Practica 3: Segmentación por colores en programa grafico

Alumno: Leonardo Ramos Espinoza

Materia: Inteligencia Artificial

Docente: Dr. Ángel Mario Lerma Sánchez

```
import cv2
import numpy as np
import tkinter as tk
from tkinter import filedialog, Scale, Button, Label, Frame, HORIZONTAL
from PIL import Image, ImageTk

class SegmentacionColores:
    def __init__(self, root):
        self.root = root
        self.root.title("Segmentación por Colores")
        self.root.geometry("1200x700")

        # Variables para almacenar imágenes
        self.imagen_original = None
        self.imagen_original_cv = None
        self.imagen_hsv = None

        # Variables para los rangos de color
        self.color_h = 0
        self.color_s = 255
        self.color_v = 255
        self.tolerancia = 15

        # Crear frames principales
        self.frame_controles = Frame(root, padx=10, pady=10)
```

```
    self.frame_controles.pack(fill=tk.X)

    self.frame_imagenes = Frame(root, padx=10, pady=10)
    self.frame_imagenes.pack(fill=tk.BOTH, expand=True)

    # Configurar controles
    self.configurar_controles()

    # Configurar área de visualización
    self.configurar_visualizacion()

def configurar_controles(self):
    # Botón para cargar imagen
    self.btn_cargar = Button(self.frame_controles, text="Cargar Imagen",
                             command=self.cargar_imagen, padx=10, pady=5)
    self.btn_cargar.grid(row=0, column=0, padx=10)

    # Selector de colores (cuadrícula de colores)
    self.label_color = Label(self.frame_controles, text="Seleccionar Color:")
    self.label_color.grid(row=0, column=1, padx=10)

    # Frame para la cuadrícula de colores
    self.frame_colores = Frame(self.frame_controles)
    self.frame_colores.grid(row=0, column=2, rowspan=2, padx=10)

    # Definir colores comunes en HSV (H en formato OpenCV de 0-179)
```

```
self.colores = [
    {"nombre": "Rojo", "hsv": (0, 255, 255), "rgb": "#FF0000"},
    {"nombre": "Naranja", "hsv": (15, 255, 255), "rgb": "#FF8000"},
    {"nombre": "Amarillo", "hsv": (30, 255, 255), "rgb": "#FFFF00"},
    {"nombre": "Verde claro", "hsv": (60, 255, 255), "rgb": "#80FF00"},
    {"nombre": "Verde", "hsv": (90, 255, 255), "rgb": "#00FF00"},
    {"nombre": "Verde azulado", "hsv": (120, 255, 255), "rgb": "#00FF80"},
    {"nombre": "Cian", "hsv": (90, 255, 255), "rgb": "#00FFFF"},
    {"nombre": "Azul claro", "hsv": (105, 255, 255), "rgb": "#0080FF"},
    {"nombre": "Azul", "hsv": (120, 255, 255), "rgb": "#0000FF"},
    {"nombre": "Púrpura", "hsv": (135, 255, 255), "rgb": "#8000FF"},
    {"nombre": "Magenta", "hsv": (150, 255, 255), "rgb": "#FF00FF"},
    {"nombre": "Rosa", "hsv": (165, 255, 255), "rgb": "#FF0080"}
]

# Crear cuadros de colores
self.botones_color = []
for i, color in enumerate(self.colores):
    fila = i // 4
    columna = i % 4
    btn = Button(self.frame_colores, bg=color["rgb"], width=5, height=2,
                 command=lambda c=color: self.seleccionar_color_predefinido(c))
    btn.grid(row=fila, column=columna, padx=2, pady=2)
    self.botones_color.append(btn)

# Mostrar el color seleccionado
```

```
        self.muestra_color = Label(self.frame_controles, text="", width=10, height=2, bg="#FFFFFF")
        self.muestra_color.grid(row=0, column=3, padx=10)

        # Etiqueta para el color seleccionado
        self.label_color_seleccionado = Label(self.frame_controles, text="Ningún color seleccionado")
        self.label_color_seleccionado.grid(row=0, column=4, padx=10)

        # Control de tolerancia
        self.label_tolerancia = Label(self.frame_controles, text="Tolerancia:")
        self.label_tolerancia.grid(row=1, column=0, padx=10, pady=10)

        self.escala_tolerancia = Scale(self.frame_controles, from_=5, to=50,
                                       orient=HORIZONTAL, length=200,
                                       command=self.actualizar_tolerancia)
        self.escala_tolerancia.set(15)
        self.escala_tolerancia.grid(row=1, column=1, padx=10)

        # Botón para segmentar
        self.btn_segmentar = Button(self.frame_controles, text="Segmentar",
                                    command=self.segmentar, padx=10, pady=5)
        self.btn_segmentar.grid(row=1, column=3, padx=10)

    def configurar_visualizacion(self):
        # Frame para las imágenes
```

```
    self.frame_original = Frame(self.frame_imagenes, width=350, height=500, bd=2, relief=tk.SUNKEN)
    self.frame_original.grid(row=0, column=0, padx=10, pady=10)
    self.frame_original.pack_propagate(False)

    self.frame_mascara = Frame(self.frame_imagenes, width=350, height=500, bd=2, relief=tk.SUNKEN)
    self.frame_mascara.grid(row=0, column=1, padx=10, pady=10)
    self.frame_mascara.pack_propagate(False)

    self.frame_resultado = Frame(self.frame_imagenes, width=350, height=500, bd=2, relief=tk.SUNKEN)
    self.frame_resultado.grid(row=0, column=2, padx=10, pady=10)
    self.frame_resultado.pack_propagate(False)

    # Labels para títulos
    Label(self.frame_original, text="Imagen Original").pack(pady=5)
    Label(self.frame_mascara, text="Máscara").pack(pady=5)
    Label(self.frame_resultado, text="Resultado").pack(pady=5)

    # Labels para imágenes
    self.label_img_original = Label(self.frame_original)
    self.label_img_original.pack(fill=tk.BOTH, expand=True)

    self.label_img_mascara = Label(self.frame_mascara)
    self.label_img_mascara.pack(fill=tk.BOTH, expand=True)
```

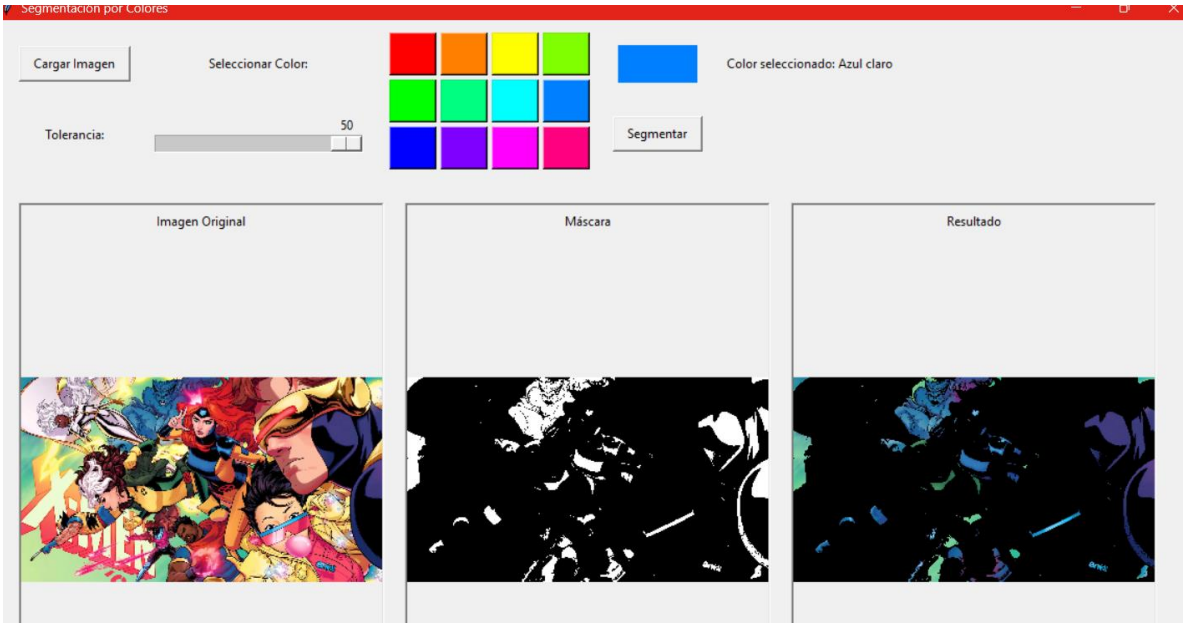
```
        self.label_img_resultado = Label(self.frame_resultado)
        self.label_img_resultado.pack(fill=tk.BOTH, expand=True)

    def seleccionar_color_predefinido(self, color):
        """Selecciona un color predefinido de la cuadrícula"""
        self.color_h, self.color_s, self.color_v = color["hsv"]
        self.muestra_color.config(bg=color["rgb"])
        self.label_color_seleccionado.config(text=f"Color seleccionado:
{color['nombre']}")

        # No segmentamos automáticamente, esperamos al botón

    def actualizar_tolerancia(self, val):
        """Actualiza el valor de tolerancia"""
        self.tolerancia = int(val)
        # No segmentamos automáticamente, esperamos al botón

    def cargar_imagen(self):
        """Carga una imagen desde el sistema de archivos"""
        ruta_archivo = filedialog.askopenfilename(
            title="Seleccionar Imagen",
            filetypes=(("Archivos de imagen", "*.jpg;*.jpeg;*.png;*.bmp"), ("Todos los
archivos", "*.*"))
        )

        if ruta_archivo:
            # Cargar imagen con OpenCV
```

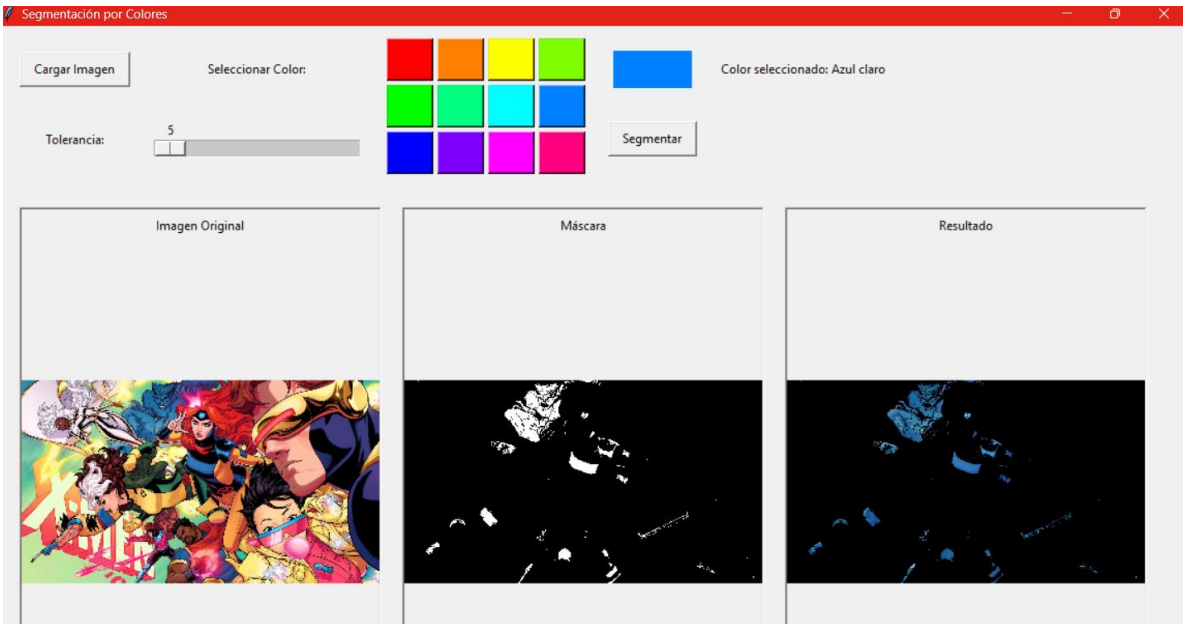
```
        self.imagen_original_cv = cv2.imread(ruta_archivo)
        if self.imagen_original_cv is None:
            print("Error al cargar la imagen")
            return

        # Redimensionar para mostrar si es muy grande
        self.imagen_original_cv = self.redimensionar_imagen(self.imagen_original_cv,
350)

        # Convertir a formato HSV para segmentación
        self.imagen_hsv = cv2.cvtColor(self.imagen_original_cv, cv2.COLOR_BGR2HSV)

        # Mostrar imagen original
        self.mostrar_imagen(self.imagen_original_cv, self.label_img_original)

        # Limpiar otras imágenes
        self.label_img_mascara.config(image="")
        self.label_img_resultado.config(image="")

    def redimensionar_imagen(self, imagen, ancho_maximo):
        """Redimensiona la imagen manteniendo la proporción"""
        altura, ancho = imagen.shape[:2]
        if ancho > ancho_maximo:
            proporcion = ancho_maximo / ancho
            nuevo_ancho = ancho_maximo
            nueva_altura = int(altura * proporcion)
```

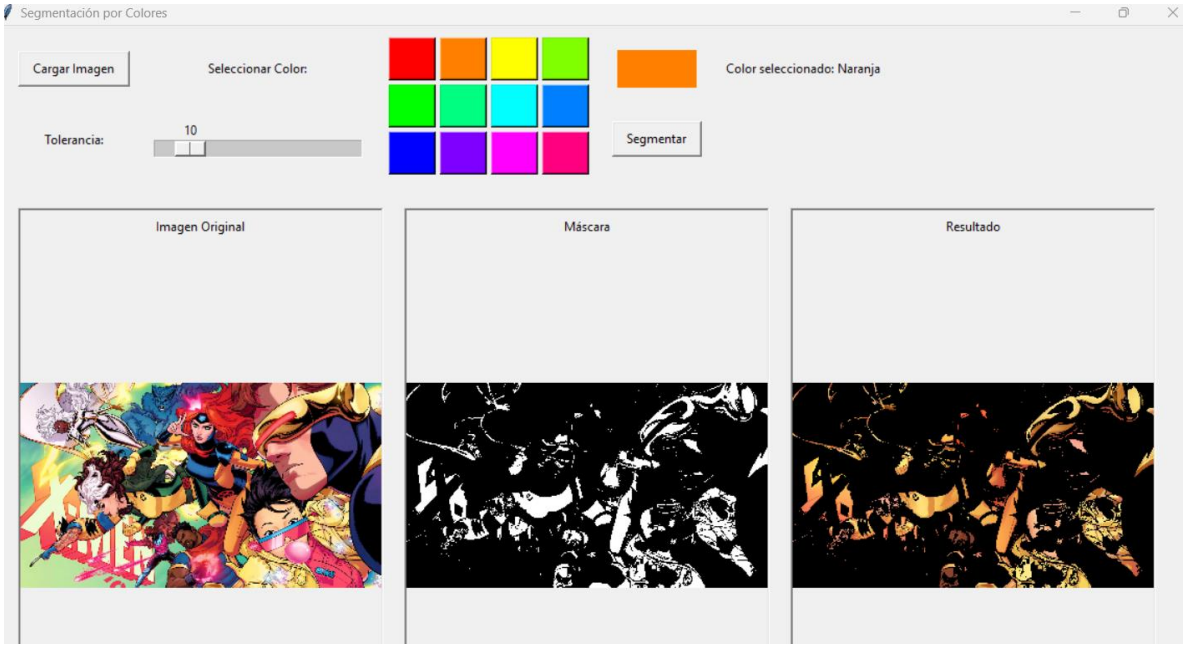
```
        return cv2.resize(imagen, (nuevo_ancho, nueva_altura))
    return imagen

def mostrar_imagen(self, imagen_cv, label):
    """Muestra una imagen OpenCV en un Label Tkinter"""
    # Convertir de BGR a RGB para Tkinter
    imagen_rgb = cv2.cvtColor(imagen_cv, cv2.COLOR_BGR2RGB)
    imagen_pil = Image.fromarray(imagen_rgb)
    imagen_tk = ImageTk.PhotoImage(image=imagen_pil)

    # Guardar referencia para evitar que el recolector de basura la elimine
    label.image = imagen_tk
    label.config(image=imagen_tk)

def segmentar(self):
    """Realiza la segmentación por color"""
    if self.imagen_hsv is None:
        return

    # Mostrar mensaje de depuración
    print(f"Segmentando con H={self.color_h}, tolerancia={self.tolerancia}")

    # Calcular rango de color con la tolerancia
    bajo_h = max(0, self.color_h - self.tolerancia)
    alto_h = min(179, self.color_h + self.tolerancia)
```

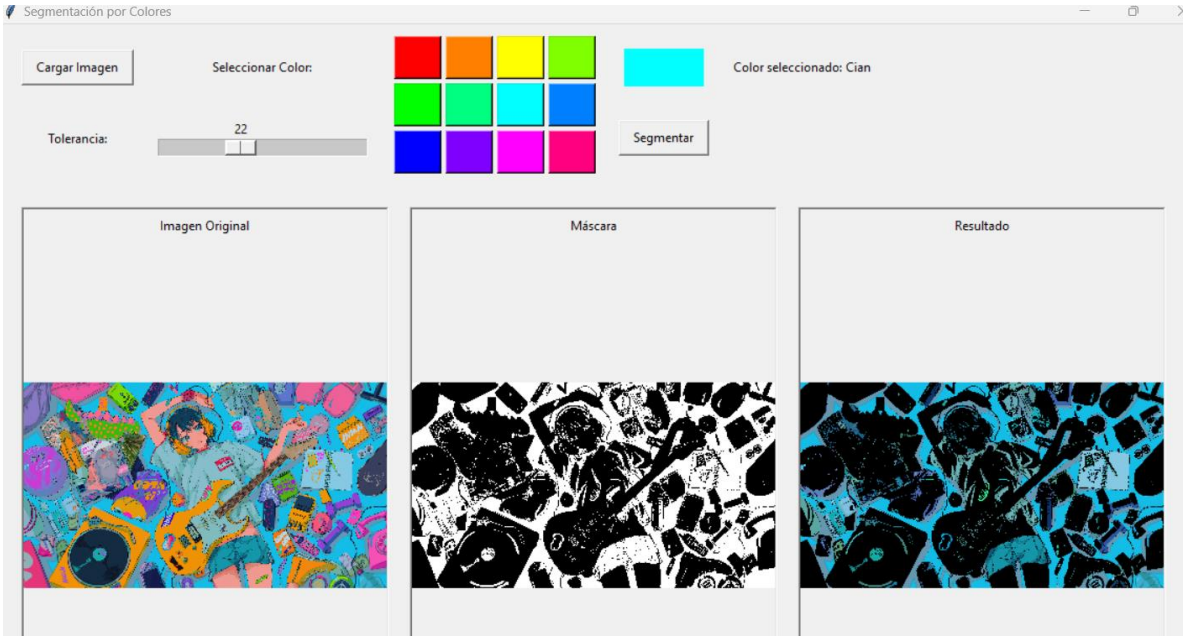
```
    # Si el rango cruza el límite de H (180/0)
    if bajo_h < 0:
        bajo_h += 180
    if alto_h > 179:
        alto_h -= 180

    # Para el caso especial del rojo (cerca del límite 0/180)
    if self.color_h < self.tolerancia or self.color_h > (179 - self.tolerancia):
        # Caso especial: el rango cruza el límite de 180/0
        mascara1 = cv2.inRange(self.imagen_hsv, np.array([0, 100, 100]),
np.array([alto_h, 255, 255]))
        mascara2 = cv2.inRange(self.imagen_hsv, np.array([bajo_h, 100, 100]),
np.array([179, 255, 255]))
        mascara = cv2.bitwise_or(mascara1, mascara2)
    else:
        # Caso normal
        mascara = cv2.inRange(self.imagen_hsv, np.array([bajo_h, 100, 100]),
np.array([alto_h, 255, 255]))

    # Aplicar la máscara a la imagen original
    resultado = cv2.bitwise_and(self.imagen_original_cv, self.imagen_original_cv,
mask=mascara)

    # Mostrar imagen de la máscara (convertir a BGR para que se vea bien)
    mascara_bgr = cv2.cvtColor(mascara, cv2.COLOR_GRAY2BGR)
    self.mostrar_imagen(mascara_bgr, self.label_img_mascara)
```

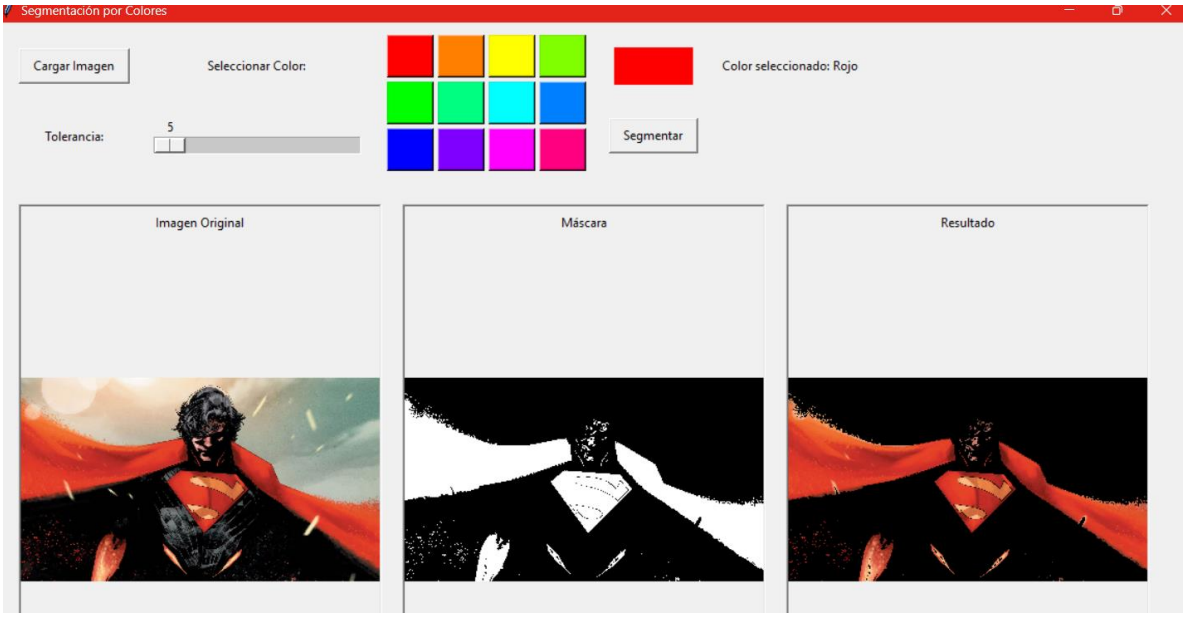
```
        # Mostrar resultado
        self.mostrar_imagen(resultado, self.label_img_resultado)

        print("Segmentación completada.")

if __name__ == "__main__":
    root = tk.Tk()
    app = SegmentacionColores(root)
    root.mainloop()
```


Segmentación por Colores
Cargar Imagen
Seleccionar Color:
Color seleccionado: Azul claro
Tolerancia:
50
Segmentar
Imagen Original
Máscara
Resultado


Segmentación por Colores
Cargar Imagen
Seleccionar Color:
Color seleccionado: Azul claro
Tolerancia:
5
Segmentar
Imagen Original
Máscara
Resultado


Segmentación por Colores
Cargar Imagen
Seleccionar Color:
Color seleccionado: Naranja
Tolerancia:
10
Segmentar
Imagen Original
Máscara
Resultado


Segmentación por Colores
Cargar Imagen
Seleccionar Color:
Color seleccionado: Cian
Tolerancia:
22
Segmentar
Imagen Original
Máscara
Resultado


Segmentación por Colores
Cargar Imagen
Seleccionar Color:
Color seleccionado: Rojo
Tolerancia:
5
Segmentar
Imagen Original
Máscara
Resultado