UNIVERSIDAD DEL BÍO-BÍO

06 Ecualización de imágenes

Tutorial Procesamiento de Imagen con webcam

Estudiantes Practicantes:
Luis Pereira

Profesor:
Luis Vera

Laboratorio CIMUBB

2023-2

Introducción a la Ecualización de Imágenes en Visión Artificial

La ecualización de imágenes es una técnica fundamental en el procesamiento de imágenes y juega un papel esencial en el ámbito de la visión artificial. Esta técnica se utiliza para mejorar la calidad y el contraste de las imágenes, lo que a su vez facilita la detección de patrones, objetos y características relevantes en una escena. La ecualización de imágenes es particularmente valiosa en entornos donde las condiciones de iluminación pueden variar significativamente o donde las imágenes pueden sufrir de problemas de contraste.

¿Qué es la Ecualización de Imágenes?

La ecualización de imágenes es un proceso que ajusta la distribución de intensidades en una imagen para que esté distribuida de manera más uniforme en todo el rango de intensidades disponibles. Este proceso ayuda a resaltar detalles y estructuras que pueden no ser claramente visibles en imágenes con distribuciones de intensidad desequilibradas.

¿Cómo funciona la Ecualización de Imágenes?

En términos sencillos, la ecualización de imágenes redistribuye las intensidades de píxeles de una imagen para que abarquen todo el rango disponible. Esto se logra mediante la aplicación de una transformación a la función de distribución acumulativa de la imagen original. El resultado es una imagen mejorada en la que las áreas de interés son más discernibles debido a un mejor contraste.

Utilidad en Visión Artificial:

1) **Mejora del Contraste**: La ecualización de imágenes es crucial en la mejora del contraste de una imagen, lo que facilita la detección de bordes, contornos y detalles importantes.

2) **Detección de Objetos**: En tareas de detección de objetos, la ecualización puede resaltar los detalles de los objetos en condiciones de iluminación variables, mejorando así la precisión de los algoritmos de detección.

3) **Reconocimiento de Patrones**: Para aplicaciones de reconocimiento de patrones y características, la ecualización puede realzar características clave, haciendo que los algoritmos de reconocimiento sean más robustos.

4) **Preprocesamiento de Imágenes**: Antes de aplicar algoritmos más avanzados, la ecualización se utiliza como una etapa de preprocesamiento para mejorar la calidad de las imágenes y, por ende, la eficiencia de los algoritmos subsiguientes.

5) **Mejora de Imágenes Médicas**: En el campo de la medicina, la ecualización de imágenes es esencial para realzar detalles en imágenes médicas, facilitando así la identificación de estructuras anatómicas.

6) **Visión Nocturna:** En sistemas de visión nocturna, donde la iluminación es limitada, la ecualización puede ser clave para resaltar objetos y detalles en entornos oscuros.

La ecualización de imágenes, por lo tanto, desempeña un papel crucial en el mejoramiento de la calidad visual de las imágenes, lo que resulta en un procesamiento más efectivo y preciso en diversas aplicaciones de visión artificial.

Implementación del Proceso de Ecualización en Código

A continuación, se presenta una implementación en código del proceso de ecualización de imágenes utilizando las bibliotecas Tkinter, OpenCV y Matplotlib. Este programa, escrito en Python, proporciona una interfaz gráfica simple para cargar imágenes, ecualizarlas y visualizar tanto las imágenes originales como las ecualizadas junto con sus histogramas correspondientes.

Observación: Antes de implementar el código, es importante tener las siguientes bibliotecas importadas y previamente instaladas.

```
import tkinter as tk
from tkinter import filedialog, messagebox
from PIL import Image, ImageTk
import cv2
from matplotlib import pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
```

Figura 1

A continuación se detallara la función principal del programa **ecualizar_imagen**():

```
def ecualizar_imagen():
    try:
        path_image = filedialog.askopenfilename(filetypes=[
            ("image", "jpg"),
            ("image", "jpeg"),
            ("image", "png")
        ])
```

Figura 2

Agregamos try por si en algún momento capturamos un error y nos lo muestre en una ventana en vez de la consola, posteriormente path_image = filedialog.askopenfilename(...): Utiliza la función askopenfilename del módulo filedialog de la biblioteca Tkinter. Esta función abre un cuadro de diálogo que permite al usuario seleccionar un archivo. En este caso, el diálogo estará configurado para mostrar solo archivos con extensiones ".jpg", ".jpeg" y ".png". La ruta del archivo seleccionado se asigna a la variable path_image.

```
if len(path_image) > 0:
    # Lee la imagen
    imagenFile = cv2.imread(path_image)
    imagenFile = cv2.cvtColor(imagenFile, cv2.COLOR_BGR2RGB)
    imagenFile = cv2.resize(imagenFile, (360, 360))

    # Visualiza la imagen original en la ventana
    imOriginal = Image.fromarray(imagenFile)
    imgOriginal = ImageTk.PhotoImage(image=imOriginal)
    lblInputImagen1.configure(image=imgOriginal)
    lblInputImagen1.image = imgOriginal
```

Figura 3

Estas líneas de código cargan una imagen, la procesan para que sea adecuada para su visualización, y luego la muestran en una etiqueta de la interfaz gráfica de Tkinter. Este proceso se repite para la imagen ecualizada en secciones posteriores del código.

```
# Muestra el histograma de la imagen original
fig_original, ax_original = plt.subplots()
ax_original.hist(imagenFile.ravel(), 256, [0, 256], color='blue')
ax_original.set_title('Histograma original')
canvasOriginal = FigureCanvasTkAgg(fig_original, master=ventana)
canvasOriginal.draw()
canvasOriginal.get_tk_widget().place(x=60, y=480, width=500, height=300)
```

Figura 4

Estas líneas de código están relacionadas con la visualización del histograma de la imagen original en la interfaz gráfica. Aquí tienes un resumen con énfasis en los aspectos clave:

fig_original, ax_original = plt.subplots(): Crea una figura (**fig_original**) y ejes **(ax_original**) para el histograma utilizando Matplotlib.

La figura es necesaria para contener el gráfico, y los ejes representan el área donde se dibujará el histograma.

ax_original.hist(imagenFile.ravel(), **255, [0, 255], color='blue'):** Genera el histograma utilizando la función **hist** de Matplotlib en los **ejes ax_original.**

imagenFile.ravel() convierte la matriz de la imagen a una única dimensión (vector). 255 especifica el número de bins (grupos) en el histograma. [0, 255] define el rango de valores para los bins.

color='blue' establece el color del histograma.

ax_original.set_title('Histograma original'): Asigna un título al histograma, en este caso, 'Histograma original'.

canvasOriginal = **FigureCanvasTkAgg(fig_original, master=ventana):** Crea un lienzo (canvas) para la figura utilizando la clase **FigureCanvasTkAgg** de Matplotlib.

master=ventana especifica que el lienzo se ubicará dentro de la ventana principal.

canvasOriginal.draw(): Dibuja la figura en el lienzo.

canvasOriginal.get_tk_widget().place(x=60, y=480, width=500, height=300): Coloca el lienzo en la interfaz gráfica de Tkinter en una posición específica (coordenadas x e y) y con dimensiones definidas (ancho y alto).

```
# Ecualiza la imagen
gray = cv2.cvtColor(imagenFile, cv2.COLOR_RGB2GRAY)
equ = cv2.equalizeHist(gray)
```

Figura 5

Esta línea de código está relacionada con el proceso de ecualización de la imagen original.

gray = **cv2.cvtColor(imagenFile, cv2.COLOR_RGB2GRAY):** Convierte la imagen original (imagenFile) de formato RGB a escala de grises.

cv2.COLOR_RGB2GRAY indica el tipo de conversión de color, transformando la imagen de un formato de tres canales (rojo, verde, azul) a escala de grises, que es de un solo canal.

equ = cv2.equalizeHist(gray): Aplica la ecualización del histograma a la imagen en escala de grises (gray) utilizando la función **equalizeHist** de **OpenCV**.

Esta función mejora el contraste de la imagen redistribuyendo las intensidades de píxeles de manera más uniforme a lo largo de todo el rango de intensidades disponible.

El resultado de la ecualización se almacena en la variable **equ**.

```
# Visualiza la imagen ecualizada en la ventana
imEcualizada = Image.fromarray(cv2.cvtColor(equ, cv2.COLOR_GRAY2RGB))
imgEcualizada = ImageTk.PhotoImage(image=imEcualizada)
lblInputImagen2.configure(image=imgEcualizada)
lblInputImagen2.image = imgEcualizada
```

Figura 6

Estas líneas de código cargan una imagen, la procesan para que sea adecuada para su visualización, y luego la muestran en una etiqueta de la interfaz gráfica de Tkinter

```
# Muestra el histograma de la imagen ecualizada
fig_ecualizada, ax_ecualizada = plt.subplots()
ax_ecualizada.hist(equ.ravel(), 256, [0, 256], color='red')
ax_ecualizada.set_title('Histograma ecualizado')
canvasEcualizado = FigureCanvasTkAgg(fig_ecualizada, master=ventana)
canvasEcualizado.draw()
canvasEcualizado.get_tk_widget().place(x=580, y=480, width=500, height=300)
```

Figura 7

Estas líneas de código hacen el histograma al igual que en la figura 4 pero ahora para la imagen ecualizada, con los pixeles distribuidos más uniformemente.

```
except Exception as e:
    messagebox.showerror(message=f"Error: {str(e)}")
```

Figura 8

Estas 2 líneas de código se encargan de capturar un error que haya sucedido durante la ejecución del programa y muestra el error capturado por pantalla para informarle al usuario.

```
def boton_salir():
    ventana.destroy()
```

Figura 9

Función botón_salir():

Esta función destruye la ventana al hacer clic en el botón salir, es recomendable presionar el botón salir para no tener que cerrar visual code o usar Ctrl + C para forzar la detención del programa.

```
# Cuadros de las Imagenes
lblInputImagen1 = tk.Label(ventana)
lblInputImagen1.place(x=60, y=100, width=360, height=360)
lblInputImagen1.configure(bg='gray')

lblInputImagen2 = tk.Label(ventana)
lblInputImagen2.place(x=580, y=100, width=360, height=360)
lblInputImagen2.configure(bg="gray")

# Botones
CImagen = tk.Button(ventana, text="Cargar Imagen", command=ecualizar_imagen)
CImagen.place(x=0, y=0, width=160, height=50)
salir = tk.Button(ventana, text="Salir", command=boton_salir)
salir.place(x=170, y=0, width=110, height=50)

ventana.mainloop()
```

Figura 10

En estas líneas se crean y diseñan los elementos que utilizaremos en la interfaz grafica, no olvidar el **ventana.mainloop().**

Resultado final

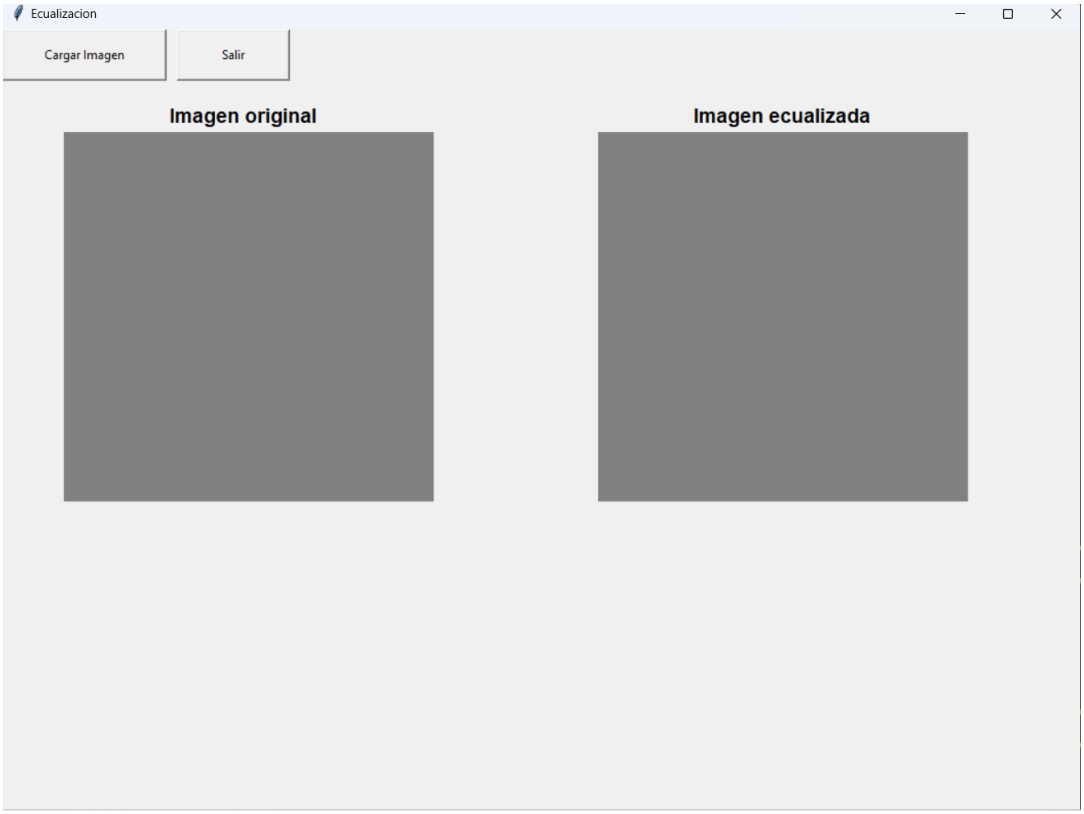

Figura 11

En ejecución…

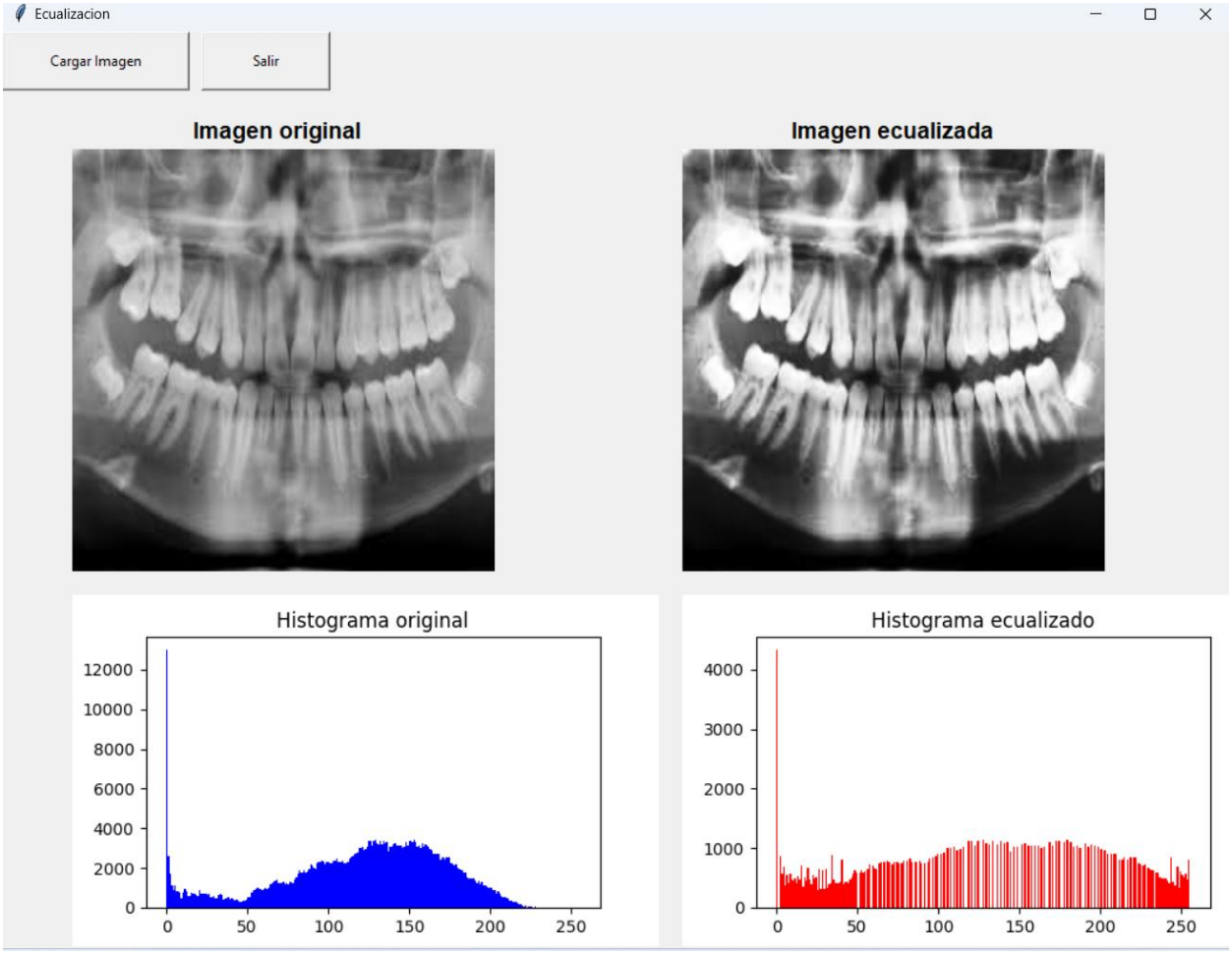

Figura 12