INGENIERÍA MECATRÓNICA

DI_CERO

DIEGO CERVANTES RODRÍGUEZ

PROGRAMACIÓN: VISIÓN ARTIFICIAL

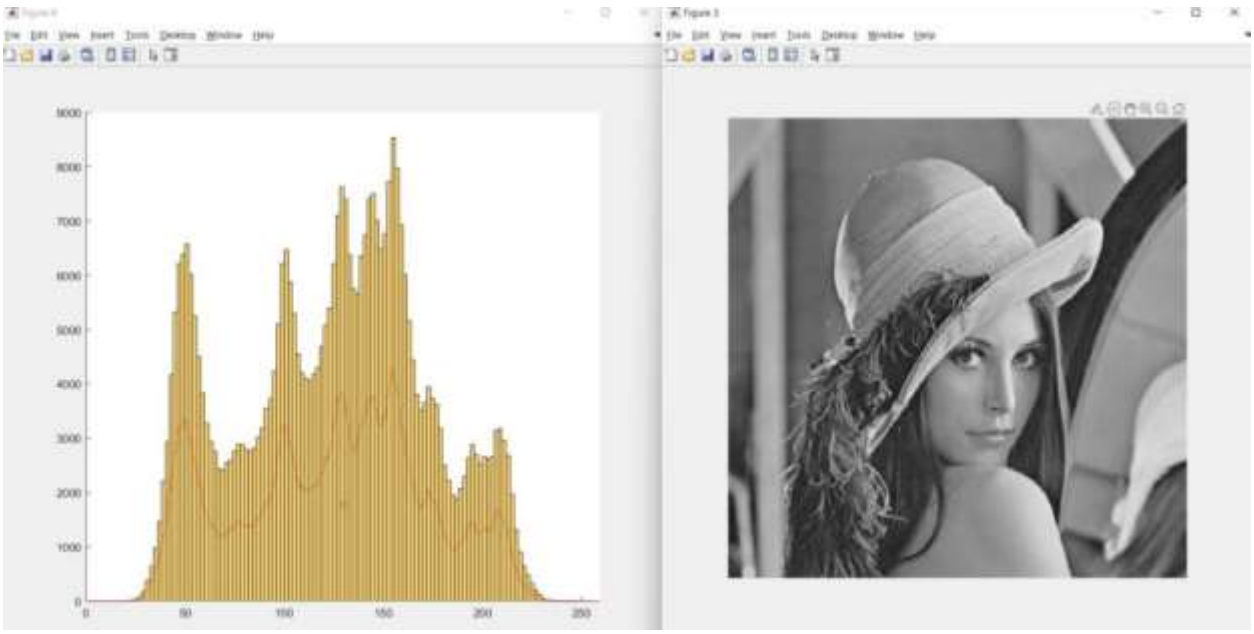
VISUAL STUDIO CODE & PYTHON 3.9.7: LIBRERÍA OPENCV

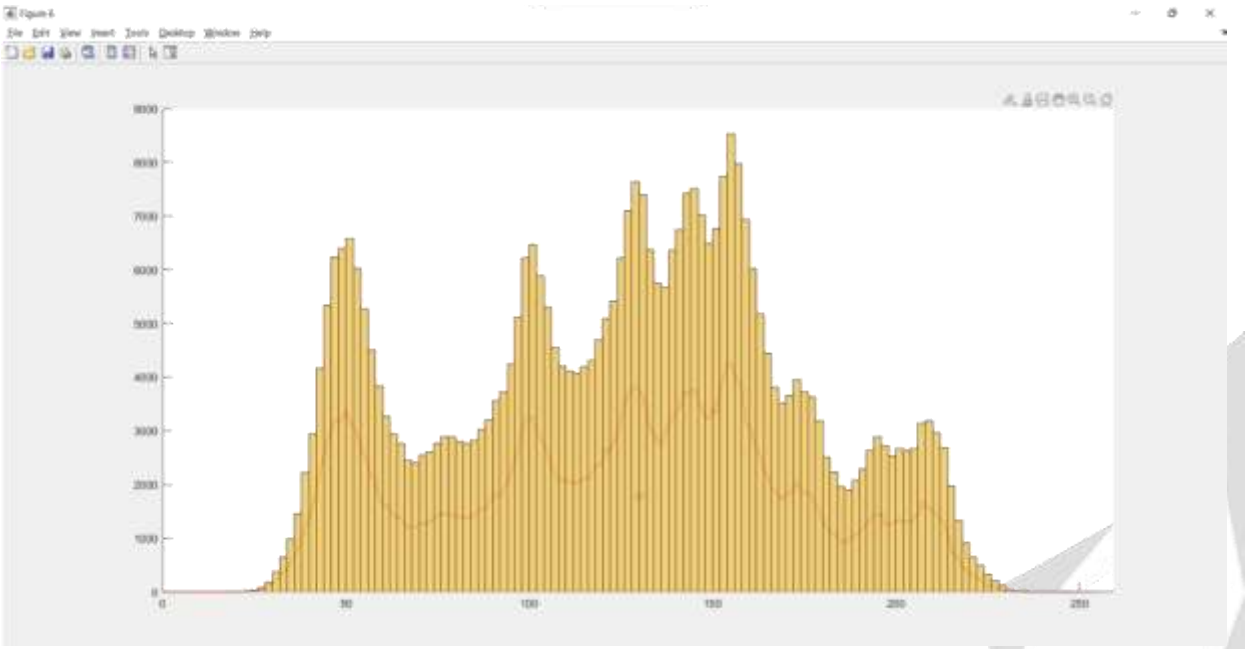
Visión Artificial: Histograma, Filtros, OpenCV, Detección y Suavizado de Bordes, Esquinas y Rostros

Contenido

Histograma

El histograma no busca transformar la imagen sino describirla por medio de términos matemáticos y medidas estadísticas, para ello se deben obtener las capas RGB de la imagen y obtener su valor de 0 a 255 en cada uno de los pixeles que conforman la imagen en el eje horizontal del histograma, en el eje vertical aparece el número de pixeles que tienen el mismo valor de 0 a 255, por lo cual no importa tanto el eje vertical sino el eje horizontal para determinar cuál color es el que más aparece en la imagen, para ello se deben analizar las capas RGB por separado.



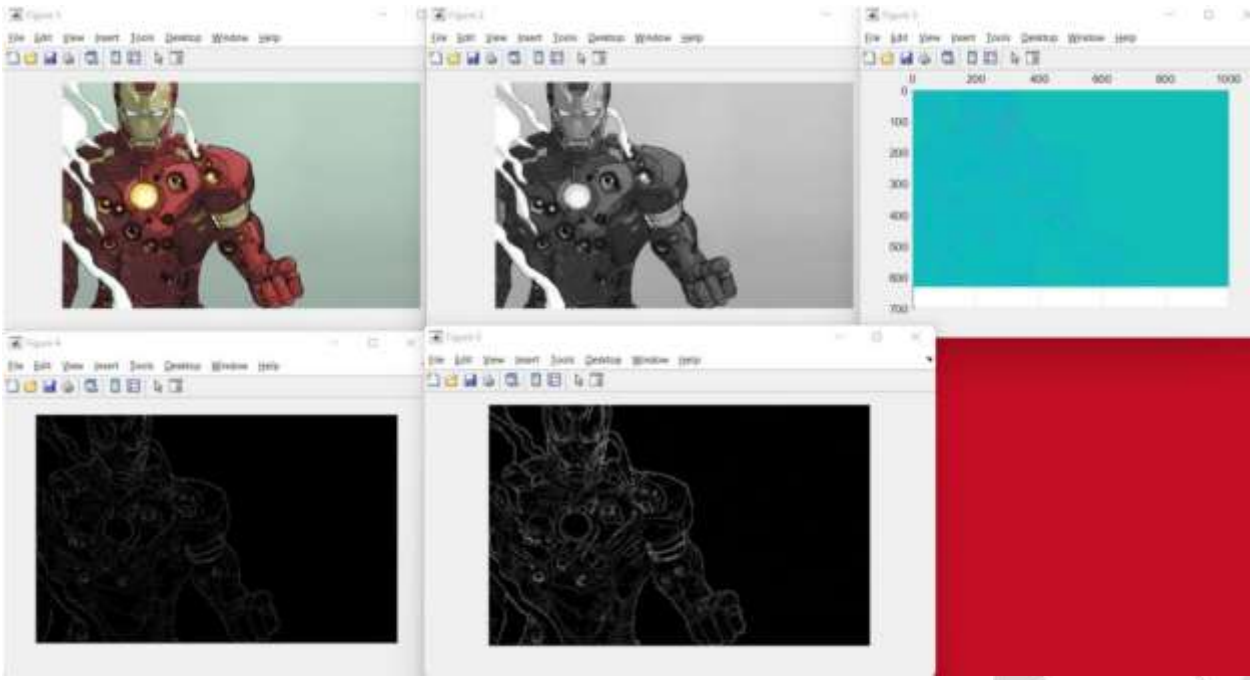


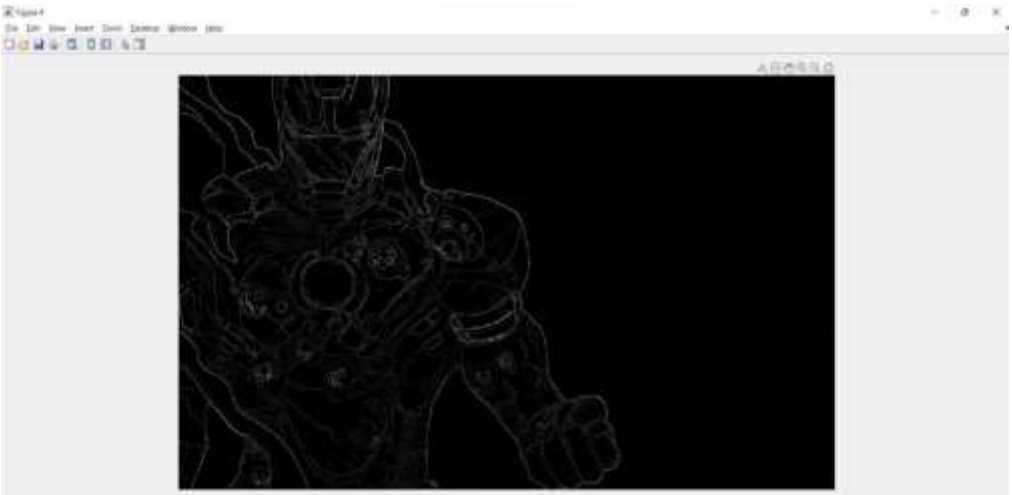
Detección de Bordes

El filtro de Laplace, así como muchos otros (como lo es la máscara de Sobel y Prewitt) sirve para identificar bordes en la imagen analizada, esto es muy importante en la visión artificial ya que de esta manera podemos delimitar el espacio que ocupa un objeto en específico. Esto se lleva a cabo a partir de la derivación de imágenes, que se realiza respecto a los ejes xy de la imagen, los valores positivos del eje x si van hacia la derecha como normalmente se acostumbra, pero los valores positivos del eje y van hacia abajo, teniendo su punto inicial en la esquina superior izquierda. Es posible realizar la derivación porque la imagen se considera como una función de dos variables f(x,y), para ello se realiza el gradiente de la función. Para poder aplicar el filtro, primero se lleva la imagen al dominio de los grises, convirtiéndola en valores decimales (double), la escala de grises de la imagen se puede obtener a partir de cualquiera de las 3 capas de la matriz que describe la imagen digital, que es un cubo 3D de 3 matrices R, G y B sobrepuestas una encima de la otra, esta es la función que depende de dos variables f(x,y).

Filtro de Laplace

El filtro de Laplace a grandes rasgos lo que hace es considerar una matriz, la cual es de tamaño 3X3 y se aplica en torno a un pixel central, el cual es el elemento de análisis y a los elementos que lo rodean se les llama vecindad, el pixel central tiene la coordenada i,j y a partir de ese punto se dan las coordenadas de los elementos de la vecindad, siendo las filas de abajo las coordenadas i+1 y las de arriba i-1 y siendo las columnas de la izquierda j-1 y las de la derecha j+1, la matriz de la escala de grises obtenida de la imagen original se debe multiplicar por la matriz del filtro de Laplace y se crea a partir de ella una nueva matriz. El filtro de Laplace lo que hace es volver cero los elementos de las esquinas en la vecindad y volver negativo el pixel de análisis.





Librerías Python para Visión Artificial

Numpy

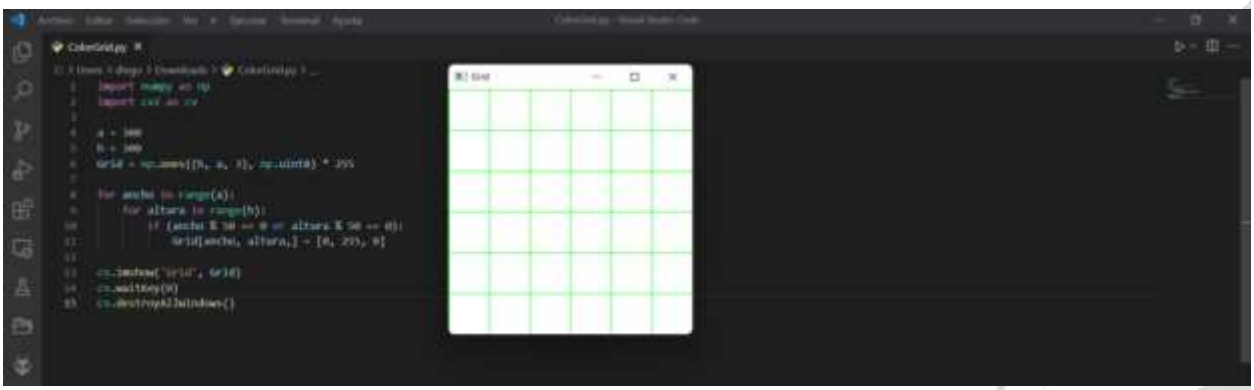
La librería numpy sirve para realizar operaciones matemáticas, como lo son funciones simples, operaciones entre matrices, operaciones binarias, etc. Se debe tener instalada la librería numpy para poder utilizar la librería OpenCV (Open Source Computer Vision).

MatPlotLib

La librería matplotlib incluye varias herramientas con las cuales se puede realizar Graficación de datos, mostrándose con distintos colores y representaciones gráficas de mostrar los datos recopilados. Se debe tener instalada la librería numpy para poder utilizar la librería OpenCV (Open Source Computer Vision).

OpenCV

OpenCV (Open Source Computer Vision) comenzó como un proyecto de investigación en Intel. Actualmente es la biblioteca de visión artificial más grande de código abierto. Es necesario tener instalada la librería numpy ya que esta permite realizar operaciones matriciales y las imágenes digitales son consideradas como una matriz de 3 capas, RGB. Además, se utiliza la librería MatplotLib para mostrar las imágenes después de haber sido modificadas.



Detección de Esquinas

Detector Beaudet

El detector Beaudet es un operador isotrópico para detectar esquinas, cuyas propiedades no dependen de la dirección en que son examinadas, basado en el cálculo del determinante Hessiano. La matriz hessiana de una función escalar o campo escalar f de n variables, es la matriz cuadrada de tamaño NXN, de las segundas derivadas parciales. Por lo que considerando su ecuación y determinante se obtiene lo siguiente:

$$det\left(H_f(x,y)\right) = I_{xx}I_{yy} - {I_{xy}}^2$$

$$B(x,y) = \frac{det\left(H_f(x,y)\right)}{\left(1 + {I_x}^2 - {I_y}^2\right)^2}$$

$$I_x = \frac{df(x,y)}{dx}$$

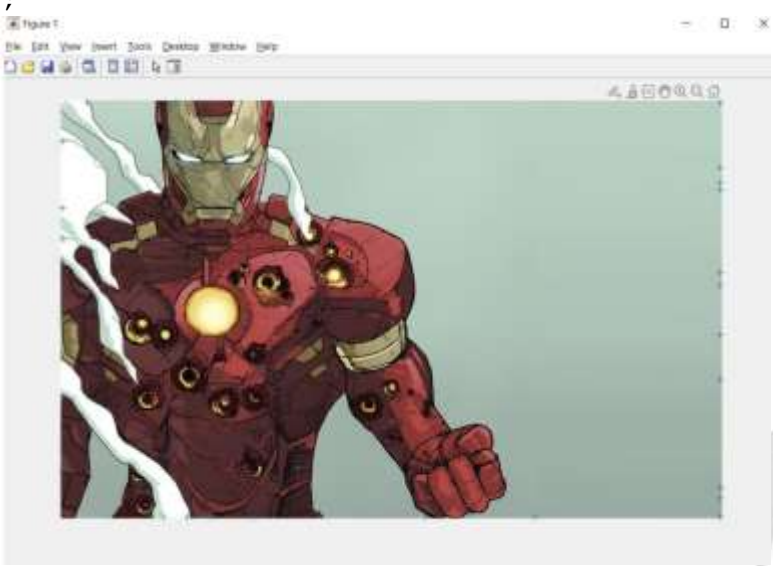
$$I_y = \frac{df(x,y)}{dy}$$

Se considera Ix e Iy como los gradientes en el sentido vertical y horizontal, de tal forma que bajo este operador serán considerados como esquinas aquellos puntos de $B(x, y)$ que sobrepasen o sean iguales a un determinado umbral definido.

Ejemplo Matlab:

```
%DIEGO CERVANTES RODRÍGUEZ
clc
clear all
close all
%DETECTOR BAUDET: DETECCIÓN DE ESQUINAS
IMA = imread('C:\Users\diego\OneDrive\Documents\Instrumentación
Virtual\Sistemas de Visión Artificial\Imagenes\Iron Man bullet.jpg');
Im=double(rgb2gray(IMA));
%Matriz H
h=ones(3)/9;
%Filtro de la imagen con la matriz H
Im = imfilter(Im,h);
%FILTRO O MÁSCARA DE SOBEL: DETECCIÓN DE BORDES
sx = [-1,0,1;-2,0,2;-1,0,1];
sy = [-1,-2,-1;0,0,0;1,2,1];
%Primera derivada parcial o gradientes en el sentido vertical y horizontal
Ix = imfilter(Im,sx); %Gradiente horizontal, primera derivada
Iy = imfilter(Im,sy); %Gradiente vertical, primera derivada
%Segunda derivada parcial
Ixx = imfilter(Ix,sx); %Gradiente horizontal, segunda derivada
Iyy = imfilter(Iy,sy); %Gradiente vertical, segunda derivada
Ixy = imfilter(Ix,sy); %Gradiente mixto
%Denominador de la matriz B(x,y)
```

```
DB = (1+Ix.*Ix+Iy.*Iy).^2;
%Numerador de la matriz B(x,y), determinante de la matriz H
NB = Ixx.*Iyy-(Ixy).^2;
%Matriz B(X,Y)
B = (NB./DB);
%Escala de la matriz
B = (1000/max(max(B)))*B;
%Binarización para crear la matriz V1
V1 = (B)>10;
pixel = 10;
%Analisis de los pixeles de la vecindad
[filas, columnas] = size(V1);
res = zeros(filas, columnas);
for r = 1:filas
    for c = 1:columnas
        if (V1(r,c))
            I1 = [r-pixel, 1]; %Limite Izquierdo
            I2 = [r+pixel, filas]; %Limite derecho
            I3 = [c-pixel, 1]; %Limite Superior
            I4 = [c+pixel, columnas]; %Limite Inferior

            MaxXi = max(I1);
            MaxXd = min(I2);
            MaxYs = max(I3);
            MaxYi = min(I4);

            tmp = B(MaxXi:MaxXd, MaxYs:MaxYi);
            maxim = max(max(tmp));
            if (B(r,c)== maxim)
                res(r,c)=1;
            end
        end
    end
end
%Impresión de la imagen con las esquinas representadas por taches
imshow(uint8(IMA));
hold on
[re,co]=find(res');
plot(re,co,'+');
```



Detector Kitchen & Rosenfeld

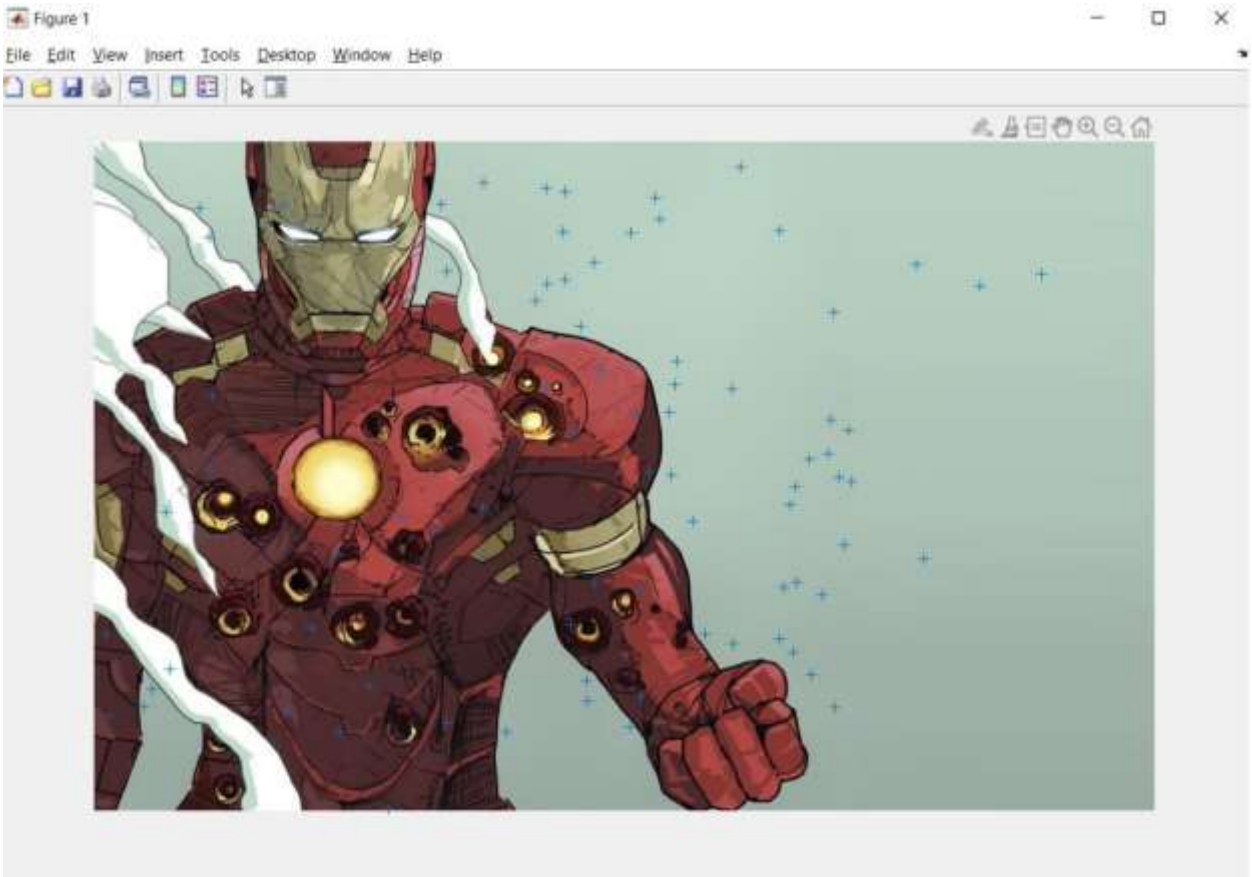
Kitchen y Rosenfeld pusieron propusieron un detector de esquinas basado en el cambio de la dirección del gradiente, a lo largo de un borde multiplicado por la dirección del gradiente local en el pixel analizado, alrededor del cual se encuentra la vecindad. Por lo que Kitchen y Rosenfeld propusieron calcular la matriz $KR(x, y)$, en la cual se considera a las esquinas como aquellos valores que sobrepasen un valor prefijado considerado como umbral:

$$KR(x,y) = \frac{I_{xx}{I_y}^2 + I_{yy}{I_x}^2 - 2I_{xy}I_y}{{I_x}^2 + {I_y}^2}$$

Ejemplo Matlab:

```
%DIEGO CERVANTES RODRÍGUEZ
clc
clear all
close all
%DETECTOR KITCHEN AND RODENFELD: DETECCIÓN DE ESQUINAS
IMA = imread('C:\Users\diego\OneDrive\Documents\Instrumentación
Virtual\Sistemas de Visión Artificial\Imagenes\Iron Man bullet.jpg');
Im=double(rgb2gray(IMA));
%Matriz H
h = ones(3)/9;
%Filtro de la imagen con la matriz H
Im = imfilter(Im,h);
%FILTRO O MÁSCARA DE SOBEL: DETECCIÓN DE BORDES
sx = [-1,0,1;-2,0,2;-1,0,1];
sy = [-1,-2,-1;0,0,0;1,2,1];
%Primera derivada parcial o gradientes en el sentido vertical y horizontal
Ix = imfilter(Im,sx); %Gradiente horizontal, primera derivada
Iy = imfilter(Im,sy); %Gradiente vertical, primera derivada
%Segunda derivada parcial
Ixx = imfilter(Ix,sx); %Gradiente horizontal, segunda derivada
Iyy = imfilter(Iy,sy); %Gradiente vertical, segunda derivada
Ixy = imfilter(Ix,sy); %Gradiente mixto
%Numerador de la matriz KR(x,y)
A = (Ixx.*(Iy.^2))+(Iyy.*(Ix.^2))-(2*Ixy.*Iy);
%Denominador de la matriz KR(x,y)
B = (Ix.^2)+(Iy.^2);
%Matriz KR(X,Y)
V = (A./B);
%Escala de la matriz
V = (1000/max(max(V)))*V;
%Binarización para crear la matriz V1
V1 = (V)>10;
pixel = 10;
%vecindad
%Analisis de los pixeles de la vecindad
[filas, columnas] = size(V1);
res = zeros(filas, columnas);
for r = 1:filas
    for c = 1:columnas
      if (V1(r,c))
          I1 = [r-pixel,1]; %Limite Izquierdo
```

```
            I2 = [r+pixel,filas]; %Limite derecho
            I3 = [c-pixel,1]; %Limite Superior
            I4 = [c+pixel,columnas]; %Limite Inferior

            MaxXi = max(I1);
            MaxXd = min(I2);
            MaxYs = max(I3);
            MaxYi = min(I4);

            tmp = V(MaxXi:MaxXd,MaxYs:MaxYi);
            maxim = max(max(tmp));
            if (V(r,c) == maxim)
                res(r,c) = 1;
            end
        end
    end
end
%Impresión de la imagen con las esquinas representadas por taches
imshow(uint8(IMA));
hold on
[re,co] = find(res');
plot(re,co,'+');
```



Suavizado de Bordes

Teorema de Douglas-Peucker

La identificación de líneas en forma de vector se aplica en mapas cartográficos porque las líneas son aproximadamente el 80% de los elementos en una representación cartográfica, las líneas se identifican por medio de dos métodos:

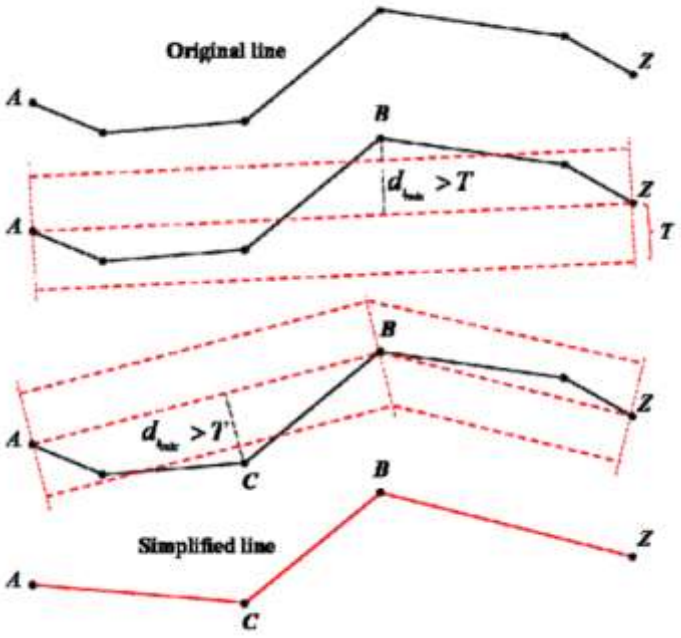
- Simplificación de Douglas-Peucker.
- Simplificación de Bézier.

Ambos para realizar el suavizado basado en curvas, identificando de esta manera las figuras en el mapa. Para identificar las figuras en los mapas es necesario identificar curvas que representan cuerpos geográficos, pero por su inmensidad es necesario reducir su tamaño, por lo cual se debe realizar un proceso de selección y reducción, llamado proceso de generalización cartográfica.

Para realizar la generalización cartográfica, una de las soluciones que mejores resultados presenta consiste en simplificar su geometría por medio de la eliminación de puntos y suavizado posterior del resultado obtenido por medio del algoritmo de Douglas-Peucker y algoritmos de filtrado de curvatura basados en las curvas de Bézier, que opera de forma finita las curvas que pueden parecer análogas.

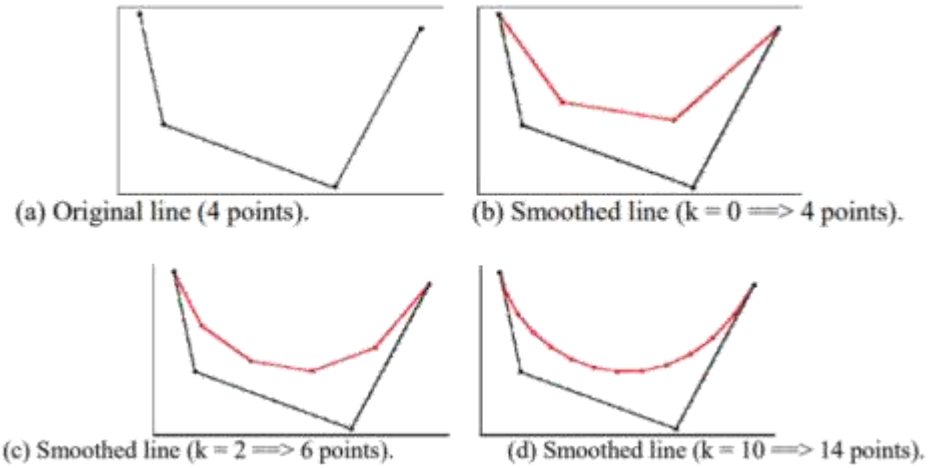
El fundamento del algoritmo consiste en seleccionar de la línea original que conforma al mapa, puntos específicos que son llamados puntos críticos o de anclaje que construirán la línea generalizada. Para seleccionarlos, se selecciona un el factor de tolerancia (umbral) mayor a cero, este se le llama simplemente tolerancia y es expresado en unidades de longitud

$$T > 0$$



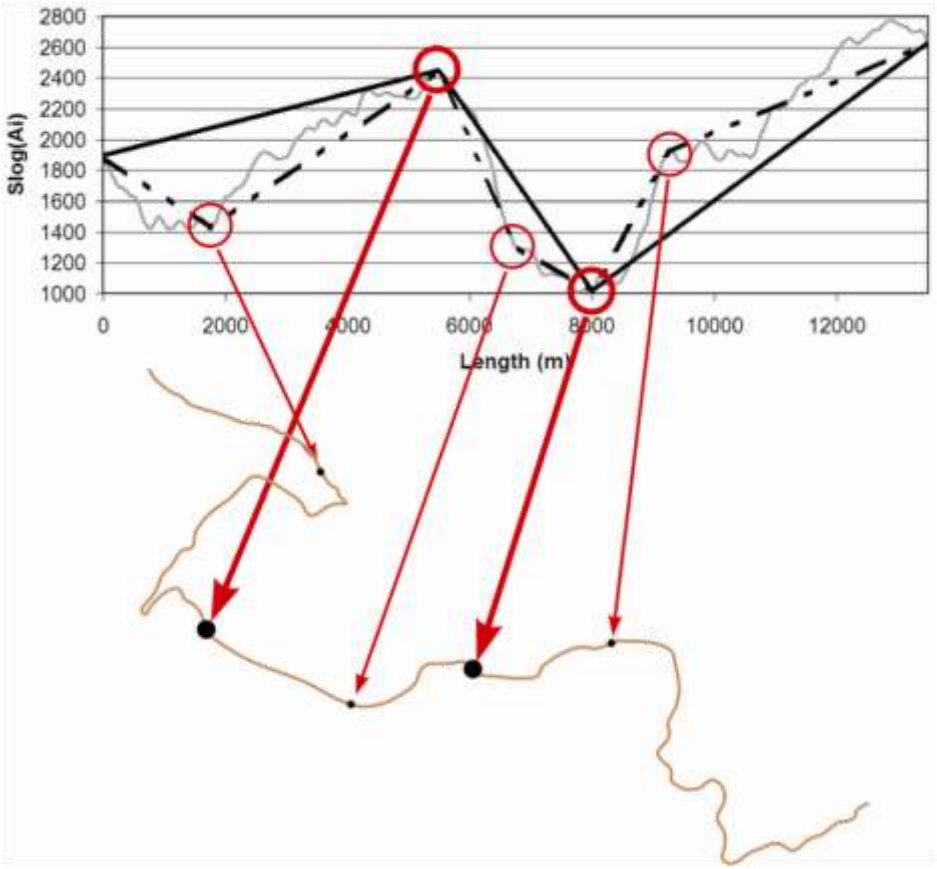
En consecuencia, para utilizar las curvas de Bézier como algoritmo de suavizado, es necesario establecer previamente el número de valores que debe adoptar dicho parámetro, considerando lo siguiente criterios:

- La curva original se puede suavizar manteniendo su tendencia a partir de una serie de valores de T igual al número de puntos de la curva, que constituirán los puntos de control de la resultante Curvas de Bezier.
- El trazado de las curvas de Bézier mejora, si se distribuyen bien los valores de t dentro de los indicados se elige el intervalo.



La aplicación informática descrita, se aplica a procesos en donde se realicen reducciones de las curvas que conforman la imagen original, pero reducidas a pequeña escala (en relación 1:2 o similar), ofreciendo dos resultados principales:

1. La generalización de los elementos lineales de un conjunto cartográfico en formato vectorial. Para Para lograr esto, la aplicación codifica el algoritmo de simplificación de Douglas-Peucker y el de suavizado basado en las curvas de Bézier, considerando el tratamiento simultáneo de diferentes clases de entidades geométricas y preservando las relaciones topológicas originales entre ellos.
2. El análisis de los resultados obtenidos tras la transformación realizada por el proceso anterior, por medio del ensayo con diferentes parámetros y el contraste de los resultados. Para conseguir eso, la aplicación permite cambiar los parámetros característicos de los algoritmos en cuestión y ofrece un informe de los resultados después de cada proceso: puntos tratados, eliminados, agregados o finales, en cada caso, así como el tiempo de procesamiento.



Webcam, Binarización, Obtención de Capas RGB, Detección y Suavizado de Bordes con Python

Se utilizó la webcam para realizar algunas de las funciones de visión artificial que se pueden aplicar con la librería OpenCV de Python, para ello en todo el código del archivo 8.- Uso de Webcam se comentó cada parte del procesamiento de las imágenes que conforman el video y además se dio instrucciones detalladas de lo que hace cada método con todo y los parámetros que recibe.

```
# -*- coding: utf-8 -*-

#Comentario de una sola linea con el simbolo #, en Python para nada se deben poner acentos sino el programa

#puede fallar o imprimir raro en consola, la primera línea de código es para que no tenga error, pero aún

#así al poner un ángulo saldrá raro en consola, la línea debe ponerse tal cual como aparece y justo al inicio.



#IMPORTACIÓN DE LIBRERÍAS:

import cv2 as cv #Librería OpenCV: Sirve para todo tipo de operaciones de visión artificial

import numpy as np #Librería numpy: Realiza operaciones matemáticas complejas

import scipy.ndimage #Librería scipy: Se utiliza para aplicar filtros de máximos y mínimos a una imagen



#SINTONIZACIÓN DE UMBRALES MÉTODO CANNY SIN SUAVIZADO GAUSSIANO:

#cv.namedWindow(): Sirve para crear una ventana donde se podrá mostrar un video o imagen recopilada, se debe declarar desde
```

```
#este punto para crear barras que varíen el valor de los umbrales del filtro Canny, para ver cómo se debe sintonizar para
#obtener de mejor manera los bordes, dando valores de 0 a 255
cv.namedWindow('Webcam - Canny') #Ventana que muestra los bordes Canny
#cv.createTrackbar(): Este método sirve para crear y mostrar una barra sobre una ventana específica que muestra una imagen o
#video, esto para variar cierto valor en tiempo real, este recibe los siguientes parámetros:
#   - Nombre de la barra (Track bar)
#   - Ventana a la cual se aplicará, esta debe tener el mismo nombre que la ventana creada con el método cv.namedWindow() y a la
#   cual se aplicará el método canny.
#   - Valor mínimo de la barra.
#   - Valor máximo de la barra.
#   - Función que a fuerza se debe ejecutar cuando se mueva la barra, esta puede ejecutar una acción o en este caso solo se
#   declara y no hace nada.
#Se debe crear la función que no hace nada en este caso para que se pueda crear la barra en la ventana antes de usar el método
#cv.createTrackbar().
def nothing (x):
    pass
#Se crea una barra con el método cv.createTrackbar() para variar el umbral inferior y superior del método Canny
cv.createTrackbar('Umbral inferior', 'Webcam - Canny', 0, 255, nothing)
cv.createTrackbar('Umbral superior', 'Webcam - Canny', 0, 255, nothing)



#SINTONIZACIÓN DE UMBRALES MÉTODO CANNY CON SUAVIZADO GAUSSIANO:
cv.namedWindow('Webcam - Canny Suavizado') #Creación de ventana
def nothing2 (x): #Función que no hace nada, para poder crear las barras en la ventana
    pass
cv.createTrackbar('Umbral inferior suavizado', 'Webcam - Canny Suavizado', 0, 255, nothing2)
cv.createTrackbar('Umbral superior suavizado', 'Webcam - Canny Suavizado', 0, 255, nothing2)



#SINTONIZACIÓN DE UMBRALIZACIÓN Y OBTENCIÓN DE BORDES:
cv.namedWindow('Webcam - Umbralizacion y obtencion de bordes') #Creación de ventana
def nothing3 (x): #Función que no hace nada, para poder crear las barras en la ventana
    pass
cv.createTrackbar('Umbralizacion', 'Webcam - Umbralizacion y obtencion de bordes', 0, 255, nothing3)



#cv.VideoCapture(): Método de python para acceder a la webcam, como parámetro solo se enlista el número de webcams a las que
#se quiere acceder, empezando a contar desde el índice 0, 1, 2, etc. O se le puede pasar entre comillas la ruta de un
#video para que lo reproduzca
videito = cv.VideoCapture(0)
#cv.VideoWriter(): Método que sirve para guardar un video, en este se pasan varios parámetros:
# 1.- Nombre del archivo que guardará el video con todo y extensión.
# 2.- Código de 4 caracteres con el que se identifica cada códec, se realiza con el método cv.VideoWriter_fourcc(), este lo que
#hace es dar un código para indicar la extensión del video dependiendo de cual se haya elegido:
#       - cv2.VideoWriter_fourcc(*'mp4v'): Para archivos .mp4, .mkv, .mov, .wmv
```

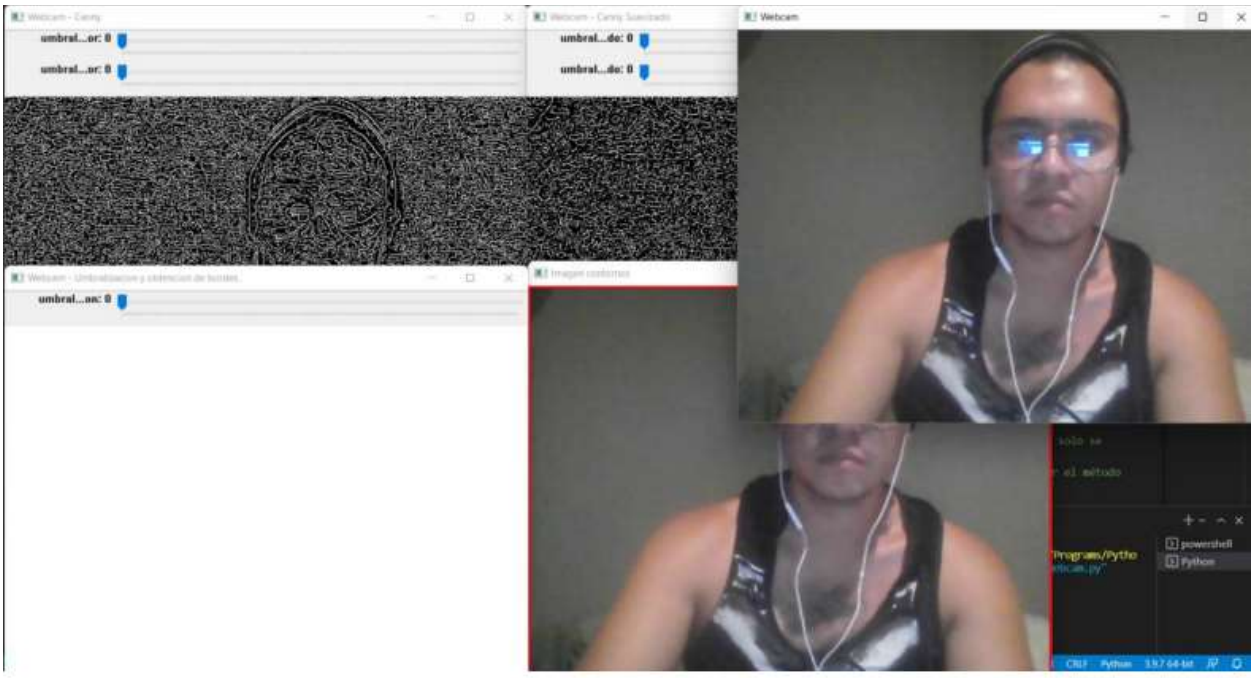
```
#       - cv2.VideoWriter_fourcc(*'DIVX'): Para archivos .avi, .mkv, .mov, .wmv
#       - cv2.VideoWriter_fourcc(*'XVID'): Para archivos .avi, .mkv, .mov, .wmv. Si es que el método anterior no funcionó.
# 3.- Velocidad de fotogramas por segundo en la secuencia del video grabado. La velocidad de fotograma estándar que solemos ver
#en los vídeos es de 24 fotogramas por segundo, si grabamos a 1 fotograma por segundo, se vería terriblemente entrecortado.
# 4.- Tamaño de los frames que componen el video, el ancho y alto en unidad de pixeles de las imagenes que conforman el video.
# 5.- Booleano que indica si queremos que el video se guarde en escala de grises o a color: True: color, False: escala de grises.
guardarVideo = cv.VideoWriter('Img/Videito.avi', cv.VideoWriter_fourcc(*'XVID'), 10, (640, 480))



#Bucle while: Cuando se analice un video, se debe realizar dentro de un bucle while para que se ejecute hasta que la ventana
#donde se muestra el video se cierre.
#.isOpened(): Método que se aplica a un objeto de OpenCV que haya abierto una webcam, este identifica si la ventana de la
#webcam está abierta o no, para que se realice el análisis de la imagen en el video.
while(videito.isOpened() == True):
    #SINTONIZACIÓN DE UMBRALES MÉTODO CANNY:
    #cv.getTrackbarPos(): Este método lo que hace es obtener el valor donde se encuentra la barra creada con el método
    #cv.createTrackbar(), como primer parámetro recibe el nombre de la barra y como segundo parámetro la ventana donde fue creada
    UmbInf = cv.getTrackbarPos('Umbral inferior', 'Webcam - Canny')
    UmbSup = cv.getTrackbarPos('Umbral superior', 'Webcam - Canny')

    #SINTONIZACIÓN DE UMBRALES MÉTODO CANNY CON SUAVIZADO GAUSSIANO:
    UmbInfSuav = cv.getTrackbarPos('Umbral inferior suavizado', 'Webcam - Canny Suavizado')
    UmbSupSuav = cv.getTrackbarPos('Umbral superior suavizado', 'Webcam - Canny Suavizado')

    #SINTONIZACIÓN DE UMBRALIZACIÓN Y OBTENCIÓN DE BORDES:
    umbral = cv.getTrackbarPos('Umbralizacion', 'Webcam - Umbralizacion y obtencion de bordes')

    #.read(): Método que se aplica a un objeto de OpenCV que haya abierto una webcam, este método obtiene dos resultados:
    # - ret: Variable booleana que puede ser true si hay un video (fotograma) capturado y false si no hay fotograma que leer,
    #este se usa más que nada para crear un if cuando se quiere leer un video ya existente en vez de grabar y mostrar uno nuevo
    #o si se quiere asegurar que la webcam está grabando.
    # - frame: La variable frame devuelve el video capturado.
    ret, frame = videito.read()
    if(ret == True):
        #FILTRO CANNY PARA DETECCIÓN DE BORDES: Lo que hace el detector Canny es mejorar la detección de bordes, ya que quita los
        #bordes que no nos interesan y deja solamente los que si nos interesan.
        #cv.Canny(): Aplica el filtro de Canny realizando los siguientes pasos de analisis de imagen:
        # 1. Suavizar la imagen (difuminarla): Reducción de ruido por medio del filtro Gaussiano.
        #cv.blur(): El método recibe la imagen o video original y luego el tamaño del Kernel que va a aplicar para realizar el
        #filtro Gaussiano de suavizado, que puede ser una matriz de tamaño: 3X3, 5X5, 7X7 o 31X31.
        suavizado = cv.blur(frame, (5, 5))
        # 2. Operador sobel (detección de bordes): Gradiente (magnitud) y orientación (ángulo) del primer borde obtenido de la
        #    imagen.
        # 3. Detector Canny: Identificar mejor los bordes de una imagen y aplicar el umbral de histéresis para preservar solamente
        #    ciertos bordes y quitar los bordes que no nos interesan.
```

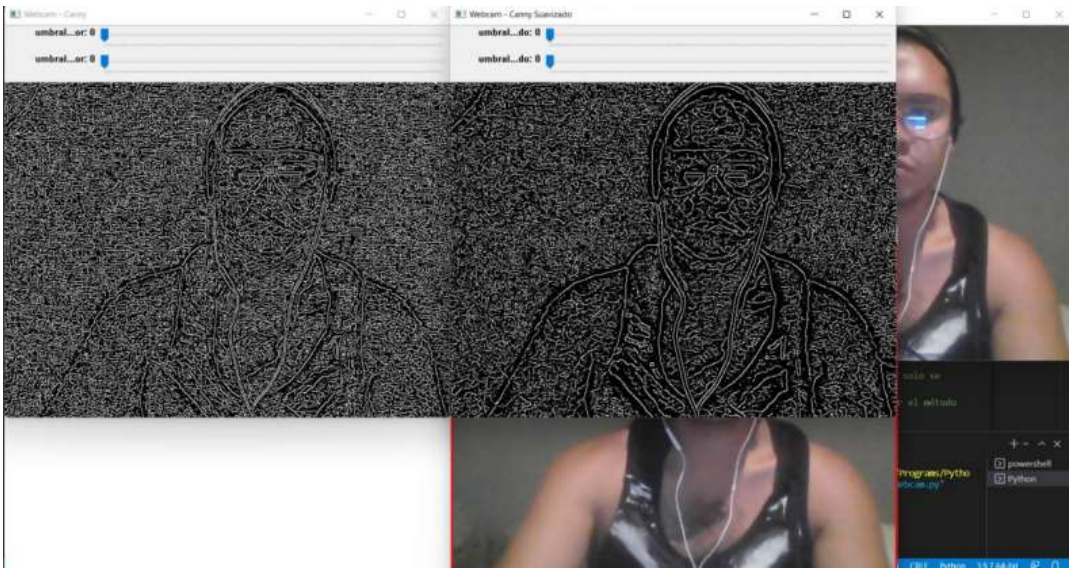
```
#cv.Canny(): Los parámetros que recibe el método son los siguientes:
# - Como primer parámetro se le pasa una imagen obtenida previamente con el método imread() o video obtenido con el
#   método VideoCapture().
# - Como segundo y tercer parámetro se le pasan umbrales para el filtrado de histéresis de la imagen, con ellos obtendremos
#   ciertos bordes solamente y despreciaremos los demás, el primer umbral es el inferior y el segundo el superior.
# - Como tercer parámetro necesitamos indicar los bordes y apertura de la imagen para el operador Sobel.
bordeCanny = cv.Canny(frame, UmbInf, UmbSup) #Filtro Canny aplicado a video original
bordeCannyBlur = cv.Canny(suavizado, UmbInfSuav, UmbSupSuav) #Filtro Canny aplicado a video con suavizado Gaussiano
#imshow(): Con el método se puede mostrar una ventana con una imagen o video, el primer parámetro es el nombre de la
#ventana donde aparecerá la imagen o video y el segundo parámetro es la imagen recopilada con el método cv.imread() o el
#video recopilado con el método .read() aplicado a un video obtenido con el método cv.VideoCapture()
cv.imshow('Webcam', frame)
cv.imshow('Webcam - Canny', bordeCanny)
cv.imshow('Webcam - Canny Suavizado', bordeCannyBlur)

#OBTENCIÓN DE CAPAS RGB:
#split(): Método que extrae los diferentes colores de la imagen en 3 distintas variables, guardándo primero el color azul,
#luego el verde y finalmente el rojo, específicamente en ese órden.
b, g, r = cv.split(frame)
#cv.imshow('Capa de azules', b) #Mostrar una ventana que tenga como título el 1er parametro y muestre la imagen del 2do
#cv.imshow('Capa de verdes', g) #Mostrar una ventana que tenga como título el 1er parametro y muestre la imagen del 2do
#cv.imshow('Capa de rojos', r) #Mostrar una ventana que tenga como título el 1er parametro y muestre la imagen del 2do

#OBTENCIÓN DE ESCALA DE GRISES
#cvtColor(): Método de la librería OpenCV que recibe como primer parámetro una imagen RGB y en su segundo parámetro recibe
#un atributo de OpenCV llamado cv.COLOR_BGR2GRAY para convertir una imagen RGB a su escala de grises.
img_gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
#cv.imshow('Escala de grises', img_gray)

"""
#El realizar el filtro de máximos, mínimos y media en video consume muchos recursos y alenta la ejecución del programa,
#pero realiza su función correctamente.
#FILTRO DE MÁXIMOS: Vuelve todo el ruido en tonos blancos, obteniendo así mayor definición en los tonos blancos de la
#imagen original, quitando negros y maximizando blancos, en función del tamaño del filtro. Recordemos que el tamaño del
#filtro se refiere al tamaño de la vecindad que rodea cada píxel de analisis en la imagen, mientras más pequeño sea el
#tamaño del vecindario, es aplicado de manera más fina el filtro. El tamaño del filtro puede ser decimal.
mask = 3.1416 #Al tamaño del filtro se le puede llamar máscara, este puede ser entero o decimal.
img2 = scipy.ndimage.maximum_filter(frame, size = mask)
cv.imshow('Imagen sin ruido - Filtro maximo', img2)

#FILTRO DE MÍNIMOS: Vuelve todo el ruido en tonos negros, obteniendo así mayor definición en los tonos negros de la imagen
#original, quitando blancos y maximizando negros, en función del tamaño de la máscara del filtro. El tamaño del filtro
#puede ser decimal.
img3 = scipy.ndimage.minimum_filter(frame, size = mask)
cv.imshow('Imagen sin ruido - Filtro minimo', img3)
```

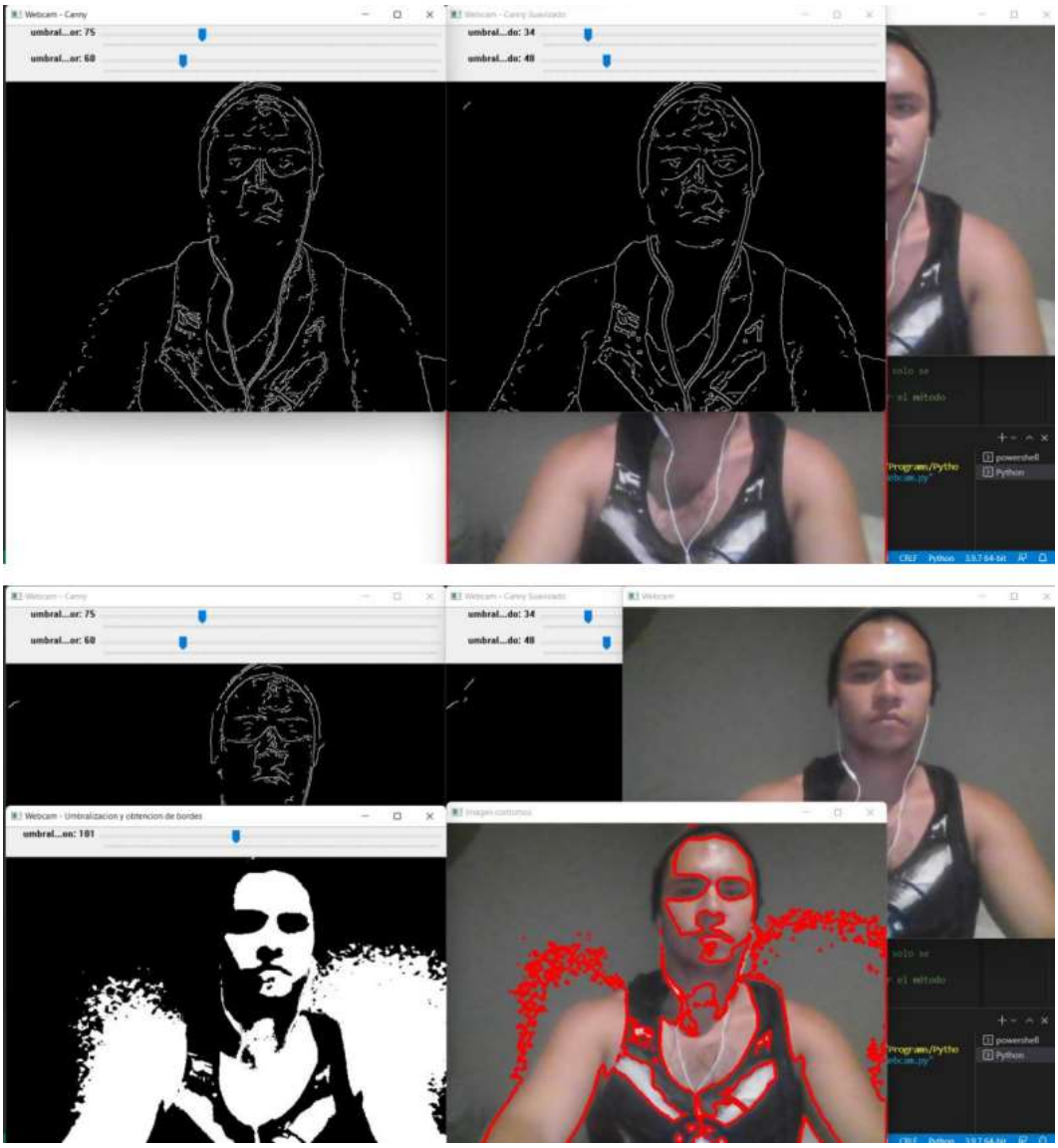
```
#FILTRO DE MEDIA: Este filtro como los dos pasados, analiza los tonos de los pixeles incluidos en la vecindad de cada píxel
#analizado, luego hace un promedio con el tono de gris de estos pixeles y deja pasar ese tono, en específico este filtro
#trata de remover el ruido de una imagen con escala de grises, conservando su forma y color original, el problema con este
#filtro es que difumina más la imagen mientras sea mayor el tamaño de la vecindad, a esto se le llama suavizado de imagen.
#El tamaño del filtro NO puede ser decimal.
img4 = scipy.ndimage.median_filter(frame, size = 6)
cv.imshow('Imagen sin ruido - Filtro medio', img4)
"""

#GUARDAR VIDEO:
#.write(): Con este método se guarda el video recopilado, para ello se debe haber usado antes el método cv.VideoWriter()
#donde se especifican las características de cómo se va a guardar el video y al método .write() se le pasa como parámetrsso
#la variable frame que devuelve el video capturado por medio del método .read()
guardarVideo.write(frame)

#UMBRALIZACIÓN Y OBTENCIÓN DE BORDES:
color_contorno = (0, 0, 255) #Variable que almacena un color BGR
grosor = 3
#cv.threshold(): Este método sirve para umbralizar una imagen, en ella se le pasa como primer parámetro la imagen en escala de
#grises que va a umbralizar, a continuación se indica el valor del umbral, el valor máximo que puede alcanzar el umbral y se
#especifica que el resultado debe ser una imagen binaria con un atributo de OpenCV llamado cv.THRESH_BINARY o si se quiere el
#inverso se puede usar cv.THRESH_BINARY_INV, en este caso se usa un guión bajo antes del nombre de la variable, porque esta es
#una variable interna.
#El guión bajo también usa para ignorar valores específicos, si no se necesita algún valor en específico del método, o no usa
#estos valores, simplemente se asignan estos valores a la "variable" representada con un guión bajo, en python el orden del
#guión importa.
_, img_umbral = cv.threshold(img_gray, umbral, 255, cv.THRESH_BINARY)
#cv.findContours(): Este método sirve para encontrar los contornos en una imagen, se puede utilizar en vez del filtro Canny, al
#método se le pasa como primer parámetro la imagen en escala de grises, en las imágenes existen varios tipos de contornos, estos
#pueden tener jerarquías de dentro hacia fuera o pueden no tenerlas, esto depende del segundo parámetro que se le pase en el
#método, en este caso vamos a utilizar el parámetro cv.RETR_LIST para que no muestre jerarquía en los bordes que encuentre,
#posteriormente se le debe indicar en el tercer parámetro la forma en la que se va a guardar la información recabada, ya que
#puede guardar todos los puntos del contorno identificado o solo guardar los puntos de las esquinas que conforman el contorno,
#para que guarde todos los puntos de los contornos se le indica con el atributo cv.CHAIN_APPROX_NONE, para guardar solo las
#esquinas se usa el atributo cv.CHAIN_APPROX_SIMPLE.
contornos, _ = cv.findContours(img_umbral, cv.RETR_LIST, cv.CHAIN_APPROX_NONE)
#cv.drawContours(): Método que sirve para dibujar los contornos recabados con el método cv.findContours(), para ello primero se
#debe indicar en qué imagen se va a dibujar el contorno, no debe ser la misma de la que se obtuvieron los contornos, después se
#indica en el segundo parámetro la variable en donde están almacenados los contornos, en el tercer parámero se establece si se
#quiere que sea hueco o sólido el contorno, como se busca que sea sólido se pone -1, si fuera hueco, en esta parte se indicaría
#los pixeles del perímetro (borde) del contorno, posteriormente se indica el color y finalmente los contornos que quiero que
#muestre.
cv.drawContours(frame, contornos, -1, color_contorno, grosor)
cv.imshow('Webcam - Umbralizacion y obtencion de bordes', img_umbral) #Mostrar una ventana que tenga como título el 1er parametro y muestre la imagen del 2do
```

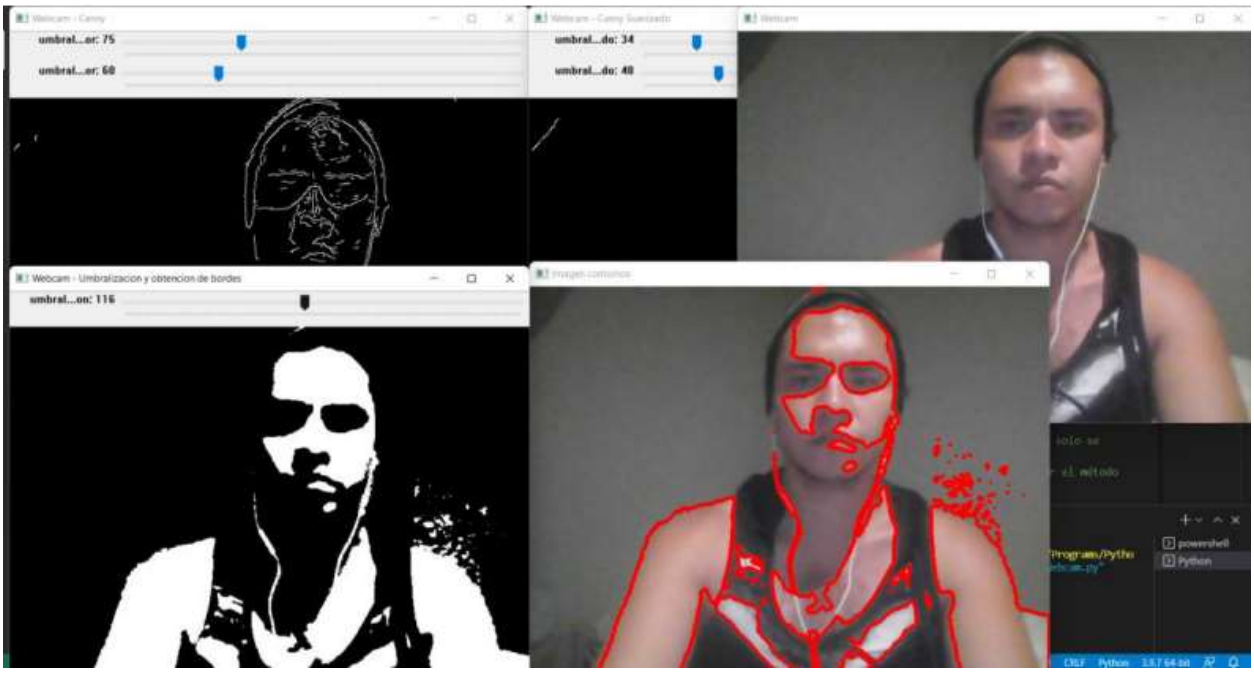
```
        cv.imshow('Imagen contornos', frame) #Mostrar una ventana que tenga como título el 1er parametro y muestre la imagen del 2do


        #Se utiliza un condicional if para saber cuando se debe dejar de mostrar el video, para ello se usan los siguientes
        #métodos:
        #waitKey(): Método que permite a los usuarios mostrar una ventana durante un número de milisegundos determinados si su
        #parámetro es un número mayor a 1 o hasta que se presione cualquier tecla. Toma tiempo en milisegundos como parámetro y
        #espera el tiempo dado para destruir la ventana:
        # - Si se pasa 0 como argumento, espera hasta que se presiona cualquier tecla.
        # - Si se pasa 1 como argumento, espera hasta que se presione una tecla específica para cerrar la ventana.
        #El método waitKey() está monitoreando si se presiona una tecla o no, este modo se activa si se le pasa como parámetro un
        #número 1, de esta manera:
        # - Si no se presiona una tecla retorna un valor -1
        # - Si se presiona una tecla, retorna un valor ASCII correspondiente a la tecla que se oprimió, con el método ord() podemos
        # indicar el código ASCII de la letra del teclado que indiquemos en su parámetro.
        if(cv.waitKey(1) == ord('s')):
            #print(): Imprime en consola el mensaje que se incluya dentro de su paréntesis en forma de string
            print("cv.waitKey(1): " + str(cv.waitKey(1)))
            print("cv.waitKey(0): " + str(cv.waitKey(0)))
            break #Cuando se presione la tecla s, se cerrará la ventana del video.
    else:
        break


#.release(): Con este método se libera la webcam, tanto en forma de software como de hardware.
guardarVideo.release() #Dejar de tener acceso al directorio donde se guardará el video.
videito.release() #Dejar de tener acceso a la webcam.
#destroyAllWindows(): Función que permite a los usuarios destruir todas las ventanas en cualquier momento. No toma ningún
#parámetro y no devuelve nada, esto se incluye para que al cerrar la ventana después del método waitKey() se destruyan para
#poder utilizarse en otra cosa.
cv.destroyAllWindows()


#El video se cierra al presionar la letra S, no cuando se presiona el tache.
#La documentación para realizar todas las operaciones de este ejercicio son las siguientes:
#FILTRO DE MÁXIMOS, MÍNIMOS Y MEDIA: Para quitar ruido.
#https://docs.scipy.org/doc/scipy/reference/generated/scipy.ndimage.maximum_filter.html#scipy.ndimage.maximum_filter


#MÉTODOS PARA CONTROLAR LOS EVENTOS DEL MOUSE: Para ver cuando se haya dado clic, se suelte el clic del mouse, etc.
#https://docs.opencv.org/3.4/d0/d90/group__highgui__window__flags.html#gga927593befdddc7e7013602bca9b079b0ad3419100fc2d7688c6dbe3da030fbfd9


#FILTRO CANNY: Detección de bordes.
#https://docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html#ga2a671611e104c093843d7b7fc46d24af


#TRANSFORMADA DE HOUGH: Detección de líneas.
#https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html
#https://docs.opencv.org/3.4/dd/d1a/group__imgproc__feature.html#ga46b4e588934f6c8dfd509cc6e0e4545a
```

```
#CONTOURS: Operaciones para medir las figuras en la imagen donde ya se haya identificado los bordes.

#https://docs.opencv.org/4.x/dd/d49/tutorial_py_contour_features.html
```






Webcam - Canny
umbral...or: 75
umbral...or: 60
Webcam - Canny Suavizado
umbral...do: 34
umbral...do: 48
Webcam - Umbralizacion y obtencion de bordes
umbral...on: 101
Imagen contornos
Webcam
powershell
Python



Reconocimiento de Rostros

Se realizó el análisis de rostros por medio de la webcam utilizando las bases de datos de la librería OpenCV, llegando a los siguientes resultados al ejecutar el archivo 9.- Detección de Rostros con Webcam, cuyo código se presenta a continuación:

```
# -*- coding: utf-8 -*-
#Comentario de una sola linea con el simbolo #, en Python para nada se deben poner acentos sino el programa
#puede fallar o imprimir raro en consola, la primera línea de código es para que no tenga error, pero aún
#así al poner un ángulo saldrá raro en consola, la línea debe ponerse tal cual como aparece y justo al inicio.

#IMPORTACIÓN DE LIBRERÍAS:
import cv2 as cv #Librería OpenCV: Sirve para todo tipo de operaciones de visión artificial
import numpy as np #Librería numpy: Realiza operaciones matemáticas complejas
import scipy.ndimage #Librería scipy: Se utiliza para aplicar filtros de máximos y mínimos a una imagen


#SINTONIZACIÓN DE UMBRALES MÉTODO CANNY SIN SUAVIZADO GAUSSIANO:
#cv.namedWindow(): Sirve para crear una ventana donde se podrá mostrar un video o imagen recopilada, se debe declarar desde
#este punto para crear barras que varíen el valor de los umbrales del filtro Canny, para ver cómo se debe sintonizar para
#obtener de mejor manera los bordes, dando valores de 0 a 255
cv.namedWindow('Identificación de Rostros') #Ventana que muestra los bordes Canny
#cv.createTrackbar(): Este método sirve para crear y mostrar una barra sobre una ventana específica que muestra una imagen o
#video, esto para variar cierto valor en tiempo real, este recibe los siguientes parámetros:
#   - Nombre de la barra (Track bar)
#   - Ventana a la cual se aplicará, esta debe tener el mismo nombre que la ventana creada con el método cv.namedWindow() y a la
#   cual se aplicará el método canny.
```

```
#   - Valor mínimo de la barra.
#   - Valor máximo de la barra.
#   - Función que a fuerza se debe ejecutar cuando se mueva la barra, esta puede ejecutar una acción o en este caso solo se
#   declara y no hace nada.
#Se debe crear la función que no hace nada en este caso para que se pueda crear la barra en la ventana antes de usar el método
#cv.createTrackbar().
def nothing (x):
    pass
#Se crea una barra con el método cv.createTrackbar() para variar el umbral inferior y superior del método Canny
cv.createTrackbar('Vecindario mínimo', 'Identificación de Rostros', 3, 30, nothing)



#DETECCIÓN DE ROSTROS CON WEBCAM:
color_rectangulo_rostro = (0, 0, 255) #Color BGR del borde del rectángulo de reconocimiento facial para rostros
grosor_rectangulo_rostro = 2 #Grosor del borde del rectángulo que identifica rostros
color_circulo_ojo = (255, 255, 0) #Color BGR del borde del círculo de reconocimiento facial para ojos
grosor_circulo_ojo = 2 #Grosor del borde del círculo que identifica ojos



#cv.VideoCapture(): Método de python para acceder a la webcam, como parámetro solo se enlista el número de webcams a las que
#se quiere acceder, empezando a contar desde el índice 0, 1, 2, etc. O se le puede pasar entre comillas la ruta de un
#video para que lo reproduzca
videito = cv.VideoCapture(0)
#cv.VideoWriter(): Método que sirve para guardar un video, en este se pasan varios parámetros:
# 1.- Nombre del archivo que guardará el video con todo y extensión.
# 2.- Código de 4 caracteres con el que se identifica cada códec, se realiza con el método cv.VideoWriter_fourcc(), este lo que
#hace es dar un código para indicar la extensión del video dependiendo de cual se haya elegido:
#       - cv2.VideoWriter_fourcc(*'mp4v'): Para archivos .mp4, .mkv, .mov, .wmv
#       - cv2.VideoWriter_fourcc(*'DIVX'): Para archivos .avi, .mkv, .mov, .wmv
#       - cv2.VideoWriter_fourcc(*'XVID'): Para archivos .avi, .mkv, .mov, .wmv. Si es que el método anterior no funcionó.
# 3.- Velocidad de fotogramas por segundo en la secuencia del video grabado. La velocidad de fotograma estándar que solemos ver
#en los vídeos es de 24 fotogramas por segundo, si grabamos a 1 fotograma por segundo, se vería terriblemente entrecortado.
# 4.- Tamaño de los frames que componen el video, el ancho y alto en unidad de pixeles de las imagenes que conforman el video.
# 5.- Booleano que indica si queremos que el video se guarde en escala de grises o a color: True: color, False: escala de grises.
guardarVideo = cv.VideoWriter('Img/Deteccion_Rostros.avi', cv.VideoWriter_fourcc(*'XVID'), 10, (640, 480))



#Bucle while: Cuando se analice un video, se debe realizar dentro de un bucle while para que se ejecute hasta que la ventana
#donde se muestra el video se cierre.
#.isOpened(): Método que se aplica a un objeto de OpenCV que haya abierto una webcam, este identifica si la ventana de la
#webcam está abierta o no, para que se realice el análisis de la imagen en el video.
while(videito.isOpened() == True):
    #SINTONIZACIÓN DE UMBRALES MÉTODO CANNY:
    #cv.getTrackbarPos(): Este método lo que hace es obtener el valor donde se encuentra la barra creada con el método
    #cv.createTrackbar(), como primer parámetro recibe el nombre de la barra y como segundo parámetro la ventana donde fue creada
```

```
vecindario_min = cv.getTrackbarPos('Vecindario mínimo', 'Identificación de Rostros')

#.read(): Método que se aplica a un objeto de OpenCV que haya abierto una webcam, este método obtiene dos resultados:
# - ret: Variable booleana que puede ser true si hay un video (fotograma) capturado y false si no hay fotograma que leer,
#este se usa más que nada para crear un if cuando se quiere leer un video ya existente en vez de grabar y mostrar uno nuevo
#o si se quiere asegurar que la webcam está grabando.
# - frame: La variable frame devuelve el video capturado.
ret, frame = videito.read()
if(ret == True):
    #OBTENCIÓN DE ESCALA DE GRISES:
    #cvtColor(): Método de la librería OpenCV que recibe como primer parámetro una imagen RGB y en su segundo parámetro recibe
    #un atributo de OpenCV llamado cv.COLOR_BGR2GRAY para convertir una imagen RGB a su escala de grises.
    img_gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
    cv.imshow('Escala de grises', img_gray)

    #DETECCIÓN DE ROSTROS:
    #cv.CascadeClassifier(): Este es un clasificador en cascada, esto se refiere a que dependiendo de ciertas características
    #del reconocimiento facial, puede ir segmentando hasta identificar un rostro, ya sea de hombre, mujer, perro, gato, etc.
    #porque cada uno tiene características especiales, para que funcione esto, se debe indicar como parámetro del método la
    #base de datos descargada de la documentación de Github de OpenCV con extensión xml.
    #https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html
    #Para la identificación de rostros, específicamente se utiliza la base de datos llamada haarcascade_frontalface_default.xml:
    #https://raw.githubusercontent.com/opencv/opencv/master/data/haarcascades/haarcascade_frontalface_default.xml
    clasificador_caras = cv.CascadeClassifier('API_Reconocimiento_Facial/7.- Reconocimiento facial OpenCV - rostros.xml')

    #objetoCascadeClassifier.detectMultiScale(): Con este método se accede a la base de datos de reconocimiento facial de
    #OpenCV para el reconocimiento de rostros, a este se le pasa como parámetro la imagen que se quiere analizar, pero antes se
    #debió haber obtenido la escala de grises de la imagen y jalado con el método cv.CascadeClassifier() la base de datos de
    #reconocimiento de rostros.
    caras = clasificador_caras.detectMultiScale(img_gray)

    #Bucle for para obtener las coordenadas y dimensiones de las caras
    for (x, y, ancho, alto) in caras:
        #cv.rectangle(): Con este método perteneciente a la librería OpenCV se puede crear un rectángulo, en él se indica
        #primero en que imagen obtenida con el método imread() se va a dibujar la figura, luego las coordenadas iniciales x,y
        #de la imagen en donde se va a dibujar la esquina superior izquierda del rectángulo, que en este caso son las
        #coordenadas x,y obtenidas con el reconocimiento facial, seguido de las coordenadas finales x,y que son las que
        #incluyen el ancho y alto del rostro identificado, posteriormente se indica su color y si se quiere que sea hueco o
        #sólido el rectángulo, si se buscara que fuera sólido se pondría un -1, si fuera hueco, en esta parte se indicaría
        #los pixeles del grosor del borde del rectángulo.
        cv.rectangle(frame, (x, y), (x + ancho, y + alto), color_rectangulo_rostro, grosor_rectangulo_rostro)
        #Para cada parte del rostro que se quiera identificar se debe crear un clasificador, todos alimentándose de la misma
        #base de datos, pero usándose para identificar distintas partes del rostro, para la parte específica de los ojos se usa
        #la base de datos llamada haarcascade_eye.xml de la documentación de OpenCV que se encuentra en el siguiente link:
        #https://raw.githubusercontent.com/opencv/opencv/master/data/haarcascades/haarcascade_eye.xml
```

```
        clasificador_ojos = cv.CascadeClassifier('API_Reconocimiento_Facial/7.- Reconocimiento facial OpenCV - ojos.xml')
        #Extraer las coordenadas de la matriz cara desde la coordenada "y" hasta "y + alto" y la coordenada "x" hasta
        #"x + ancho" de la imagen en escala de grises:
        caraIndividual = img_gray[y:y+alto, x:x+ancho]
        #objetoCascadeClassifier.detectMultiScale(): Con este método se accede a la base de datos de reconocimiento facial de
        #OpenCV para el reconocimiento de ojos, a este se le pasa como parámetro la imagen que se quiere analizar, pero antes
        #se debió haber obtenido la colección de datos de identificación de rostros y jalado con el método cv.CascadeClassifier()
        #la base de datos de reconocimiento ocular. Si no se está haciendo correctamente el análisis del rostro, en esta parte
        #es donde se puede sintonizar la detección de rostros para mejorarla con los siguientes parámetros:
        # - Matriz a analizar: Que en el caso de los ojos es una submatriz de la matriz que identifica rostros.
        # - Factor de escala: Parámetro que especifica cuánto se reduce el tamaño de la imagen en cada escala de la imagen
        #   original, el factor de escala por default es de 1.1 (scaleFactor). Este valor se sube para que detecte cosas más
        #   grandes y se reduce cuando se quiere identificar partes más pequeñas.
        # - Vecindario mínimo: Parámetro que especifica cuántos pixeles vecinos debe tener cada rectángulo candidato para
        #   conservarlo, el vecindario mínimo por default es de 3 (minNeighbors).
        # - Tamaño mínimo: Tamaño mínimo posible del vecindario (minSize).
        ojos = clasificador_ojos.detectMultiScale(caraIndividual, scaleFactor = 1.01, minNeighbors = vecindario_min, minSize = (10, 10))
        #Con la imagen 'Lena.jpg': scaleFactor = 1.1, minNeighbors = 10, minSize = (10, 10)
        #Con la imagen 'Familia.jpg': scaleFactor = 1.01, minNeighbors = 8, minSize = (10, 10)
        for (x1, y1, ancho1, alto1) in ojos:
            #cv.circle(): Con este método perteneciente a la librería OpenCV se puede crear un círculo, en él se indica primero
            #en que imagen obtenida con el método imread() se va a dibujar la figura, luego la coordenada x,y de la imagen en
            #donde se va a mostrar, el radio en pixeles del círculo, su color y si se quiere que sea hueco o sólido el círculo,
            #como se busca que sea sólido se pone -1, si fuera hueco, en esta parte se indicaría los pixeles del perímetro
            #(borde) del círculo.
            #En la interfaz se mostrará un círculo para mostrar de dónde se está obteniendo el color de la imagen.
            radio = int((ancho1 + alto1)/4)
            cv.circle(frame, (x+x1+radio, y+y1+radio), radio, color_circulo_ojo, grosor_circulo_ojo)

    cv.imshow("Identificación de Rostros", frame) #Mostrar una ventana que tenga como título el 1er parametro y muestre la imagen del 2do

    #GUARDAR VIDEO:
    #.write(): Con este método se guarda el video recopilado, para ello se debe haber usado antes el método cv.VideoWriter()
    #donde se especifican las características de cómo se va a guardar el video y al método .write() se le pasa como parámetro
    #la variable frame que devuelve el video capturado por medio del método .read()
    guardarVideo.write(frame)

    #Se utiliza un condicional if para saber cuando se debe dejar de mostrar el video, para ello se usan los siguientes
    #métodos:
    #waitKey(): Método que permite a los usuarios mostrar una ventana durante un número de milisegundos determinados si su
    #parámetro es un número mayor a 1 o hasta que se presione cualquier tecla. Toma tiempo en milisegundos como parámetro y
    #espera el tiempo dado para destruir la ventana:
    # - Si se pasa 0 como argumento, espera hasta que se presiona cualquier tecla.
    # - Si se pasa 1 como argumento, espera hasta que se presione una tecla específica para cerrar la ventana.
    #El método waitKey() está monitoreando si se presiona una tecla o no, este modo se activa si se le pasa como parámetro un
```

```
        #número 1, de esta manera:
        # - Si no se presiona una tecla retorna un valor -1
        # - Si se presiona una tecla, retorna un valor ASCII correspondiente a la tecla que se oprimió, con el método ord() podemos
        # indicar el código ASCII de la letra del teclado que indiquemos en su parámetro.
        if(cv.waitKey(1) == ord('s')):
            #print(): Imprime en consola el mensaje que se incluya dentro de su paréntesis en forma de string
            print("cv.waitKey(1): " + str(cv.waitKey(1)))
            print("cv.waitKey(0): " + str(cv.waitKey(0)))
            break #Cuando se presione la tecla s, se cerrará la ventana del video.

        #GUARDAR UNA FOTO DEL VIDEO: Cuando se presione la tecla a, toma una foto del video
        if (cv.waitKey(1) == ord('a')):
            print("Captura de video tomada cuando se presionó la letra a")
            #GUARDAR UNA FOTO DEL VIDEO:
            #imwrite(): Método de la librería OpenCV que sirve para guardar una imagen, como primer parámetro tengo que poner el
            #nombre con el que se guardará la imagen con todo y extensión y en el segundo debo poner la variable de la que extrae
            #la imagen.
            cv.imwrite("Img/Fotovideo.jgp", frame)
    else:
        break

#.release(): Con este método se libera la webcam, tanto en forma de software como de hardware.
guardarVideo.release() #Dejar de tener acceso al directorio donde se guardará el video.
videito.release() #Dejar de tener acceso a la webcam.
#destroyAllWindows(): Función que permite a los usuarios destruir todas las ventanas en cualquier momento. No toma ningún
#parámetro y no devuelve nada, esto se incluye para que al cerrar la ventana después del método waitKey() se destruyan para
#poder utilizarse en otra cosa.
cv.destroyAllWindows()

#El video se cierra al presionar la letra S, no cuando se presiona el tache.

#La documentación para realizar todas las operaciones de este ejercicio son las siguientes:
#CLASIFICADOR EN CASCADA: Clasificador que se usa en la identificación de rostros
#https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html
#https://docs.opencv.org/3.4/d1/de5/classcv_1_1CascadeClassifier.html#aaf8181cb63968136476ec4204ffca498

#BASE DE DATOS DE RECONOCIMIENTO FACIAL:
#https://github.com/opencv/opencv/tree/master/data/haarcascades

#IDENTIFICACIÓN DE ROSTROS POR MEDIO DE UNA LIBRERÍA DE PYTHON QUE NO ES OPENCV:
#https://pypi.org/project/face-recognition/
```