DOCUMENTACIÓN DEL CÓDIGO

Manipulación de Imágenes con Transformaciones Lineales

Autores

Berumen Martinez Juan Luis

Herrera Monroy Abraham Andre

Pérez Méndez Nancy Esmeralda

Romero Rosales Lino Ehecatl

Descripción

El proyecto consiste en crear una herramienta que permita al usuario cargar una imagen, aplicar diversas transformaciones geométricas y observar los efectos resultantes.

Tecnologías Utilizadas:

- ✓ Python

Librerias

import numpy as np

import cv2

import tkinter as tk

from tkinter import filedialog, messagebox

from PIL import Image, ImageTk

def cargar_imagen(ruta):

Carga una imagen desde la ruta especificada.

Parámetros:

ruta (str): La ruta del archivo de imagen a cargar.

Retorna:

numpy.ndarray: Imagen cargada en formato RGB.

Lanza:

FileNotFoundError: Si no se puede cargar la imagen desde la ruta proporcionada.

Codigo:

```
imagen = cv2.imread(ruta)
if imagen is None:
    raise FileNotFoundError("No se pudo cargar la imagen. Verifica la ruta.")
return cv2.cvtColor(imagen, cv2.COLOR_BGR2RGB)  # Convertir de BGR (OpenCV) a RGB
```

def rotar_imagen(imagen, angulo):

Aplica una rotación a la imagen.

Parámetros:

imagen (numpy.ndarray): Imagen a rotar.

angulo (float): Ángulo en grados para rotar la imagen.

Retorna:

numpy.ndarray: Imagen rotada.

Codigo:

```
filas, columnas, _ = imagen.shape
```

matriz_rotacion = cv2.getRotationMatrix2D((columnas / 2, filas / 2), angulo, 1)

return cv2.warpAffine(imagen, matriz_rotacion, (columnas, filas))

def escalar_imagen(imagen, factor_x, factor_y):

Escala la imagen por los factores especificados.

Parámetros:

imagen (numpy.ndarray): Imagen a escalar.

factor_x (float): Factor de escala en el eje X.

factor_y (float): Factor de escala en el eje Y.

Retorna:

numpy.ndarray: Imagen escalada.

Lanza:

ValueError: Si alguno de los factores de escala es mayor a 5.

Codigo:

filas, columnas, _ = imagen.shape

if factor_x > 5 or factor_y > 5:

raise ValueError("El factor de escala no puede ser mayor a 5 para evitar problemas de memoria.")

nuevo_ancho = max(1, int(columnas * factor_x)) # Asegurarse de que el ancho sea al menos 1

nuevo_alto = max(1, int(filas * factor_y)) # Asegurarse de que el alto sea al menos 1

return cv2.resize(imagen, (nuevo_ancho, nuevo_alto), interpolation=cv2.INTER_LINEAR)

def reflejar_imagen(imagen, eje):

Refleja la imagen sobre el eje especificado.

Parámetros:

imagen (numpy.ndarray): Imagen a reflejar.

eje (str): Eje de reflexión, puede ser 'horizontal' o 'vertical'.

Retorna:

numpy.ndarray: Imagen reflejada.

Lanza:

ValueError: Si el eje no es 'horizontal' o 'vertical'.

Codigo:

```
if eje == 'horizontal':
    return cv2.flip(imagen, 0)
elif eje == 'vertical':
    return cv2.flip(imagen, 1)
else:
    raise ValueError("El eje debe ser 'horizontal' o 'vertical'.")
```

def trasladar_imagen(imagen, desplazamiento_x, desplazamiento_y):

Aplica una traslación a la imagen.

Parámetros:

imagen (numpy.ndarray): Imagen a trasladar.

desplazamiento_x (int): Desplazamiento en el eje X (en píxeles).

desplazamiento_y (int): Desplazamiento en el eje Y (en píxeles).

Retorna:

numpy.ndarray: Imagen trasladada.

Codigo:

```
filas, columnas, _ = imagen.shape
matriz_traslacion = np.float32([[1, 0, desplazamiento_x], [0, 1, desplazamiento_y]])
return cv2.warpAffine(imagen, matriz_traslacion, (columnas, filas))
```

def mostrar_imagen(imagen_cv):

Convierte la imagen de OpenCV (formato BGR) a un formato compatible con Tkinter y la muestra.

Parámetros:

imagen_cv (numpy.ndarray): Imagen en formato BGR (OpenCV).

Retorna:

ImageTk.PhotoImage: Objeto de imagen compatible con Tkinter.

Codigo:

```
imagen_pil = Image.fromarray(imagen_cv)
```

```
    return ImageTk.PhotoImage(imagen_pil)
```

def cargar_archivo():

Abre un cuadro de diálogo para seleccionar una imagen y cargarla en la interfaz gráfica.

Carga la imagen seleccionada, la convierte al formato RGB y actualiza la interfaz con la imagen original.

Codigo:

```
global imagen_original, imagen_transformada, tk_imagen_original, historial_transformaciones
    ruta = filedialog.askopenfilename(filetypes=[("Imágenes", ".jpg;.png")])
    if ruta:
        try:
            imagen_original = cargar_imagen(ruta)
            imagen_transformada = imagen_original.copy()  # Inicializar la imagen transformada
            historial_transformaciones = [imagen_original.copy()]  # Inicializar historial
            tk_imagen_original = mostrar_imagen(imagen_original)
            etiqueta_imagen_original.config(image=tk_imagen_original)
            etiqueta_imagen_original.image = tk_imagen_original
        except FileNotFoundError as e:
            messagebox.showerror("Error", str(e))
```

def actualizar_campos(*args):

Actualiza los campos visibles de la interfaz según la transformación seleccionada.

Según la opción seleccionada (Rotar, Escalar, Reflejar, Trasladar), actualiza la interfaz

para mostrar los campos de entrada necesarios.

Codigo:

```
    seleccion = opcion_transformacion.get()


    # Ocultar todos los elementos primero
```

```
entrada_parametro_1.pack_forget()
entrada_parametro_2.pack_forget()
etiqueta_parametro_1.pack_forget()
etiqueta_parametro_2.pack_forget()
boton_aplicar.pack_forget()
boton_guardar.pack_forget()

if seleccion == "Rotar":
    etiqueta_parametro_1.config(text="Ángulo (grados):")
    etiqueta_parametro_1.pack(pady=5)
    entrada_parametro_1.pack(pady=5, ipadx=5, ipady=5)
elif seleccion == "Escalar":
    # Mostrar primero los campos X e Y
    etiqueta_parametro_1.config(text="Factor de escala en X (máx. 5):")
    etiqueta_parametro_1.pack(pady=5)
    entrada_parametro_1.pack(pady=5, ipadx=5, ipady=5)
    etiqueta_parametro_2.config(text="Factor de escala en Y (máx. 5):")
    etiqueta_parametro_2.pack(pady=5)
    entrada_parametro_2.pack(pady=5, ipadx=5, ipady=5)
elif seleccion == "Reflejar":
    etiqueta_parametro_1.config(text="Eje (horizontal/vertical):")
    etiqueta_parametro_1.pack(pady=5)
    entrada_parametro_1.pack(pady=5, ipadx=5, ipady=5)
elif seleccion == "Trasladar":
    # Mostrar primero los campos X e Y
    etiqueta_parametro_1.config(text="Desplazamiento en X (px):")
    etiqueta_parametro_1.pack(pady=5)
    entrada_parametro_1.pack(pady=5, ipadx=5, ipady=5)
    etiqueta_parametro_2.config(text="Desplazamiento en Y (px):")
    etiqueta_parametro_2.pack(pady=5)
    entrada_parametro_2.pack(pady=5, ipadx=5, ipady=5)
```

```
# Mostrar los botones al final
boton_aplicar.pack(pady=10, ipadx=5, ipady=5)
boton_guardar.pack(pady=10, ipadx=5, ipady=5)
```

def aplicar_transformacion():

Aplica la transformación seleccionada a la imagen transformada.

Si no se ha cargado una imagen, muestra un error. Aplica la rotación, escalado, reflexión o traslación según la opción seleccionada y actualiza la interfaz.

Codigo:

```
global imagen_transformada, tk_imagen_transformada, historial_transformaciones
if imagen_transformada is None:
    messagebox.showerror("Error", "Primero carga una imagen.")
    return
try:
    if opcion_transformacion.get() == "Rotar":
        angulo = float(entrada_parametro_1.get())
        imagen_transformada = rotar_imagen(imagen_transformada, angulo)
    elif opcion_transformacion.get() == "Escalar":
        factor_x = float(entrada_parametro_1.get())
        factor_y = float(entrada_parametro_2.get())
        imagen_transformada = escalar_imagen(imagen_transformada, factor_x, factor_y)
    elif opcion_transformacion.get() == "Reflejar":
        eje = entrada_parametro_1.get().strip().lower()
        imagen_transformada = reflejar_imagen(imagen_transformada, eje)
    elif opcion_transformacion.get() == "Trasladar":
        desplazamiento_x = int(entrada_parametro_1.get())
        desplazamiento_y = int(entrada_parametro_2.get())
        imagen_transformada = trasladar_imagen(imagen_transformada, desplazamiento_x, desplazamiento_y)
    else:
```

```
            messagebox.showerror("Error", "Selecciona una transformación válida.")
            return

        historial_transformaciones.append(imagen_transformada.copy())  # Guardar en historial
        tk_imagen_transformada = mostrar_imagen(imagen_transformada)
        etiqueta_imagen_transformada.config(image=tk_imagen_transformada)
        etiqueta_imagen_transformada.image = tk_imagen_transformada
    except ValueError as e:
        messagebox.showerror("Error", "Los parámetros deben ser números válidos.")
    except Exception as e:
        messagebox.showerror("Error", str(e))
```

def deshacer_transformacion():

Deshace la última transformación aplicada y restaura la imagen a su estado anterior.

Codigo:

```
    global imagen_transformada, tk_imagen_transformada, historial_transformaciones
    if len(historial_transformaciones) > 1:
        historial_transformaciones.pop()  # Eliminar la última transformación
        imagen_transformada = historial_transformaciones[-1]  # Recuperar la imagen previa
        tk_imagen_transformada = mostrar_imagen(imagen_transformada)
        etiqueta_imagen_transformada.config(image=tk_imagen_transformada)
        etiqueta_imagen_transformada.image = tk_imagen_transformada
    else:
        messagebox.showinfo("Información", "No hay transformaciones para deshacer.")
```

def restaurar_imagen_original():

Restaura la imagen al estado original, sin ninguna transformación aplicada.

```
    global imagen_transformada, tk_imagen_transformada, historial_transformaciones
    imagen_transformada = imagen_original.copy()
    tk_imagen_transformada = mostrar_imagen(imagen_transformada)
    etiqueta_imagen_transformada.config(image=tk_imagen_transformada)
    etiqueta_imagen_transformada.image = tk_imagen_transformada
```

```
    historial_transformaciones = [imagen_original.copy()]  # Limpiar historial
```

def guardar_imagen():

Permite guardar la imagen transformada en el sistema de archivos.

Codigo:

```
    global imagen_transformada
    if imagen_transformada is None:
        messagebox.showerror("Error", "No hay imagen para guardar.")
        return
    ruta_guardado = filedialog.asksaveasfilename(defaultextension=".png", filetypes=[("PNG", "*.png"), ("JPEG", "*.jpg")])
    if ruta_guardado:
        try:
            imagen_guardada = cv2.cvtColor(imagen_transformada, cv2.COLOR_RGB2BGR)  # Convertir de RGB a BGR para OpenCV
            cv2.imwrite(ruta_guardado, imagen_guardada)
            messagebox.showinfo("Éxito", f"Imagen guardada en {ruta_guardado}")
        except Exception as e:
            messagebox.showerror("Error", f"No se pudo guardar la imagen. {str(e)}")


# Crear la ventana principal de la interfaz gráfica
ventana = tk.Tk()
ventana.title("Editor de Imágenes")
ventana.geometry("800x600")


# Crear un contenedor de imagen para mostrar la imagen cargada y transformada
frame_imagen = tk.Frame(ventana)
frame_imagen.pack(pady=20)


etiqueta_imagen_original = tk.Label(frame_imagen)
etiqueta_imagen_original.grid(row=0, column=0, padx=10)
```

```
etiqueta_imagen_transformada = tk.Label(frame_imagen)
etiqueta_imagen_transformada.grid(row=0, column=1, padx=10)

# Crear el cuadro de selección de transformación
opcion_transformacion = tk.StringVar()
opcion_transformacion.set("Rotar")  # Selección por defecto
menu_transformacion = tk.OptionMenu(ventana, opcion_transformacion, "Rotar", "Escalar", "Reflejar", "Trasladar")
menu_transformacion.pack(pady=10)

# Crear los campos para ingresar los parámetros de la transformación
entrada_parametro_1 = tk.Entry(ventana)
entrada_parametro_2 = tk.Entry(ventana)

etiqueta_parametro_1 = tk.Label(ventana)
etiqueta_parametro_2 = tk.Label(ventana)

# Crear los botones para aplicar transformaciones y guardar
boton_aplicar = tk.Button(ventana, text="Aplicar", command=aplicar_transformacion)
boton_guardar = tk.Button(ventana, text="Guardar Imagen", command=guardar_imagen)

# Crear los botones para cargar una imagen y deshacer transformaciones
boton_cargar = tk.Button(ventana, text="Cargar Imagen", command=cargar_archivo)
boton_deshacer = tk.Button(ventana, text="Deshacer", command=deshacer_transformacion)
boton_restaurar = tk.Button(ventana, text="Restaurar Imagen Original", command=restaurar_imagen_original)

# Empacar los botones de cargar imagen y deshacer
boton_cargar.pack(pady=5)
boton_deshacer.pack(pady=5)
boton_restaurar.pack(pady=5)
```

```
# Empacar los campos y botones para las transformaciones
opcion_transformacion.pack(pady=5)
boton_aplicar.pack_forget()  # Inicialmente oculto
boton_guardar.pack_forget()  # Inicialmente oculto

# Configurar la actualización de campos dependiendo de la transformación seleccionada
opcion_transformacion.trace_add("write", actualizar_campos)

# Iniciar la interfaz
ventana.mainloop()
```

Proyectos de ejemplo

1. Proyecto: Cargar y Visualizar Imágenes

Descripción:

Este proyecto se enfoca en cargar una imagen desde el sistema de archivos del usuario y mostrarla en una interfaz gráfica.

Objetivo:

- Usar la función `cargar_imagen` para leer una imagen desde una ruta dada.
- Mostrar la imagen cargada en una interfaz gráfica utilizando Tkinter y PIL.

Código de Ejemplo:

```
importtkinterastk
fromtkinterimportfiledialog
importcv2
fromPILimportImage,ImageTk

defcargar_imagen(ruta):
imagen=cv2.imread(ruta)
ifimagenisNone:
raise FileNotFoundError("No se pudo cargar la imagen. Verifica la ruta.")
returncv2.cvtColor(imagen,cv2.COLOR_BGR2RGB)

defmostrar_imagen(imagen_cv):
imagen_pil=Image.fromarray(imagen_cv)
returnImageTk.PhotoImage(imagen_pil)
```

```
defcargar_archivo():
ruta=filedialog.askopenfilename(filetypes=[("Imágenes", ".jpg;.png")])
ifruta:
imagen=cargar_imagen(ruta)
tk_imagen=mostrar_imagen(imagen)
etiqueta.config(image=tk_imagen)
etiqueta.image=tk_imagen

ventana = tk.Tk()
ventana.title("Cargar Imagen")

etiqueta = tk.Label(ventana)
etiqueta.pack()

boton_cargar=tk.Button(ventana,text="CargarImagen",
command=cargar_archivo)
boton_cargar.pack()

ventana.mainloop()
```

2. Proyecto: Rotar Imágenes

Descripción:

Este proyecto permite a los usuarios rotar una imagen cargada en cualquier ángulo especificado por ellos.

Objetivo:

- Implementar la función `rotar_imagen` para rotar la imagen cargada por un ángulo dado por el usuario.
- Mostrar la imagen rotada en la interfaz.

Código de Ejemplo:

```
def rotar_imagen(imagen, angulo):
    filas, columnas, _ = imagen.shape
    matriz_rotacion = cv2.getRotationMatrix2D((columnas / 2, filas / 2), angulo, 1)
    return cv2.warpAffine(imagen, matriz_rotacion, (columnas, filas))

def aplicar_rotacion():
    angulo = float(entry_angulo.get())
    imagen_rotada = rotar_imagen(imagen, angulo)
```

```
    tk_imagen_rotada = mostrar_imagen(imagen_rotada)
    etiqueta.config(image=tk_imagen_rotada)
    etiqueta.image = tk_imagen_rotada
```

3. Proyecto: Escalar Imágenes

Descripción:

Este proyecto permite a los usuarios escalar una imagen según los factores de escala en los ejes X e Y.

Objetivo:

- Usar la función `escalar_imagen` para modificar el tamaño de la imagen basada en los factores de escala proporcionados.
- Ver la imagen escalada en la interfaz gráfica.

Código de Ejemplo:

```
def escalar_imagen(imagen, factor_x, factor_y):
    filas, columnas, _ = imagen.shape
    nuevo_ancho = int(columnas * factor_x)
    nuevo_alto = int(filas * factor_y)
    return cv2.resize(imagen, (nuevo_ancho, nuevo_alto), interpolation=cv2.INTER_LINEAR)

def aplicar_escala():
    factor_x = float(entry_factor_x.get())
    factor_y = float(entry_factor_y.get())
    imagen_escalada = escalar_imagen(imagen, factor_x, factor_y)
    tk_imagen_escalada = mostrar_imagen(imagen_escalada)
    etiqueta.config(image=tk_imagen_escalada)
    etiqueta.image = tk_imagen_escalada
```

4. Proyecto: Reflejar Imágenes

Descripción:

En este proyecto, se permite al usuario reflejar la imagen sobre un eje (horizontal o vertical).

Objetivo:

- Utilizar la función `reflejar_imagen` para voltear la imagen según el eje especificado por el usuario.
- Mostrar la imagen reflejada en la interfaz.

Código de Ejemplo:

```
defreflejar_imagen(imagen,eje):
ifeje=='horizontal':
returncv2.flip(imagen,0)
elifeje=='vertical':returncv2.flip(imagen,1)
else:
raise ValueError("El eje debe ser 'horizontal' o 'vertical'.")

defaplicar_reflejo():
eje=entry_eje.get().strip().lower()
imagen_reflejada=reflejar_imagen(imagen,eje)

tk_imagen_reflejada=mostrar_imagen(imagen_reflejada)
etiqueta.config(image=tk_imagen_reflejada)
 etiqueta.image = tk_imagen_reflejada
```

5. Proyecto: Trasladar Imágenes

Descripción:

Este proyecto permite trasladar la imagen en el espacio 2D a través de valores de desplazamiento en los ejes X e Y.

Objetivo:

- Usar la función `trasladar_imagen` para mover la imagen a una nueva ubicación según el desplazamiento en los ejes X e Y.
- Mostrar la imagen trasladada en la interfaz gráfica.

Código de Ejemplo:

```
def trasladar_imagen(imagen, desplazamiento_x, desplazamiento_y):
filas, columnas, _ =imagen.shape
matriz_traslacion=np.float32([[1,0,desplazamiento_x],[0,1,desplazamien
to_y]])
return cv2.warpAffine(imagen, matriz_traslacion, (columnas, filas))

def aplicar_traslacion():
```

```
    desplazamiento_x        =        int(entry_desplazamiento_x.get())
    desplazamiento_y        =        int(entry_desplazamiento_y.get())
    imagen_trasladada   =   trasladar_imagen(imagen,   desplazamiento_x,
desplazamiento_y)
    tk_imagen_trasladada      =      mostrar_imagen(imagen_trasladada)
    etiqueta.config(image=tk_imagen_trasladada)
    etiqueta.image                =                tk_imagen_trasladada
```

6. Proyecto: Guardar Imagen Transformada

Descripción:

Este proyecto permite guardar la imagen transformada (después de aplicar cualquier transformación) en el disco.

Objetivo:

- Implementar la función `guardar_imagen` para permitir al usuario guardar la imagen transformada en formato JPG o PNG.
- Almacenar la imagen en el directorio seleccionado por el usuario.

Código de Ejemplo:

```
defguardar_imagen():
ifimagen_transformadaisNone:
messagebox.showerror("Error",  "No  hay  una  imagen  transformada  para
guardar.")
return
ruta=filedialog.asksaveasfilename(defaultextension=".jpg",filetypes=[(
"Imágenes",                                                  ".jpg;.png")])
    if                                                              ruta:
cv2.imwrite(ruta,cv2.cvtColor(imagen_transformada,  cv2.COLOR_RGB2BGR))
#         Convertir         a         BGR         para         guardar
messagebox.showinfo("Éxito",    "Imagen    guardada    correctamente.")
```

Documentación del Código de Manipulación de Imágenes

1. `cargar_imagen(ruta)`

- **Descripción:**

- Esta función carga una imagen desde la ruta especificada por el usuario y la convierte de formato BGR a RGB

- **Parámetros:**
  - `ruta` (str): La ruta completa del archivo de imagen que se desea cargar.
- **Valor de Retorno:**
  - `numpy.ndarray`: La imagen cargada y convertida de BGR a RGB.
- **Excepciones:**
  - Lanza un `FileNotFoundError` si no se puede cargar la imagen desde la ruta especificada.

2. `rotar_imagen(imagen, angulo)`

- **Descripción:**

Esta función rota la imagen especificada por un ángulo determinado en grados alrededor del centro de la imagen.

- **Parámetros:**
  - `imagen` (numpy.ndarray): La imagen sobre la que se realizará la rotación.
  - `angulo` (float): El ángulo en grados para la rotación de la imagen. Los valores negativos rotan la imagen en sentido horario y los positivos en sentido antihorario.
- **Valor de Retorno:**
  - `numpy.ndarray`: La imagen rotada.
- **Excepciones:**
  - No se esperan excepciones, pero si se pasa una imagen no válida, puede generar errores.

3. `escalar_imagen(imagen, factor_x, factor_y)`

- **Descripción:**

Esta función escala la imagen según los factores proporcionados en los ejes X e Y.

- **Parámetros:**
  - `imagen` (numpy.ndarray): La imagen que se desea escalar.
  - `factor_x` (float): El factor de escala en el eje X (ancho de la imagen).
  - `factor_y` (float): El factor de escala en el eje Y (alto de la imagen).

- **Valor de Retorno:**
  - `numpy.ndarray`: La imagen escalada según los factores proporcionados.
- **Excepciones:**
  - Lanza un `ValueError` si los factores de escala `factor_x` o `factor_y` son mayores que 5 para evitar problemas de memoria o distorsión excesiva de la imagen.

4. `reflejar_imagen(imagen, eje)`

- **Descripción:**

Esta función refleja la imagen a lo largo de uno de los ejes especificados: horizontal o vertical.

- **Parámetros:**
  - `imagen` (numpy.ndarray): La imagen que se desea reflejar.
  - `eje` (str): El eje sobre el que se realizará el reflejo. Puede ser `'horizontal'` o `'vertical'`.
- **Valor de Retorno:**
  - `numpy.ndarray`: La imagen reflejada sobre el eje especificado.
- **Excepciones:**
  - Lanza un `ValueError` si el valor del parámetro `eje` no es `'horizontal'` ni `'vertical'`.

5.`trasladar_imagen(imagen,desplazamiento_x, desplazamiento_y)`

- **Descripción:**

Esta función mueve (traslada) la imagen en el espacio 2D a través de los desplazamientos dados en los ejes X e Y.

- **Parámetros:**
  - `imagen` (numpy.ndarray): La imagen que se desea trasladar.
  - `desplazamiento_x` (int): El desplazamiento de la imagen en el eje X (horizontal).
  - `desplazamiento_y` (int): El desplazamiento de la imagen en el eje Y (vertical).
- **Valor de Retorno:**
  - `numpy.ndarray`: La imagen trasladada según los valores proporcionados.
- **Excepciones:**
  - No se esperan excepciones, pero si los desplazamientos son muy grandes, la imagen puede moverse fuera del área visible.

6. `mostrar_imagen(imagen_cv)`

- **Descripción:**

Convierte una imagen de formato OpenCV (`numpy.ndarray`) a un formato que sea compatible con Tkinter para su visualización en la interfaz gráfica.

- **Parámetros:**
  - `imagen_cv` (numpy.ndarray): La imagen que se desea mostrar, en formato OpenCV (BGR/RGB).
- **Valor de Retorno:**
  - `ImageTk.PhotoImage`: Una imagen convertida en un formato que Tkinter puede manejar.

7. `cargar_archivo()`

- **Descripción:**

Abre un cuadro de diálogo para que el usuario seleccione una imagen desde su sistema de archivos y la carga para su visualización y posterior manipulación.

- **Parámetros:**

No tiene parámetros.

- **Valor de Retorno:**

No tiene valor de retorno.

- **Excepciones:**

Muestra un mensaje de error en un cuadro de diálogo si la imagen no puede cargarse o si la ruta proporcionada es inválida.

8. `actualizar_campos(*args)`

- **Descripción:**

Actualiza los campos visibles en la interfaz gráfica de acuerdo con la transformación seleccionada por el usuario (rotar, escalar, reflejar, trasladar).

- **Parámetros:**

No tiene parámetros, pero se activa automáticamente cuando se cambia la opción de transformación en el menú desplegable.

- **Valor de Retorno:**
  No tiene valor de retorno.
- **Excepciones:**

No se esperan excepciones, aunque si se selecciona una opción de transformación incorrecta, se podría mostrar un mensaje de error.

9. `aplicar_transformacion()`

- **Descripción:**

Aplica la transformación seleccionada (rotar, escalar, reflejar, trasladar) a la imagen cargada y actualiza la visualización de la imagen transformada en la interfaz gráfica.

- **Parámetros:**

No tiene parámetros.

- **Valor de Retorno:**

No tiene valor de retorno.

- **Excepciones:**

Lanza un `ValueError` si los parámetros de la transformación no son válidos (por ejemplo, si no son números), o muestra un mensaje de error si la transformación no se puede aplicar.

10. `deshacer_transformacion()`

- **Descripción:**

Deshace la última transformación aplicada y restaura la imagen al estado anterior.

- **Parámetros:**

No tiene parámetros.

- **Valor de Retorno:**

No tiene valor de retorno.

- **Excepciones:**

Muestra un mensaje si no hay más transformaciones para deshacer.

11. `restaurar_imagen_original()`

- **Descripción:**

Restaura la imagen a su estado original, es decir, la imagen cargada inicialmente, sin ninguna transformación aplicada.

- **Parámetros:**

No tiene parámetros.

- **Valor de Retorno:**

No tiene valor de retorno.

- **Excepciones:**

Muestra un mensaje si no se ha cargado una imagen original para restaurar.

12. `guardar_imagen()`

- **Descripción:**

Permite al usuario guardar la imagen transformada en un archivo en el sistema de archivos.

- **Parámetros:**

No tiene parámetros.

- **Valor de Retorno:**

No tiene valor de retorno.

- **Excepciones:**

Muestra un mensaje de error si no hay una imagen transformada para guardar.