PROCESAMIENTO DE IMÁGENES

CLASE 3: Librería OPENCV

A continuación se presentan algunos ejemplos básicos de las posibilidades de la librería OPENCV. La ejecución se realizó con el núcleo (kernel) de Jupyter Notebook de Anaconda3 para Python 3.9.13

Se recomienda realizar la ejecución por cada celda desde el principio, se generará una ventana en la mayoría de los casos. Para detener la ejecución y cerrar la ventana, solo debe presionar una tecla.

A continuación se detalla el proceso de ejecución de cada celda de la notebook compartida en el repositorio.

- Celda 1: Importación cv2, carga y visualización de imagen

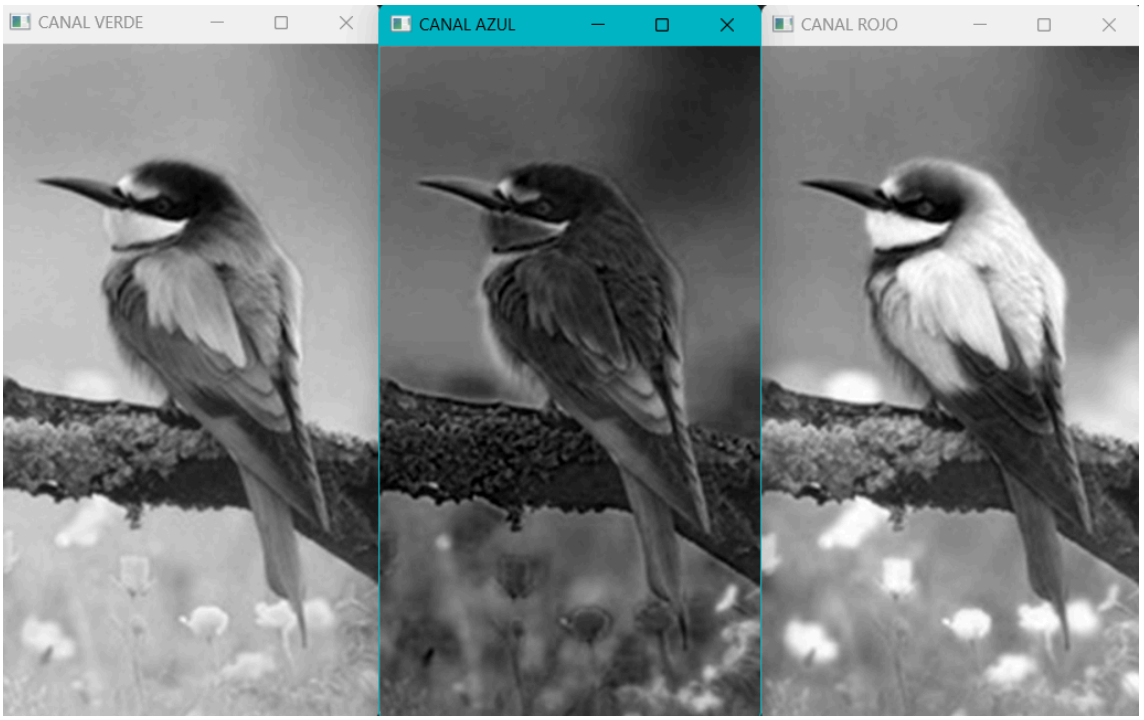
```
#CARGA DE LIBRERIA OPENCV - VISUALIZACIÓN DE IMAGEN ORIGINAL
import cv2
img = cv2.imread('imagen.jpg',1)                    #carga la imagen original
cv2.imshow('ORIGINAL',img)
cv2.waitKey(0)
cv2.destroyAllWindows()                              #presionar una TECLA para TERMINAR
```

- Celda 2: Conversión de una imagen a escala de grises

```
#VISUALIZACIÓN DE IMAGEN EN ESCALA DE GRISES
img_byn = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)     # convierte la imagen en escala de grises
cv2.imshow('ESCALA DE GRISES',img_byn)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- Celda 3: Visualización de los canales azul, verde y rojo, en escala de grises

```
#VISUALIZACIÓN DE CADA CANAL (azul, verde, rojo) EN ESCALA DE GRISES
img_azul, img_verde, img_roja = cv2.split(img)     #se muestran los 3 canales en escala de grises
cv2.imshow('CANAL AZUL', img_azul)
cv2.imshow('CANAL VERDE', img_verde)
cv2.imshow('CANAL ROJO', img_roja)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



- Celda 4: Información de una imagen

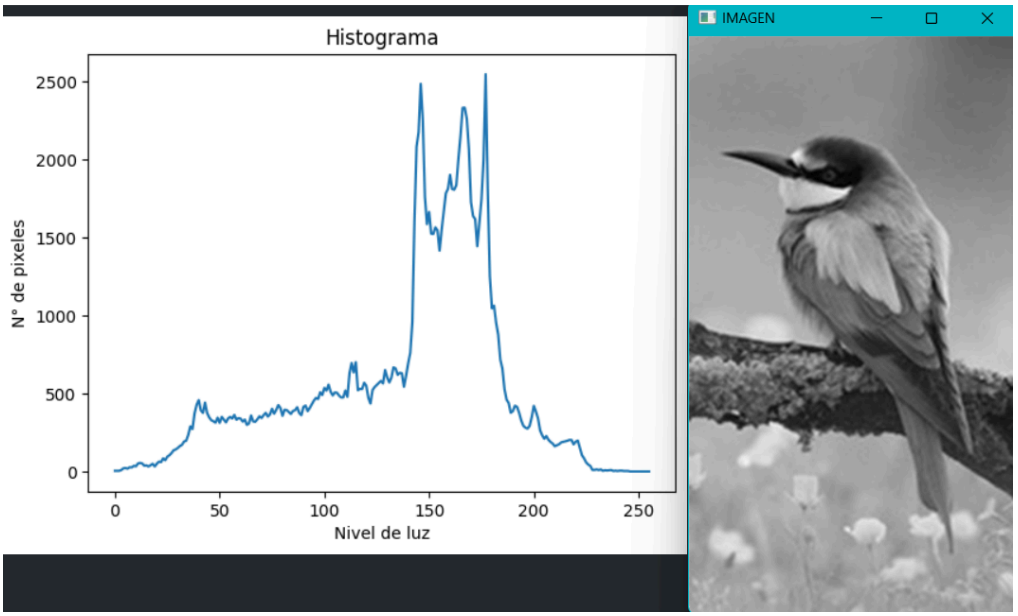
```
#INFORMACIÓN DE IMAGEN
tamanio = img.size                          #obtención de las características de la imagen original
alto, ancho, canales = img.shape
tipo = img.dtype
print("Tamaño: " + str(tamanio) + " bytes")
print("Ancho: " + str(ancho) + " pixeles")
print("Alto: " + str(alto) + " pixeles")
print("N° de Canales: " + str(canales))
print("Tipo: " + str(tipo))
```

✓ 0.0s

```
Tamaño: 405720 bytes
Ancho: 276 pixeles
Alto: 490 pixeles
N° de Canales: 3
Tipo: uint8
```

- Celda 5: Histograma

```
#HISTOGRAMA
from matplotlib import pyplot as plt
img0 = cv2.imread('imagen.jpg',0)       #carga de imagen en blanco y negro
hist = cv2.calcHist([img0], [0], None, [256], [0, 256])   #imagenes, canales(0=img b&n, 1-2-3(canales img color)),
                                                            #región(none=total), histsize, rango nivel de intensidad.
cv2.imshow('IMAGEN', img0)
plt.xlabel('Nivel de luz')
plt.ylabel('N° de pixeles')
plt.title('Histograma')
plt.plot(hist)
plt.show()
cv2.waitKey(0)
cv2.destroyAllWindows()
```

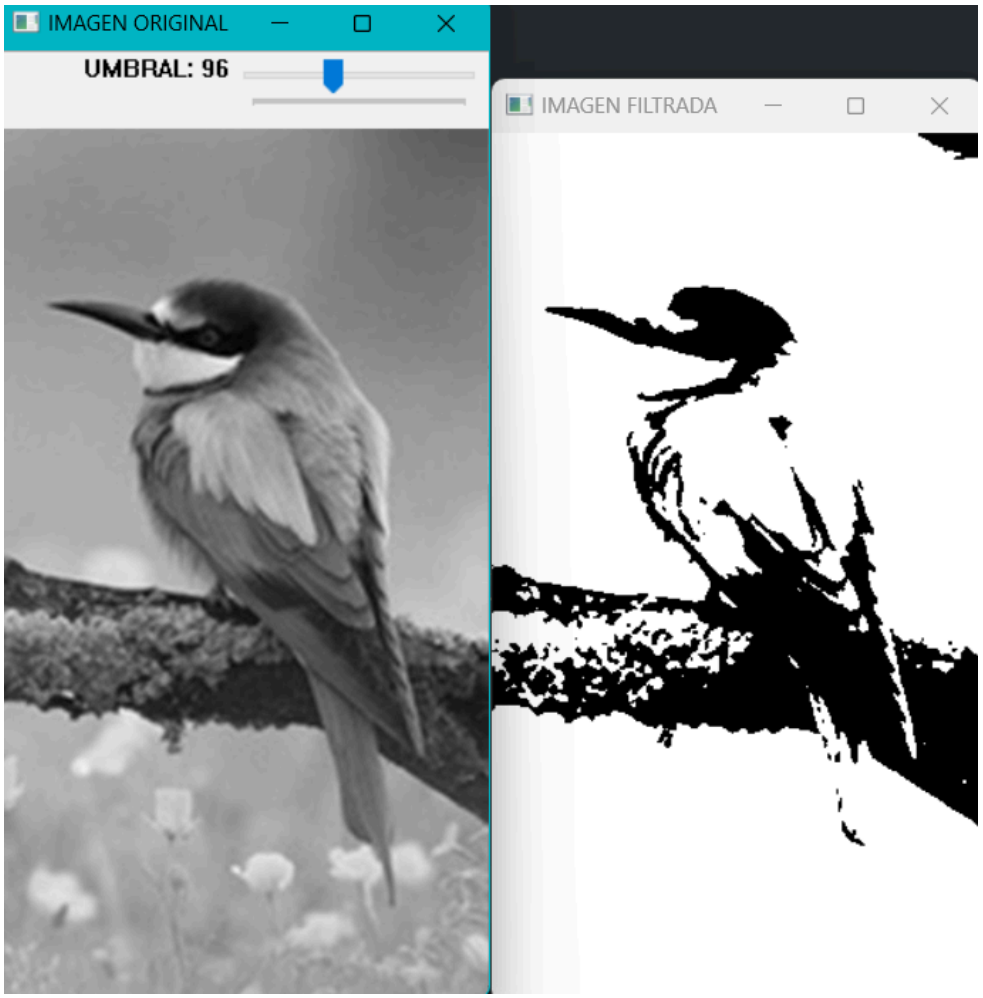


- Celda 6: Segmentación por filtros basados en el umbral

  Umbral simple(binario). Se utiliza para separar un objeto del fondo.

```
#FILTRO BASADO EN EL UMBRAL
#UMBRAL SIMPLE
def actualizar_imagen(umbral):
    _, img_umbral = cv2.threshold(img0, umbral, 255, cv2.THRESH_BINARY)       #primer argumento no se utiliza, threshold=filtro binario -
                                                                             #(imagen, valor umbral, valor máximo, tipo umbral)
    cv2.imshow('IMAGEN FILTRADA', img_umbral)
cv2.imshow('IMAGEN ORIGINAL', img0)
cv2.createTrackbar('UMBRAL', 'IMAGEN ORIGINAL', 0, 255, actualizar_imagen)   #barra de desplazamiento
actualizar_imagen(0)                                                          #llamada a la función
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Mover la barra de desplazamiento para fijar el valor umbral.



- Celda 7: Filtro CANNY, para detección de bordes

```
#FILTRO CANNY - DETECCIÓN DE BORDES
img_canny = cv2.Canny(img0, 80, 150)      #imagen, umbral inferior, umbral superior
cv2.imshow('IMAGEN FILTRADA', img_canny)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OPENCV - Documentación.

https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html