Practica 2: Aplicación de filtros en programa grafico

Alumno: Leonardo Ramos Espinoza

Materia: Inteligencia Artificial

Docente: Dr. Ángel Mario Lerma Sánchez

```
import tkinter as tk
from tkinter import filedialog, ttk
import cv2
import numpy as np
from PIL import Image, ImageTk

class FiltrosApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Aplicación de Filtros")
        self.root.geometry("1200x700")

        # Variables
        self.imagen_original = None
        self.ruta_imagen = None

        # Crear el marco principal
        self.frame_principal = ttk.Frame(root)
        self.frame_principal.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        # Crear el marco para los controles
        self.frame_controles = ttk.LabelFrame(self.frame_principal, text="Controles")
        self.frame_controles.pack(side=tk.LEFT, fill=tk.Y, padx=10, pady=10)

        # Botón para cargar imagen
```

```
        self.btn_cargar = ttk.Button(self.frame_controles, text="Cargar Imagen",
command=self.cargar_imagen)
        self.btn_cargar.pack(fill=tk.X, padx=10, pady=10)

        # Lista de filtros
        self.frame_filtros = ttk.LabelFrame(self.frame_controles, text="Filtros
Disponibles")
        self.frame_filtros.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        self.filtros = [
            "Original",
            "Suavizado (Blur)",
            "Convolución 2D",
            "Filtro Promedio",
            "Filtro Gaussiano",
            "Filtro Mediana",
            "Umbralización Simple",
            "Umbralización Adaptativa",
            "Binarización de Otsu",
            "Laplaciano",
            "Sobel X",
            "Sobel Y",
            "Canny"
        ]

        self.filtro_seleccionado = tk.StringVar()
        self.filtro_seleccionado.set(self.filtros[0])
```

```
        # Crear los radio buttons para los filtros
        for filtro in self.filtros:
            rb = ttk.Radiobutton(
                self.frame_filtros,
                text=filtro,
                value=filtro,
                variable=self.filtro_seleccionado,
                command=self.aplicar_filtro
            )
            rb.pack(anchor=tk.W, padx=10, pady=5)

        # Marco para mostrar las imágenes
        self.frame_imagenes = ttk.Frame(self.frame_principal)
        self.frame_imagenes.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True, padx=10, pady=10)

        # Panel para imagen original
        self.frame_original = ttk.LabelFrame(self.frame_imagenes, text="Imagen Original")
        self.frame_original.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=5, pady=5)

        self.panel_original = ttk.Label(self.frame_original)
        self.panel_original.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

        # Panel para imagen filtrada
```

```
    self.frame_filtrada = ttk.LabelFrame(self.frame_imagenes, text="Imagen con Filtro")
    self.frame_filtrada.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True, padx=5, pady=5)

    self.panel_filtrada = ttk.Label(self.frame_filtrada)
    self.panel_filtrada.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

    # Se inicializa con una imagen de ejemplo
    self.mostrar_mensaje("Cargue una imagen para empezar")

  def cargar_imagen(self):
    """Abrir diálogo para seleccionar y cargar una imagen"""
    self.ruta_imagen = filedialog.askopenfilename(
      title="Seleccionar Imagen",
      filetypes=(
        ("Archivos de Imagen", "*.jpg *.jpeg *.png *.bmp"),
        ("Todos los archivos", "*.*")
      )
    )

    if self.ruta_imagen:
      # Cargar la imagen con OpenCV
      self.imagen_original = cv2.imread(self.ruta_imagen)

      if self.imagen_original is None:
        self.mostrar_mensaje("Error al cargar la imagen")
```

```
            return

        # Mostrar imagen original
        self.mostrar_imagen_original()

        # Aplicar filtro seleccionado
        self.aplicar_filtro()

    def mostrar_imagen_original(self):
        """Mostrar la imagen original en el panel izquierdo"""
        if self.imagen_original is None:
            return

        # Convertir BGR a RGB para mostrar con PIL
        imagen_rgb = cv2.cvtColor(self.imagen_original, cv2.COLOR_BGR2RGB)

        # Redimensionar si es necesario
        alto, ancho = imagen_rgb.shape[:2]
        max_size = 500

        if alto > max_size or ancho > max_size:
            ratio = min(max_size / ancho, max_size / alto)
            nuevo_ancho = int(ancho * ratio)
            nuevo_alto = int(alto * ratio)
            imagen_rgb = cv2.resize(imagen_rgb, (nuevo_ancho, nuevo_alto))
```

```
    # Convertir a formato para Tkinter
    img = Image.fromarray(imagen_rgb)
    img_tk = ImageTk.PhotoImage(image=img)

    # Actualizar panel
    self.panel_original.configure(image=img_tk)
    self.panel_original.image = img_tk

def aplicar_filtro(self):
    """Aplicar el filtro seleccionado a la imagen original"""
    if self.imagen_original is None:
        return

    filtro = self.filtro_seleccionado.get()
    imagen_filtrada = None

    try:
        if filtro == "Original":
            imagen_filtrada = self.imagen_original.copy()

        elif filtro == "Suavizado (Blur)":
            imagen_filtrada = cv2.blur(self.imagen_original, (5, 5))

        elif filtro == "Convolución 2D":
            kernel = np.ones((5, 5), np.float32) / 25
            imagen_filtrada = cv2.filter2D(self.imagen_original, -1, kernel)
```

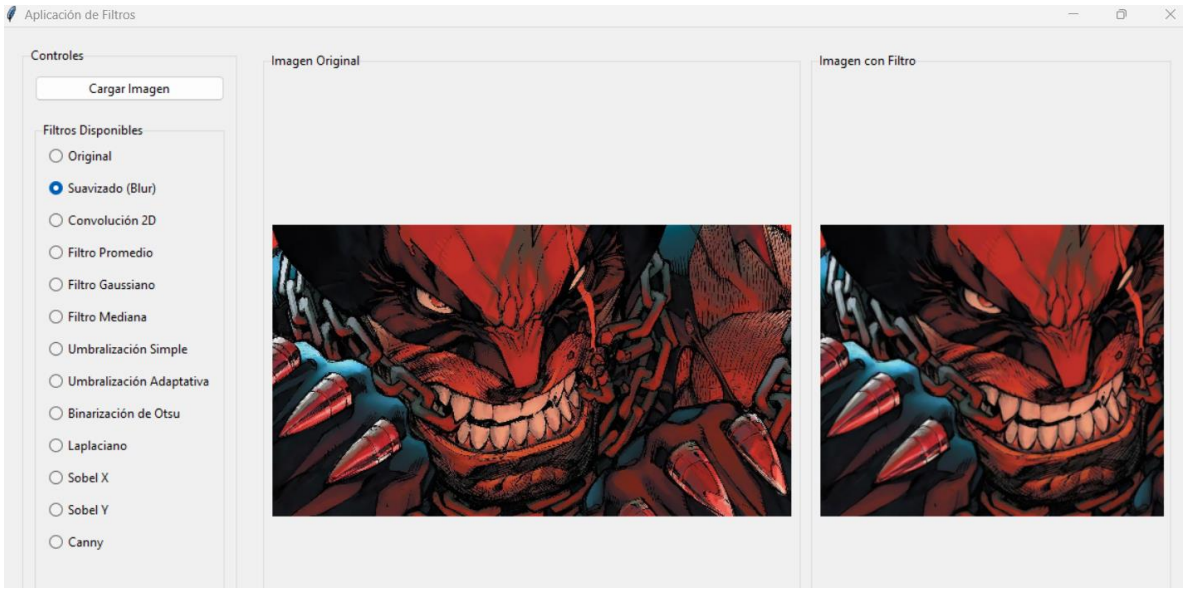
```
elif filtro == "Filtro Promedio":
    imagen_filtrada = cv2.boxFilter(self.imagen_original, -1, (5, 5))

elif filtro == "Filtro Gaussiano":
    imagen_filtrada = cv2.GaussianBlur(self.imagen_original, (5, 5), 0)

elif filtro == "Filtro Mediana":
    imagen_filtrada = cv2.medianBlur(self.imagen_original, 5)

elif filtro == "Umbralización Simple":
    gris = cv2.cvtColor(self.imagen_original, cv2.COLOR_BGR2GRAY)
    _, imagen_filtrada = cv2.threshold(gris, 127, 255, cv2.THRESH_BINARY)
    imagen_filtrada = cv2.cvtColor(imagen_filtrada, cv2.COLOR_GRAY2BGR)

elif filtro == "Umbralización Adaptativa":
    gris = cv2.cvtColor(self.imagen_original, cv2.COLOR_BGR2GRAY)
    imagen_filtrada = cv2.adaptiveThreshold(
        gris, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
        cv2.THRESH_BINARY, 11, 2
    )
    imagen_filtrada = cv2.cvtColor(imagen_filtrada, cv2.COLOR_GRAY2BGR)

elif filtro == "Binarización de Otsu":
    gris = cv2.cvtColor(self.imagen_original, cv2.COLOR_BGR2GRAY)
    _, imagen_filtrada = cv2.threshold(
```

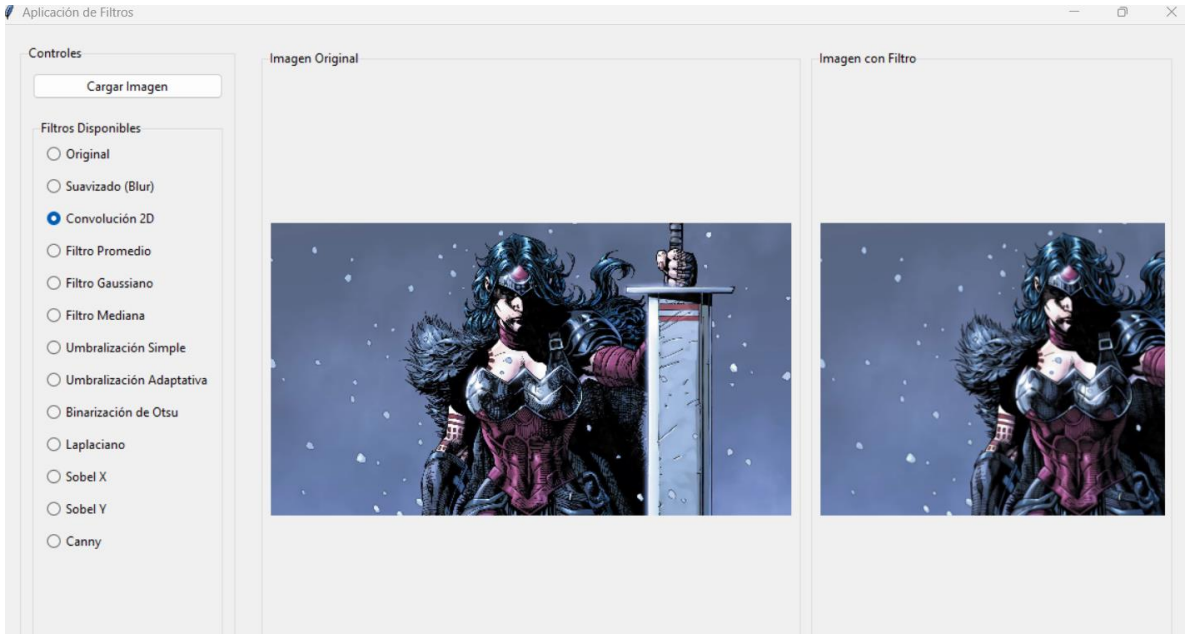
```
            gris, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU
        )
        imagen_filtrada = cv2.cvtColor(imagen_filtrada, cv2.COLOR_GRAY2BGR)

    elif filtro == "Laplaciano":
        gris = cv2.cvtColor(self.imagen_original, cv2.COLOR_BGR2GRAY)
        imagen_filtrada = cv2.Laplacian(gris, cv2.CV_64F)
        imagen_filtrada = cv2.convertScaleAbs(imagen_filtrada)
        imagen_filtrada = cv2.cvtColor(imagen_filtrada, cv2.COLOR_GRAY2BGR)

    elif filtro == "Sobel X":
        gris = cv2.cvtColor(self.imagen_original, cv2.COLOR_BGR2GRAY)
        imagen_filtrada = cv2.Sobel(gris, cv2.CV_64F, 1, 0, ksize=5)
        imagen_filtrada = cv2.convertScaleAbs(imagen_filtrada)
        imagen_filtrada = cv2.cvtColor(imagen_filtrada, cv2.COLOR_GRAY2BGR)

    elif filtro == "Sobel Y":
        gris = cv2.cvtColor(self.imagen_original, cv2.COLOR_BGR2GRAY)
        imagen_filtrada = cv2.Sobel(gris, cv2.CV_64F, 0, 1, ksize=5)
        imagen_filtrada = cv2.convertScaleAbs(imagen_filtrada)
        imagen_filtrada = cv2.cvtColor(imagen_filtrada, cv2.COLOR_GRAY2BGR)

    elif filtro == "Canny":
        imagen_filtrada = cv2.Canny(self.imagen_original, 100, 200)
        imagen_filtrada = cv2.cvtColor(imagen_filtrada, cv2.COLOR_GRAY2BGR)
```

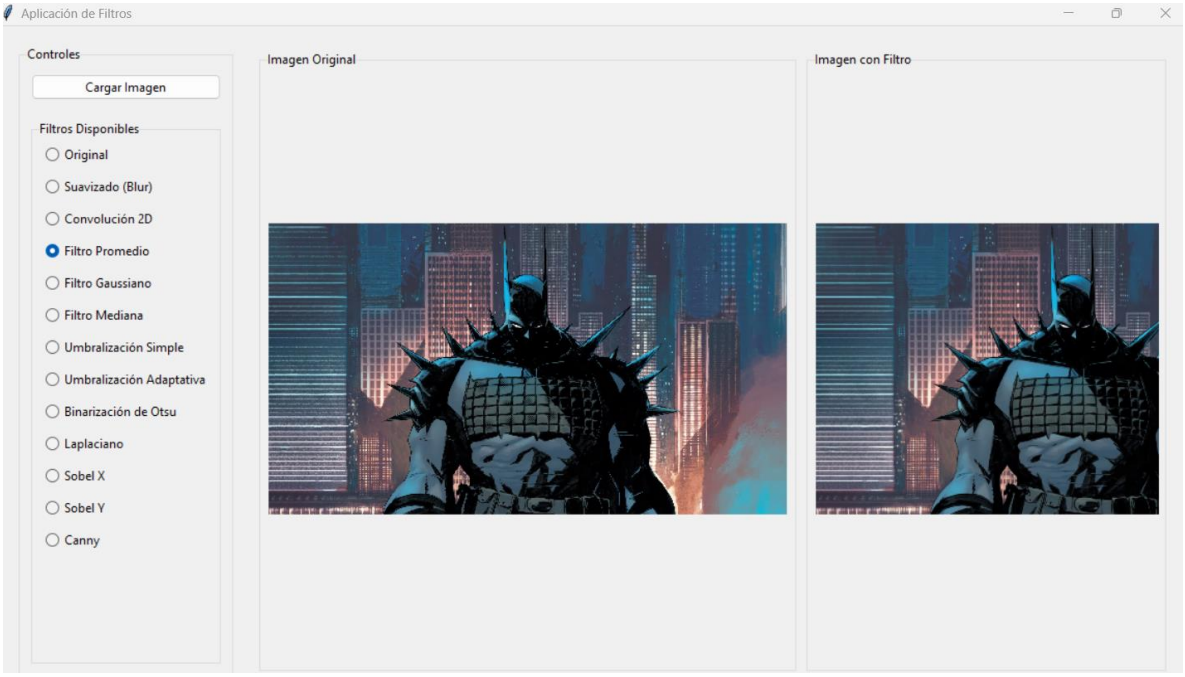
```
            # Mostrar la imagen filtrada
            self.mostrar_imagen_filtrada(imagen_filtrada)

        except Exception as e:
            self.mostrar_mensaje(f"Error al aplicar filtro: {e}")

    def mostrar_imagen_filtrada(self, imagen):
        """Mostrar la imagen filtrada en el panel derecho"""
        if imagen is None:
            return

        # Convertir BGR a RGB para mostrar con PIL
        imagen_rgb = cv2.cvtColor(imagen, cv2.COLOR_BGR2RGB)

        # Redimensionar si es necesario
        alto, ancho = imagen_rgb.shape[:2]
        max_size = 500

        if alto > max_size or ancho > max_size:
            ratio = min(max_size / ancho, max_size / alto)
            nuevo_ancho = int(ancho * ratio)
            nuevo_alto = int(alto * ratio)
            imagen_rgb = cv2.resize(imagen_rgb, (nuevo_ancho, nuevo_alto))

        # Convertir a formato para Tkinter
        img = Image.fromarray(imagen_rgb)
```

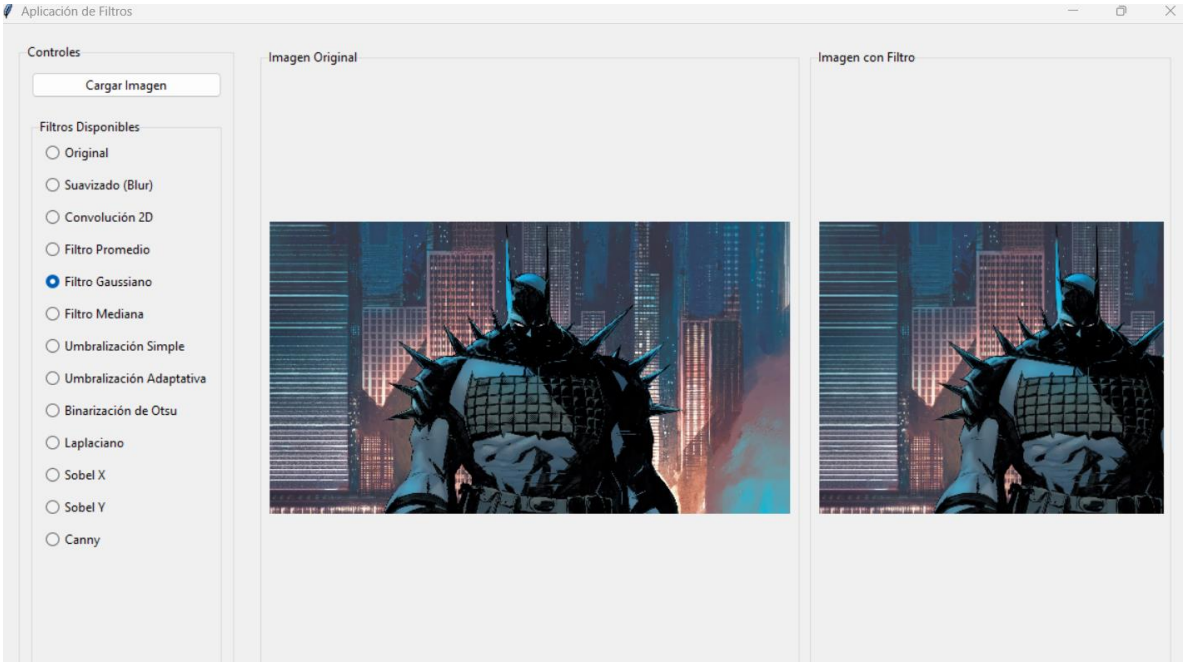
```
        img_tk = ImageTk.PhotoImage(image=img)

        # Actualizar panel
        self.panel_filtrada.configure(image=img_tk)
        self.panel_filtrada.image = img_tk

    def mostrar_mensaje(self, mensaje):
        """Mostrar un mensaje en los paneles"""
        # Crear una imagen con el mensaje
        img = np.ones((300, 400, 3), dtype=np.uint8) * 255
        cv2.putText(img, mensaje, (30, 150), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0),
2)

        # Mostrar la imagen en ambos paneles
        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        img_pil = Image.fromarray(img_rgb)
        img_tk = ImageTk.PhotoImage(image=img_pil)

        self.panel_original.configure(image=img_tk)
        self.panel_original.image = img_tk

        self.panel_filtrada.configure(image=img_tk)
        self.panel_filtrada.image = img_tk

if __name__ == "__main__":
```

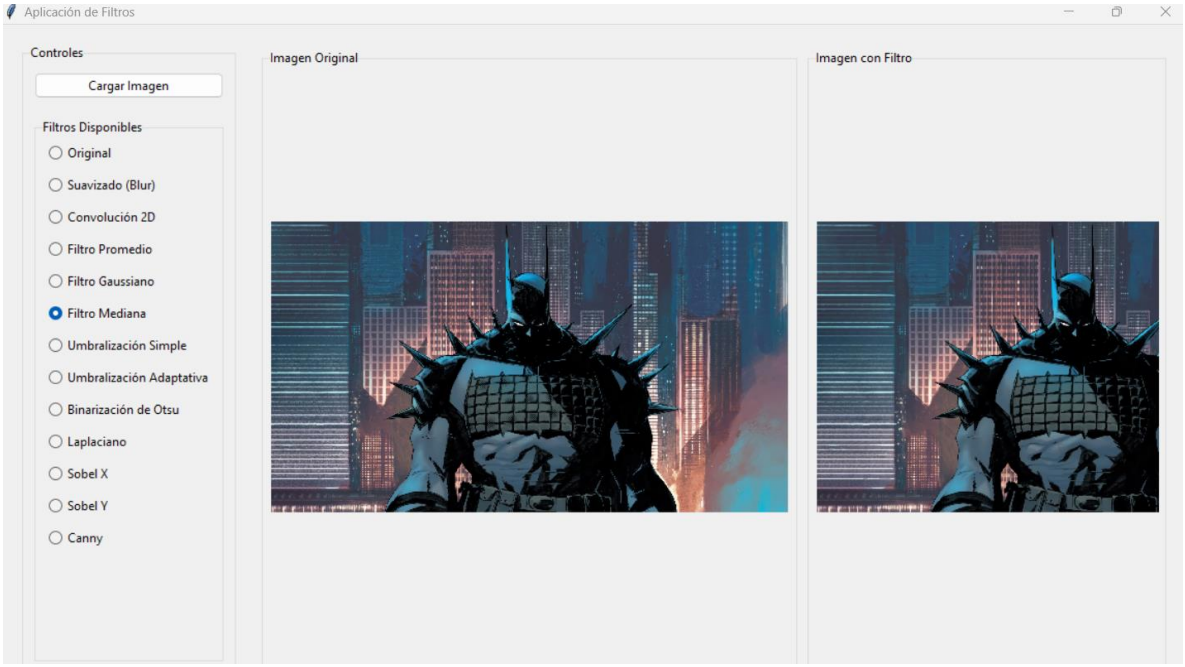
```
root = tk.Tk()

app = FiltrosApp(root)

root.mainloop()
```

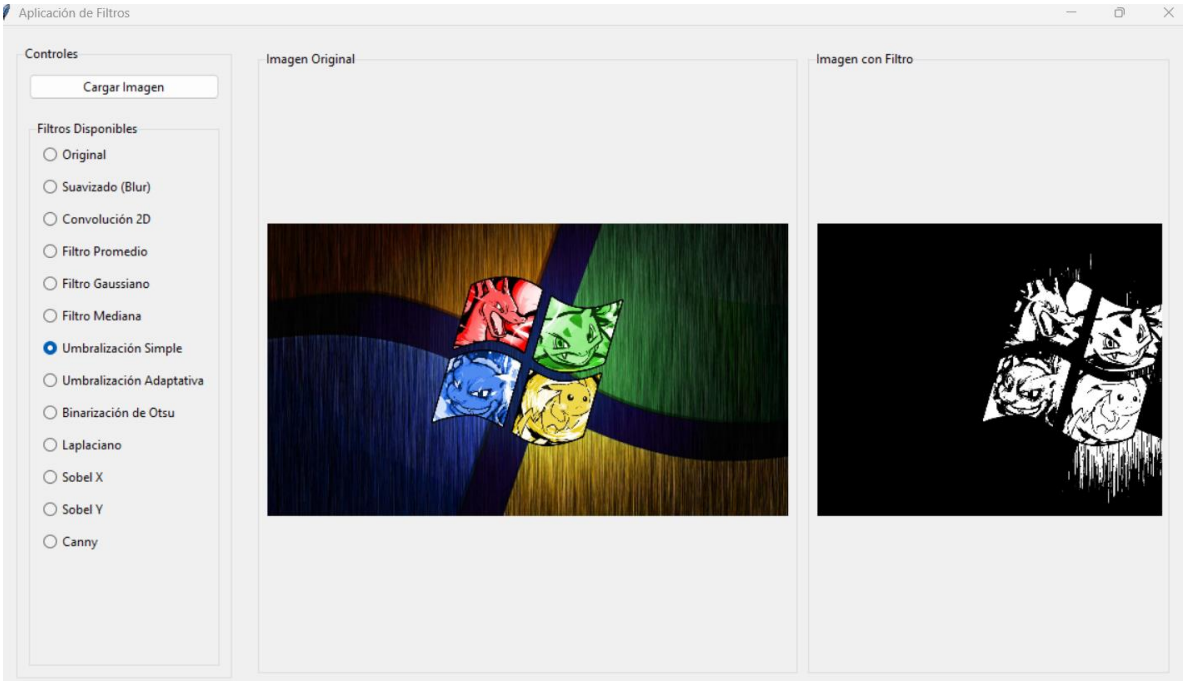

Aplicación de Filtros
Controles
Cargar Imagen
Filtros Disponibles
Original
Suavizado (Blur)
Convolución 2D
Filtro Promedio
Filtro Gaussiano
Filtro Mediana
Umbralización Simple
Umbralización Adaptativa
Binarización de Otsu
Laplaciano
Sobel X
Sobel Y
Canny
Imagen Original
Imagen con Filtro

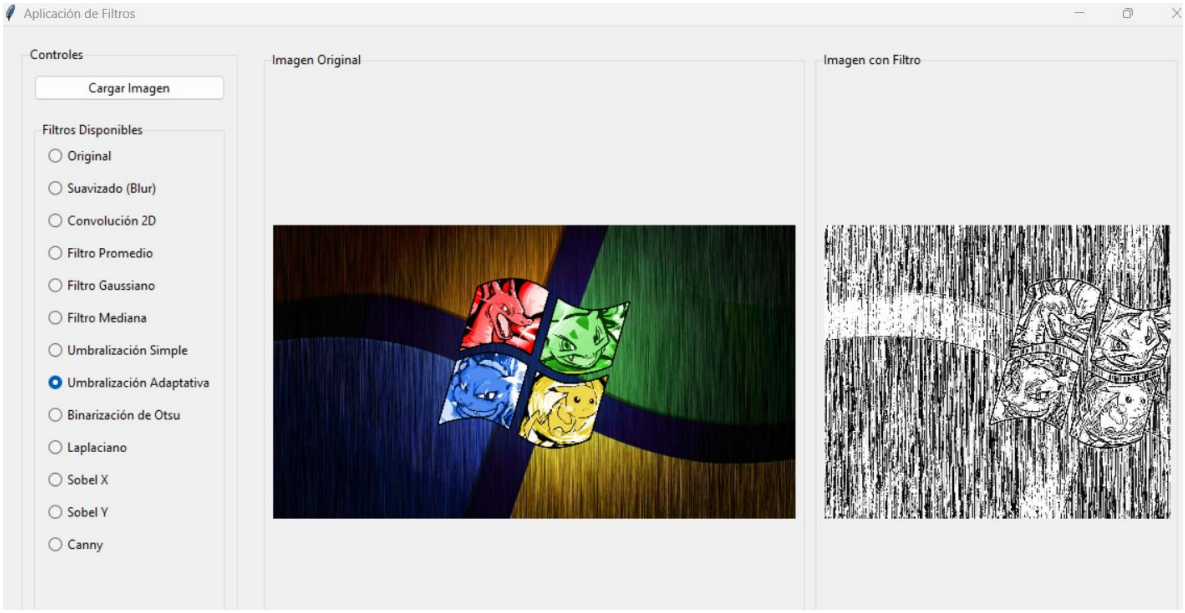

Aplicación de Filtros
Controles
Cargar Imagen
Filtros Disponibles
Original
Suavizado (Blur)
Convolución 2D
Filtro Promedio
Filtro Gaussiano
Filtro Mediana
Umbralización Simple
Umbralización Adaptativa
Binarización de Otsu
Laplaciano
Sobel X
Sobel Y
Canny
Imagen Original
Imagen con Filtro

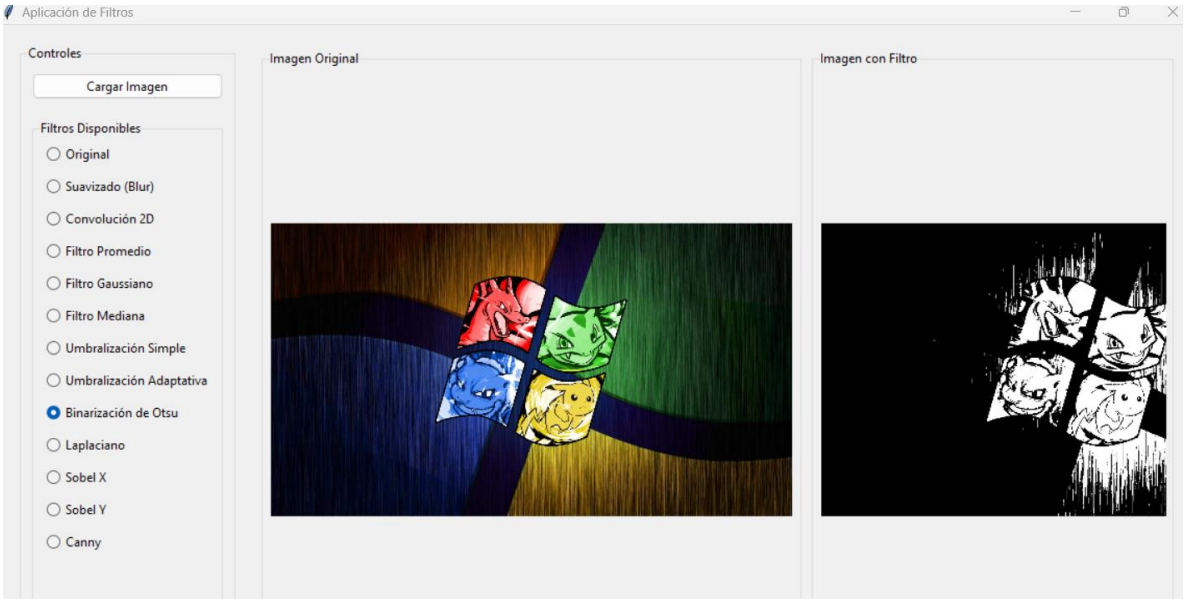

Aplicación de Filtros
Controles
Cargar Imagen
Filtros Disponibles
Original
Suavizado (Blur)
Convolución 2D
Filtro Promedio
Filtro Gaussiano
Filtro Mediana
Umbralización Simple
Umbralización Adaptativa
Binarización de Otsu
Laplaciano
Sobel X
Sobel Y
Canny
Imagen Original
Imagen con Filtro

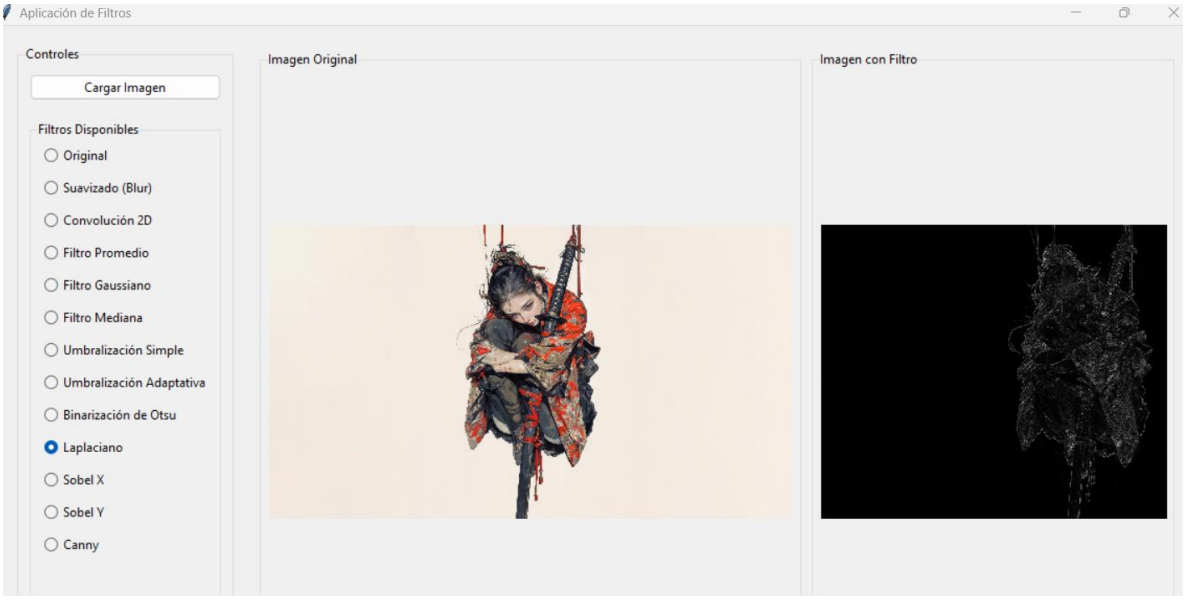

Aplicación de Filtros
Controles
Cargar Imagen
Filtros Disponibles
Original
Suavizado (Blur)
Convolución 2D
Filtro Promedio
Filtro Gaussiano
Filtro Mediana
Umbralización Simple
Umbralización Adaptativa
Binarización de Otsu
Laplaciano
Sobel X
Sobel Y
Canny
Imagen Original
Imagen con Filtro

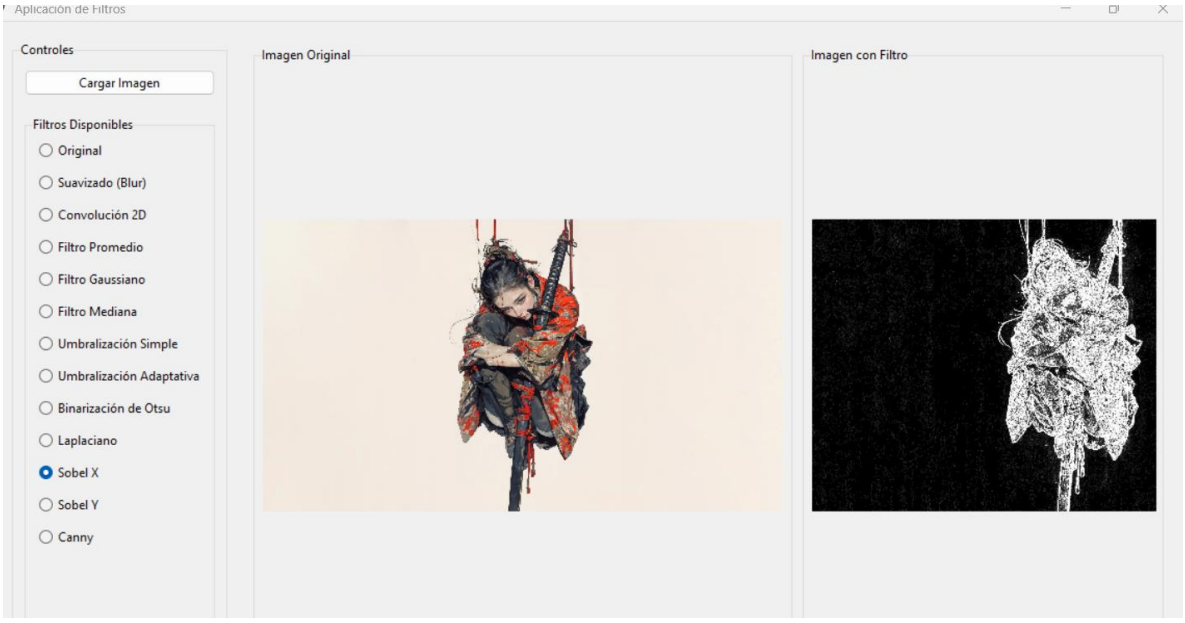

Aplicación de Filtros
Controles
Cargar Imagen
Filtros Disponibles
Original
Suavizado (Blur)
Convolución 2D
Filtro Promedio
Filtro Gaussiano
Filtro Mediana
Umbralización Simple
Umbralización Adaptativa
Binarización de Otsu
Laplaciano
Sobel X
Sobel Y
Canny
Imagen Original
Imagen con Filtro

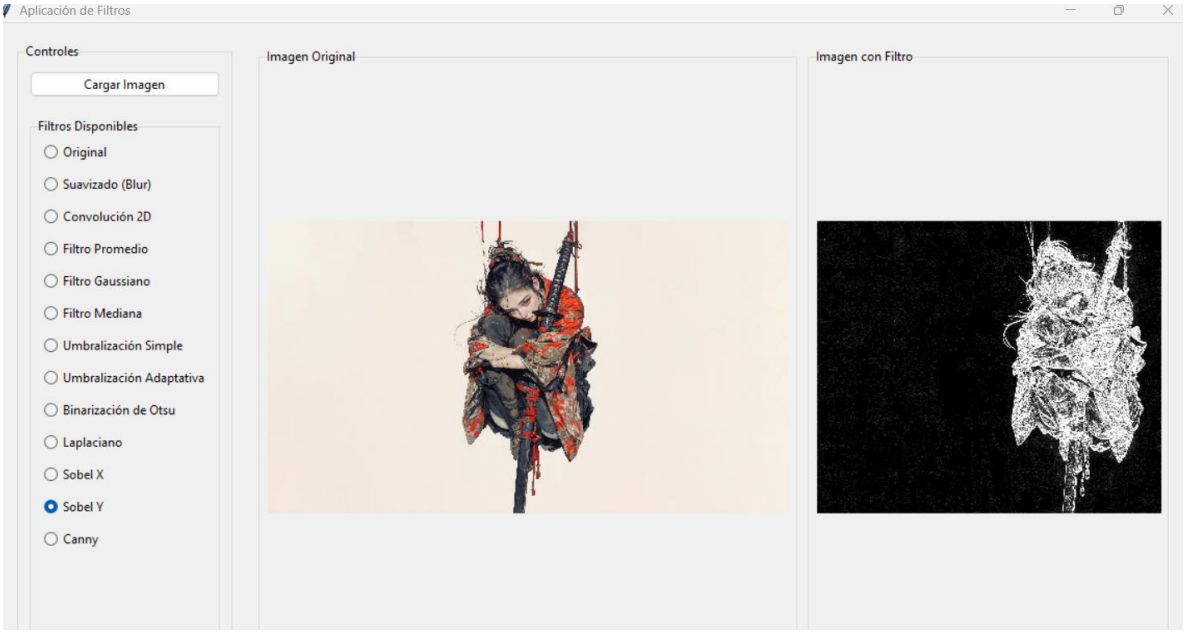

Aplicación de Filtros
Controles
Cargar Imagen
Filtros Disponibles
Original
Suavizado (Blur)
Convolución 2D
Filtro Promedio
Filtro Gaussiano
Filtro Mediana
Umbralización Simple
Umbralización Adaptativa
Binarización de Otsu
Laplaciano
Sobel X
Sobel Y
Canny
Imagen Original
Imagen con Filtro

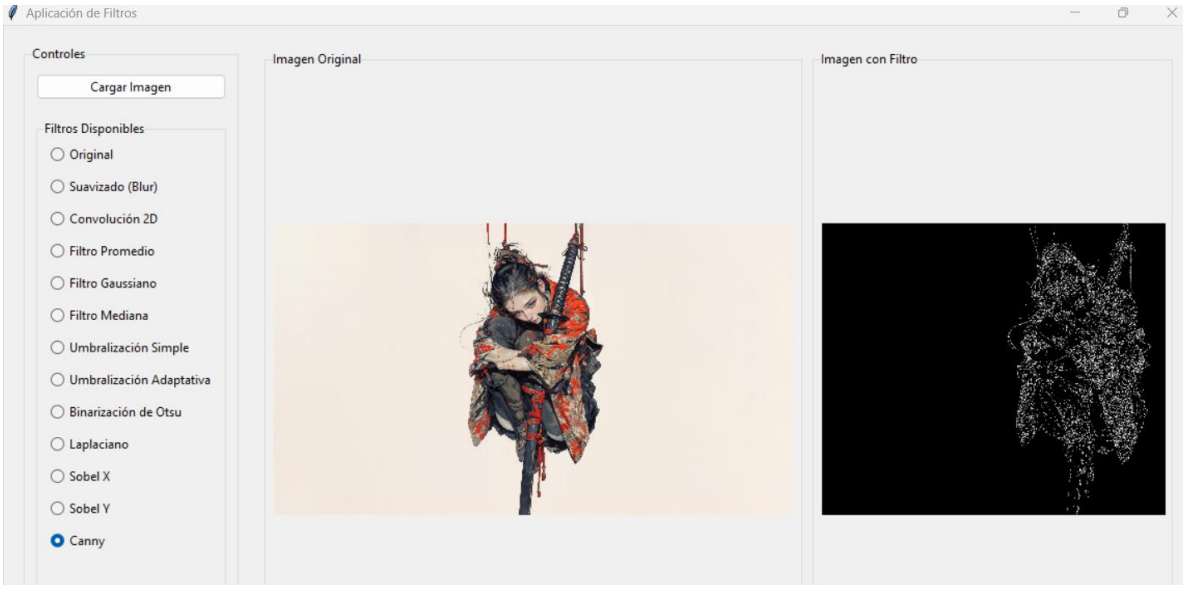

Aplicación de Filtros
Controles
Cargar Imagen
Filtros Disponibles
Original
Suavizado (Blur)
Convolución 2D
Filtro Promedio
Filtro Gaussiano
Filtro Mediana
Umbralización Simple
Umbralización Adaptativa
Binarización de Otsu
Laplaciano
Sobel X
Sobel Y
Canny
Imagen Original
Imagen con Filtro


Aplicación de Filtros
Controles
Cargar Imagen
Filtros Disponibles
Original
Suavizado (Blur)
Convolución 2D
Filtro Promedio
Filtro Gaussiano
Filtro Mediana
Umbralización Simple
Umbralización Adaptativa
Binarización de Otsu
Laplaciano
Sobel X
Sobel Y
Canny
Imagen Original
Imagen con Filtro


Aplicación de Filtros
Controles
Cargar Imagen
Filtros Disponibles
Original
Suavizado (Blur)
Convolución 2D
Filtro Promedio
Filtro Gaussiano
Filtro Mediana
Umbralización Simple
Umbralización Adaptativa
Binarización de Otsu
Laplaciano
Sobel X
Sobel Y
Canny
Imagen Original
Imagen con Filtro


Aplicación de Filtros
Controles
Cargar Imagen
Filtros Disponibles
Original
Suavizado (Blur)
Convolución 2D
Filtro Promedio
Filtro Gaussiano
Filtro Mediana
Umbralización Simple
Umbralización Adaptativa
Binarización de Otsu
Laplaciano
Sobel X
Sobel Y
Canny
Imagen Original
Imagen con Filtro


Aplicación de Filtros
Controles
Cargar Imagen
Filtros Disponibles
Original
Suavizado (Blur)
Convolución 2D
Filtro Promedio
Filtro Gaussiano
Filtro Mediana
Umbralización Simple
Umbralización Adaptativa
Binarización de Otsu
Laplaciano
Sobel X
Sobel Y
Canny
Imagen Original
Imagen con Filtro


Aplicación de Filtros
Controles
Cargar Imagen
Filtros Disponibles
Original
Suavizado (Blur)
Convolución 2D
Filtro Promedio
Filtro Gaussiano
Filtro Mediana
Umbralización Simple
Umbralización Adaptativa
Binarización de Otsu
Laplaciano
Sobel X
Sobel Y
Canny
Imagen Original
Imagen con Filtro