

Práctica 0: Manejo Básico de Imágenes.

Serapio Hernández Alexis Arturo

Facultad de Ingeniería
UNAM

Ciudad de México, México

alexis.serapio@ingenieria.unam.edu

García López Erik
Facultad de Ingeniería
UNAM

Ciudad de México, México
erikpumas999@gmail.com

Reyes Herrera Rogelio

Facultad de Ingeniería
UNAM

Ciudad de México, México

masterfive5of@gmail.com

Abstract—This article is about the making of a practical document at the Faculty of Engineering at UNAM and our first approach to the Basic Image Processing using the programming language “Python” for these given exercises.

Keywords—Basic Images Processing, Python, Pattern, Pattern Recognition.c

I. INTRODUCCIÓN (HEADING 1)

El manejo básico de imágenes en el ámbito de la informática y la inteligencia artificial implica técnicas y herramientas que permiten procesar, analizar y manipular imágenes digitales. Un área clave dentro del campo del reconocimiento de patrones. Este se refiere a la capacidad de un sistema para identificar y clasificar objetos y características en una imagen basándose en patrones predefinidos.[1]

Algunos conceptos generales para abordar la práctica son:

Imagen Digital: Una imagen digital es una representación visual que se almacena como una matriz de píxeles, donde cada píxel tiene un valor que representa su color o su intensidad.

Procesamiento de Imágenes: Este proceso incluye operaciones básicas como la transformación de formatos de imagen, la mejora del contraste, el filtrado, la segmentación y la detección de bordes.

Reconocimiento de Patrones: El reconocimiento de patrones tiene aplicaciones en muchas áreas, como la visión computacional, la robótica, la biometría y más. Por ejemplo, se usa en el reconocimiento facial, en la detección de objetos en imágenes y en la clasificación de imágenes médicas.

El reconocimiento de patrones es esencial en el manejo de imágenes, ya que permite a los sistemas no solo ver, sino también comprender y reaccionar a la información visual. Con el avance de las técnicas de inteligencia artificial, esta área sigue creciendo en complejidad y aplicaciones.

Una de estas técnicas para el reconocimiento de imágenes es el **uso de un filtro paso bajas**.

Un filtro pasabajas aplicado a una imagen es una técnica utilizada en el procesamiento de imágenes para suavizar, lo que implica reducir la cantidad de detalles y eliminar o atenuar las altas frecuencias espaciales. Las altas frecuencias se encuentran asociadas con cambios bruscos de intensidad, como los bordes y detalles finos de la imagen.

Sus efectos principales que tiene este tipo de filtros son:

Suavizado

Elimina el ruido y las fluctuaciones rápidas en la imagen, haciéndola parecer más suave. Esto es útil, por ejemplo, en la reducción del ruido de las imágenes

Difuminado

Los detalles finos, como los bordes, se atenúan, lo que hace que la imagen parezca más borrosa.

Promediado de píxeles

En la práctica, el filtro reemplaza cada píxel de la imagen con un promedio ponderado de los píxeles vecinos.

Un ejemplo común de filtro paso bajas es un filtro Gaussiano.

Una distribución gaussiana con desviación típica σ y media μ viene dada por:

$$g_{\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

La convolución de la función anterior con una señal $f(x)$ da lugar a una nueva señal suavizada $h(x)$, donde el valor de cada punto del resultado de promediar con distintos pesos los valores vecinos a ambos lados de dicho punto. En este suavizado, la desviación típica juega un papel importante a la hora de controlar el grado de suavizado de este operador.[2]

Cuanto mayor sea, más se tienen en cuenta los puntos lejanos, y por tanto, mayor será el suavizado resultante.

Extendiendo la función anterior a dos dimensiones tenemos:

$$g_{\sigma}(x) = \frac{1}{2\pi\sigma^2} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Donde además, se ha supuesto que la campana de Gauss se centra sobre la media y tiene simetría de revolución.

Teniendo como resultado:



a)



b)

a) Imagen de Lena en escala de grises, b) Imagen de Lena con el filtro de gauss aplicado.

De igual manera existen otros tipos de filtro paso bajas para lograr efectos similares, como la Media ponderada, la Moda y el K Nearest.

II. DESARROLLO PRÁCTICA 0

A.- Desarrolla un script para leer y desplegar cada imagen con los paquetes de Matplotlib, OpenCV, Scikit-Image y PIL.

Este código carga y muestra una imagen usando cuatro diferentes bibliotecas de Python (Matplotlib, OpenCV, Scikit-Image y PIL). Luego, organiza las imágenes en una cuadrícula de 2x2, mostrando cómo cada biblioteca maneja y visualiza la misma imagen.

Es importante remarcar que la biblioteca que menos nos proporcionó errores fue “CV2” ya que este nos permitió desplegar completamente todas las imágenes, incluso antes de que necesitaran ser modificadas por la profesora.



Lena Matplot



Lena CV2



Lena SciKit



Lena PIL

Para observar el código del programa, ver en el apéndice I.

B.- Imprimir el tipo de imagen, el tamaño y el tipo de dato.

Se eligió el módulo Scikit-Image y Matplotlib ya que en el tutorial se usaron estas bibliotecas para la lectura de las imágenes. Explicado brevemente, usando el módulo os para agregar a una lista todo el directorio con nuestras imágenes de la siguiente forma:

```
for file_name in  
sorted(os.listdir(images_directory)):  
    file_path =  
os.path.join(images_directory,  
file_name)
```

Se itera sobre el directorio y detecta si hay .tif, .tiff, dcm y los agrega a una lista, la lectura es con:

```
image = io.imread(file_path)
```

Además, se crearon listas para ir agregando el tipo de imagen, su nombre y tamaño.

Por ejemplo:



Es importante observar que tenemos definida una función que compara el número de dimensiones para saber el modo de impresión, si es a color o en escala de grises.

```
if len(img.shape) == 2:
```

Para el desplazamiento únicamente se identifican eventos del teclado que serían la flecha izquierda y derecha variando los índices de nuestra lista de imágenes.

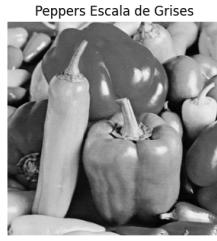
Para observar el código del programa, ver en el apéndice II.

C.- De las imágenes

“lena_color_512.tif”, “peppers_color.tif”. Desarrolla un script con OpenCV y Scikit-Image para cambiar el espacio de color de:

1. RGB a Escala de grises

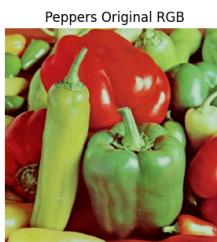
Primero se cargaron las imágenes utilizando OpenCV, se cargan por defecto en BGR por lo que se convierten a RGB para ahora si convertirlas a escala de grises con ayuda de la función COLOR_RGB2GRAY.



2. RGB a YUV

Primero se cargaron las imágenes utilizando OpenCV, se cargan por defecto en BGR por lo que se convierten a RGB para ahora si convertirlas a YUV con ayuda de la función COLOR_RGB2YUV.

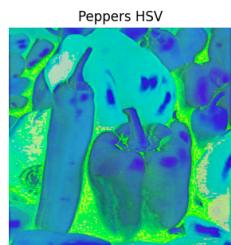
YUV es un espacio de color que separa la imagen en luminancia (**Y**) y crominancia (**U** y **V**), donde **Y** maneja el brillo y **U/V** la información de color. Es común en video porque permite optimizar el almacenamiento reduciendo detalles de color sin perder calidad visual significativa.



3. RGB a HSV

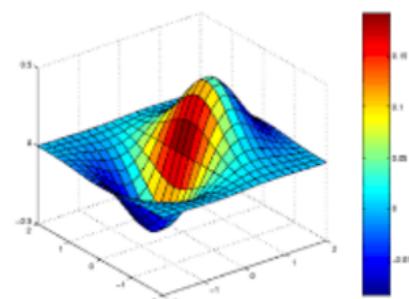
Primero se cargaron las imágenes utilizando OpenCV, se cargan por defecto en BGR por lo que se convierten a RGB para ahora si convertirlas a HSV con ayuda de la función COLOR_RGB2HSV.

HSV es un espacio de color que describe los colores mediante tono (**H**), saturación (**S**) y valor (**V**). Es útil en el procesamiento de imágenes porque permite manipular el color de manera más intuitiva, facilitando tareas como la selección y segmentación de colores.



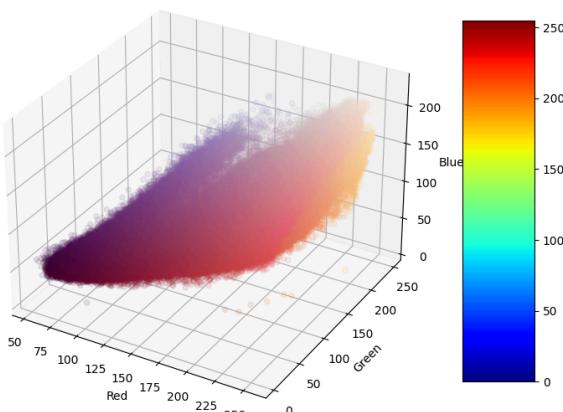
Para observar el código del programa, ver en el apéndice III.

D.- Despliega la paleta de colores de RGB por separado, ver figura siguiente, la barra de la derecha con valores es la paleta de colores.

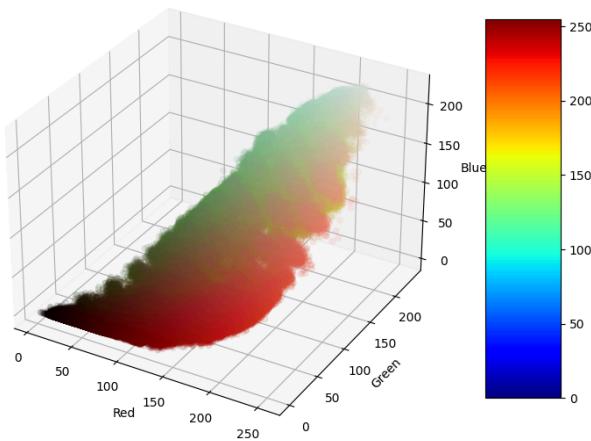


Para lograr esto se convierte la imagen a un arreglo de numpy que contiene los valores RGB de cada píxel, después los componentes rojo (R), verde (G) y azul (B) se extraen de la matriz y se apllanan en vectores 1D y finalmente se utiliza scatter para graficar cada píxel en el espacio 3D de RGB, coloreando cada punto con su color correspondiente. La transparencia de los puntos se ajusta con alpha = 0.1 para facilitar la visualización.

Resultados lena_color:



Resultados peppers_color:



Para observar el código del programa, ver en el apéndice IV.

E.- De una imagen que usted escoja, dejarla en escala de grises y procure que sea igual en renglones y en columnas. Programe una función que realice decimación de una imagen, reduciéndola a la mitad de su tamaño original. Y promediando en grupos de 4 píxeles. Pruebe con su imagen.

Nuestro código se asegura de que la imagen sea cuadrada y aplica un proceso de decimación que reduce su tamaño a la mitad. La decimación se realiza promediando bloques de 4 píxeles adyacentes (2×2) y asignando ese valor promedio a la posición correspondiente en la nueva imagen de menor resolución. Finalmente, se muestran tanto la imagen original como la imagen decimada utilizando matplotlib.

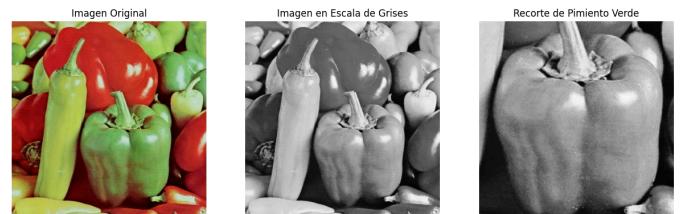


Convierte la imagen peppers_color.tif a escala de grises:

1. Recortela de manera que solo quede uno de los pimientos verdes en ese recorte
2. Guárdela en formato jpg.

Primero se leyó la imagen con cv2, se convirtió a escala de grises como ya se había visto. Se usan las variables x & y para las coordenadas, así como w & h para el tamaño del recorte, se jugaron con estos valores para lograr que solo saliera un pimiento verde.

Finalmente se guardó la imagen en formato jpg con cv2.imwrite

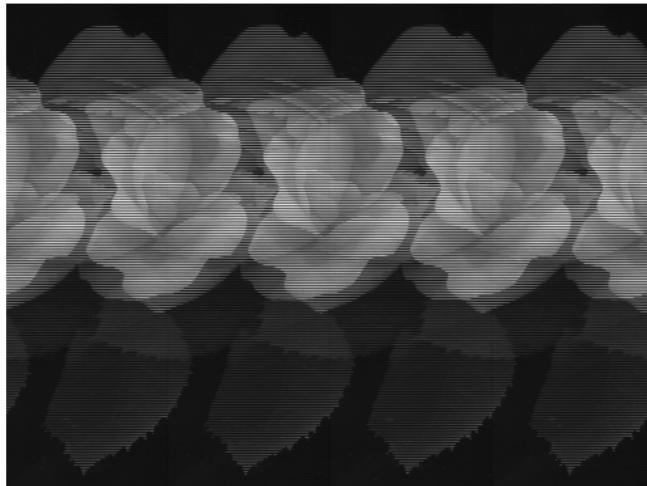


Para observar el código del programa, ver en el apéndice V.

F.- Un formato de imágenes sin ningún tipo de codificación se conoce como formato crudo (RAW). De la imagen "rosa800x600.raw" lea y despliegue la imagen. Tome en cuenta que esta imagen maneja la precisión de integer8 y el tamaño es de 600x800 píxeles.

Dado que .raw es un formato crudo y sin compresión se empezó definiendo el ancho, la altura (800x600) y el tipo de dato (np.uint8) de la imagen, indicando que cada píxel se representa con un entero de 8 bits, abrimos el archivo RAW en modo binario , leemos los datos en un arreglo. Después, reorganiza estos datos en una matriz 2D para representar la imagen y la despliega en escala de grises usando matplotlib.

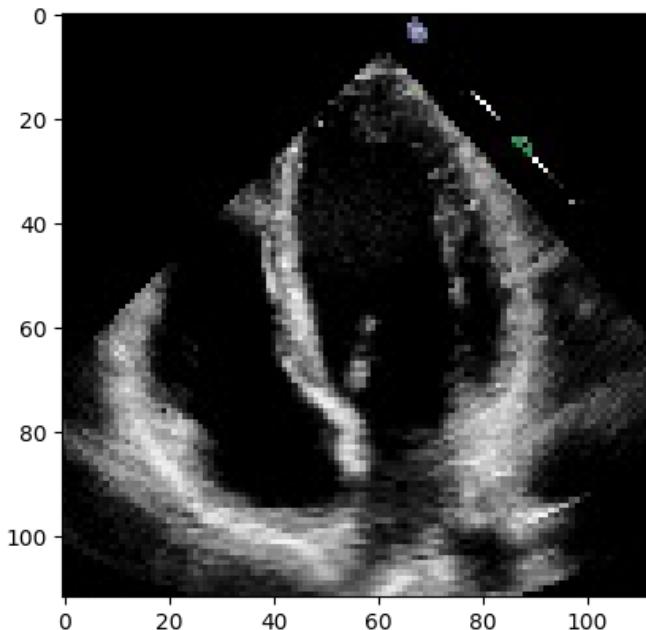
Imagen RAW: rosa800x600



Para observar el código del programa, ver en el apéndice VI.

G.- *Lectura de Video archivo “.avi”. Probar el siguiente código y muestre el video “0X2A8498756C4D6E82_corto.avi”. Si existe un error corregirlo.*

Se usa cv2.VideoCapture para abrir y leer el archivo de video, se lee el primer frame del video. ret es un valor booleano que indica si la lectura fue exitosa, y frame contiene la imagen del primer frame. Finalmente se muestra el primer frame utilizando matplotlib



Para observar el código del programa, ver en el apéndice VII.

H.- Imagen DICOM

Mostrar la información del encabezado de los formatos, investigar cómo se despliega.

1. Usar imagen DICOM:

Realiza un script que lea las imágenes de la carpeta DICOM y muestra en una misma ventana las 280 imágenes de tal manera que se pueda simular el movimiento del corazón. Utiliza ciclos for y el comando pause, para leer la ruta completa de la carpeta.

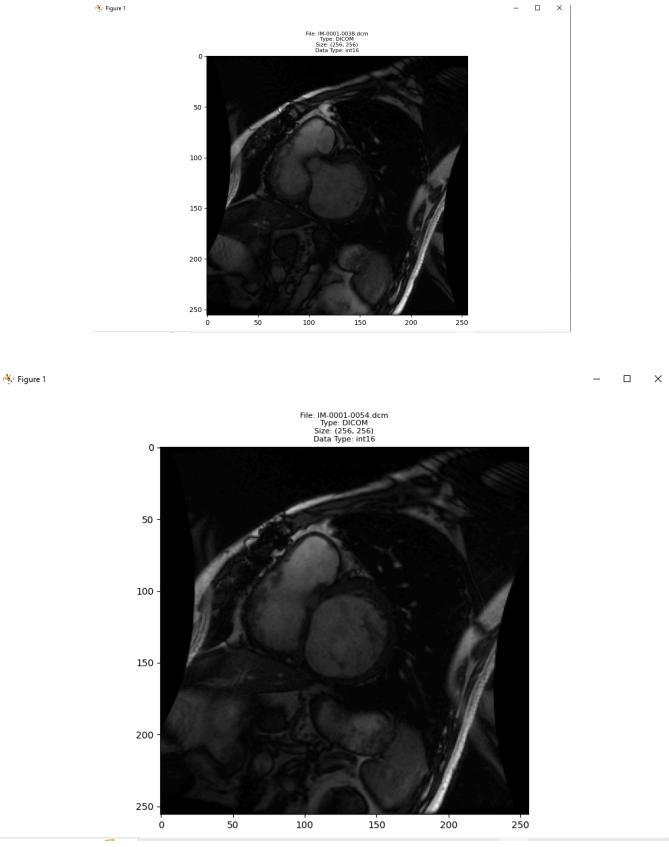
Para esta parte, se usó un programa similar al de la actividad 4.1 solo que con la biblioteca imageio, en la actividad 4.1 se leía un dcm con el modulo pydicom

```
dicom_image = pydicom.dcmread(file_path)
```

En este caso no es necesaria ninguna biblioteca adicional:

```
image = read_image(file_path)
images.append(image)
file_names.append(file_name)
image_types.append(
    'TIFF' if
file_name.lower().endswith('.tif',
'.tiff') else
    'RAW' if
file_name.lower().endswith('.raw') else
    'DICOM' if
file_name.lower().endswith('.dcm') else
    'Unknown'
)
image_shapes.append(image.shape)
image_dtypes.append(image.dtype)
```

Solo le agregan las imágenes y sus datos a las listas, teniendo como resultado la secuencia del corazón:



Por último para obtener los encabezados de cada imagen se usa `dicom_data = pydicom.dcmread(file_path)` y se agregan a una lista nueva, aquí cada que se actualice la imagen en el plot, nos imprime en la consola de salida el encabezado de dicha imagen, no se anexa al reporte ya que son muy extensos. Pero con fines ilustrativos se muestra un fragmento de la imagen 1.

```
(0043, 1006) [Number of EPI shots] SS: 0
(0043, 1007) [Views per segment] SS: 16
(0043, 1008) [Respiratory rate, bpm] SS: 0
(0043, 1009) [Respiratory trigger point] SS: 0
(0043, 100a) [Type of receiver used] SS: 1
(0043, 100b) [dB/dt Peak rate of change of gradi DS: '0.0'
(0043, 100c) [dB/dt Limits in units of percent] DS: '80.0'
```

Para observar el código del programa, ver en el apéndice VIII.

III. CONCLUSIONES

Tras realizar esta práctica hemos aprendido a manejar de manera básica las imágenes en python, desde leerlas con distintas librerías como Matplotlib, OpenCV, Scikit-Image y PIL. Aprendimos cómo convertir de RGB a escala de grises, YUV y HSV. Hacer recortes sobre una imágenes y a manejar distintos formato de imagen como es el tif, jpg, raw, avi y DICOM.

Finalmente podemos concluir que aprendimos bastante del manejo básico de imágenes y estamos listos para seguir avanzando.

REFERENCES

- [1] Reynaga R. & Mayta W. (2009) *Introducción al reconocimiento de patrones*. Fides et Ratio - Revista de difusión cultural y científica de la universidad La Salle en Bolivia. 41-44. Recuperado el 19 de Agosto del 2024 en https://www.scielo.org.bo/scielo.php?script=sci_arttext&pid=S2071-081X2009000100005&lng=es&tlang=es
- [2] Arribas, J. I. (2018). Filtros paso-bajo. Univ. Valladolid Electrical Engineering Dep. https://www.lpi.tel.uva.es/~nacho/docencia/ing_ond_1/trabajos_03_04/sanitificacion/cabroa_archivos/pasobajo.html

IV. APÉNDICE

En este apéndice, se anexará el contenido de los códigos de los ejercicios desarrollados en la presente.

- I. Desarrolla un script para leer y desplegar cada imagen con los paquetes de Matplotlib, OpenCV, Scikit-Image y PIL.

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import cv2
from skimage import io
from PIL import Image

# Mostrar las imágenes originales y las
# convertidas a escala de grises
fig, axs = plt.subplots(2, 2, figsize=(10, 8))
imgMatplot=mpimg.imread('cameraman.tif')

imgCV2=cv2.imread('cameraman.tif')
imagen_rgb = cv2.cvtColor(imgCV2,
cv2.COLOR_BGR2RGB)

imgSci=io.imread('cameraman.tif')

imgPIL = Image.open('cameraman.tif')

axs[0, 0].imshow(imgMatplot, cmap = 'gray')
axs[0, 0].set_title("Lena Matplot")
axs[0, 0].axis('off')

axs[0, 1].imshow(imagen_rgb)
axs[0, 1].set_title("Lena CV2")
axs[0, 1].axis('off')

axs[1, 0].imshow(imgSci,cmap = 'gray')
axs[1, 0].set_title("Lena SciKit")
axs[1, 0].axis('off')

axs[1, 1].imshow(imgPIL,cmap = 'gray')
axs[1, 1].set_title("Lena PIL")
axs[1, 1].axis('off')
```

- II. Imprimir el tipo de imagen, el tamaño y el tipo de dato

- III. De las imágenes “lena_color_512.tif”, “peppers_color.tif”. Desarrolla un script con OpenCV y Scikit-Image para cambiar el espacio de color de:

4.3.1.RGB a Escala de grises

```
import cv2
import matplotlib.pyplot as plt

# Cargar las imágenes usando OpenCV
lena_image = cv2.imread('lena_color_512.tif')
peppers_image = cv2.imread('peppers_color2.jpg')

# Convertir de BGR (por defecto en OpenCV) a RGB
lena_image_rgb      =      cv2.cvtColor(lena_image,
cv2.COLOR_BGR2RGB)
peppers_image_rgb   =      cv2.cvtColor(peppers_image,
cv2.COLOR_BGR2RGB)

# Convertir de RGB a Escala de Grises
lena_gray      =      cv2.cvtColor(lena_image_rgb,
cv2.COLOR_RGB2GRAY)
peppers_gray   =      cv2.cvtColor(peppers_image_rgb,
cv2.COLOR_RGB2GRAY)

# Mostrar las imágenes originales y las
# convertidas a escala de grises
fig, axs = plt.subplots(2, 2, figsize=(10, 8))

axs[0, 0].imshow(lena_image_rgb)
axs[0, 0].set_title("Lena Original RGB")
axs[0, 0].axis('off')

axs[0, 1].imshow(lena_gray, cmap='gray')
axs[0, 1].set_title("Lena Escala de Grises")
axs[0, 1].axis('off')

axs[1, 0].imshow(peppers_image_rgb)
axs[1, 0].set_title("Peppers Original RGB")
axs[1, 0].axis('off')

axs[1, 1].imshow(peppers_gray, cmap='gray')
axs[1, 1].set_title("Peppers Escala de Grises")
axs[1, 1].axis('off')

plt.show()
```

4.3.2.RGB a YUV

```
import cv2
import matplotlib.pyplot as plt

# Cargar las imágenes usando OpenCV
lena_image = cv2.imread('lena_color_512.tif')
peppers_image = cv2.imread('peppers_color2.jpg')

# Convertir de BGR (por defecto en OpenCV) a RGB
```

```

lena_image_rgb      = cv2.cvtColor(lena_image,
cv2.COLOR_BGR2RGB)
peppers_image_rgb  = cv2.cvtColor(peppers_image,
cv2.COLOR_BGR2RGB)

# Convertir de RGB a YUV
lena_yuv           = cv2.cvtColor(lena_image_rgb,
cv2.COLOR_RGB2YUV)
peppers_yuv         = cv2.cvtColor(peppers_image_rgb,
cv2.COLOR_RGB2YUV)

# Mostrar las imágenes originales y las
convertidas a YUV
fig, axs = plt.subplots(2, 2, figsize=(10, 8))

axs[0, 0].imshow(lena_image_rgb)
axs[0, 0].set_title("Lena Original RGB")
axs[0, 0].axis('off')

axs[0, 1].imshow(lena_yuv)
axs[0, 1].set_title("Lena YUV")
axs[0, 1].axis('off')

axs[1, 0].imshow(peppers_image_rgb)
axs[1, 0].set_title("Peppers Original RGB")
axs[1, 0].axis('off')

axs[1, 1].imshow(peppers_yuv)
axs[1, 1].set_title("Peppers YUV")
axs[1, 1].axis('off')

plt.show()

```

4.3.3. RGB a HSV

```

import cv2
import matplotlib.pyplot as plt

# Cargar las imágenes usando OpenCV
lena_image = cv2.imread('lena_color_512.tif')
peppers_image = cv2.imread('peppers_color2.jpg')

# Convertir de BGR (por defecto en OpenCV) a RGB
lena_image_rgb      = cv2.cvtColor(lena_image,
cv2.COLOR_BGR2RGB)
peppers_image_rgb   = cv2.cvtColor(peppers_image,
cv2.COLOR_BGR2RGB)

# Convertir de RGB a HSV
lena_hsv           = cv2.cvtColor(lena_image_rgb,
cv2.COLOR_RGB2HSV)
peppers_hsv         = cv2.cvtColor(peppers_image_rgb,
cv2.COLOR_RGB2HSV)

```

```

# Mostrar las imágenes originales y las
convertidas a HSV
fig, axs = plt.subplots(2, 2, figsize=(10, 8))

axs[0, 0].imshow(lena_image_rgb)
axs[0, 0].set_title("Lena Original RGB")
axs[0, 0].axis('off')

axs[0, 1].imshow(lena_hsv)
axs[0, 1].set_title("Lena HSV")
axs[0, 1].axis('off')

axs[1, 0].imshow(peppers_image_rgb)
axs[1, 0].set_title("Peppers Original RGB")
axs[1, 0].axis('off')

axs[1, 1].imshow(peppers_hsv)
axs[1, 1].set_title("Peppers HSV")
axs[1, 1].axis('off')

plt.show()

```

IV. Despliega la paleta de colores de RGB por separado, ver figura siguiente, la barra de la derecha con valores es la paleta de colores.

Paleta de Colores de Peppers/Lena:

```

import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from PIL import Image

# Cargar la imagen
#image_path = "lena_color_512.tif"
image_path = "peppers_color2.jpg"
im = Image.open(image_path)

# Convertir la imagen en una matriz de valores RGB
rgb_data = np.array(im)

# Crear figura
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')

# Desglosar los componentes R, G y B
r, g, b = rgb_data[:, :, 0].flatten(), rgb_data[:, :, 1].flatten(), rgb_data[:, :, 2].flatten()

# Graficar los puntos RGB
ax.scatter(r, g, b, c=np.stack((r/255, g/255, b/255), axis=-1), marker="o", alpha=0.1)

```

```

# Etiquetas de los ejes
ax.set_xlabel('Red')
ax.set_ylabel('Green')
ax.set_zlabel('Blue')

# Agregar la barra de colores
mappable = plt.cm.ScalarMappable(cmap='jet')
mappable.set_array(np.arange(256))
plt.colorbar(mappable, ax=ax, shrink=0.8,
             aspect=5)

plt.show()

```

V. De una imagen que usted escoja, dejarla en escala de grises y procure que sea igual en renglones y en columnas. Programe una función que realice decimación de una imagen, reduciéndola a la mitad de su tamaño original. Y promediando en grupos de 4 pixeles. Pruebe con su imagen.

```

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

# Función para realizar la decimación de la imagen
def decimate_image(image):
    # Obtener dimensiones de la imagen
    height, width = image.shape

    # Verificar que las dimensiones sean pares
    if height % 2 != 0 or width % 2 != 0:
        raise ValueError("La imagen debe tener dimensiones pares.")

    # Crear la imagen decimada con la mitad del tamaño original
    new_height, new_width = height // 2, width // 2
    decimated_image = np.zeros((new_height, new_width), dtype=np.uint8)

    # Promediar los grupos de 4 pixeles
    for i in range(0, height, 2):
        for j in range(0, width, 2):
            pixel_block = image[i:i+2, j:j+2]
            decimated_image[i//2, j//2] = np.mean(pixel_block)

    return decimated_image

# Cargar la imagen que deseas (por ejemplo, lena)

```

```

image      = cv2.imread('lena_color_512.tif',
cv2.IMREAD_GRAYSCALE)

# Asegurarse de que la imagen sea cuadrada
size = min(image.shape)
image = cv2.resize(image, (size, size))

# Aplicar la decimación
decimated_image = decimate_image(image)

# Mostrar la imagen original y la decimada
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Imagen Original (Escala de Grises)')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(decimated_image, cmap='gray')
plt.title('Imagen Decimada')
plt.axis('off')

plt.show()

```

VI. Convierte la imagen peppers_color.tif a escala de grises,
4.6.1. Recortela de manera que solo quede uno de los pimientos verdes en ese recorte
4.6.2. Guardela en formato jpg.

```

import cv2
import matplotlib.pyplot as plt

# Cargar la imagen en color
image = cv2.imread('peppers_color2.jpg')

# Convertir la imagen a escala de grises
image_gray      = cv2.cvtColor(image,
cv2.COLOR_BGR2GRAY)

# Recortar la región que contiene uno de los pimientos verdes
# Aquí se debe ajustar manualmente los valores de recorte según la posición del pimiento
# (x, y, w, h) son las coordenadas y el tamaño del recorte
x, y, w, h = 160, 200, 275, 275 # Valores

# Realizar el recorte
cropped_image = image_gray[y:y+h, x:x+w]

# Guardar la imagen recortada en formato .jpg

```

```

cv2.imwrite('pepper_green_cropped.jpg',
cropped_image)

# Mostrar la imagen original, la imagen en escala
de grises y el recorte
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.title('Imagen Original')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(image_gray, cmap='gray')
plt.title('Imagen en Escala de Grises')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(cropped_image, cmap='gray')
plt.title('Recorte de Pimiento Verde')
plt.axis('off')

plt.show()

```

VII. Un formato de imágenes sin ningún tipo de codificación se conoce como formato crudo (RAW). De la imagen “rosa800x600.raw” lea y despliegue la imagen. Tome en cuenta que esta imagen maneja la precisión de integer8 y el tamaño es de 600x800 pixeles.

```

import numpy as np
import matplotlib.pyplot as plt

# Especificaciones de la imagen
width = 800 # Ancho de la imagen
height = 600 # Altura de la imagen
dtype = np.uint8 # Tipo de dato (8-bit entero)

# Leer los datos crudos de la imagen
with open('rosa800x600.raw', 'rb') as file:
    img_data = np.fromfile(file, dtype=dtype)

# Asegurarse de que los datos tienen el tamaño
esperado
if img_data.size != width * height:
    raise ValueError("El tamaño de la imagen no
coincide con las dimensiones especificadas.")

# Remodelar los datos en una matriz 2D
image = img_data.reshape((height, width))

# Mostrar la imagen

```

```

plt.imshow(image, cmap='gray')
plt.title('Imagen RAW: rosa800x600')
plt.axis('off')
plt.show()

```

VIII. Lectura de Video archivo “.avi”. Probar el siguiente código y muestre el video “0X2A8498756C4D6E82_corto.avi”. Si existe un error corregirlo.

```

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

# Read video
cap = cv2.VideoCapture('0X2A8498756C4D6E82_corto.avi')

# Leer el primer frame del video
ret, frame = cap.read()

# Mostrar el primer frame
plt.figure()
plt.imshow(frame)

```

IX. Imagen DICOM. Mostrar la información del encabezado de los formatos, investigar como se despliega.

a. Usar imagen DICOM: Realiza un script que lea las imágenes de la carpeta DICOM y muestra en una misma ventana las 280 imágenes de tal manera que se pueda simular el movimiento del corazón. Utiliza ciclos for y el comando pause, para leer la ruta completa de la carpeta.

```

import os
import numpy as np
import matplotlib.pyplot as plt
import imageio
import pydicom

def read_image(file_path):
    _, ext = os.path.splitext(file_path)
    ext = ext.lower()

    try:

```

```

if ext in ['.tif', '.tiff']:
    img_array = imageio.imread(file_path)
elif ext == '.dcm':
    pydicom.dcmread(file_path)           dicom_data =
    img_array = dicom_data.pixel_array
    if img_array.shape[2] == 2: img_array.ndim == 3 and
        img_array = img_array[:, :, 0]
    return img_array, dicom_data
else:
    raise ValueError(f"Formato no soportado: {ext}")

except Exception as e:
    raise ValueError(f"Error al leer {file_path}: {e}")

images_directory = 'Imagenes' # Directorio local

# Carga las imágenes
images = []
file_names = []
image_types = []
image_shapes = []
image_dtypes = []
dicom_data_list = []

for file_name in os.listdir(images_directory):
    file_path = os.path.join(images_directory,
    try:
        image, dicom_data = read_image(file_path)
        images.append(image)
        file_names.append(file_name)
        image_types.append(
            file_name.lower().endswith('.tif', '.TIF') else
            file_name.lower().endswith('.dcm') else 'DICOM' if
            )
        image_shapes.append(image.shape)
        image_dtypes.append(image.dtype)
        dicom_data_list.append(dicom_data)
    except Exception as e:
        print(f"Error al cargar {file_name}: {e}")
        continue

fig, ax = plt.subplots(figsize=(10, 8)) # Ajusta

```

```

current_index = 0

def update_image(index):
    ax.clear()
    img = images[index]
    if len(img.shape) == 2:
        # Imagen en escala de grises
        ax.imshow(img, cmap='gray')
    else:
        # Imagen en color
        ax.imshow(img)

    ax.set_title(
        f"File: {file_names[index]}\n"
        f"Type: {image_types[index]}\n"
        f"Size: {image_shapes[index]}\n"
        f"Data Type: {image_dtypes[index]}",
        fontsize=8
    )

actual# si imprime el encabezado DICOM de la imagen
dicom_data_list[index].type != 'DICOM' and
{file_names[index]}: print(f"Encabezado DICOM de
    print(dicom_data_list[index])

plt.draw()

update_image(current_index)

def on_key(event):
    global current_index
    if event.key == 'right':
        if current_index < len(images) - 1:
            current_index += 1
            update_image(current_index)
    elif event.key == 'left':
        if current_index > 0:
            current_index -= 1
            update_image(current_index)

fig.canvas.mpl_connect('key_press_event', on_key)

```

```
plt.show()
```



García López Erik
Reyes Herrera Robelio
Serapio Hernández Alexis Arturo

Manejo Básico de Imágenes

Resumen: Este pequeño fragmento del trabajo total titulado manejo básico de imágenes tiene como objetivo comprender las funciones más básicas para la creación, carga manipulación de imágenes en Python; inicialmente se planeta con el modulo scipy, pero al estar depreciado se adecuó el tutorial con el modulo skimage sin alterar ninguna línea de código. Se hace énfasis en el aspecto práctico sin dejar de lado los fundamentos teóricos.

La primera parte “Writing an array to a file” consiste en guardar en archivo de tipo imagen un arreglo de píxeles.

El código original es el siguiente:

```
from scipy import misc
f = misc.face()
misc.imsave('face.png', f) # uses the Image module (PIL)
import matplotlib.pyplot as plt
plt.imshow(f)
plt.show()
```

Este código carga una imagen muestra de un mapache y la guarda en un png; la adecuación necesaria carga una imagen muestra de un gato y realiza el mismo procedimiento.

```
1  from skimage import data, io
2  from PIL import Image
3  import matplotlib.pyplot as plt
4
5  f = data.chelsea()
6
7  ie.imsave( fname: 'face.png' , f )
8  |
9  plt.imshow(f)
10 plt.show()
```



```
/usr/local/lib/python3.10/dist-packages/scipy/misc/_init_.py in __getattr__(name)
    43     def __getattr__(name):
    44         if name not in _all_:
--> 45             raise AttributeError(
    46                 "scipy.misc is deprecated and has no attribute "
    47                 f"{{name}}.")
```

AttributeError: scipy.misc is deprecated and has no attribute imsave.



La segunda parte “Creating a numpy array from an image file”, consiste en crear un arreglo del módulo numpy a partir de una imagen, que en este caso será la misma; esto se hace con:

```
face = io.imread('face.png')
type(face)
face.shape, face.dtype
```

La segunda línea nos dice que, en este caso, face debería ser un array de NumPy mientras que la tercera retorna una tupla que representa las dimensiones del array, para una imagen en color, face.shape suele ser una tupla de la forma (alto, ancho, canales).

La tercera parte “Opening raw files (camera, 3-D images)” generó un error:

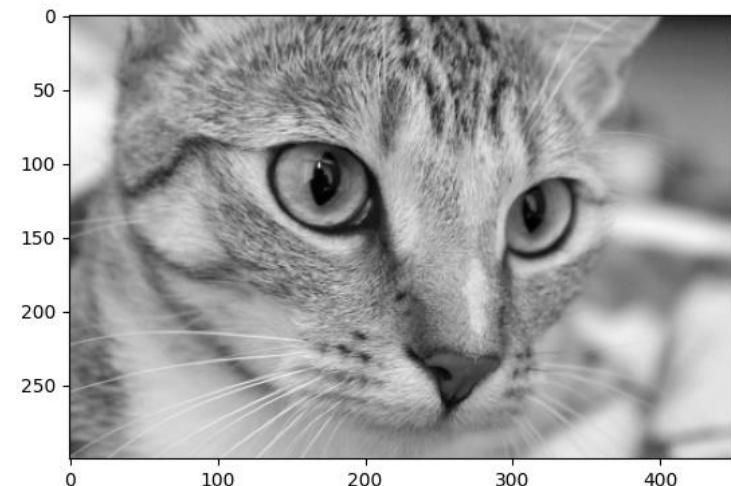
```
face_from_raw.shape = (768, 1024, 3)
^^^^^^^^^^^^^^^^^^^^^
ValueError: cannot reshape array of size 405900 into shape (768,1024,3)
```

Este es debido a que las dimensiones del arreglo no son las que se le están tratando de dar, por tanto, con ayuda de face.shape obtenemos los datos correctos.

Pasando a la escala de grises:

```
1  from skimage import data, io, color, img_as_ubyte
2  from PIL import Image
3  import matplotlib.pyplot as plt
4
5  f = data.chelsea()
6  f_gray = color.rgb2gray(f)
7  f_gray = img_as_ubyte(f_gray)
8
9  io.imsave(fname: 'facades.png', f_gray)
10
11 plt.imshow(f_gray, cmap=plt.cm.gray)
12 plt.show()
13
14
15
```

Figure 1

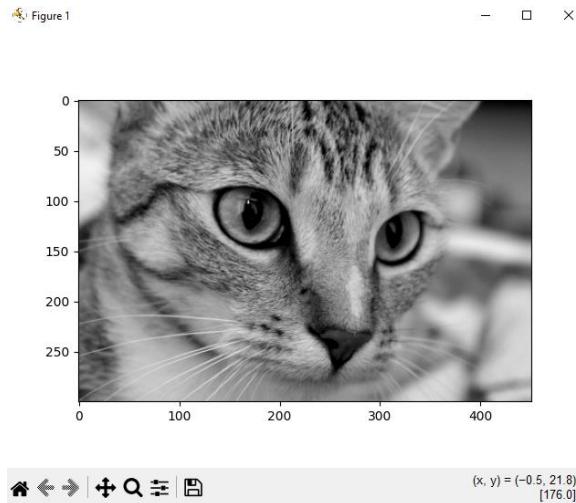




García López Erik

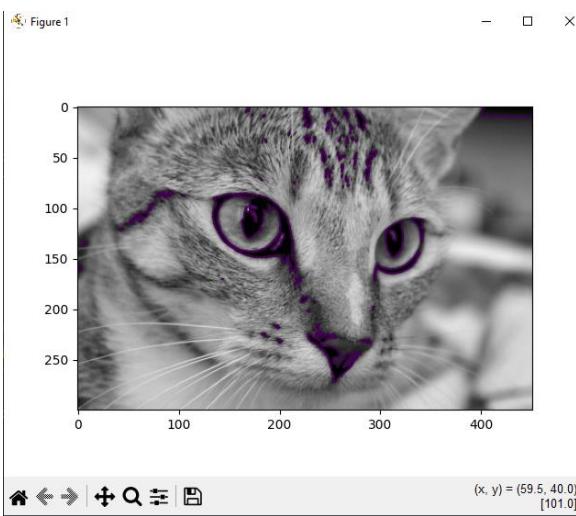
Reyes Herrera Rogelio
Serapio Hernández Alexis Arturo

Posteriormente jugando con el contraste:



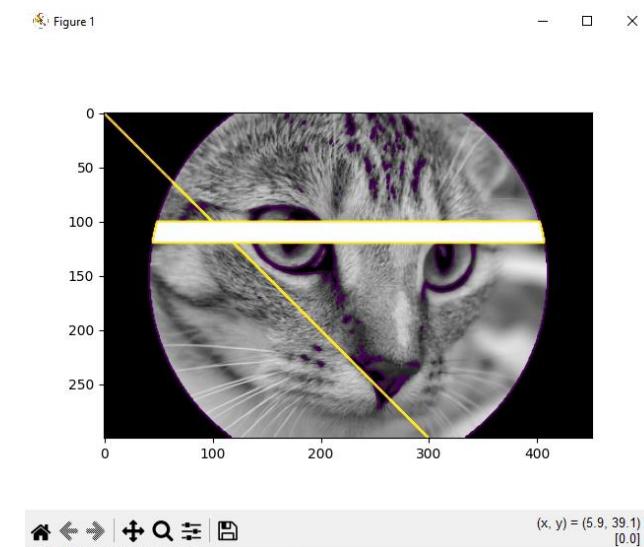
Se nota una imagen más vívida.

“Draw contour lines”:



Usando ciertos pixeles y rangos de pixeles con:

```
f_gray[10:13, 20:23]
f_gray[100:120] = 255
lx, ly = f_gray.shape
X, Y = np.ogrid[0:lx, 0:ly]
mask = (X - lx / 2) ** 2 + (Y - ly / 2) ** 2 > lx * ly / 4
# Masks
f_gray[mask] = 0
# Fancy indexing
max_range = min(lx, ly)
f_gray[range(max_range), range(max_range)] = 255
```



Y por último haciendo algunas transformaciones básicas sobre la imagen con este código tenemos los siguientes resultados:

```
#crop_face = f_gray[lx // 4: - lx // 4, ly // 4: - ly // 4]
```

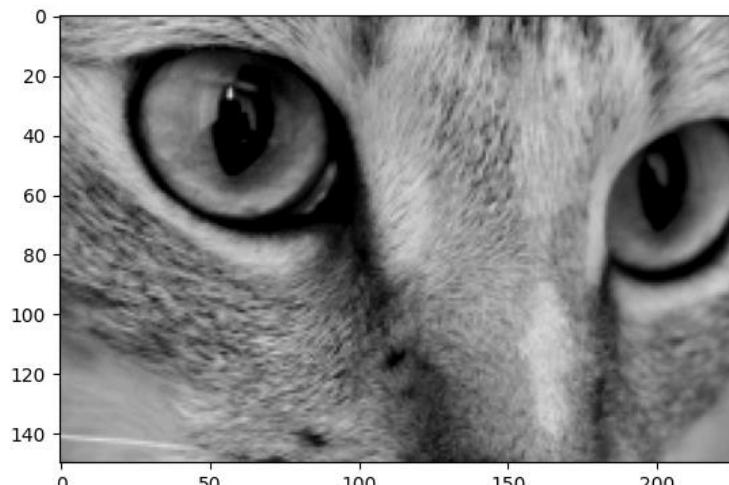


García López Erik

Reyes Herrera Rogelio
Serapio Hernández Alexis Arturo

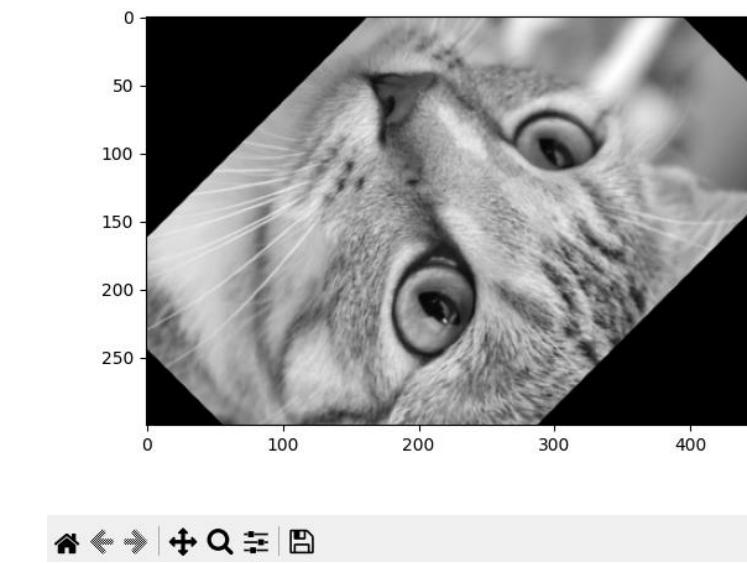
Figure 1

Figure 1



```
flip_f = np.fliplr(f_gray)  
rot = rotate(flip_f, 45, resize=False)  
max_range = min(lx, ly)
```

Se hace una rotación de 45 grados y un flip a la imagen, sin el zoom generado con anterioridad:



Referencias

- [1] «3.3. Scikit-image: image processing,» [En línea]. Available: <https://scipy-lectures.org/packages/scikit-image/index.html#scikit-image>. [Último acceso: 18 Agosto 2024].