lo que nuestro programa no funcionará correctamente. Para subsanar esto lo que hemos realizado ha sido una pequeña comprobación para verificar que la webcam esté funcionando correctamente:

```
10      # Verificamos que la webcam esté funcionando correctamente
11      if not captura.isOpened():
12          raise IOError("No se puede abrir la webcam")
13
```

Ilustración 4:Código que comprueba el funcionamiento de la webcam

Ahora viene la parte del código donde se realiza todo el trabajo de detección donde:

- **Primero**: mediante un bucle while vamos capturando frame a frame del vídeo.
    - En caso de que no pueda capturar un frame salimos del bucle.
- **Segundo**: convertimos la imagen a escala de grises que ayuda a la detección de las formas.
- **Tercero**: detectamos las caras usando el clasificador de caras pre-entrenado y el frame de la imagen en gris. Es aquí donde podemos modificar los calores de "scaleFactor", "minNeighbors" y "minSize" praa mejorar la calidad de la detección. Cada atributo significa lo siguiente:
    - ***scaleFactor***: Controla la sensibilidad del detector ajustando la escala de reducción de la imagen durante la detección de objetos. Un valor más pequeño aumenta la sensibilidad, pero puede aumentar los falsos positivos, mientras que un valor más grande disminuye la sensibilidad.
    - ***minNeighbors***: Especifica el número mínimo de vecinos requeridos para clasificar una región como objeto válido. Un valor más alto ayuda a filtrar falsos positivos, pero puede perder objetos válidos. Es un mecanismo para reducir falsas detecciones y mejorar la precisión.
    - ***minSize***: Indica el tamaño mínimo del objeto que se puede detectar. Se utiliza para filtrar detecciones demasiado pequeñas que podrían ser ruido o falsos positivos. Permite descartar detecciones no deseadas y enfocarse en objetos de un tamaño específico.
- **Cuarto**: iteramos sobre cada cara detectada para:
    - Dibujar un rectángulo de color verde alrededor de cada cara.
    - Obtenemos la región e interés de la cara en la imagen en escala de grises y en la imagen original

- Una vez tenemos la cara detectamos los ojos en dicha región e iteramos sobre cada ojo dibujando un rectángulo de color azul alrededor de cada ojo.

- **Quinto**: Mostramos el frame con las detecciones realizadas en el paso Cuarto.
- **Sexto**: Si se pulsa la tecla “s” salimos del bucle y por tanto se apaga la cámara.

reconocimientoCaraStreamingOpenCV.py

```
while True:
    # Capturamos un frame de video
    ret, imagen = captura.read()

    # Si no se pudo capturar el frame, salimos del bucle
    if not ret:
        break

    # Convertimos la imagen a escala de grises
    imagen_gris = cv2.cvtColor(imagen, cv2.COLOR_BGR2GRAY)

    # Detectamos caras en la imagen en escala de grises
    caras = clasificador_cara.detectMultiScale(imagen_gris, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

    # Iteramos sobre cada cara detectada
    for (x, y, w, h) in caras:
        # Dibujamos un rectángulo verde alrededor de la cara
        cv2.rectangle(imagen, (x, y), (x+w, y+h), (0, 255, 0), 2)

        # Obtenemos la región de interés (ROI) de la cara en la imagen en escala de grises y en la imagen original
        roi_gris = imagen_gris[y:y+h, x:x+w]
        roi_color = imagen[y:y+h, x:x+w]

        # Detectamos ojos en la región de interés de la cara
        ojos = clasificador_ojo.detectMultiScale(roi_gris)

        # Iteramos sobre cada ojo detectado
        for (ex, ey, ew, eh) in ojos:
            # Dibujamos un rectángulo azul alrededor del ojo
            cv2.rectangle(roi_color, (ex, ey), (ex+ew, ey+eh), (255, 0, 0), 2)

    # Mostramos el frame con las detecciones en una ventana llamada 'Video en Streaming'
    cv2.imshow('Video en Streaming', imagen)

    # Esperamos a que se presione la tecla 's' para salir del bucle
    if cv2.waitKey(1) == ord('s'):
        break
```

Ilustración 5:Parte del código donde se realiza la detección de la cara y de los ojos

Finalmente liberamos los recursos de la cámara y cerramos todas las ventanas abiertas:

```
# Liberamos los recursos de la cámara y cerramos todas las ventanas abiertas
captura.release()
cv2.destroyAllWindows()
```

Ilustración 6:Liberamos los rescursos de la cámaray cerramos las ventanas

Resultado del código

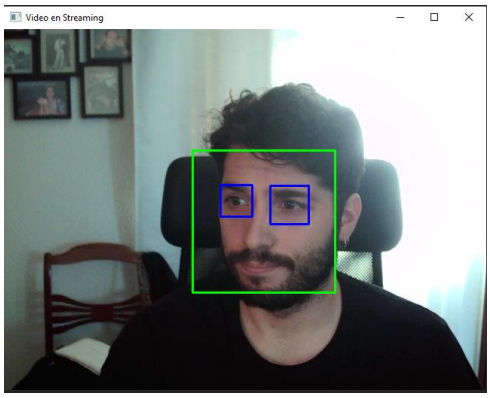

Ilustración 7:Captura normal

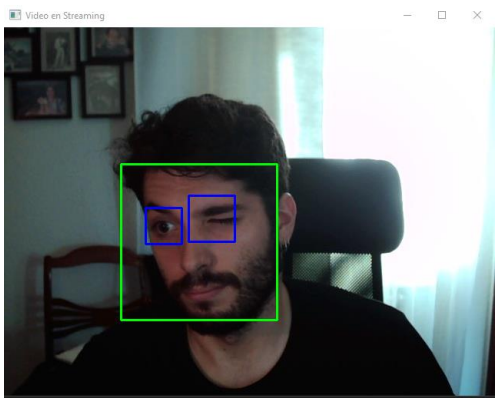

Ilustración 8:Captura ojo guiñado

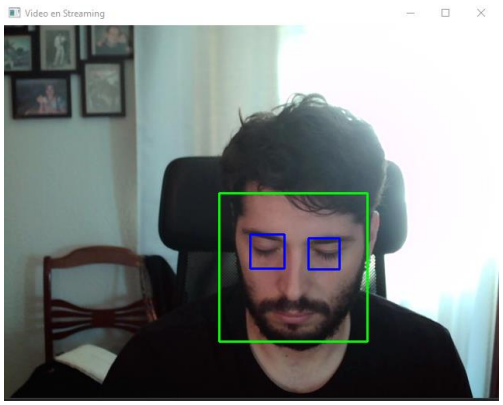

Ilustración 9:Captura ojos cerrados

Conclusiones y Reflexiones

Durante este laboratorio, hemos aprendido a utilizar la biblioteca OpenCV para realizar la detección de caras y ojos en tiempo real utilizando clasificadores basados en características. OpenCV proporciona una variedad de clasificadores pre-entrenados que se basan en el algoritmo de Haar Cascade.

Mediante el uso de la detección de caras y ojos, podemos identificar y ubicar automáticamente estas regiones en imágenes o videos. Esto tiene muchas aplicaciones prácticas, como en sistemas de seguridad y vigilancia, autenticación biométrica, análisis de emociones faciales, detección de somnolencia en conductores, entre otros.

Una limitación del enfoque basado en Haar Cascade es que puede haber falsos positivos o falsos negativos en la detección de caras y ojos, especialmente en condiciones de iluminación y ángulos difíciles. Sin embargo, el rendimiento de detección se puede mejorar ajustando los parámetros del clasificador y utilizando técnicas adicionales, como el seguimiento de objetos.

OpenCV es una poderosa biblioteca de visión por computadora que va más allá de la detección de caras y ojos. Ofrece una amplia gama de funcionalidades para procesamiento de imágenes, segmentación, reconocimiento de objetos, seguimiento de movimiento, calibración de cámaras, entre otros. Es una herramienta esencial para desarrolladores y científicos que trabajan en aplicaciones de visión artificial.

Bibliografía

opencv. (80 de junio de 2023). *docs.opencv.org*. Obtenido de https://docs.opencv.org/4.x/index.html#gsc.tab=0

programarfacil. (01 de enero de 2023). *programarfacil*. Obtenido de https://programarfacil.com/podcast/81-vision-artificial-opencv-phyton/

Rodríguez, H. (08 de abril de 2021). *crehana*. Obtenido de https://www.crehana.com/blog/transformacion-digital/que-es-opencv/