Universidad Veracruzana

Maestría en Inteligencia Artificial
Visión por computadora

Tarea 3
Mejora de imágenes mediante ecualización de histograma

Ulises Jiménez Guerrero

6 de marzo de 2025

Reporte tarea 3

1. Objetivos

- Crear un programa de computadora que genere el histograma de una imagen de entrada.
- Con base al programa anterior, mejorar la calidad de la imagen mediante el proceso de ecualización de histograma.
- Evaluar el desempeño de los programas desarrollados.

2. Metodología

2.1. Materiales utilizados

Se utilizó el lenguaje de programación Python, versión 3.12.8. Para el manejo de las imágenes se utilizó la paquetería de código libre opencv para Python, junto a la paquetería Numpy para el manejo de las matrices generadas por las imágenes. Finalmente, se utilizó la paquetería de Matplotlib para el despliegue de imágenes en la pantalla y para guardar los resultados. A su vez, esta librería instala varias dependencias. Todas las paqueterías junto a las versiones utilizadas se encuentran listadas en el anexo.

Para la comprobación del algoritmo, se utilizaron las imágenes 3.15 (1, 2, 3) del libro [2] . Estas imágenes se pueden observar en la figura 1 . Todas tienen un tamaño de 500×500 píxeles, y se utilizó una escala de grises con 8 bits.

(a) Imagen 3.15(1) (b) Imagen 3.15(2) (c) Imagen 3.15(3)

Figura 1: Imágenes utilizadas para evaluar el programa desarrollado.

2.2. Algoritmos e implementación

2.2.1. Histograma de una imagen

El histograma de una imagen es el conteo del total de píxeles de cierta intensidad presentes en dicha imagen. En nuestro caso, dado que se maneja una escala de grises con 8 bits, la

intensidad toma valores entre 0 y 255. Además, dado que todas las imágenes de prueba tienen un tamaño de 500×500, el total de píxeles será de $250,000$. En las aplicaciones de visión por computadora resulta de más interés el histograma normalizado, donde se divide el conteo de cada valor de intensidad entre el total de píxeles, de forma que estos valores se restringen en un valor entre 0 y 1. Dado que las imágenes solo toman valores discretos, esto es una aproximación a la *probabilidad* de que cierta intensidad aparezca en la imagen.

Para el desarrollo del histograma, se utilizó *opencv* para leer la imagen en escala de grises como una matriz de 500×500. Esta matriz se aplanó a un vector de tamaño $1 \times 25,000$ para simplificar cálculos. Se creó otro vector de ceros con índices de 0 a 255, de tal forma que cada índice se corresponde con cierta intensidad. Este es el utilizado para generar el histograma. Posteriormente, se recorrió el vector de la imagen aplanada, aumentando en 1 el valor del histograma en el índice correspondiente a la intensidad del píxel. Al final de este proceso, se tiene que en el vector histograma el valor correspondiente a cada índice indica el total de píxeles con dicha intensidad en la imagen.

Este programa se implementó como una función de Python que toma como entrada la matriz imagen. Cabe destacar que las paqueterías instaladas incluyen funciones que realizan este proceso, pero se decidió implementarlo desde cero por el valor didáctico.

2.2.2. Ecualización de histograma

Para cada imagen, su histograma correspondiente nos da información acerca de la distribución de tonos. Imágenes oscuras tendrán histogramas centrados a la izquierda del rango de intensidades, imágenes claras tendrán histogramas centrados a la derecha, e imágenes con poco contraste tendrán histogramas ubicados alrededor del centro de este rango. Para la mejora de la calidad de una imagen, buscamos aumentar su rango dinámico, de manera que utilice todas las intensidades disponibles. Esto se traduce en un histograma distribuido de manera uniforme.

Por lo tanto, utilizaremos transformaciones de histogramas para la mejora de la calidad de imágenes. En especifico, se utilizará aquella dada por [2]

$$s_k = T(r_k) = (L-1)\sum_{j=0}^{k} p_r(r_j) \quad K = 0, \ldots, L-1$$

donde s_k indica el valor del píxel resultante, r_k el valor del píxel original, $p_r(r_j)$ la probabilidad de encontrar la intensidad r_j en la imagen, y L es el número total de intensidades posibles. Notemos que, como las imágenes digitales están dadas en valores discretos, la probabilidad de una intensidad dada se puede encontrar con la fórmula

$$p_r(r_k) = \frac{n_k}{\text{Total de pixeles}}$$

donde n_k indica cuántos píxeles en la imagen tienen la intensidad especificada. Como se mencionó anteriormente, esta información se encuentra de forma explícita en el histograma normalizado. De esta forma, la transformación mostrada se puede aplicar de manera directa teniendo la imagen original y su histograma normalizado.

La función es la *distribución de probabilidad acumulativa discreta.* Resulta de interés dado que, como se demuestra en [2] , su equivalente continuo da como resultado una imagen

donde las intensidades tienen una probabilidad uniforme, de forma que su histograma es plano, abarcando todo el rango de tonos. Esto no se cumple para el caso discreto, pero sí se logra que el histograma abarque todo el rango de intensidades de tal forma que se mejora el contraste de la imagen en una *gran cantidad de casos*. Como es una transformación global, presenta malos resultados en imágenes donde las tonalidades cambian de forma drástica en diferentes secciones.

Para obtener la probabilidad acumulativa, simplemente debemos realizar una suma acumulativa sobre nuestro histograma normalizado. Aunque esto se encuentra implementado en la paquetería de *numpy*, se implemento desde cero. El resultado de esta suma se escala multiplicándolo por 255, y finalmente se realiza un mapeo sobre la imagen original para obtener la imagen *ecualizada*. Para ello, se utilizan el indexado avanzado de *numpy* sobre esta distribución, guardada como un *array*. Se utiliza como arreglo de índices la imagen original aplanada, de forma que los valores originales de cada píxel llevan a los valores escalados indicados por la distribución acumulativa. Al array resultante se le dan las dimensiones de la imagen original, obteniendo la imagen ecualizada.

2.3. Mejora de las imágenes

Para cada una de las imágenes de prueba se aplicó el siguiente proceso, de acuerdo al script incluido en el anexo:

1. Lectura de la imagen como matriz utilizando *opencv*. A su vez, se crea una copia aplanada de la matriz y se guardan las dimensiones originales.

2. Obtención del histograma de la imagen original utilizando la función desarrollada. Este se normaliza dividiendo entre el total de píxeles de la imagen.

3. Calculo de la distribución de probabilidad acumulada mediante una suma acumulativa sobre el histograma normalizado. Esta se escala multiplicando por el rango dinámico, 255.

4. Creación de la nueva imagen ecualizada mediante la probabilidad acumulada escalada. Esta se crea primero como una matriz aplanada, y después se le dan las dimensiones de la imagen original.

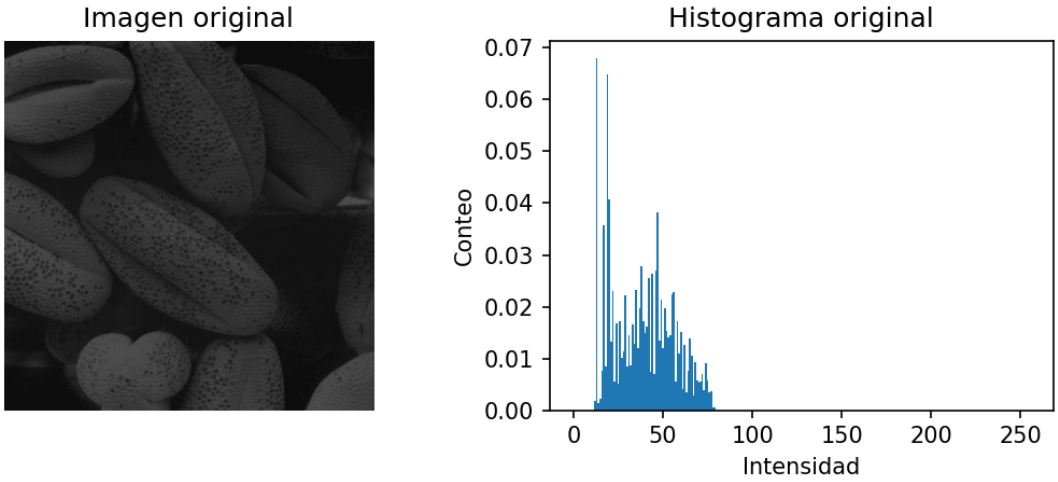
5. Se grafica la imagen original junto a su histograma con la función desarrollada para ello. Se hace lo mismo con la imagen ecualizada.

6. Se grafica la función de ecualización, esto es, la distribución de probabilidad escalada.

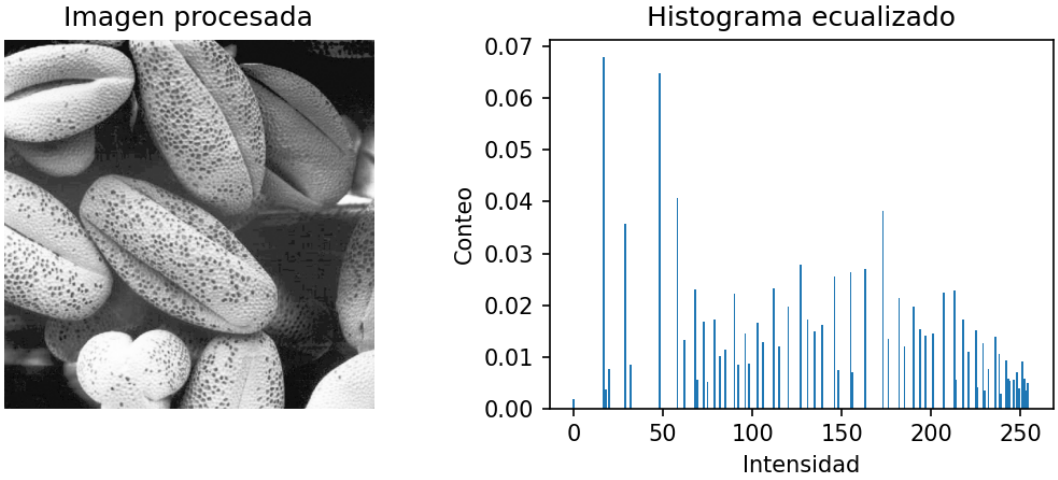
Todas las operaciones de matrices se realizaron con *numpy*, y las gráficas se obtuvieron mediante *matplotlib*. Se utilizaron de consulta las fuentes [1], [3] y [4].

3. Resultados

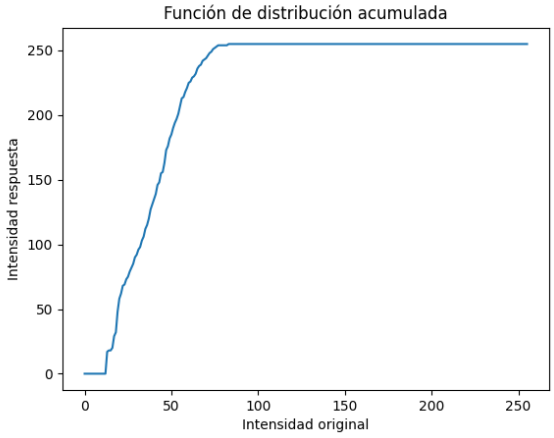
Se muestran las imágenes originales, los resultados de la ecualización, los histogramas correspondientes a cada imagen y su función de distribución escalada en las figuras 2, 3, 4.



(a) Imagen original y su histograma

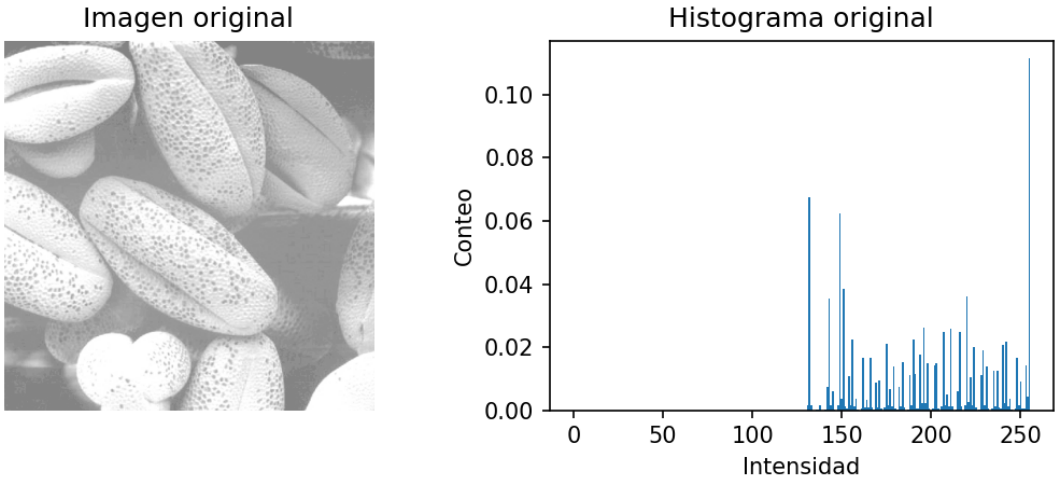


(b) Imagen ecualizada y su histograma

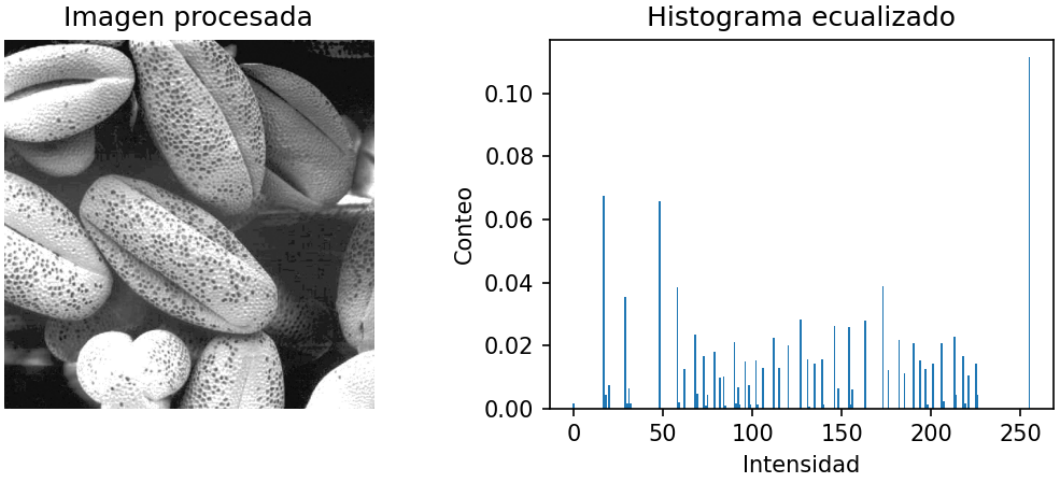


(c) Función de transformación del histograma

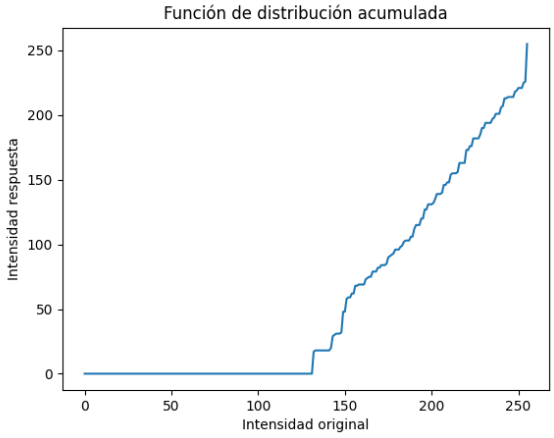
Figura 2: Resultados para la imagen 3.15(1).



(a) Imagen original y su histograma

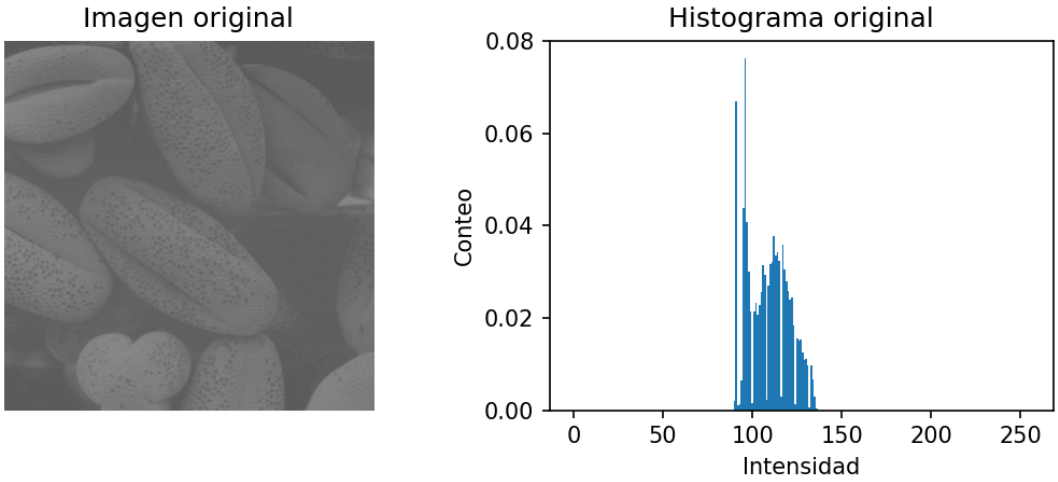


(b) Imagen ecualizada y su histograma

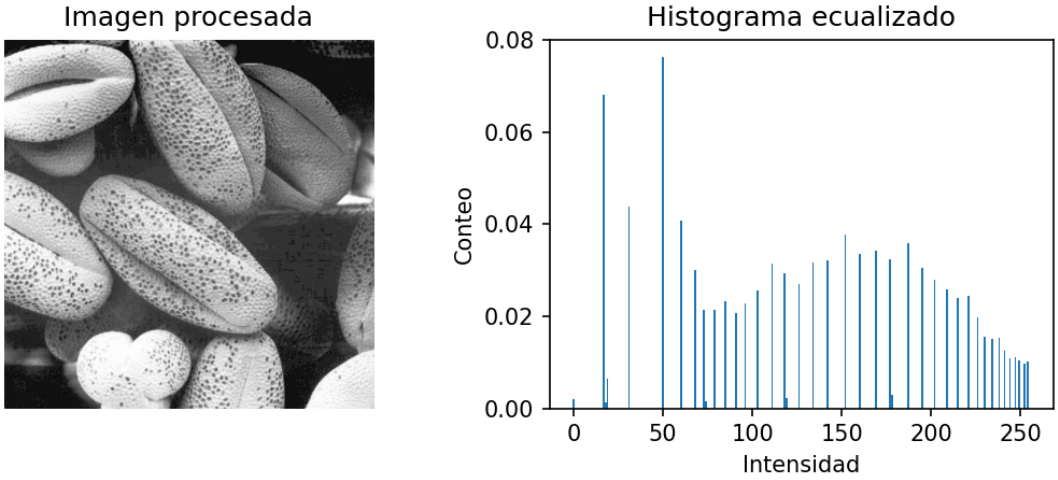


(c) Función de transformación del histograma

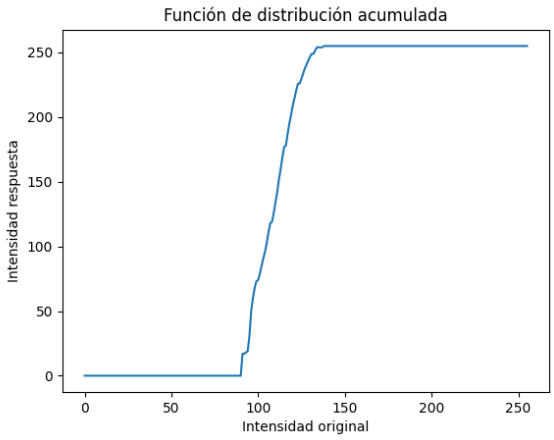
Figura 3: Resultados para la imagen 3.15(2).



(a) Imagen original y su histograma



(b) Imagen ecualizada y su histograma



(c) Función de transformación del histograma

Figura 4: Resultados para la imagen 3.15(3).

A pesar de tener histogramas diferentes, las tres imágenes resultantes son extremadamente similares, sin diferencias perceptibles a simple vista. Esto es de esperarse, dado que todas toman como base la misma imagen, con diferentes contrastes. Al eliminar las deficiencias causadas por estos contrastes extremos, obtenemos la imagen base. Todos los histogramas resultantes comparten el que se abarca todo el rango dinámico, desde el 0 hasta el 255, y el que una gran cantidad de los valores intermedios se encuentran vacíos. Esto es evidente al observar las funciones de transformación, donde todas presentas zonas planas que envían todo un rango de valores a la misma respuesta.

4. Conclusiones

Para las imágenes de prueba seleccionadas, la ecualización de histograma tuvo resultados excepcionales, permitiendo una gran mejora de los detalles observables en la imagen, aumentando el rango dinámico y en general mejorando la calidad de imagen. Sin embargo, esto funciona dado que todas presentan un tono similar de manera global: toda la imagen 3.15(1) es oscura, toda la imagen 3.15(2) es clara, y toda la imagen 3.15(3) presenta un tono apagado. Si estas tuvieran regiones muy marcadas dentro de la misma imagen, esto es, una zona oscura contrastada con una zona muy clara, el filtro sería incapaz de manejarlo de manera adecuada. Esto se puede resolver aplicando la ecualización por zonas sobre la imagen, o utilizando otros métodos de histograma más sofisticados. Aunque en este trabajo se limitó su uso a imágenes en escala de gris, la expansión a imágenes RGB se da de manera natural, aplicando el mismo proceso a cada uno de los canales de color.

Referencias

[1] solem's vision blog: Histogram equalization with Python and NumPy. https://web.archive.org/web/20151219221513/http://www.janeriksolem.net/2009/06/histogram-equalization-with-python-and.html, December 2015.

[2] Rafael C. Gonzalez and Richard E. Woods. *Digital image processing.* Pearson, New York, fourth edition, global edition edition, 2017.

[3] Animesh Karnewar. Back-to-basics Part 1: Histogram Equalization in Image Processing. https://medium.com/@animeshsk3/back-to-basics-part-1-histogram-equalization-in-image-processing-f607f33c5d55, October 2018.

[4] Tory Walker. Histogram Equalization in Python from Scratch. https://medium.com/hackernoon/histogram-equalization-in-python-from-scratch-ebb9c8aa3f23, April 2023.

5. Anexo

5.1. Lista de paqueterías y versiones

- contourpy==1.3.1
- cycler==0.12.1
- fonttools==4.56.0
- kiwisolver==1.4.8
- matplotlib==3.10.0
- numpy==2.2.3
- opencv-python==4.11.0.86
- packaging==24.2
- pillow==11.1.0
- pyparsing==3.2.1
- python-dateutil==2.9.0.post0
- six==1.17.0

5.2. Código

```
import numpy as np
import cv2
import matplotlib.pyplot as plt

# Reading the files and asserting a correct path
img = cv2.imread('reporte/imgs/Fig3.15(a)3.jpg', 0)
assert img is not None, "file could not be read, check path"

# Getting the dimensions of the image and creating a flat
# copy for efficiency
width, height = img.shape
img_flat = img.flatten()
# Function for creating the histogram. We try not to use
# the predefined functions
def generate_hist(img, int_levels=256):
    img_flat = img.flatten()
    # Creating an array of zeros for the histogram. Each
    # index corresponds to an intensity value, going from
    # 0 to 255
    hist = np.zeros(256, dtype=int)

    # We iterate over every pixel in the image, and when
```

```
    # we encounter a specific intensity value we augment
    # its counter in the histogram by 1
    for pixel in img_flat:
        hist[pixel] += 1

    return hist

# Function for plotting an image next to its histogram
def plot_hist(img, hist, img_title:str, hist_title:str, file_name:str =
    None):
    fig, (ax1, ax2) = plt.subplots(1,2, layout='constrained')
    # Indicating the colormap and the correct range for the intensity
       values
    ax1.imshow(img, cmap='gray', vmin=0, vmax=255)
    ax1.set_title(img_title)
    ax1.axis('off')

    ax2.bar(np.arange(0,256,1), hist, width=1)
    ax2.set_title(hist_title)
    ax2.set_xlabel('Intensidad')
    ax2.set_ylabel('Conteo')
    fig.set_size_inches(7, 3, forward=True)
    # Given a file name, it saves the resulting image
    if file_name:
        fig.savefig(file_name, dpi=150)
    return

# Generating an histogram from the image
hist = generate_hist(img)
# Normalizing the histogram dividing by the total amount
# of pixels in the image
hist_norm = hist/(width*height)

# Creating an empty array for the cumulative probability function.
# Same size as the histogram
cpf = np.zeros(len(hist))
# We make the first value equal to the normalized histogram
cpf[0] = hist_norm[0]

# We calculate the cumulative sum iterating over the histogram
# and adding the value at the current index to the previous sum
for i in range(1, len(hist_norm)):
    cpf[i] = hist_norm[i] + cpf[i-1]

# We scale the cpf to 0-255, so that it gives us the required
# intensity values
scaled_cpf = cpf*255
# Correct typing for image plotting
scaled_cpf = scaled_cpf.astype(np.uint8)

# We create the new image as a flat array. Using the original
# flat image and advanced indexing, we map the original value
# to the equalized one, following the cpf
# For example, if the original pixel value is 0, it will look
```

```
# at the corresponding index in the cpf and assign the equalized
# value
img_new = scaled_cpf[img_flat]
# Correcting the shape of the new image
img_new = np.reshape(img_new, img.shape)

# Generating an histogram for the equalized image
new_hist = generate_hist(img_new)
new_hist_norm = new_hist / (width*height)

# Plotting results
plot_hist(img, hist_norm, 'Imagen␣original', 'Histograma␣original')
plot_hist(img_new, new_hist_norm, 'Imagen␣procesada', 'Histograma␣
    ecualizado')

fig, ax = plt.subplots()
ax.plot(scaled_cpf)
ax.set_title('Funcion␣de␣distribucion␣acumulada')
ax.set_xlabel('Intensidad␣original')
ax.set_ylabel('Intensidad␣respuesta')
#fig.savefig('cdf3.png')

plt.show()
```