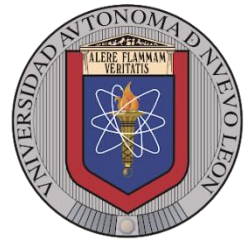


UANL

UNIVERSIDAD AUTONOMA DE NUEVO LEON

FACULTAD DE INGENIERIA MECANICA Y ELECTRICA

FIME

Nombre: Roberto Erick Aguilar Morales

Matricula: 1871004

Carrera: Ingeniero en Tecnologías del Software

4.– Umbrales

Materia: VISION COMPUTACIONAL LABORATORIO

Docente: RAYMUNDO SAID ZAMORA PEQUEÑO

Hora: N1-N2 **Días: Miércoles**

Fecha: 06/10/24

Objetivo

Utilizar umbrales para discretizar una imagen.

- Varianza
- Entropía
- Valle global

Utilice los umbrales para generar una imagen con 3, 4, 8 y 16 tonos

Marco teórico

Procesamiento de Imágenes

El procesamiento de imágenes es una disciplina que se ocupa de la manipulación y análisis de imágenes digitales a través de computadoras. Se emplea en diversas áreas, desde la medicina y la astronomía, hasta la inteligencia artificial y el control de calidad industrial. Dentro de esta área, el propósito principal es extraer información útil de las imágenes o transformarlas en representaciones más adecuadas para su análisis.

El primer paso en el procesamiento de imágenes digitales suele ser la conversión de la imagen a escala de grises, lo que simplifica el análisis y permite un tratamiento más eficiente de los píxeles. Las imágenes en escala de grises son representaciones bidimensionales donde cada píxel tiene un valor de intensidad que oscila entre 0 (negro) y 255 (blanco).

Umbralización de Imágenes

La umbralización es una técnica fundamental en el procesamiento de imágenes que consiste en transformar una imagen de múltiples tonos a una imagen binaria (blanco y negro) o con un número reducido de tonos. Se utiliza principalmente para resaltar o segmentar regiones de interés en una imagen, separando los objetos del fondo. Esto se logra mediante la selección de uno o más **umbrales** que determinan los puntos de corte de los niveles de intensidad de los píxeles.

Umbrales Globales y Locales

Un umbral global utiliza un único valor de umbral para toda la imagen, mientras que un umbral local se ajusta de manera dinámica según diferentes regiones de la imagen. En este trabajo nos enfocamos en métodos globales que establecen un único umbral para segmentar la imagen.

Métodos de Umbralización Automática

Para determinar de manera automática un umbral adecuado para la segmentación de imágenes, se han desarrollado diversos métodos que consideran aspectos estadísticos y de entropía de la distribución de los niveles de gris en la imagen.

Umbralización por Entropía

El método de Umbralización por Entropía utiliza la teoría de la información para maximizar la entropía de la imagen segmentada en dos clases: los píxeles de intensidad baja y los de intensidad alta. El objetivo es encontrar el umbral que mejor separa la imagen en términos de la cantidad de información que contienen ambas clases. Este método es particularmente útil cuando las intensidades de los píxeles en las regiones de interés y fondo son comparables, y se busca maximizar la cantidad de información en la segmentación.

La entropía se define como: $H(X) = -\sum_{i=0}^{n} p(x_i)\, log_2 p(x_i)$
donde $p(x_i)$ es la probabilidad de ocurrencia del nivel de intensidad x_i. El umbral óptimo es el que maximiza la entropía de la imagen segmentada.

Umbralización por Grupo de Varianza

La Umbralización por Grupo de Varianza se basa en la minimización de la varianza intragrupo y la maximización de la varianza intergrupo, lo que significa que busca el umbral que mejor separa los píxeles en dos clases: los de baja intensidad y los de alta intensidad. Este método es una implementación del criterio de Otsu, que es un algoritmo ampliamente utilizado en la umbralización automática.

La fórmula para calcular la varianza entre clases es: $\sigma_B^2 = w1(t)(\mu 1(t) - \mu T)^2 + w2(t)(\mu 2(t) - \mu T)^2$ donde $w1(t)$ y $w2(t)$ son las proporciones de los píxeles en las dos clases, $\mu 1(t)$ y $\mu 2(t)$ son las medias de las intensidades en las dos clases, y μT es la media total de la imagen.

Aproximación de Valle Global

La Aproximación de Valle Global se basa en la forma del histograma de la imagen. Este método busca los puntos de inflexión en el histograma para determinar un umbral adecuado. Los valles en el histograma representan las transiciones entre regiones claras y oscuras en la imagen, y el umbral óptimo se encuentra en el valle más profundo, lo que indica una buena separación entre el fondo y los objetos.

Este enfoque es particularmente útil cuando el histograma tiene una distribución bimodal o multimodal, lo que facilita la identificación de un umbral que se ajuste adecuadamente a la transición entre las regiones de la imagen.

Reducción de Tonos en Imágenes

La reducción de tonos es el proceso de simplificar la representación de una imagen mediante la disminución de los niveles de gris a un número específico de tonos. Este proceso es útil para simplificar la segmentación y el análisis de imágenes, especialmente cuando la imagen original tiene mucho ruido o variabilidad en los tonos.

En este trabajo, hemos aplicado la reducción de tonos a **3, 4, 8 y 16** tonos, para cada uno de los métodos de umbralización presentados (Entropía, Grupo de Varianza y Valle Global). Este enfoque permite comparar el rendimiento de los métodos y la calidad de la segmentación en diferentes niveles de tonalidades.

Introducción

Los 3 programas de umbrales semi-adaptativos llamados grupo varianza, umbral por entropía y aproximación por valle global están diseñados para encontrar el umbral optimo en una misma imagen, a su vez esta se dividirá en 3, 4, 8 y 16 tonos, por lo que su mayor cambio será su método principal para dividir los colores dependiendo su método diferente, aclarando esto podemos explicar de manera mas general la explicación del código de 3 programas en un solo documento

Umbral por Entropía

El método de Umbral por Entropía utiliza la teoría de la información para encontrar el umbral óptimo que maximiza la entropía entre las dos clases de píxeles (claros y oscuros). A continuación, se muestra el fragmento de código responsable del cálculo del umbral óptimo basado en la entropía:

```
def calcular_entropia(histograma, total_pixeles):
    # Cálculo de la entropía
    probabilidad = histograma / total_pixeles
    probabilidad = probabilidad[probabilidad > 0]  # Filtrar ceros
    entropia = -np.sum(probabilidad * np.log2(probabilidad))
    return entropia

def encontrar_mejor_umbral_entropia(imagen):
    # Calcular el histograma de la imagen
```

```
    histograma = calcular_histograma(imagen)
    total_pixeles = imagen.size

    # Variables para encontrar el mejor umbral
    mejor_entropia = 0
    mejor_umbral = 0
```

Este fragmento implementa la búsqueda del umbral óptimo al calcular la entropía de las dos clases de la imagen. Este método es eficiente cuando se busca maximizar la información contenida en la segmentación de la imagen.

Grupo de Varianza

El método de Grupo de Varianza busca minimizar la varianza dentro de las clases y maximizar la varianza entre clases. Este enfoque es especialmente útil en imágenes bimodales, donde los niveles de gris se agrupan en dos clases claramente definidas.

```
def calcular_grupo_varianza(histograma, total_pixeles):
    # Cálculo de varianza entre los grupos
    suma_total = np.sum([i * histograma[i] for i in range(256)])
    suma_b = 0
    w_b = 0
    varianza_max = 0
    mejor_umbral = 0

    for umbral in range(256):
        w_b += histograma[umbral]
        w_f = total_pixeles - w_b
        if w_b == 0 or w_f == 0:
            continue

        suma_b += umbral * histograma[umbral]
        m_b = suma_b / w_b if w_b != 0 else 0
        m_f = (suma_total - suma_b) / w_f if w_f != 0 else 0

        varianza_entre_clases = w_b * w_f * (m_b - m_f) ** 2

        if varianza_entre_clases > varianza_max:
            varianza_max = varianza_entre_clases
            mejor_umbral = umbral

    return mejor_umbral
```

Este código identifica el umbral óptimo basado en la varianza entre clases. La función calcular_grupo_varianza es clave para evaluar el umbral que maximiza la separación entre las clases de la imagen.

Aproximación de Valle Global

La Aproximación de Valle Global identifica los puntos de transición entre objetos y fondo en una imagen basada en los valles del histograma. El umbral óptimo se encuentra en el valle más profundo, lo que sugiere una buena separación entre las dos clases de píxeles.

```
def calcular_desviacion_estandar(imagen):
    # Cálculo de la desviación estándar
    return np.std(imagen)

def encontrar_picos(histograma):
    # Encontrar picos en el histograma (máximos locales)
    picos = []
    for i in range(1, len(histograma) - 1):
        if histograma[i] > histograma[i - 1] and histograma[i] >
histograma[i + 1]:
            picos.append(i)
    return picos

def calcular_valle_global(imagen):
    # Obtener el histograma
    histograma = calcular_histograma(imagen)
    total_pixeles = imagen.size

    # Calcular el umbral utilizando grupo varianza
    mejor_umbral = calcular_grupo_varianza(histograma, total_pixeles)

    # Calcular la desviación estándar
    desviacion_estandar = calcular_desviacion_estandar(imagen)

    # Encontrar los picos del histograma
    picos = encontrar_picos(histograma)

    return mejor_umbral, desviacion_estandar, picos
```

Este fragmento de código busca los valles en el histograma de la imagen y selecciona el valle más profundo como el umbral óptimo para la segmentación. Este método es útil cuando el histograma de la imagen presenta una clara distribución multimodal.

Generación de Imágenes con Diferentes Tonos

Para los tres métodos, se ha implementado una funcionalidad que permite aplicar los umbrales obtenidos para generar imágenes con **3, 4, 8 y 16 tonos**. A continuación, se muestra el fragmento común que se utiliza en los tres programas:

```
ef aplicar_tonos(imagen, num_tonos):
    # Definir los umbrales
    max_val = 255
    umbrales = np.linspace(0, max_val, num_tonos + 1)
    imagen_tonos = np.zeros_like(imagen)

    # Asignar tonos basados en los umbrales definidos
    for i in range(num_tonos):
        lower_bound = umbrales[i]
        upper_bound = umbrales[i + 1]
        imagen_tonos[(imagen >= lower_bound) & (imagen < upper_bound)] =
int((lower_bound + upper_bound) / 2)

    return imagen_tonos
```

Este código es responsable de reducir los tonos de la imagen a 3, 4, 8 o 16 niveles de gris, lo que simplifica la representación de la imagen y facilita su análisis en diferentes escalas de tonos.

Visualización del Histograma y Guardado de Imágenes

Finalmente, en cada uno de los métodos, se ha implementado una sección para visualizar los histogramas y guardar las matrices resultantes en archivos CSV, lo cual permite un análisis cuantitativo de los resultados:

```
# Mostrar el histograma y las imágenes umbralizadas
    plt.figure(figsize=(15, 10))

    # Mostrar el histograma
    plt.subplot(2, len(tonos), 1)
    plt.bar(range(256), histograma, color='black')
    plt.title('Histograma Original')
    plt.xlim([0, 255])

    # Mostrar las imágenes umbralizadas
    for idx, num_tonos in enumerate(tonos):
        plt.subplot(2, len(tonos), idx + 2)
        plt.imshow(imagenes_tonos[idx], cmap='gray')

        # Crear nombre de archivo dinámicamente usando f-string
```

```
        nombre_archivo = f"imagen_{num_tonos}_tonos.csv"
        np.savetxt(nombre_archivo, imagenes_tonos[idx], delimiter=',',
fmt='%d')

        plt.title(f'{num_tonos} tonos')

    for num_tonos in tonos:
        imagen_tonos = aplicar_tonos(imagen, num_tonos)
        # Guardar la imagen umbralizada con diferentes tonos
        cv2.imwrite(f'imagen_{num_tonos}_tonos.png', imagen_tonos)

    # Mostrar las imágenes con el histograma
    plt.tight_layout()
    plt.show()
```

Este fragmento de código es común en los tres métodos y permite visualizar las imágenes resultantes y guardar las matrices en formato CSV para su posterior análisis.

Cálculos y Resultados

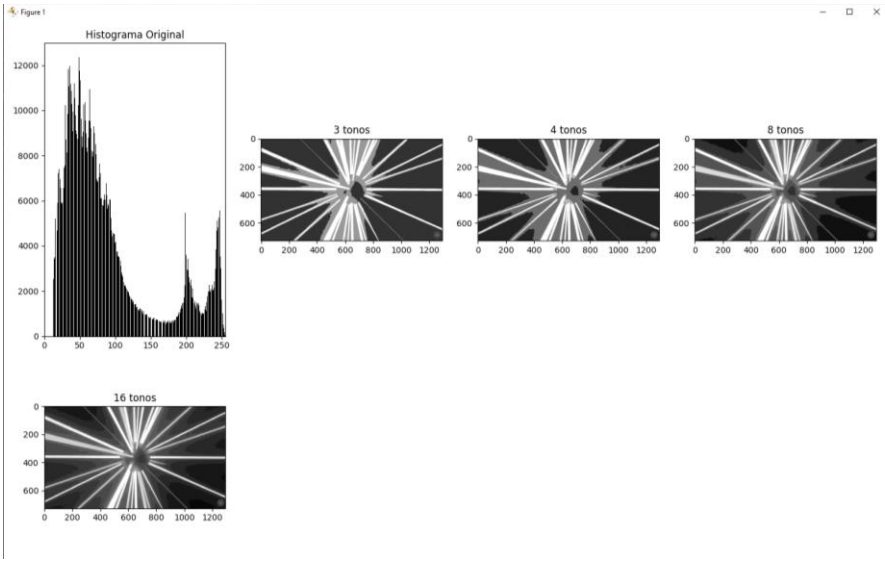
P04UmbralAproximacionDeValleGlobal.py

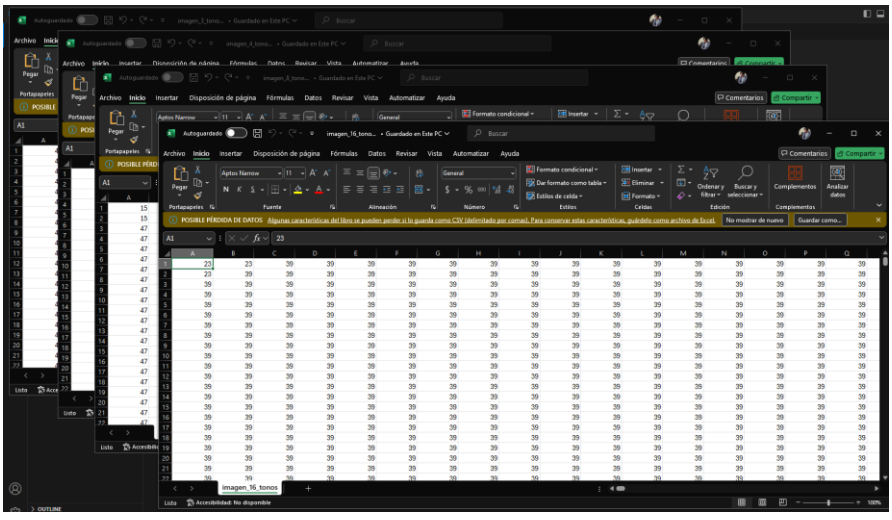
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage import io

def calcular_histograma(imagen):
    # Calcula el histograma de la imagen en escala de grises
    histograma, _ = np.histogram(imagen.ravel(), bins=256, range=(0, 256))
    return histograma

def calcular_grupo_varianza(histograma, total_pixeles):
    # Cálculo de varianza entre los grupos
    suma_total = np.sum([i * histograma[i] for i in range(256)])
    suma_b = 0
    w_b = 0
    varianza_max = 0
    mejor_umbral = 0

    for umbral in range(256):
        w_b += histograma[umbral]
        w_f = total_pixeles - w_b
        if w_b == 0 or w_f == 0:
            continue
```

```
        suma_b += umbral * histograma[umbral]
        m_b = suma_b / w_b if w_b != 0 else 0
        m_f = (suma_total - suma_b) / w_f if w_f != 0 else 0

        varianza_entre_clases = w_b * w_f * (m_b - m_f) ** 2

        if varianza_entre_clases > varianza_max:
            varianza_max = varianza_entre_clases
            mejor_umbral = umbral

    return mejor_umbral

def calcular_desviacion_estandar(imagen):
    # Cálculo de la desviación estándar
    return np.std(imagen)

def encontrar_picos(histograma):
    # Encontrar picos en el histograma (máximos locales)
    picos = []
    for i in range(1, len(histograma) - 1):
        if histograma[i] > histograma[i - 1] and histograma[i] >
histograma[i + 1]:
            picos.append(i)
    return picos

def calcular_valle_global(imagen):
    # Obtener el histograma
    histograma = calcular_histograma(imagen)
    total_pixeles = imagen.size

    # Calcular el umbral utilizando grupo varianza
    mejor_umbral = calcular_grupo_varianza(histograma, total_pixeles)

    # Calcular la desviación estándar
    desviacion_estandar = calcular_desviacion_estandar(imagen)

    # Encontrar los picos del histograma
    picos = encontrar_picos(histograma)

    return mejor_umbral, desviacion_estandar, picos

def aplicar_tonos(imagen, num_tonos):
    # Definir los umbrales
    max_val = 255
    umbrales = np.linspace(0, max_val, num_tonos + 1)
```

```
    imagen_tonos = np.zeros_like(imagen)

    # Asignar tonos basados en los umbrales definidos
    for i in range(num_tonos):
        lower_bound = umbrales[i]
        upper_bound = umbrales[i + 1]
        imagen_tonos[(imagen >= lower_bound) & (imagen < upper_bound)] =
int((lower_bound + upper_bound) / 2)

    return imagen_tonos

# Cargar la imagen
imagen_a_color = cv2.imread("imagen a color.png")
if imagen_a_color is None:
    print("La imagen no se cargó correctamente, favor de verificar la ruta")
else:
    # Convertir la imagen a escala de Grises
    imagen = cv2.cvtColor(imagen_a_color, cv2.COLOR_BGR2GRAY)

    # Calcular umbrales
    mejor_umbral, desviacion_estandar, picos = calcular_valle_global(imagen)

    # Obtener el histograma de la imagen original
    histograma = calcular_histograma(imagen)

    # Aplicar diferentes tonos
    tonos = [3, 4, 8, 16]
    imagenes_tonos = [aplicar_tonos(imagen, num_tonos) for num_tonos in
tonos]

    # Mostrar el histograma y las imágenes umbralizadas
    plt.figure(figsize=(15, 10))

    # Mostrar el histograma
    plt.subplot(2, len(tonos), 1)
    plt.bar(range(256), histograma, color='black')
    plt.title('Histograma Original')
    plt.xlim([0, 255])

    # Mostrar las imágenes umbralizadas
    for idx, num_tonos in enumerate(tonos):
        plt.subplot(2, len(tonos), idx + 2)
        plt.imshow(imagenes_tonos[idx], cmap='gray')

        # Crear nombre de archivo dinámicamente usando f-string
```

```
        nombre_archivo = f"imagen_{num_tonos}_tonos.csv"
        np.savetxt(nombre_archivo, imagenes_tonos[idx], delimiter=',',
fmt='%d')

        plt.title(f'{num_tonos} tonos')

    for num_tonos in tonos:
        imagen_tonos = aplicar_tonos(imagen, num_tonos)
        # Guardar la imagen umbralizada con diferentes tonos
        cv2.imwrite(f'imagen_{num_tonos}_tonos.png', imagen_tonos)

    # Mostrar las imágenes con el histograma
    plt.tight_layout()
    plt.show()
```



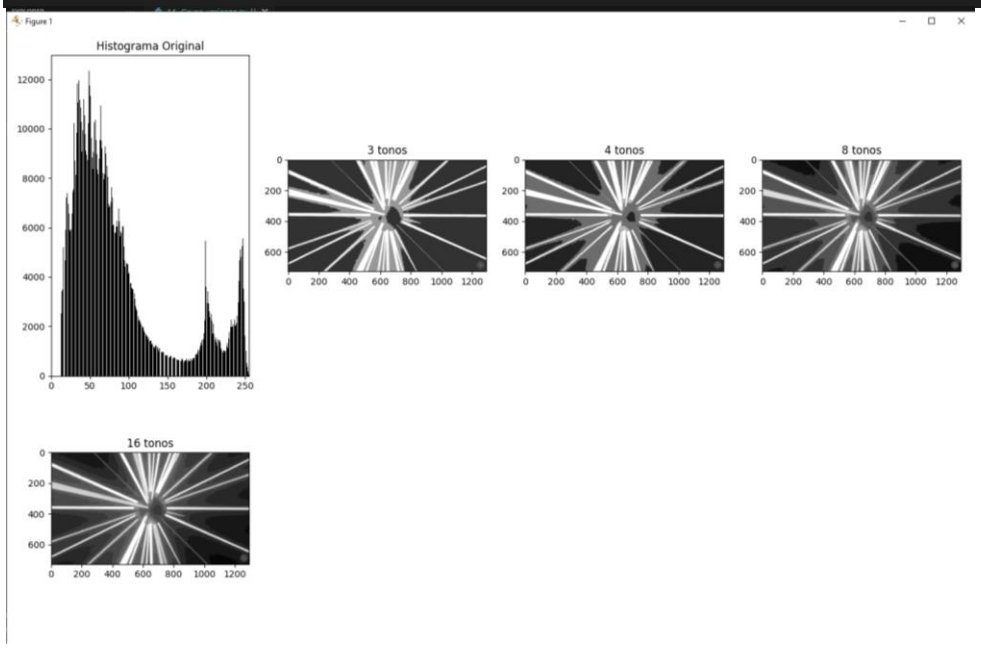


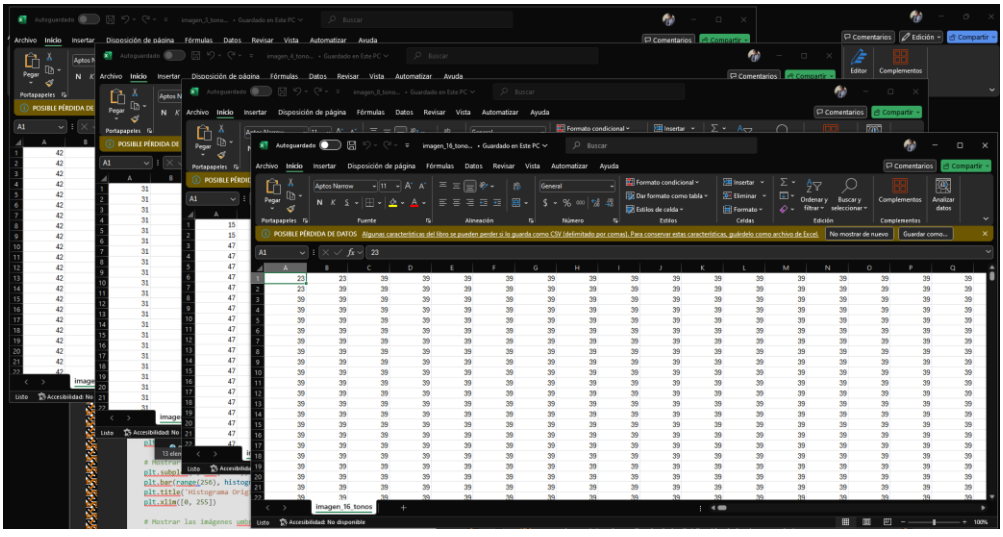
P04UmbralGrupoVarianza.py

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage import io

def calcular_histograma(imagen):
    # Calcula el histograma de la imagen en escala de grises
    histograma, _ = np.histogram(imagen.ravel(), bins=256, range=(0, 256))
    return histograma

def calcular_grupo_varianza(histograma, total_pixeles):
    # Cálculo de varianza entre los grupos
    suma_total = np.sum([i * histograma[i] for i in range(256)])
    suma_b = 0
    w_b = 0
    varianza_max = 0
    mejor_umbral = 0

    for umbral in range(256):
        w_b += histograma[umbral]
        w_f = total_pixeles - w_b
        if w_b == 0 or w_f == 0:
            continue

        suma_b += umbral * histograma[umbral]
        m_b = suma_b / w_b if w_b != 0 else 0
        m_f = (suma_total - suma_b) / w_f if w_f != 0 else 0

        varianza_entre_clases = w_b * w_f * (m_b - m_f) ** 2

        if varianza_entre_clases > varianza_max:
            varianza_max = varianza_entre_clases
            mejor_umbral = umbral

    return mejor_umbral

def aplicar_tonos(imagen, num_tonos):
    # Definir los umbrales
    max_val = 255
    umbrales = np.linspace(0, max_val, num_tonos + 1)
    imagen_tonos = np.zeros_like(imagen)

    # Asignar tonos basados en los umbrales definidos
    for i in range(num_tonos):
```

```
        lower_bound = umbrales[i]
        upper_bound = umbrales[i + 1]
        imagen_tonos[(imagen >= lower_bound) & (imagen < upper_bound)] =
int((lower_bound + upper_bound) / 2)

    return imagen_tonos

# Cargar la imagen
imagen_a_color = cv2.imread("imagen a color.png")
if imagen_a_color is None:
    print("La ruta de la imagen no es correcta, porfavor ingrese la ruta
adecuada")
else:
    # Convertrir la imagen a escala de grises
    imagen = cv2.cvtColor(imagen_a_color,cv2.COLOR_BGR2GRAY)

    # Calcular el histograma de la imagen original
    histograma = calcular_histograma(imagen)
    total_pixeles = imagen.size

    # Calcular el mejor umbral usando el método de grupo varianza
    mejor_umbral = calcular_grupo_varianza(histograma, total_pixeles)

    # Aplicar diferentes tonos (3, 4, 8, 16)
    tonos = [3, 4, 8, 16]
    imagenes_tonos = [aplicar_tonos(imagen, num_tonos) for num_tonos in
tonos]

    # Mostrar el histograma y las imágenes umbralizadas
    plt.figure(figsize=(15, 10))

    # Mostrar el histograma original
    plt.subplot(2, len(tonos), 1)
    plt.bar(range(256), histograma, color='black')
    plt.title('Histograma Original')
    plt.xlim([0, 255])

    # Mostrar las imágenes umbralizadas
    for idx, num_tonos in enumerate(tonos):
        plt.subplot(2, len(tonos), idx + 2)
        plt.imshow(imagenes_tonos[idx], cmap='gray')

        # Crear nombre de archivo dinámicamente usando f-string
        nombre_archivo = f"imagen_{num_tonos}_tonos.csv"
```

```
        np.savetxt(nombre_archivo, imagenes_tonos[idx], delimiter=',', fmt='%d')

        plt.title(f'{num_tonos} tonos')

    for num_tonos in tonos:
        imagen_tonos = aplicar_tonos(imagen, num_tonos)
        # Guardar la imagen umbralizada con diferentes tonos
        cv2.imwrite(f'imagen_{num_tonos}_tonos.png', imagen_tonos)

    # Mostrar las imágenes y el histograma
    plt.tight_layout()
    plt.show()
```



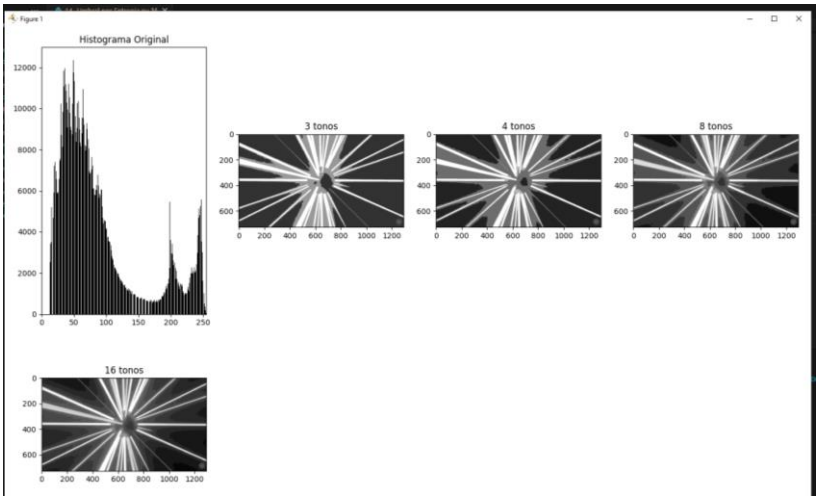


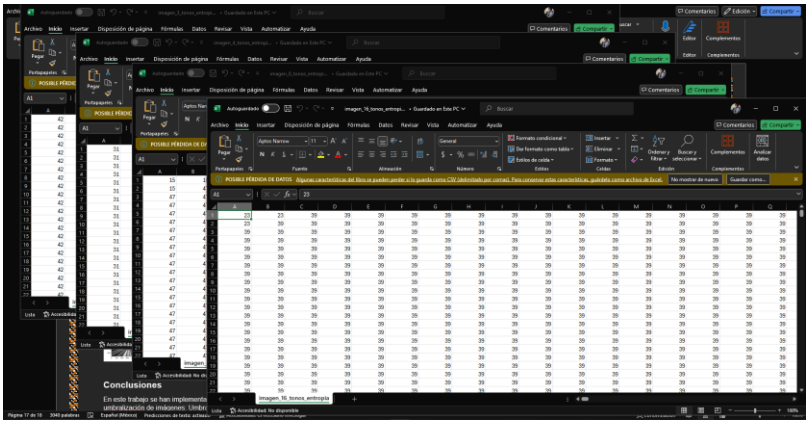
P04UmbralPorEntropia.py

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage import io

def calcular_histograma(imagen):
    # Calcula el histograma de la imagen en escala de grises
    histograma, _ = np.histogram(imagen.ravel(), bins=256, range=(0, 256))
    return histograma

def calcular_entropia(histograma, total_pixeles):
    # Cálculo de la entropía
    probabilidad = histograma / total_pixeles
    probabilidad = probabilidad[probabilidad > 0]  # Filtrar ceros
    entropia = -np.sum(probabilidad * np.log2(probabilidad))
    return entropia

def encontrar_mejor_umbral_entropia(imagen):
    # Calcular el histograma de la imagen
    histograma = calcular_histograma(imagen)
    total_pixeles = imagen.size

    # Variables para encontrar el mejor umbral
    mejor_entropia = 0
    mejor_umbral = 0

    # Calcular la entropía para cada posible umbral
    for umbral in range(256):
        hist_bajo = histograma[:umbral]
        hist_alto = histograma[umbral:]

        # Cálculo de la entropía para las dos clases
        entropia_bajo = calcular_entropia(hist_bajo, total_pixeles)
        entropia_alto = calcular_entropia(hist_alto, total_pixeles)

        # Entropía total
        entropia_total = entropia_bajo + entropia_alto

        if entropia_total > mejor_entropia:
            mejor_entropia = entropia_total
            mejor_umbral = umbral

    return mejor_umbral
```

```
def aplicar_tonos(imagen, num_tonos):
    # Definir los umbrales
    max_val = 255
    umbrales = np.linspace(0, max_val, num_tonos + 1)
    imagen_tonos = np.zeros_like(imagen)

    # Asignar tonos basados en los umbrales definidos
    for i in range(num_tonos):
        lower_bound = umbrales[i]
        upper_bound = umbrales[i + 1]
        imagen_tonos[(imagen >= lower_bound) & (imagen < upper_bound)] =
int((lower_bound + upper_bound) / 2)

    return imagen_tonos

# Cargar la imagen
imagen_a_color = cv2.imread("imagen a color.png")
if imagen_a_color is None:
    print("La imagen no se cargo correctamente, favor de verificar su ruta")
else:
    imagen = cv2.cvtColor(imagen_a_color,cv2.COLOR_BGR2GRAY)

    # Calcular el histograma de la imagen original
    histograma = calcular_histograma(imagen)

    # Calcular el mejor umbral usando el método de entropía
    mejor_umbral = encontrar_mejor_umbral_entropia(imagen)

    # Aplicar diferentes tonos (3, 4, 8, 16)
    tonos = [3, 4, 8, 16]
    imagenes_tonos = [aplicar_tonos(imagen, num_tonos) for num_tonos in
tonos]

    # Mostrar el histograma y las imágenes umbralizadas
    plt.figure(figsize=(15, 10))

    # Mostrar el histograma original
    plt.subplot(2, len(tonos), 1)
    plt.bar(range(256), histograma, color='black')
    plt.title('Histograma Original')
    plt.xlim([0, 255])

    # Mostrar las imágenes umbralizadas y guardar las matrices
    for idx, num_tonos in enumerate(tonos):
        plt.subplot(2, len(tonos), idx + 2)
```

```
        plt.imshow(imagenes_tonos[idx], cmap='gray')

        # Guardar la matriz de cada imagen en un archivo CSV
        nombre_archivo = f"imagen_{num_tonos}_tonos_entropia.csv"
        np.savetxt(nombre_archivo, imagenes_tonos[idx], delimiter=',',
fmt='%d')

        plt.title(f'{num_tonos} tonos')

    for num_tonos in tonos:
        imagen_tonos = aplicar_tonos(imagen, num_tonos)
        # Guardar la imagen umbralizada con diferentes tonos
        cv2.imwrite(f'imagen_{num_tonos}_tonos.png', imagen_tonos)

    # Mostrar las imágenes y el histograma
    plt.tight_layout()
    plt.show()
```





Conclusiones

En este trabajo se han implementado y comparado tres métodos automáticos de umbralización de imágenes: Umbral por Entropía, Grupo de Varianza y Aproximación de Valle Global. Cada uno de estos métodos aborda la segmentación de imágenes desde diferentes enfoques matemáticos y estadísticos, lo que nos ha permitido observar cómo cada técnica tiene ventajas y desventajas dependiendo de la distribución de los tonos en la imagen.

En conjunto, estos tres métodos de umbralización ofrecen una gama de opciones para la segmentación automática de imágenes, permitiendo seleccionar el enfoque más adecuado según las características particulares de cada imagen. Los resultados obtenidos, tanto en la visualización de las imágenes umbralizadas como en los histogramas y matrices exportadas, permiten una evaluación cuantitativa y cualitativa que puede ser extendida a otros contextos de procesamiento de imágenes.

Bibliografía

OpenCV modules. OpenCV. (n.d.). https://docs.opencv.org/4.x/index.html

NumPy Org. (2024, June 17). NumPy. https://numpy.org/

Matplotlib 3.5.3 documentation#. Matplotlib documentation - Matplotlib 3.5.3 documentation. (2012). https://matplotlib.org/3.5.3/index.html