UNIVERSIDAD DEL BÍO-BÍO

19 Procesamiento de Rectángulos con OpenCV y Tkinter

Estudiantes Practicantes:
Luis Pereira

Profesor:
Luis Vera

Laboratorio CIMUBB

2023-2

1. Importar Librerías:

Importamos las librerías necesarias para nuestro proyecto. cv2 es para OpenCV, tkinter para la interfaz gráfica, y PIL para manipular imágenes. imutils proporciona funciones útiles para trabajar con imágenes.

```
import cv2
import tkinter as tk
from tkinter import *
from tkinter import ttk
from PIL import Image
from PIL import ImageTk
import imutils
```

2. Variables Globales:

Declaramos variables globales para la captura de la cámara (capture) y la imagen recortada (ImagenRecortada). Esto nos permite acceder y modificar estas variables desde diferentes funciones.

```
# Variables globales
global capture, ImagenRecortada
```

3. Iniciar la Cámara:

La función camara() inicializa la cámara web utilizando OpenCV. Llama a la función iniciar() para comenzar el procesamiento de la imagen.

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

4. Procesamiento de Imágenes:

La función **iniciar**() realiza varias tareas:

1) Captura un frame de la cámara.

```
ret, frame = capture.read()
```

2) Convierte la imagen a escala de grises y la binariza.

```
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
# Volver la imagen gris a una binaria
_, th = cv2.threshold(gray, 140, 240, cv2.THRESH_BINARY)
```

3) Encuentra los contornos en la imagen binarizada.

```
contornos, _ = cv2.findContours(th, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

4) Identifica y cuenta los rectángulos en la imagen.

```
for c in contornos:
    area = cv2.contourArea(c)
    if area > 1700:
        peri = cv2.arcLength(c, True)
        approx = cv2.approxPolyDP(c, 0.02 * peri, True)
        if len(approx) == 4:
            # Dibujar un contorno de un rectángulo alrededor del objeto
            cv2.drawContours(frame, [approx], -1, (255, 0, 0), 2, cv2.LINE_AA)
            total += 1

            # Recortar el rectángulo
            x, y, w, h = cv2.boundingRect(approx)
            ImagenRecortada = frame[y:y + h, x:x + w]
```

5) Muestra la imagen original con los rectángulos dibujados.

```
cv2.drawContours(frame, [approx], -1, (255, 0, 0), 2, cv2.LINE_AA)
```

6) Muestra la imagen recortada en otro label.

```
# Mostrar la imagen recortada en el segundo label
ImagenRecortada = cv2.cvtColor(ImagenRecortada, cv2.COLOR_BGR2RGB)
img = Image.fromarray(cv2.cvtColor(ImagenRecortada, cv2.COLOR_BGR2GRAY))
img = ImageTk.PhotoImage(image=img)
GImagenROI.configure(image=img)
GImagenROI.image = img
```

5. Mostrar Imágenes en Tkinter:

Utilizamos la biblioteca PIL para convertir las imágenes de formato OpenCV a formato compatible con Tkinter y mostrarlas en labels de la interfaz.

```
        # Mostrar la imagen con los contornos en el primer label
        frame = imutils.resize(frame, width=311)
        ImagenCamara = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        img = Image.fromarray(ImagenCamara)
        img = ImageTk.PhotoImage(image=img)
        LImagen.configure(image=img)
        LImagen.image = img

        LImagen.after(1, iniciar)
    else:
        LImagen.image = None
        capture.release()
```

6. Capturar Rectángulo y Umbralización:

capturar_rectangulo() captura el rectángulo identificado en la imagen procesada y lo muestra en un tercer label.

```
def capturar_rectangulo():
    # Mostrar la imagen recortada en el tercer label (Umbralización)
    global rectanguloRecortado
    rectanguloRecortado = ImagenRecortada
    img = Image.fromarray(ImagenRecortada)
    img = ImageTk.PhotoImage(image=img)
    UImagen.configure(image=img)
    UImagen.image = img
```

umbralizar_escala_grises() realiza la umbralización de la imagen recortada en escala de grises.

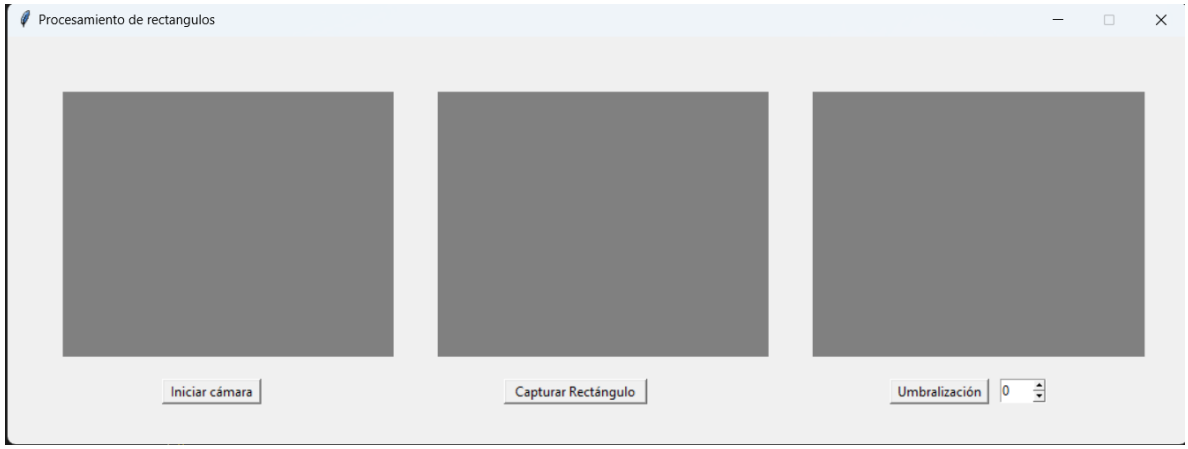
```
def umbralizar_escala_grises():
    global ImagenRecortada
    valor = int(numeroUmbra.get())
    ImagenRecortadaG = cv2.cvtColor(rectanguloRecortado , cv2.COLOR_BGR2GRAY)
    ret, thresh1 = cv2.threshold(ImagenRecortadaG, valor, 255, cv2.THRESH_BINARY)
    Umbral = Image.fromarray(thresh1)
    Umbral = ImageTk.PhotoImage(image=Umbral)
    UImagen.configure(image=Umbral)
    UImagen.image = Umbral
```

7. Interfaz de Usuario (UI):

Creamos una interfaz de usuario con botones para iniciar la cámara, capturar rectángulo y realizar umbralización. Un SpinBox permite ajustar el valor de umbralización.

```
ventana = tk.Tk()
ventana.geometry("1070x370")
ventana.resizable(0, 0)
ventana.title("Procesamiento de rectangulos")

# Botones
BCamara = tk.Button(ventana, text="Iniciar cámara", command=camara)
BCamara.place(x=140, y=310, width=90, height=23)

BCapturarRectangulo = tk.Button(ventana, text="Capturar Rectángulo", command=capturar_rectangulo)
BCapturarRectangulo.place(x=450, y=310, width=130, height=23)
BBinary = tk.Button(ventana, text="Umbralización", command=umbralizar_escala_grises)
BBinary.place(x=800, y=310, width=90, height=23)

# SpinBox
numeroUmbra = tk.Spinbox(ventana, from_=0, to=255)
numeroUmbra.place(x=900, y=310, width=42, height=22)

# Cuadros de Imagen grises
LImagen = tk.Label(ventana, background="gray")
LImagen.place(x=50, y=50, width=300, height=240)
GImagenROI = tk.Label(ventana, background="gray")
GImagenROI.place(x=390, y=50, width=300, height=240)
UImagen = tk.Label(ventana, background="gray")
UImagen.place(x=730, y=50, width=301, height=240)
# Texto con contador de rectangulos
texto_rectangulo = tk.Label(ventana, text="", font=("Arial", 12), fg="red")
texto_rectangulo.place(x=50, y=10)
```

Resultado Final



En funcionamiento: