UNIVERSIDAD DEL BÍO-BÍO

17 Programa final – Procesamiento de imágenes usando webcam

Tutorial Procesamiento de Imagen con webcam

Estudiantes Practicantes:
Luis Pereira

Profesor:
Luis Vera

Laboratorio CIMUBB

2023-2

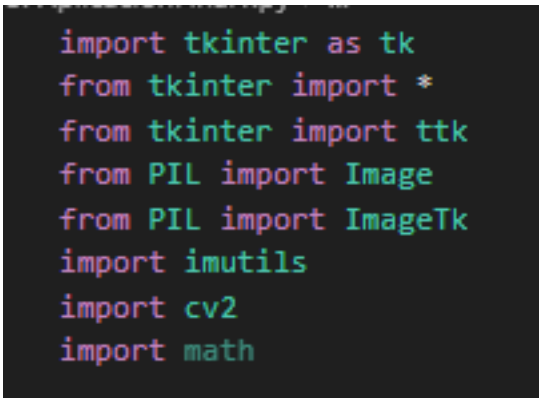
Procesamiento de imágenes usando webcam

Como proyecto final, se unirán los proyectos anteriores, creando un solo programa que contenga todo el análisis de imágenes.

Para ellos, creamos otro archivo .py y le añadimos las siguientes librerías.

Importación de Bibliotecas

1) **tkinter as tk**: Se utiliza para crear la interfaz gráfica de usuario.
2) **from tkinter import *:** Se importa el módulo principal de tkinter para acceder a las clases y funciones básicas.
3) **from tkinter import ttk**: Importa el módulo de tkinter para agregar estilos y temas.
4) **from PIL import Image, ImageTk**: Estas bibliotecas se utilizan para trabajar con imágenes y conversiones de formatos de imágenes.
5) **import imutils:** Imutils es una biblioteca útil para procesar imágenes y video de manera más eficiente.
6) **import cv2**: OpenCV es una biblioteca ampliamente utilizada para procesar imágenes y vídeo.
7) **import math**: Importa el módulo de matemáticas de Python, que no parece estar siendo utilizado en el código.



Inicio de la Ventana

El código crea una ventana de interfaz de usuario llamada ventana con las siguientes características:

1) **Tamaño**: 1320x800 píxeles.
2) La ventana no es redimensionable.
3) **Título**: "Proyecto procesamiento de imagen con Webcam".

```
ventana = tk.Tk()
ventana.geometry("1320x800")
ventana.resizable(0, 0)
ventana.title("Proyecto de procesamiento de imágenes con Webcam")
```

Función camara()

Esta función inicia la cámara web al abrir una conexión con la cámara de índice 0. Luego, llama a la función iniciar().

```
def camara():
    global capture
    capture = cv2.VideoCapture(0)
    iniciar()
```

Función iniciar()

La función **iniciar()** se ejecuta de forma continua después de iniciar la cámara. Realiza las siguientes acciones:

1) Captura imágenes de la cámara en tiempo real.
2) Redimensiona la imagen capturada.
3) Convierte la imagen de BGR (Blue-Green-Red) a RGB (Red-Green-Blue) utilizando OpenCV.
4) Actualiza la etiqueta de imagen (LImagen) en la interfaz de usuario con la imagen capturada.
5) Llama a sí misma después de un breve retraso (10 ms) para lograr una vista en tiempo real.

```
def iniciar():
    global capture
    if capture is not None:
        BCapturar.place(x=250, y=330, width=91, height=23)
        ret, frame = capture.read()
        if ret == True:
            frame = imutils.resize(frame, width=311)
            frame = imutils.resize(frame, height=241)
            ImagenCamara = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            im = Image.fromarray(ImagenCamara)
            img = ImageTk.PhotoImage(image=im)
            LImagen.configure(image=img)
            LImagen.image = img
            LImagen.after(10, iniciar)
        else:
            LImagen.image = ""
            capture.release()
```

Función Capturar()

La función **Capturar**() captura una imagen de la cámara en tiempo real y realiza las siguientes acciones:

1) Redimensiona la imagen capturada.
2) Crea una versión en escala de grises de la imagen capturada (CapturaG) y una versión en color (Captura).
3) Actualiza dos etiquetas de imagen en la interfaz de usuario (GImagenROI y LImagenRecorte) con las imágenes capturadas.

```
def Capturar():
    global valor, Captura, CapturaG
    camara = capture
    return_value, image = camara.read()
    frame = imutils.resize(image, width=301)
    frame = imutils.resize(frame, height=221)
    CapturaG = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    Captura = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    im = Image.fromarray(Captura)
    img = ImageTk.PhotoImage(image=im)
    imG = Image.fromarray(CapturaG)
    imgG = ImageTk.PhotoImage(image=imG)
    GImagenROI.configure(image=imgG)
    GImagenROI.image = imgG
    LImagenRecorte.configure(image=img)
    LImagenRecorte.image = img
```

Funciones de Umbralización

Las funciones **rgb()**, **umbralizar_escala_grises()**, **y umbralizar_rgb_a_escala_grises()** son componentes clave en el procesamiento de imágenes en esta aplicación. Cada una de estas funciones realiza umbralización en la imagen capturada en contextos específicos y configura etiquetas de imagen para mostrar los resultados de manera clara y comprensible para el usuario.

La función **rgb()** permite al usuario definir rangos de valores para los canales de color rojo (R), verde (G) y azul (B) y, a continuación, aplica umbrales para identificar las áreas en la imagen que cumplen con estos criterios de color. Posteriormente, muestra el resultado en una etiqueta de imagen en la interfaz de usuario.

Por otro lado, la función **umbralizar_escala_grises()** se utiliza para umbralizar la imagen en escala de grises, permitiendo al usuario especificar un valor de umbral. Esta función convierte los píxeles con valores superiores al umbral en blanco y los píxeles por debajo del umbral en negro, lo que resalta las áreas de interés en la imagen en escala de grises.

La función **umbralizar_rgb_a_escala_grises()** realiza una umbralización similar a la anterior, pero en una versión en escala de grises de la imagen original fusionada. El usuario puede ajustar el valor de umbral para destacar áreas específicas de la imagen en escala de grises.

```
def rgb():
    global img_mask, img_aux, bin_imagen
    Minimos = (int(SRedI.get()), int(SGreenI.get()), int(SBlueI.get()))
    maximos = (int(SRedD.get()), int(SGreenD.get()), int(SBlueD.get()))
    img_mask = cv2.inRange(ImgRec, Minimos, maximos)
    img_aux = img_mask
    img_mask = Image.fromarray(img_mask)
    img_mask = ImageTk.PhotoImage(image=img_mask)
    LImagenManchas.configure(image=img_mask)
    LImagenManchas.image = img_mask
    _, bin_imagen = cv2.threshold(img_aux, 0, 255, cv2.THRESH_BINARY_INV)

def umbralizar_escala_grises():
    global thresh1
    valor = int(numeroUmbra.get())
    ret, thresh1 = cv2.threshold(CapturaG, valor, 255, cv2.THRESH_BINARY)
    Umbral = Image.fromarray(thresh1)
    Umbral = ImageTk.PhotoImage(image=Umbral)
    UImagen.configure(image=Umbral)
    UImagen.image = Umbral

def umbralizar_rgb_a_escala_grises():
    global thresh2
    valor = int(numeroUmbraR.get())
    ret1, thresh2 = cv2.threshold(imagen_umbralizadaG, valor, 255, cv2.THRESH_BINARY)
    Umbral = Image.fromarray(thresh2)
    Umbral = ImageTk.PhotoImage(image=Umbral)
    LImagenManchas.configure(image=Umbral)
    LImagenManchas.image = Umbral
```

Funciones de Análisis de Manchas

Las funciones **contar_manchas_blancas()** y **contar_manchas_negras()** realizan análisis de manchas en las imágenes umbralizadas. Calculan la cantidad de manchas blancas o negras, así como el porcentaje del área de la imagen cubierta por estas manchas. Los resultados se muestran en cuadros de texto en la interfaz de usuario.

```python
def contar_manchas_blancas(boton):

    if boton == 1:
        num_pixels_con_manchas_blancas = cv2.countNonZero(thresh1)
        porcentaje_manchas = 100 - (num_pixels_con_manchas_blancas / thresh1.size) * 100
        contornos_blancos, _ = cv2.findContours(thresh1, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cantidad_manchas_blancas = len(contornos_blancos)
        Cadena = f"Cantidad de manchas blancas: {cantidad_manchas_blancas}\nPorcentaje de área con manchas: {round(100 - porcentaje_manchas, 2)}%"
        CajaTexto1.configure(state='normal')
        CajaTexto1.delete(1.0, tk.END)
        CajaTexto1.insert(1.0, Cadena)
        CajaTexto1.configure(state='disabled')
    elif boton == 2:
        num_pixels_con_manchas_blancas = cv2.countNonZero(thresh2)
        porcentaje_manchas = 100 - (num_pixels_con_manchas_blancas / thresh2.size) * 100
        contornos_blancos, _ = cv2.findContours(thresh2, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cantidad_manchas_blancas = len(contornos_blancos)
        Cadena = f"Cantidad de manchas blancas: {cantidad_manchas_blancas}\nPorcentaje de área con manchas: {round(100 - porcentaje_manchas, 2)}%"
        CajaTexto3.configure(state='normal')
        CajaTexto3.delete(1.0, tk.END)
        CajaTexto3.insert(1.0, Cadena)
        CajaTexto3.configure(state='disabled')

def contar_manchas_negras(boton):

    if boton == 1:
        imagen_invertida = cv2.bitwise_not(thresh1)
        num_pixels_con_manchas_negras = cv2.countNonZero(imagen_invertida)
        porcentaje_manchas = 100 - (num_pixels_con_manchas_negras / thresh1.size) * 100
        contornos_negros, _ = cv2.findContours(imagen_invertida, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cantidad_manchas_negras = len(contornos_negros)
        Cadena = f"Cantidad de manchas negras: {cantidad_manchas_negras}\nPorcentaje de área con manchas: {round(100 - porcentaje_manchas, 2)}%"
        CajaTexto2.configure(state='normal')
        CajaTexto2.delete(1.0, tk.END)
        CajaTexto2.insert(1.0, Cadena)
        CajaTexto2.configure(state='disabled')
    if boton == 2:
        imagen_invertida = cv2.bitwise_not(thresh2)
        num_pixels_con_manchas_negras = cv2.countNonZero(imagen_invertida)
        porcentaje_manchas = 100 - (num_pixels_con_manchas_negras / thresh2.size) * 100
        contornos_negros, _ = cv2.findContours(imagen_invertida, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cantidad_manchas_negras = len(contornos_negros)
        Cadena = f"Cantidad de manchas negras: {cantidad_manchas_negras}\nPorcentaje de área con manchas: {round(100 - porcentaje_manchas, 2)}%"
        CajaTexto4.configure(state='normal')
        CajaTexto4.delete(1.0, tk.END)
        CajaTexto4.insert(1.0, Cadena)
        CajaTexto4.configure(state='disabled')
```

Función umbralizar_rgb()

La **función umbralizar_rgb()** es esencial en el procesamiento de imágenes de esta aplicación. En primer lugar, divide la imagen capturada en sus tres canales de color (rojo, verde y azul). Luego, aplica umbrales de forma independiente a cada canal, donde los valores de umbral son configurados por el usuario. Posteriormente, se combinan los canales umbralizados para obtener una imagen en color fusionada. Además, se convierte esta imagen en una versión en escala de grises para su visualización en la interfaz de usuario. Por último, se actualiza la interfaz gráfica con la imagen fusionada, lo que permite a los usuarios visualizar cómo cambian los colores de la imagen en función de los umbrales configurados.

```python
def umbralizar_rgb():
    # Dividimos la matriz en los 3 canales
    global imagen_umbralizada, imagen_umbralizadaG
    canal_r, canal_g, canal_b = cv2.split(ImgRec)
    umbral_r_min = int(SRedI.get())
    umbral_g_min = int(SGreenI.get())
    umbral_b_min = int(SBlueI.get())

    umbral_r_max = int(SRedD.get())
    umbral_g_max = int(SGreenD.get())
    umbral_b_max = int(SBlueD.get())

    umbralizado_r = cv2.threshold(canal_r, umbral_r_min, umbral_r_max, cv2.THRESH_BINARY)[1]
    umbralizado_g = cv2.threshold(canal_g, umbral_g_min, umbral_g_max, cv2.THRESH_BINARY)[1]
    umbralizado_b = cv2.threshold(canal_b, umbral_b_min, umbral_b_max, cv2.THRESH_BINARY)[1]
    imagen_umbralizada = cv2.merge((umbralizado_r, umbralizado_g, umbralizado_b))
    imagen_umbralizadaG = cv2.cvtColor(imagen_umbralizada, cv2.COLOR_RGB2GRAY)
    im = Image.fromarray(imagen_umbralizada)
    img = ImageTk.PhotoImage(image=im)
    LImagenROI.configure(image=img)
    LImagenROI.image = img
```

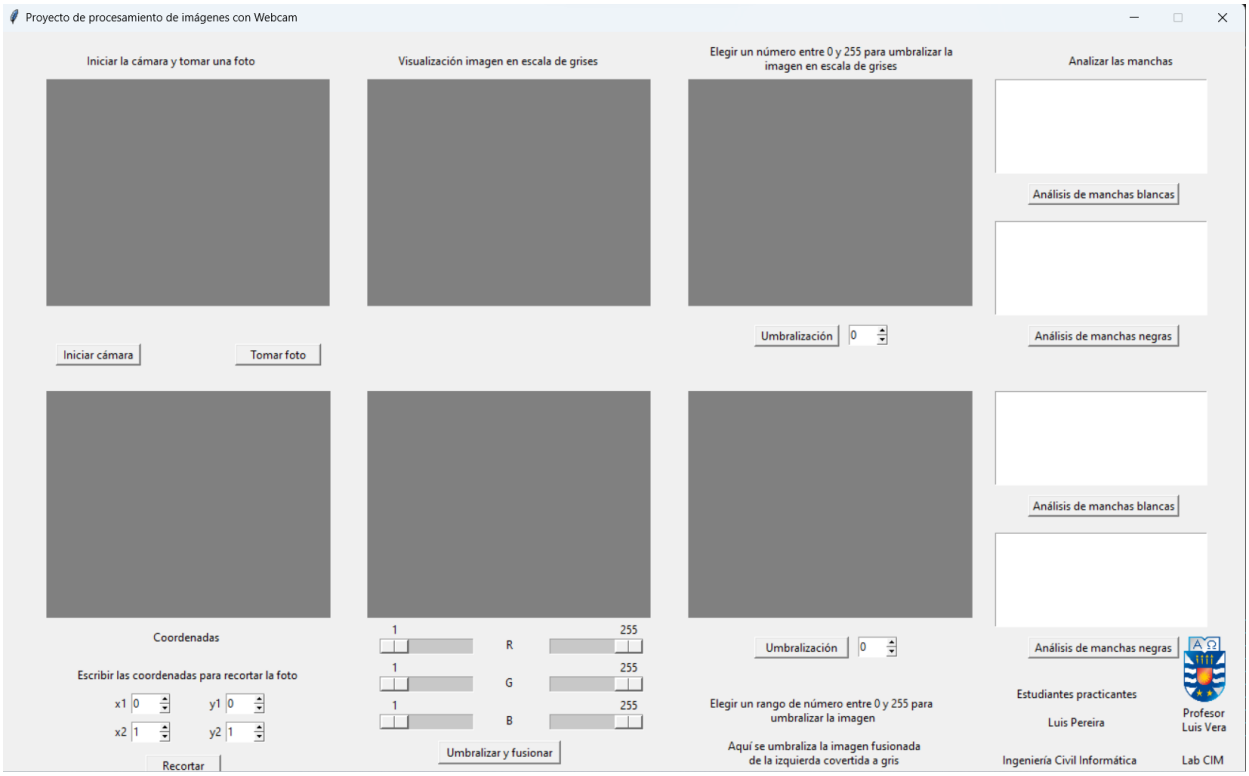
Funciones de Coordenadas y Recorte

Las funciones **mostrar_coordenadas()** y **recortar()** permiten al usuario definir las coordenadas para recortar una región específica de la imagen capturada.

```python
def mostrar_coordenadas(event):
    coordenadas['text'] = f'x = {event.x}    y = {event.y}'

def recortar():
    global ImgRec
    Vx1 = int(x1.get())
    Vy1 = int(y1.get())
    Vx2 = int(x2.get())
    Vy2 = int(y2.get())
    ImgRec = Captura[Vx1:Vx2, Vy1:Vy2]
    Im = Image.fromarray(ImgRec)
    ImRec = ImageTk.PhotoImage(image=Im)
    LImagenRecorte.configure(image=ImRec)
    LImagenRecorte.image = ImRec
```

Resultado Final



Y en funcionamiento…