

TP2

Luciana Diaz Kralj
Agustina Sol Ortú
Paula Oseroff

4 de septiembre de 2024

Profesor: Daniel Jacoby



ex1

September 4, 2024

1 Ejercicio 1

```
[ ]: from PIL import Image
      from pathlib import Path
      import matplotlib.pyplot as plt

[ ]: ASSETS_FOLDER_PATH = "../assets"
      OUTPUT_FOLDER_PATH = "./"

[ ]: Path(OUTPUT_FOLDER_PATH).mkdir(parents=True, exist_ok=True)

[ ]: rose_b_w_1024x1024 = Image.open(f"{ASSETS_FOLDER_PATH}/Fig0219(rose1024).tif")
      rose_b_w_1024x1024.save(f"{OUTPUT_FOLDER_PATH}/rose_b_w_1024x1024.bmp")
      display(f"{'rose_b_w_1024x1024.width'}x{'rose_b_w_1024x1024.height'}")
      display(rose_b_w_1024x1024)

'1024x1024'
```



```
[ ]: building = Image.open(f"{ASSETS_FOLDER_PATH}/building.jpg")
display(f"{building.width}x{building.height}")
display(building)
```

'512x384'



```
[ ]: skull_256k = Image.open(f'{ASSETS_FOLDER_PATH}/Fig0221(a)(ctskull-256).tif")
display(f'{skull_256k.width}x{skull_256k.height}')
display(skull_256k)
```

'374x452'



```
[ ]: def subsampling(image: Image.Image) -> Image.Image:
    result_y = Image.new(mode=image.mode, size=(int(image.width), int(image.
    ↴height/2)))
    result_y_pixels = result_y.load()
    image_pixels = image.load()

    y_res = 0
    for y in range(image.height):
        if y % 2 == 0:
            for x in range(image.width):
```

```

        result_y_pixels[x, y_res] = image_pixels[x, y]
        y_res += 1

    result = Image.new(mode=image.mode, size=(int(image.width/2), int(image.
    ↪height/2)))
    result_pixels = result.load()
    x_res = 0
    for x in range(result_y.width):
        if x % 2 == 0:
            for y in range(result_y.height):
                result_pixels[x_res, y] = result_y_pixels[x, y]
        x_res += 1

    return result

```

```
[ ]: def replicating(image: Image.Image) -> Image.Image:
    result_y = Image.new(mode=image.mode, size=(int(image.width), int(image.
    ↪height*2)))
    result_y_pixels = result_y.load()
    image_pixels = image.load()

    y_res = 0
    for y in range(image.height):
        for _ in range(2):
            for x in range(image.width):
                result_y_pixels[x, y_res] = image_pixels[x, y]
            y_res += 1

    result = Image.new(mode=image.mode, size=(int(image.width*2), int(image.
    ↪height*2)))
    result_pixels = result.load()
    x_res = 0
    for x in range(result_y.width):
        for _ in range(2):
            for y in range(result_y.height):
                result_pixels[x_res, y] = result_y_pixels[x, y]
        x_res += 1

    return result

```

```
[ ]: def scale_down_luminance(image: Image.Image, levels: int) -> Image.Image:
    result = image.copy()
    result_pixels = result.load()
    for x in range(result.width):
        for y in range(result.height):
            result_pixels[x, y] = round(round(result_pixels[x, y]*(levels-1) / ↪
    ↪255)*(255/(levels-1)))

```

```
    return result
```

```
[ ]: rose_b_w_512x512 = subsampling(rose_b_w_1024x1024)
rose_b_w_512x512.save(f"{OUTPUT_FOLDER_PATH}/rose_b_w_512x512.bmp")
display(f'{rose_b_w_512x512.width}x{rose_b_w_512x512.height}')
display(rose_b_w_512x512)
```

'512x512'



```
[ ]: rose_b_w_256x256 = subsampling(rose_b_w_512x512)
rose_b_w_256x256.save(f"{OUTPUT_FOLDER_PATH}/rose_b_w_256x256.bmp")
display(f'{rose_b_w_256x256.width}x{rose_b_w_256x256.height}')
display(rose_b_w_256x256)
```

'256x256'



```
[ ]: rose_b_w_128x128 = subsampling(rose_b_w_256x256)
      rose_b_w_128x128.save(f"{OUTPUT_FOLDER_PATH}/rose_b_w_128x128.bmp")
      display(f"{rose_b_w_128x128.width}x{rose_b_w_128x128.height}")
      display(rose_b_w_128x128)
```

'128x128'



```
[ ]: rose_b_w_64x64 = subsampling(rose_b_w_128x128)
      rose_b_w_64x64.save(f"{OUTPUT_FOLDER_PATH}/rose_b_w_64x64.bmp")
      display(f"{rose_b_w_64x64.width}x{rose_b_w_64x64.height}")
      display(rose_b_w_64x64)
```

'64x64'



```
[ ]: rose_b_w_32x32 = subsampling(rose_b_w_64x64)
rose_b_w_32x32.save(f"{OUTPUT_FOLDER_PATH}/rose_b_w_32x32.bmp")
display(f"{rose_b_w_32x32.width}x{rose_b_w_32x32.height}")
display(rose_b_w_32x32)
```

'32x32'



```
[ ]: rose_b_w_from_512x512 = replicating(rose_b_w_512x512)
rose_b_w_from_512x512.save(f"{OUTPUT_FOLDER_PATH}/rose_b_w_from_512x512.bmp")
display(f"{rose_b_w_from_512x512.width}x{rose_b_w_from_512x512.height}")
display(rose_b_w_from_512x512)
```

'1024x1024'



```
[ ]: rose_b_w_from_256x256 = replicating(replicating(rose_b_w_256x256))
rose_b_w_from_256x256.save(f"{OUTPUT_FOLDER_PATH}/rose_b_w_from_256x256.bmp")
display(f"{rose_b_w_from_256x256.width}x{rose_b_w_from_256x256.height}")
display(rose_b_w_from_256x256)
```

'1024x1024'



```
[ ]: rose_b_w_from_128x128 = replicating(replicating(replicating(rose_b_w_128x128)))
rose_b_w_from_128x128.save(f"{OUTPUT_FOLDER_PATH}/rose_b_w_from_128x128.bmp")
display(f"{rose_b_w_from_128x128.width}x{rose_b_w_from_128x128.height}")
display(rose_b_w_from_128x128)
```

'1024x1024'



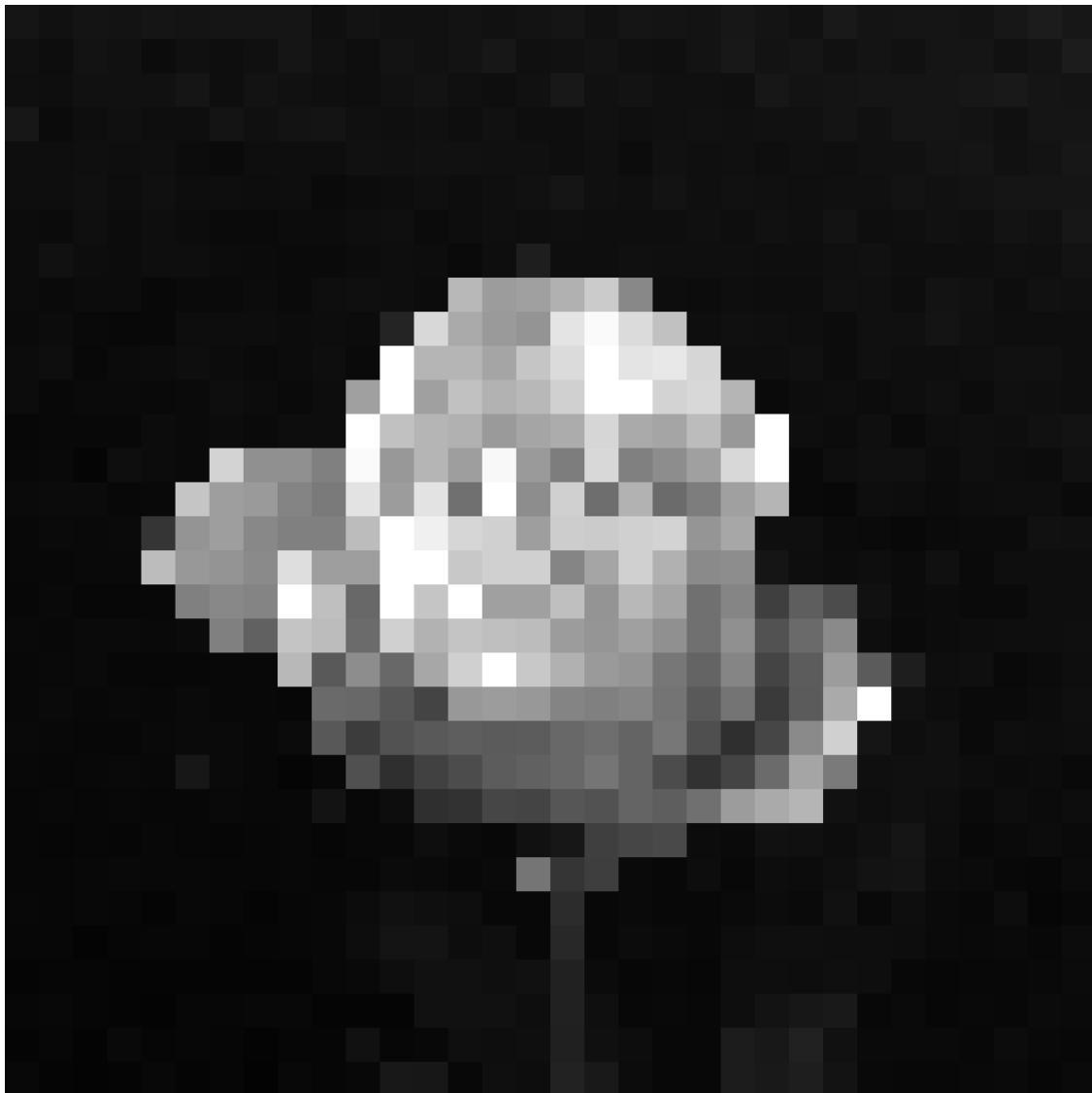
```
[ ]: rose_b_w_from_64x64 =_
    ↪replicating(replicating(replicating(replicating(rose_b_w_64x64))))
rose_b_w_from_64x64.save(f"{OUTPUT_FOLDER_PATH}/rose_b_w_from_64x64.bmp")
display(f"{rose_b_w_from_64x64.width}x{rose_b_w_from_64x64.height}")
display(rose_b_w_from_64x64)
```

'1024x1024'



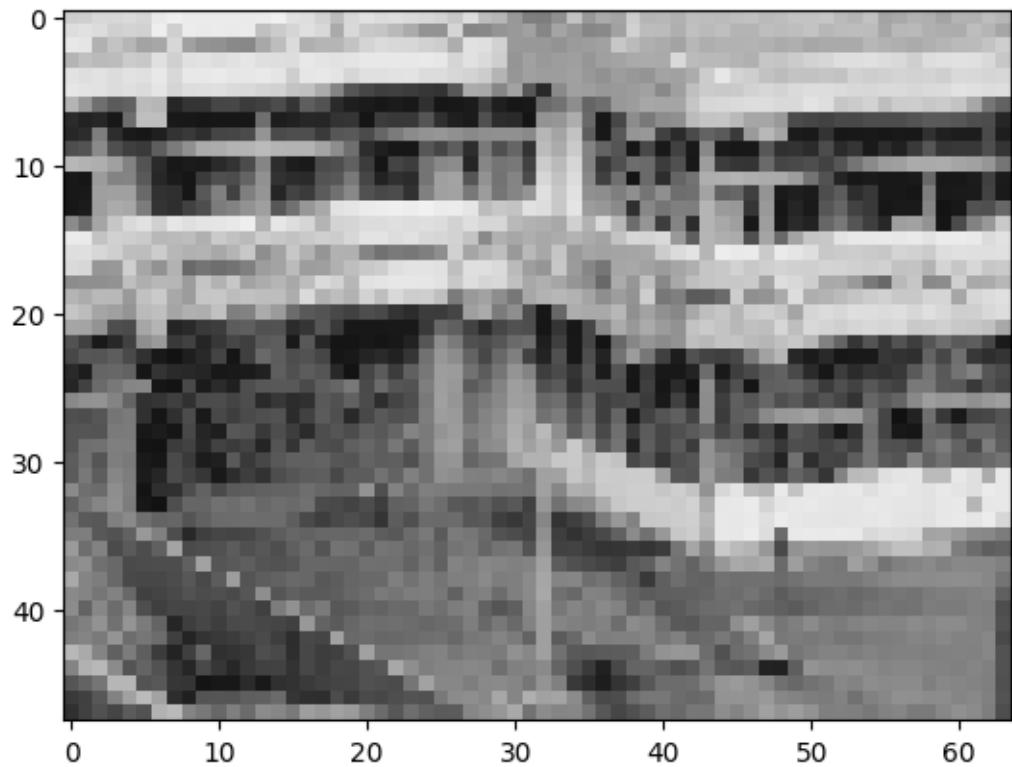
```
[ ]: rose_b_w_from_32x32 =_
    ↪replicating(replicating(replicating(replicating(replicating(rose_b_w_32x32)))))
rose_b_w_from_32x32.save(f"{OUTPUT_FOLDER_PATH}/rose_b_w_from_32x32.bmp")
display(f"{rose_b_w_from_32x32.width}x{rose_b_w_from_32x32.height}")
display(rose_b_w_from_32x32)
```

'1024x1024'



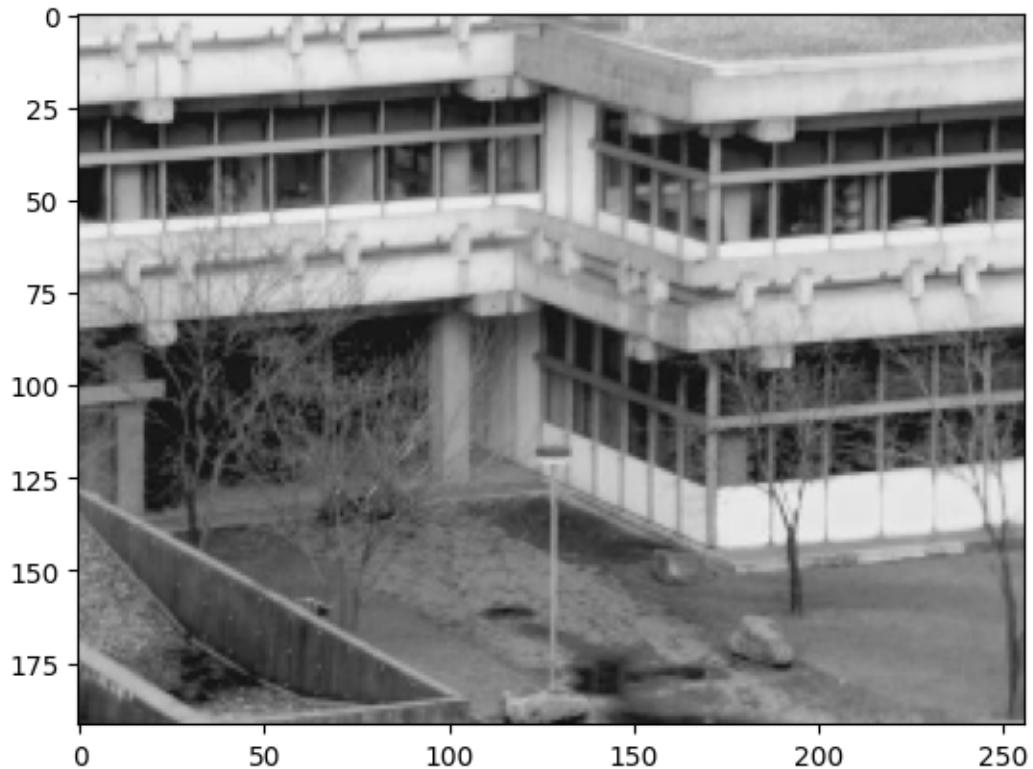
```
[ ]: building_48x64 = subsampling(subsampling(subsampling(building)))
building_48x64.save(f"{OUTPUT_FOLDER_PATH}/building_48x64.bmp")
plt.imshow(building_48x64, cmap="gray", vmin=0, vmax=255)
```

```
[ ]: <matplotlib.image.AxesImage at 0x79b6c7f18ee0>
```



```
[ ]: building_192x256 = subsampling(building)
building_192x256.save(f"{OUTPUT_FOLDER_PATH}/building_192x256.bmp")
plt.imshow(building_192x256, cmap="gray", vmin=0, vmax=255)
```

```
[ ]: <matplotlib.image.AxesImage at 0x79b6c7fe6260>
```



```
[ ]: skull_128k = scale_down_luminance(skull_256k, 128)
skull_128k.save(f"{OUTPUT_FOLDER_PATH}/skull_128k.bmp")
display(skull_128k)
```



```
[ ]: skull_64k = scale_down_luminance(skull_256k, 64)
skull_64k.save(f"{OUTPUT_FOLDER_PATH}/skull_64k.bmp")
display(skull_64k)
```



```
[ ]: skull_32k = scale_down_luminance(skull_256k, 32)
skull_32k.save(f"{OUTPUT_FOLDER_PATH}/skull_32k.bmp")
display(skull_32k)
```



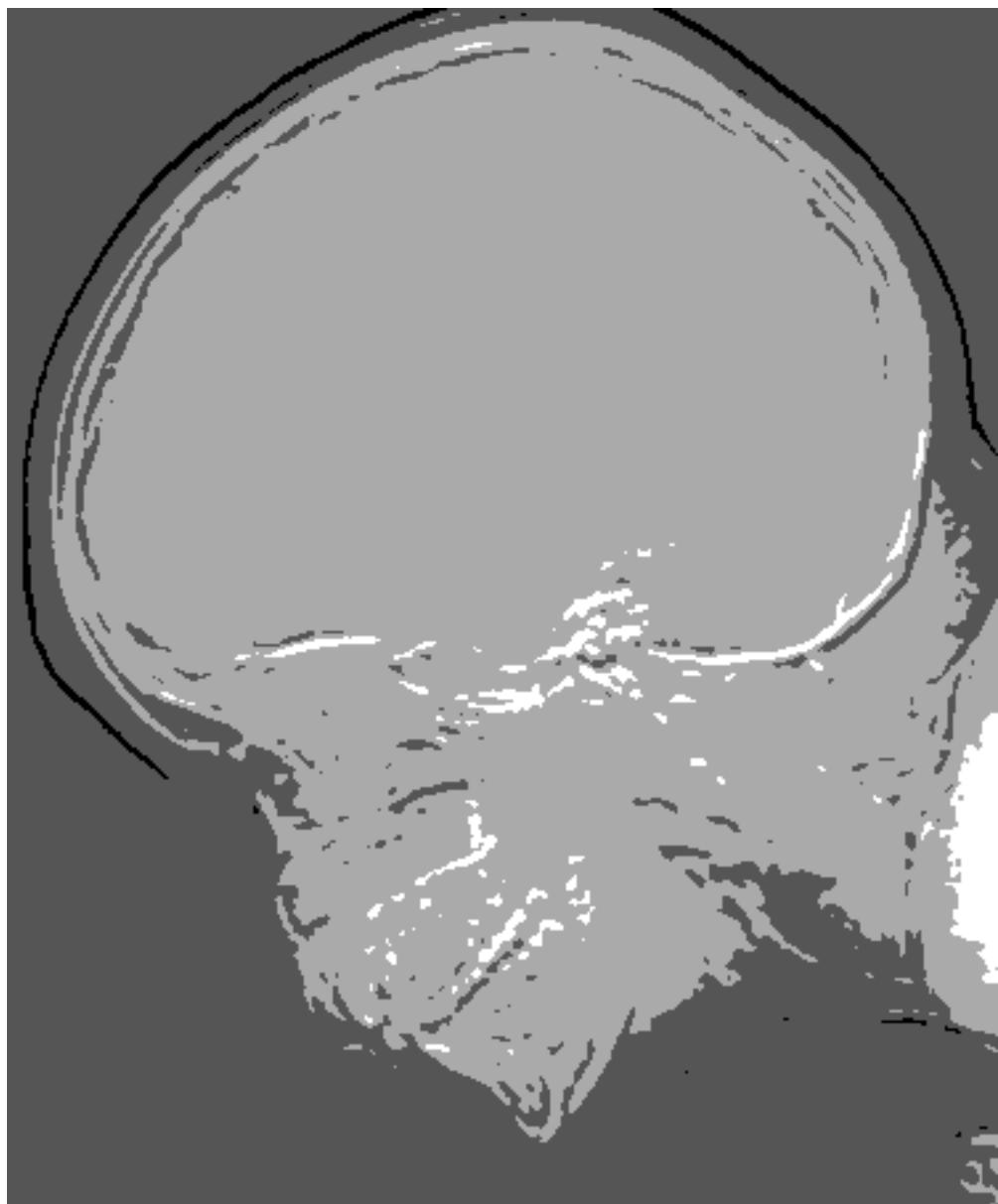
```
[ ]: skull_16k = scale_down_luminance(skull_256k, 16)
skull_16k.save(f"{OUTPUT_FOLDER_PATH}/skull_16k.bmp")
display(skull_16k)
```



```
[ ]: skull_8k = scale_down_luminance(skull_256k, 8)
skull_8k.save(f"{OUTPUT_FOLDER_PATH}/skull_8k.bmp")
display(skull_8k)
```



```
[ ]: skull_4k = scale_down_luminance(skull_256k, 4)
skull_4k.save(f"{OUTPUT_FOLDER_PATH}/skull_4k.bmp")
display(skull_4k)
```



```
[ ]: skull_2k = scale_down_luminance(skull_256k, 2)
skull_2k.save(f"{OUTPUT_FOLDER_PATH}/skull_2k.bmp")
display(skull_2k)
```



2. Ejercicio 2

python-ex1

September 4, 2024

0.1 ex1.m

```
[ ]: # ! pip install -U scikit-image
# ! pip install numpy matplotlib opencv-python
```

```
[ ]: import cv2
import numpy as np
import scipy.fft as fft
import matplotlib.pyplot as plt
import plotly.graph_objects as go

from scipy.signal import convolve2d
from skimage.draw import disk

N = 256
```

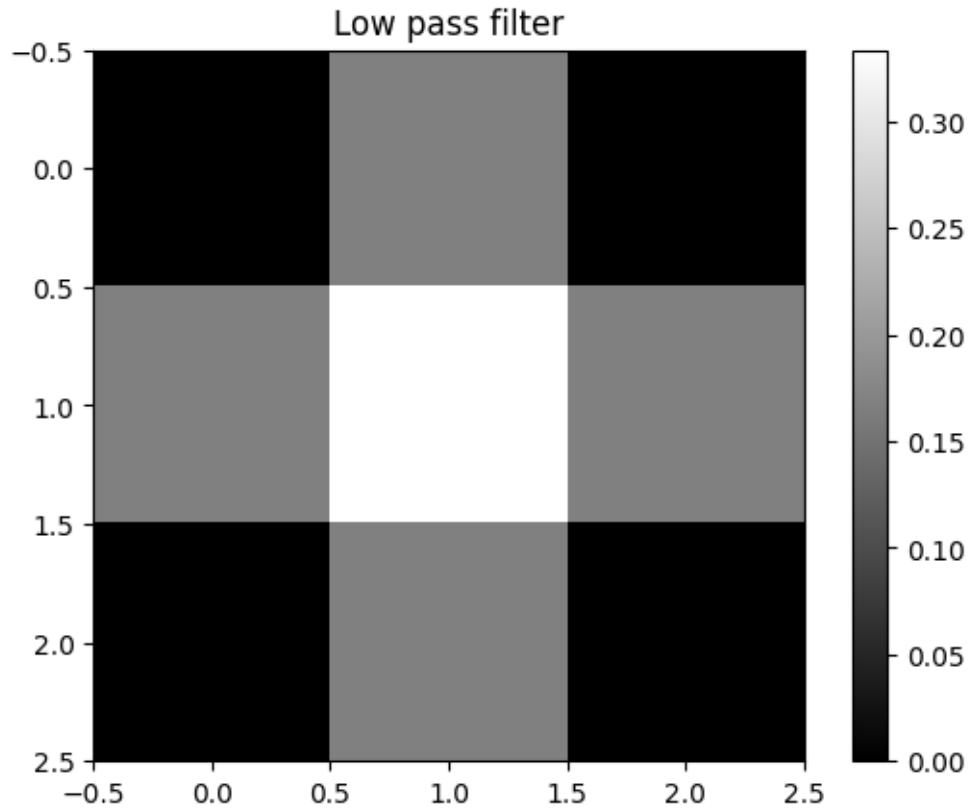
Definimos los filtros y los visualizamos:

```
[ ]: def filterVisual(h, title):
    plt.imshow(h, cmap='gray')
    plt.colorbar()
    plt.title(title)
    plt.show()

    f = fft.fftshift(fft.fftfreq(N))
    H = fft.fftshift(fft.fft2(h, [N, N]))
    fig = go.Figure(data=[go.Surface(x=f, y=f, z=np.abs(H))])
    fig.update_layout(title=title+' FFT', autosize=False, width=700,
                     height=500, margin=dict(l=40, r=10, b=80, t=60),
                     scene = dict(xaxis_title='Fx', yaxis_title='Fy',
                     xaxis_title='Magnitude'))
    fig.show()
```

- Pasa bajos:

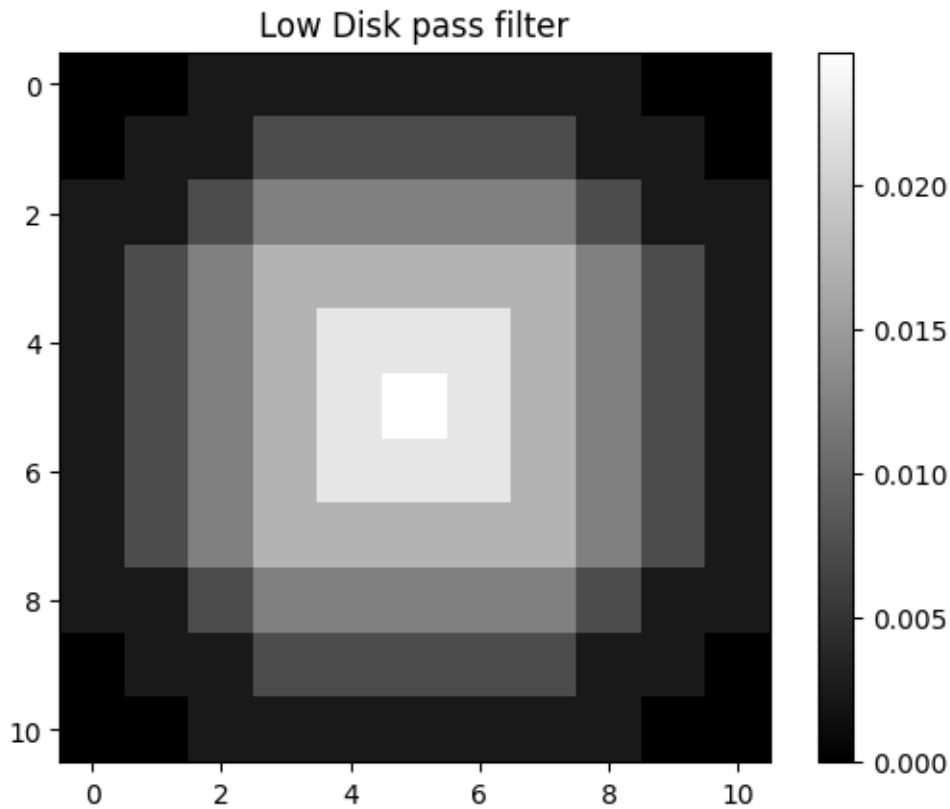
```
[ ]: h_low = np.array([[0, 1/6, 0], [1/6, 1/3, 1/6], [0, 1/6, 0]])
filterVisual(h_low, 'Low pass filter')
```



- Pasa bajos en forma de disco:

```
[ ]: def fspecial_disk(radius):
    size = 2 * radius + 1
    h = np.zeros((size, size), dtype=np.float32)
    rr, cc = disk((size // 2, size // 2), radius+1)
    h[rr, cc] = 0.1
    rr, cc = disk((size // 2, size // 2), radius)
    h[rr, cc] = 0.3
    rr, cc = disk((size // 2, size // 2), radius-1)
    h[rr, cc] = 0.5
    rr, cc = disk((size // 2, size // 2), radius-2)
    h[rr, cc] = 0.7
    rr, cc = disk((size // 2, size // 2), radius-3)
    h[rr, cc] = 0.9
    rr, cc = disk((size // 2, size // 2), radius-4)
    h[rr, cc] = 1
    h /= np.sum(h)
    return h
```

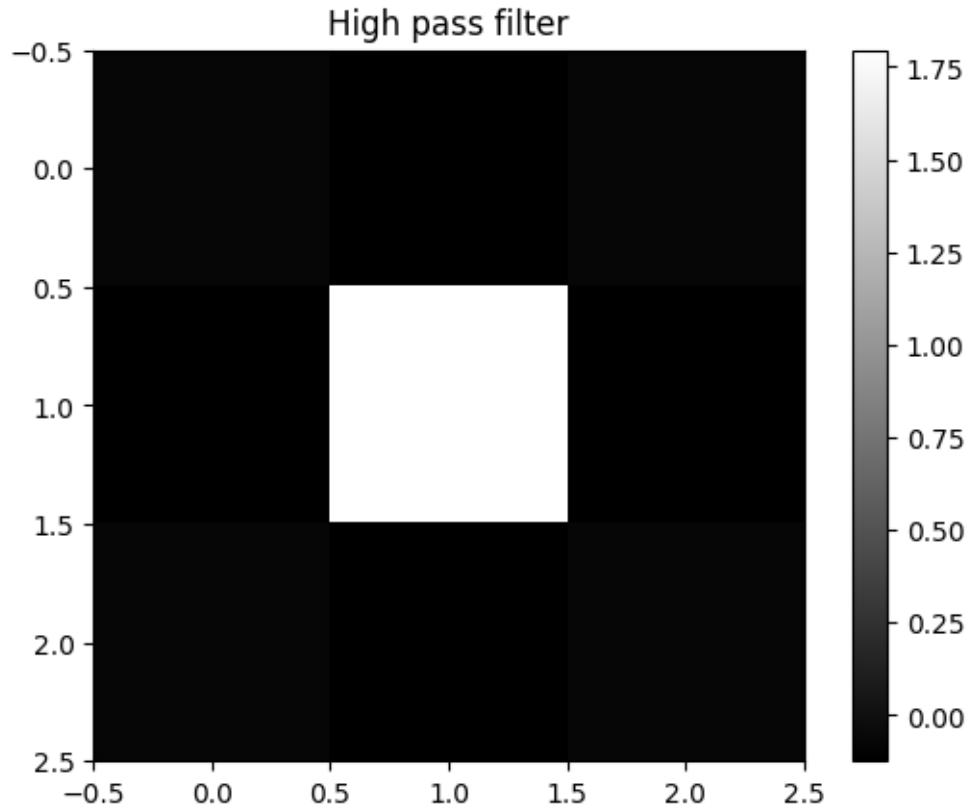
```
[ ]: h_lowDisk = fspecial_disk(radius = 5)
filterVisual(h_lowDisk, 'Low Disk pass filter')
```



- Pasa altos:

```
[ ]: def fspecial_unsharp(size, sigma):
    gaussian_kernel = cv2.getGaussianKernel(size, sigma)
    gaussian_kernel = gaussian_kernel @ gaussian_kernel.T
    unsharp_kernel = np.zeros_like(gaussian_kernel)
    unsharp_kernel[size // 2, size // 2] = 2
    unsharp_kernel -= gaussian_kernel
    return unsharp_kernel
```

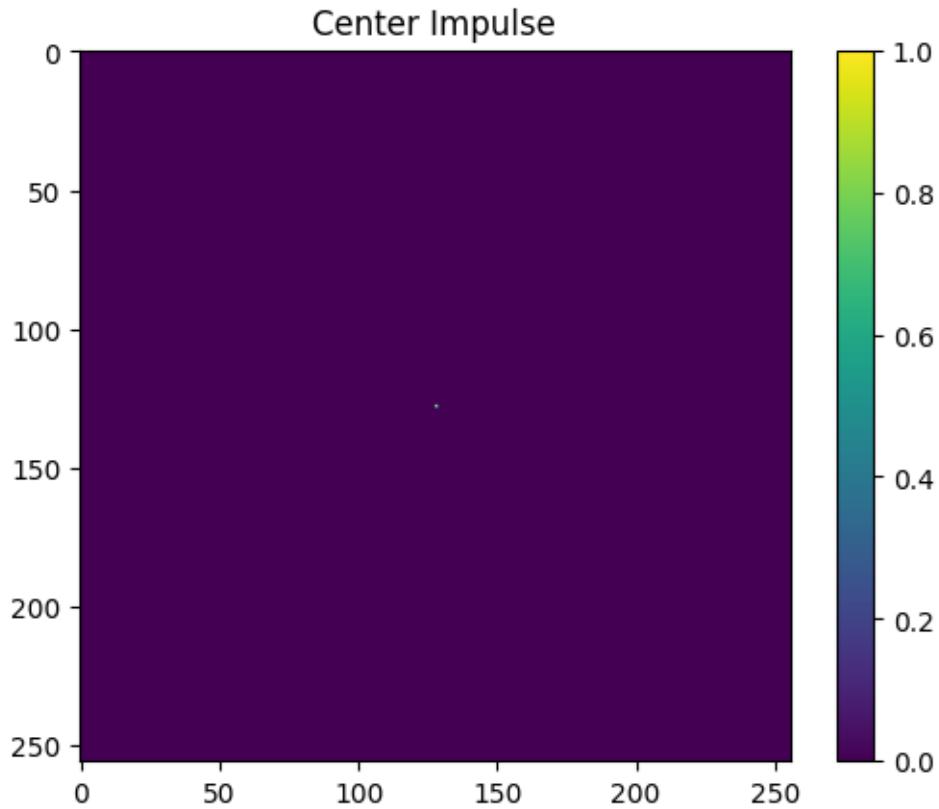
```
[ ]: h_high = fspecial_unsharp(size = 3, sigma = 1)
filterVisual(h_high, 'High pass filter')
```



Construimos una imagen (que es una delta en 0) suficientemente grande para poder bien el espectro.

```
[ ]: big = np.zeros((N, N))
big[N//2, N//2] = 1

plt.imshow(big)
plt.colorbar()
plt.title("Center Impulse")
plt.show()
```



Convolucionamos la imagen con los filtros y graficamos el espectro de la imagen filtrada:

```
[ ]: def showSpect(src, kernel, title):
    fig = plt.figure(figsize=(10, 7))
    fig.add_subplot(1, 2, 1)

    # h1 = cv2.filter2D(src, -1, kernel) # ddepth=-1 keeps src depth
    h1 = convolve2d(src, kernel)
    S = fft.fft2(h1)
    SM = np.abs(S)

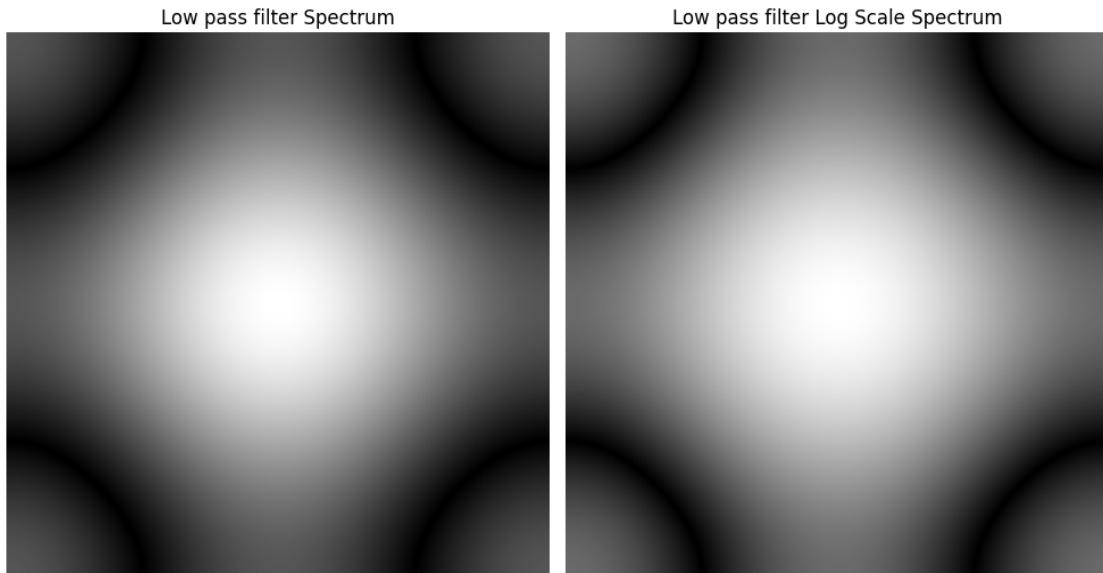
    plt.imshow(fft.fftshift(SM / np.max(np.max(SM))), cmap='gray')
    plt.axis('off')
    plt.title(title+' Spectrum')

    IMd = np.log(1 + np.abs(SM))

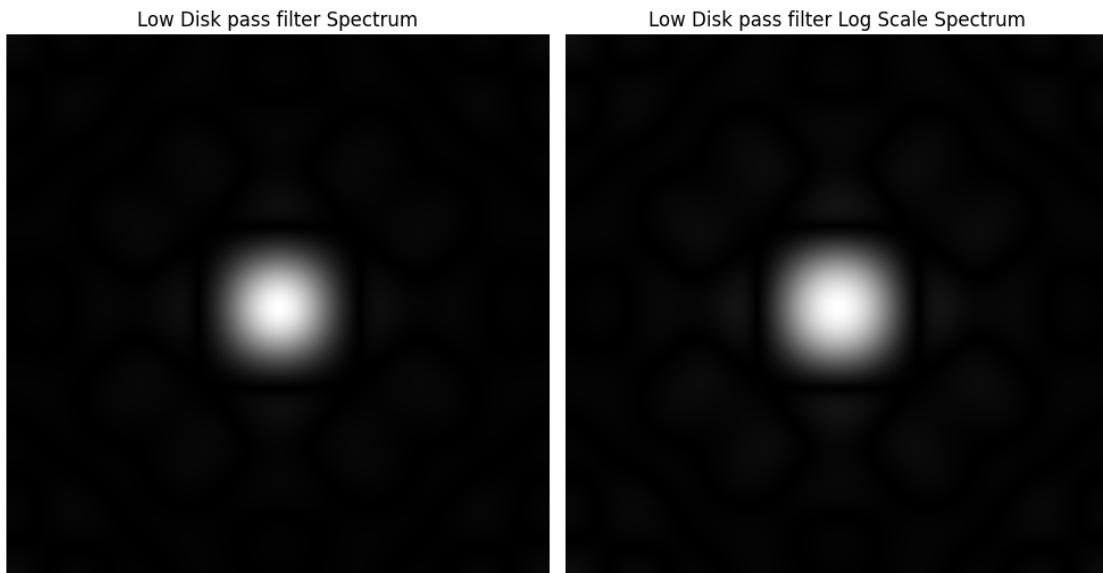
    fig.add_subplot(1, 2, 2)
    plt.imshow(fft.fftshift(IMd / np.max(np.max(IMd))), cmap='gray')
    plt.axis('off')
    plt.title(title+' Log Scale Spectrum')
```

```
plt.tight_layout()  
plt.show()
```

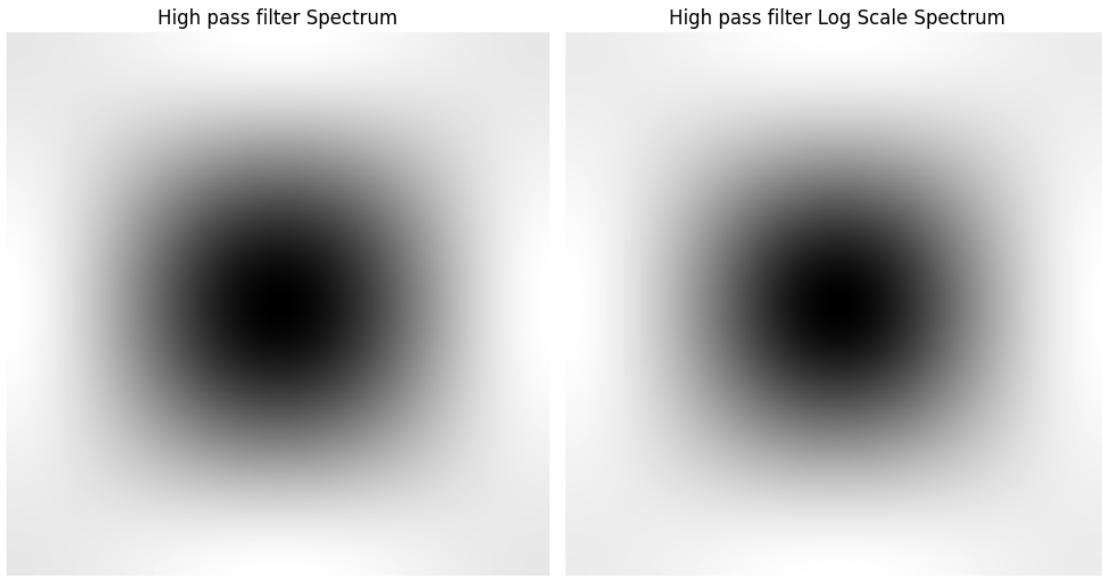
```
[ ]: showSpect(src=big, kernel=h_low, title='Low pass filter')
```



```
[ ]: showSpect(src=big, kernel=h_lowDisk, title='Low Disk pass filter')
```



```
[ ]: showSpect(src=big, kernel=h_high, title='High pass filter')
```



python-ex2

September 4, 2024

0.1 ex2.m

```
[ ]: import numpy as np
      import matplotlib.pyplot as plt
      import cv2
      from pathlib import Path

[ ]: from PIL import Image

[ ]: ASSETS_FOLDER_PATH = "../assets"
      OUTPUT_FOLDER_PATH = "."

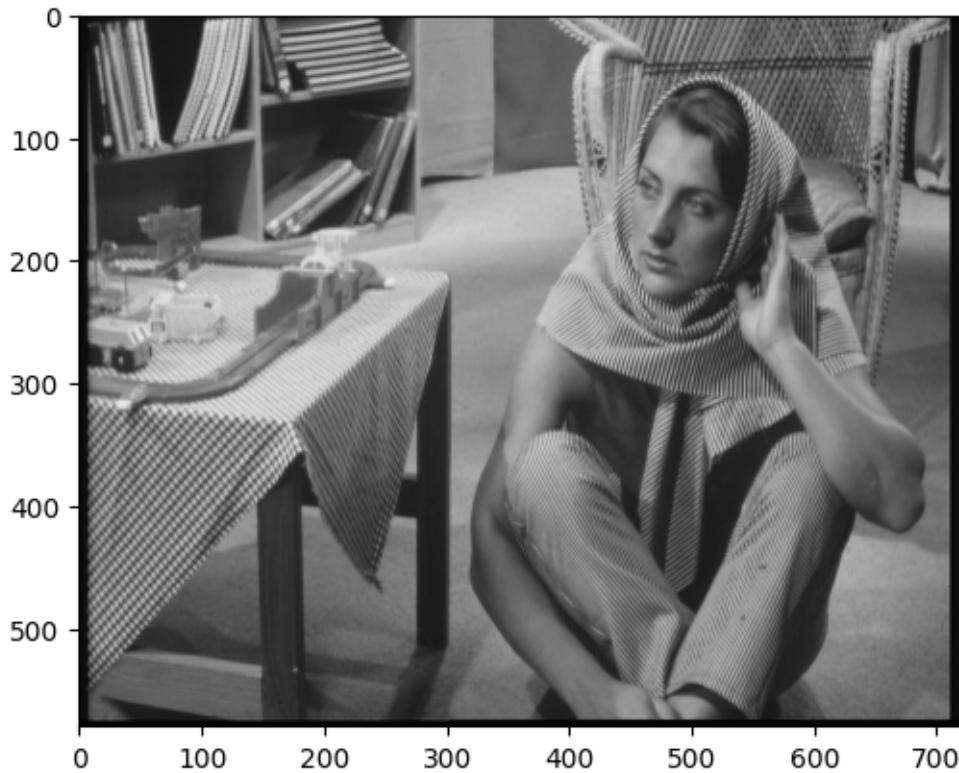
[ ]: Path(OUTPUT_FOLDER_PATH).mkdir(parents=True, exist_ok=True)
```

Cargamos la imagen

```
[ ]: b = Image.open(f"{ASSETS_FOLDER_PATH}/barbara.gif")

[ ]: plt.imshow(b, cmap="gray", vmin=0, vmax=255)

[ ]: <matplotlib.image.AxesImage at 0x77b4e854efb0>
```



```
[ ]: b = np.array(b)
```

Definimos un filtro

```
[ ]: h = np.array([
    [0, 1/6, 0],
    [1/6, 1/3, 1/6],
    [0, 1/6, 0]
])
```

```
[ ]: h = h / np.sum(h[:])
```

Definimos los filtros de disk y unsharp tal cual los usa matlab

```
[ ]: h_unsharp = np.array([
    [-1/6, -2/3, -1/6],
    [-2/3, 13/3, -2/3],
    [-1/6, -2/3, -1/6]
])

h_disk = np.array([
```

```

[      0,          0,          0, 67/53617,    3/604, 63/10064,    3/604, 67/
˓→53617,          0,          0,          0],
[      0, 4/124819, 94/15267, 61/4921, 113/8875, 113/8875, 113/8875, 61/
˓→4921, 94/15267, 4/124819,          0],
[      0, 94/15267, 113/8875, 113/8875, 113/8875, 113/8875, 113/
˓→8875, 113/8875, 94/15267,          0],
[ 67/53617, 61/4921, 113/8875, 113/8875, 113/8875, 113/8875, 113/
˓→8875, 113/8875, 61/4921, 67/53617],
[ 3/604, 113/8875, 113/8875, 113/8875, 113/8875, 113/8875, 113/8875, 113/
˓→8875, 113/8875, 113/8875, 3/604],
[ 63/10064, 113/8875, 113/8875, 113/8875, 113/8875, 113/8875, 113/8875, 113/
˓→8875, 113/8875, 113/8875, 63/10064],
[ 3/604, 113/8875, 113/8875, 113/8875, 113/8875, 113/8875, 113/8875, 113/
˓→8875, 113/8875, 113/8875, 3/604],
[ 67/53617, 61/4921, 113/8875, 113/8875, 113/8875, 113/8875, 113/8875, 113/
˓→8875, 113/8875, 61/4921, 67/53617],
[      0, 94/15267, 113/8875, 113/8875, 113/8875, 113/8875, 113/8875, 113/
˓→8875, 113/8875, 94/15267,          0],
[      0, 4/124819, 94/15267, 61/4921, 113/8875, 113/8875, 113/8875, 61/
˓→4921, 94/15267, 4/124819,          0],
[      0,          0,          0, 67/53617,    3/604, 63/10064,    3/604, 67/
˓→53617,          0,          0,          0]
])

```

Aplicamos el filtro

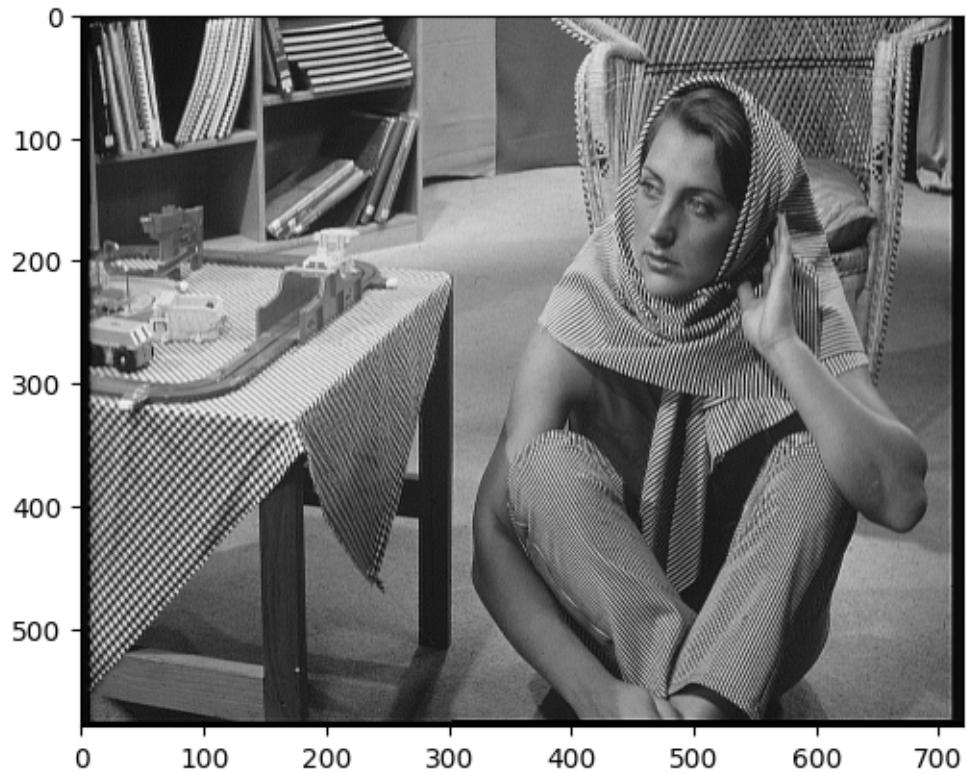
```

[ ]: h1 = cv2.filter2D(src=b, ddepth=-1, kernel=h_unsharp)
h2 = cv2.filter2D(src=b, ddepth=-1, kernel=h_disk)
h3 = cv2.filter2D(src=b, ddepth=-1, kernel=h)

[ ]: plt.imshow(h1, cmap="gray", vmin=0, vmax=255)

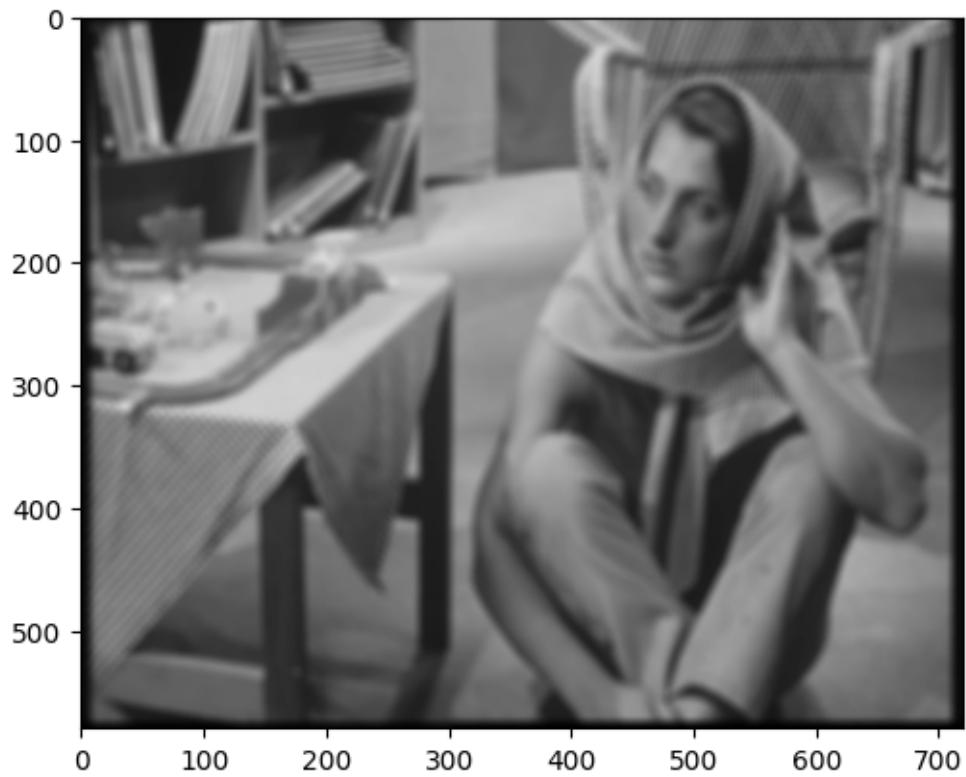
[ ]: <matplotlib.image.AxesImage at 0x77b4dffef490>

```



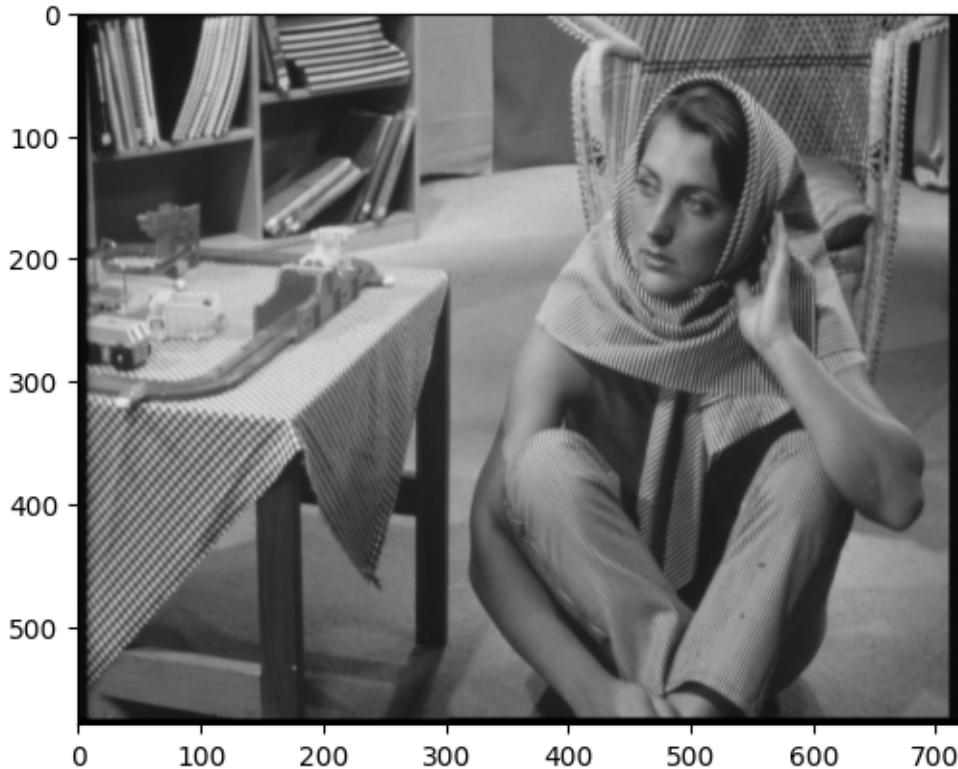
```
[ ]: plt.imshow(h2, cmap="gray", vmin=0, vmax=255)
```

```
[ ]: <matplotlib.image.AxesImage at 0x77b4ddd86350>
```



```
[ ]: plt.imshow(h3, cmap="gray", vmin=0, vmax=255)
```

```
[ ]: <matplotlib.image.AxesImage at 0x77b4dd3124d0>
```



```
[ ]: h1_image = Image.fromarray(h1)
h1_image.save(f"{OUTPUT_FOLDER_PATH}/python-ex2-unsharp.gif")
```

```
[ ]: h2_image = Image.fromarray(h2, mode="L")
h2_image.save(f"{OUTPUT_FOLDER_PATH}/python-ex2-disk.gif")
```

```
[ ]: h3_image = Image.fromarray(h3, mode="L")
h3_image.save(f"{OUTPUT_FOLDER_PATH}/python-ex2-other.gif")
```

Observamos que el disk vuelve más borrosa la imagen mientras que el unsharp acentua los cambios de intensidad. El último filtro, acentua las líneas verticales y horizontales pero pierden distinguibilidad las diagonales

python-spect

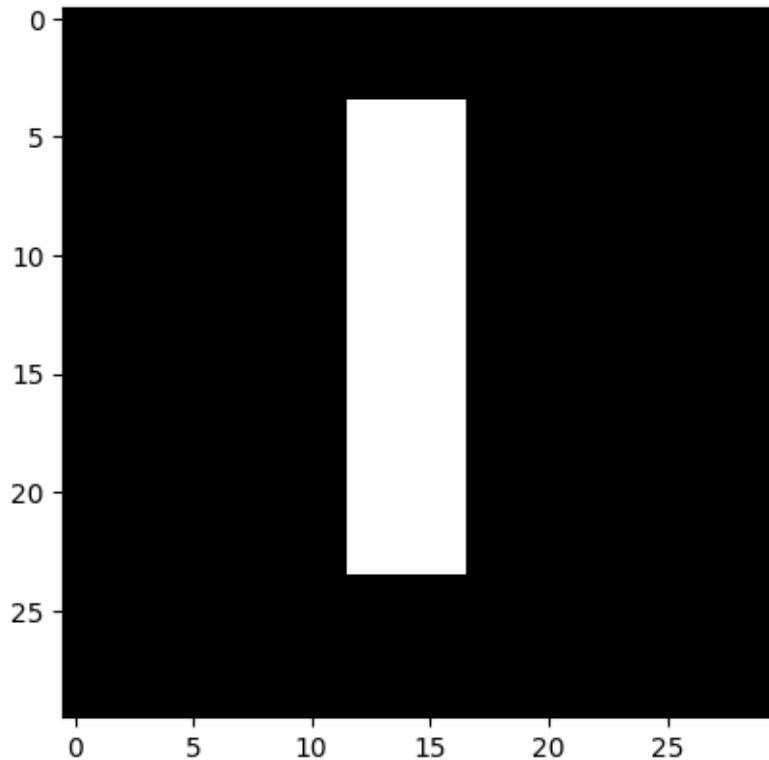
September 4, 2024

0.1 spect.m

```
[ ]: import numpy as np
      import matplotlib.pyplot as plt

[ ]: # Prepare image
      f = np.zeros((30, 30))
      f[4:24, 12:17] = 1  # Note: Python uses 0-based indexing so 5:24 converts into 4:24 for example
      plt.figure()
      plt.imshow(f, cmap='gray')
```

```
[ ]: <matplotlib.image.AxesImage at 0x701e3c0dd420>
```



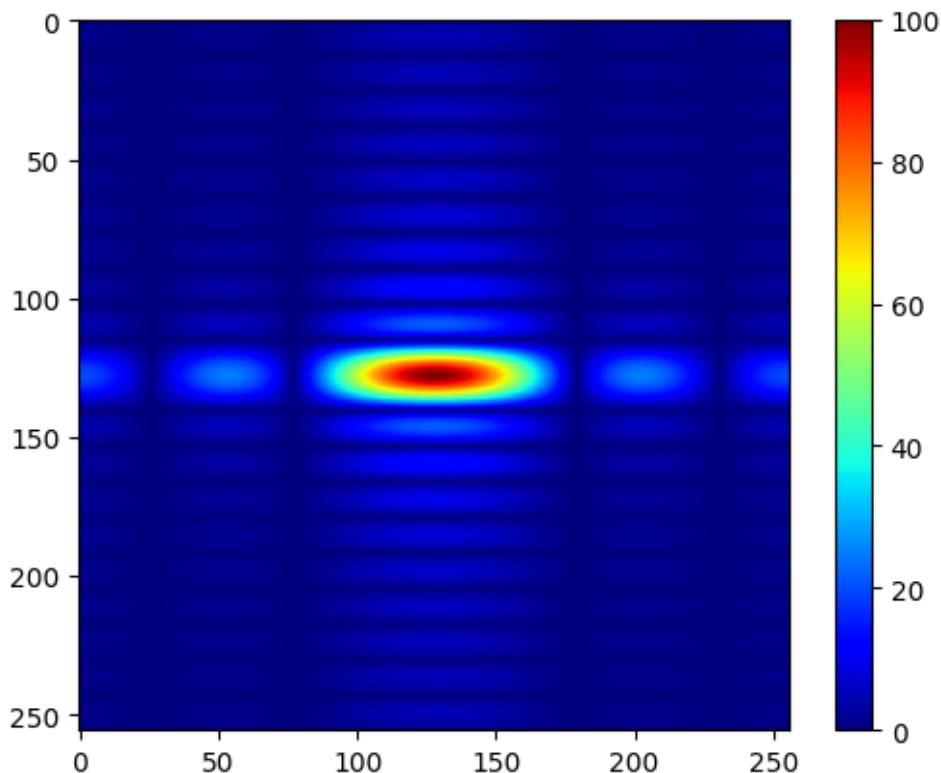
```
[ ]: # Compute Fourier Transform
F = np.fft.fft2(f, (256, 256))
F = np.fft.fftshift(F) # Center FFT
```

```
[ ]: # Measure the minimum and maximum value of the transform amplitude
print(f"Min amplitude: {np.min(np.abs(F))}")
print(f"Max amplitude: {np.max(np.abs(F))}")
```

Min amplitude: 0.0
Max amplitude: 100.0

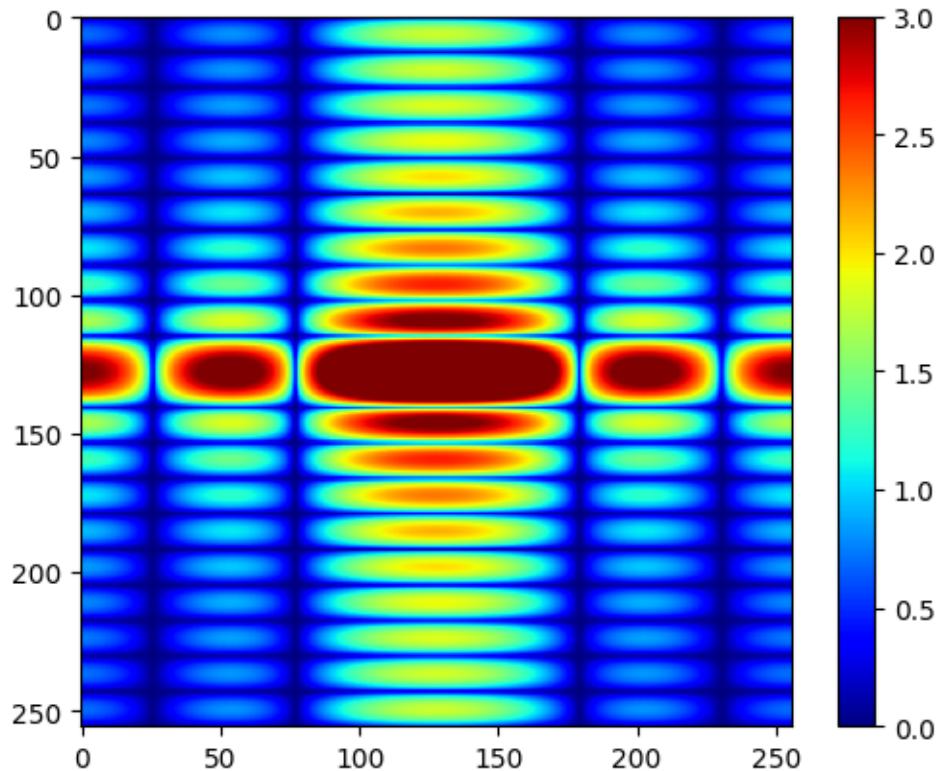
```
[ ]: # Show amplitude
plt.figure()
plt.imshow(np.abs(F), cmap='jet', vmin=0, vmax=100)
plt.colorbar()
```

```
[ ]: <matplotlib.colorbar.Colorbar at 0x701e3373f1c0>
```



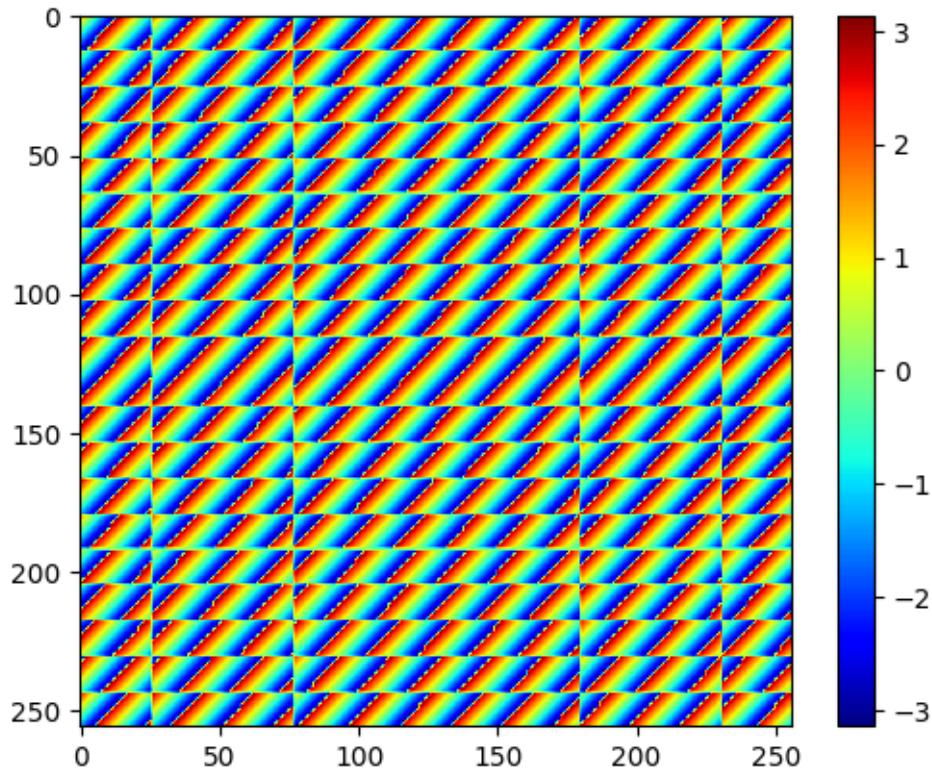
```
[ ]: # Show log amplitude
plt.figure()
plt.imshow(np.log1p(np.abs(F)), cmap='jet', vmin=0, vmax=3)
plt.colorbar()
```

```
[ ]: <matplotlib.colorbar.Colorbar at 0x701e3150fd90>
```



```
[ ]: # Show phases
plt.figure()
plt.imshow(np.angle(F), cmap='jet', vmin=-np.pi, vmax=np.pi)
plt.colorbar()
```

```
[ ]: <matplotlib.colorbar.Colorbar at 0x701e315d3dc0>
```



Lo que se analiza es una simple imagen de un rectangulo vertical blanco en el centro de un cuadrado negro a la cuál se le calcula la FFT y se grafica el modulo y el logaritmo de la misma.

Para explicar lo que sucede al graficar el espectro del módulo se debe primero recorrer la imagen original de forma horizontal y se puede apreciar un pulso de duración 4 en la zona blanca que si le hacemos la FFT es una sinc. Luego lo recorremos de forma vertical para ver otro pulso ahora de duración 19 por lo que el ancho del lóbulo principal de la sinc será mas angosto en esta dirección.

Luego se plotea el logaritmo de la FFT para poder conseguir mejor rango dinámico y poder diferenciar aún más los cambios.

python-viasampling

September 4, 2024

0.1 viasampling.m

```
[ ]: import numpy as np
      import matplotlib.pyplot as plt
      from PIL import Image
      from scipy import fftpack

[ ]: def samplingLoop(N, M, w, F):
      for i in range(N):
          for j in range(M):
              r2 = (i - N//2)**2 + (j - M//2)**2

              if r2 > round(N//2 * w)**2:
                  F[i, j] = 0
      return F

[ ]: def downsampling(I, m, filter_option):
      # Downsample the square image I by a factor of m
      N, M = I.shape

      # Apply ideal filter
      w = 1/m
      F = fftpack.fftshift(fftpack.fft2(I))

      if filter_option == 'FILTER_ON':
          F = samplingLoop(N, M, w, F)

      Idown = np.real(fftpack.ifft2(fftpack.ifftshift(F)))

      # Now downsample
      Idown = np.array(Image.fromarray(Idown).resize((N // m, M // m), Image.
      NEAREST))

      return Idown

[ ]: def upsampling(I, m):
      # Upsample the square image I by a factor of m
      N, M = I.shape
```

```

Iup = np.zeros((m*N, m*N))

# Expand input image
for i in range(N):
    for j in range(N):
        Iup[m*i, m*j] = I[i,j]

# Ideal filter
N, M = Iup.shape
w = 1/m
F = fftpack.fftshift(fftpack.fft2(Iup))

F = samplingLoop(N, M, w, F)

Iup = (m*m) * np.abs(fftpack.ifft2(fftpack.ifftshift(F)))

return Iup

```

```

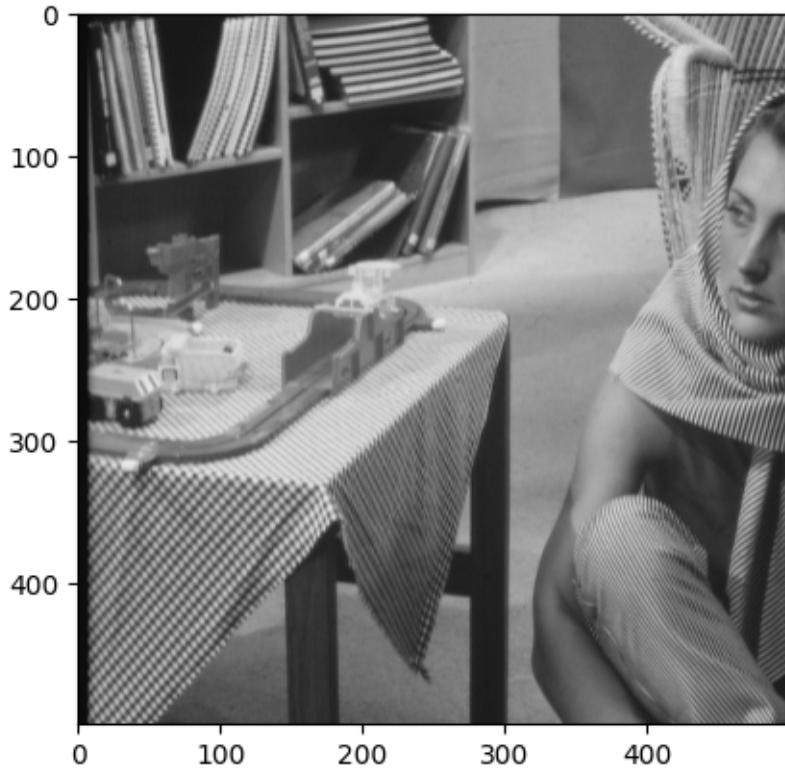
[ ]: m = 2

f1 = np.array(Image.open('../assets/barbara.gif').convert('L'))
f1 = f1[:500, :500]

plt.figure()
plt.imshow(f1, cmap='gray', vmin=0, vmax=255)

```

```
[ ]: <matplotlib.image.AxesImage at 0x79430f1be6b0>
```



```
[ ]: # Downsample
f2 = downsampling(f1, m, 'FILTER_ON')

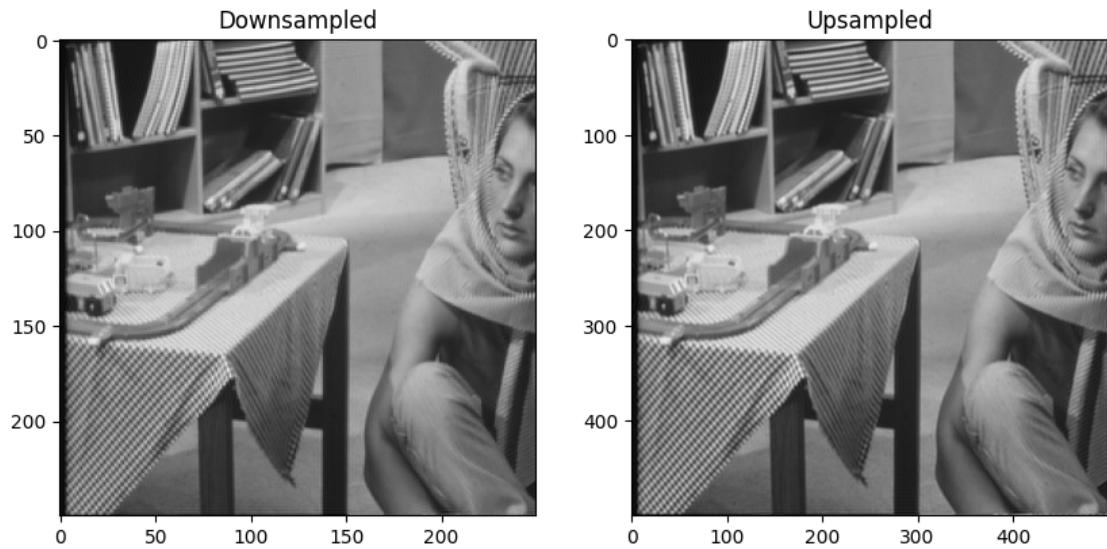
# Crear una figura con dos subgráficos
plt.figure(figsize=(10, 5)) # El tamaño puede ajustarse según se necesite

# Primer subgráfico (Imagen de downsampling)
plt.subplot(1, 2, 1)
plt.imshow(f2, cmap='gray', vmin=0, vmax=255)
plt.title("Downsampled")

# Upsample
f3 = upsampling(f2, m)

# Segundo subgráfico (Imagen de upsampling)
plt.subplot(1, 2, 2)
plt.imshow(f3, cmap='gray', vmin=0, vmax=255)
plt.title("Upsampled")

# Mostrar las imágenes
plt.show()
```



```
[ ]: # Downsample
f2 = downsampling(f1, m, 'FILTER_OFF')

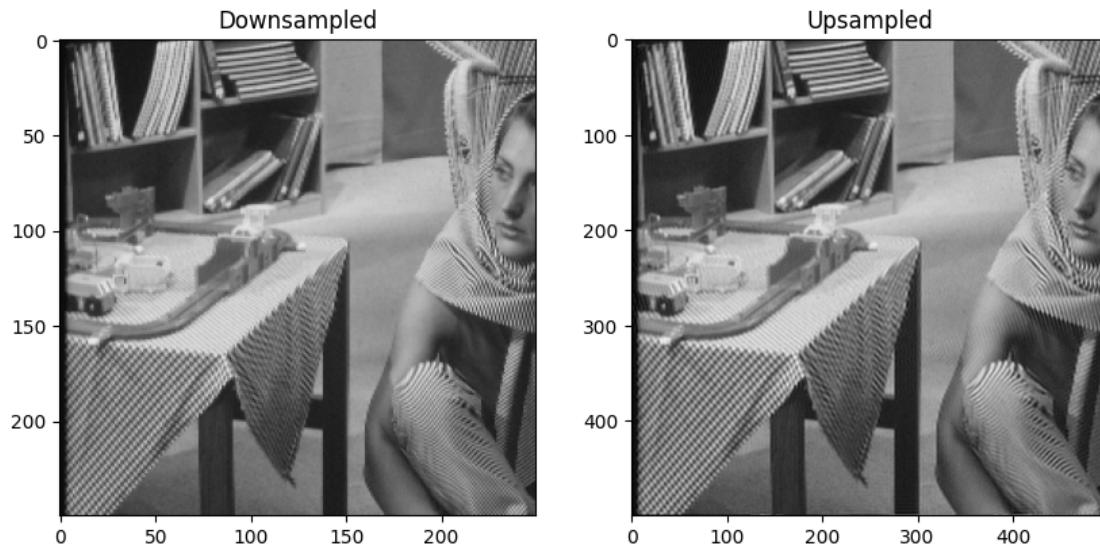
# Crear una figura con dos subgráficos
plt.figure(figsize=(10, 5)) # El tamaño puede ajustarse según se necesite

# Primer subgráfico (Imagen de downampling)
plt.subplot(1, 2, 1)
plt.imshow(f2, cmap='gray', vmin=0, vmax=255)
plt.title("Downsampled")

# Upsample
f3 = upsampling(f2, m)

# Segundo subgráfico (Imagen de upsampling)
plt.subplot(1, 2, 2)
plt.imshow(f3, cmap='gray', vmin=0, vmax=255)
plt.title("Upsampled")

# Mostrar las imágenes
plt.show()
```



Se toma una imagen original de 500x500 pixeles que se puede pensar como una señal sobre un espacio de 2D resultado de un muestreo espacial con frecuencia de muestreo desconocida.

Se plantean dos casos donde se hace un downsampling con factor 2, es decir que se reduce la frecuencia de muestreo espacial a la mitad, esto lo que hace es remover la mitad de las muestras de la imagen.

En el primer caso se aplica el filtro pasabajo previo al downsampling evitando así el efecto de aliasing pero generando una perdida de informacion de alta frecuencia de la informacion original.

En el segundo caso no se aplica el filtro pasabajo entonces al momento de ejecutar el downsampling se provoca el efecto de aliasing.

Finalmente en ambos casos se aplica el upsampling que busca aumentar la frecuencia de muestreo de la señal.

python-image_aliasing_new

September 5, 2024

0.1 image_aliasing_new.m

```
[ ]: import cv2
import numpy as np
import matplotlib.pyplot as plt
import scipy.fft as fft

[ ]: import skvideo
skvideo.setFFmpegPath("C:/Users/74005/Downloads/
↪ffmpeg-20210728-0068b3d0f0-win64-shared/bin")

[ ]: import skvideo.io
```

Definimos `write_video` para escribir una secuencia de frames en un archivo de video. Esta función ajusta el tamaño de cada frame para que todos tengan el mismo tamaño, rellenando con ceros si es necesario, y luego los escribe en un archivo de video utilizando la biblioteca `skvideo`.

```
[ ]: def write_video(file_path, frames):
    # Calculate the maximum height and width of all frames
    max_height = max(frame.shape[0] for frame in frames)
    max_width = max(frame.shape[1] for frame in frames)

    # Pad each frame to the maximum height and width
    padded_frames = [np.pad(frame, ((0, max_height - frame.shape[0]), (0, ↪
    ↪max_width - frame.shape[1]))) for frame in frames]

    skvideo.io.vwrite(file_path, np.array(padded_frames, dtype=np.uint8))
```

Con `centered_affine_transform` se aplica una transformación afín a una imagen, asegurando que la transformación esté centrada. Esta función calcula la nueva posición de la imagen transformada y aplica la transformación para obtener la imagen resultante.

```
[ ]: def centered_affine_transform(t, src_im):

    h, w = src_im.shape[:2]

    # Extract translation components from the transformation matrix
    trans = t[:-1]
```

```

inv_t = np.linalg.inv(t)
inv_trans = inv_t[:-1]

# Define source points for the transformation
src_pts = np.float32([[0, 0], [w-1, 0], [0, h-1], [w-1, h-1]])

# Apply the transformation to the source points
dst_pts = cv2.transform(np.array([src_pts]), trans)[0]

# Calculate the bounds of the transformed image
min_x, max_x = np.min(dst_pts[:, 0]), np.max(dst_pts[:, 0])
min_y, max_y = np.min(dst_pts[:, 1]), np.max(dst_pts[:, 1])

# Calculate the size of the transformed image
dst_w, dst_h = int(max_x - min_x + 1), int(max_y - min_y + 1)

# Calculate the center of the transformed image
dst_center = np.float32([(dst_w-1.0)/2, (dst_h-1.0)/2])

# Project the center of the transformed image back onto the source image
src_projected_center = cv2.transform(np.array([dst_center]), inv_trans)[0]

# Calculate the translation needed to center the transformation
translation = src_projected_center - np.float32([(w-1.0)/2, (h-1.0)/2])

# Update the translation components of the transformation matrix
trans[:, 2] = translation

# Apply the centered affine transformation to the image
return cv2.warpAffine(src_im, trans, (dst_w, dst_h))

```

Con `scale_shrink` definimos las matrices de la transformación:

```
[ ]: def scale_shrink(xshrink, xsize):
    scale_shrink = (xsize - xshrink) / xsize
    return np.array([
        [scale_shrink, 0, 0],
        [0, scale_shrink, 0],
        [0, 0, 1],
    ])
```

Con `scale_boost` creamos una matriz de transformación que vuelve a escalar la imagen a su tamaño original después de haber sido reducida por la función `scale_shrink`:

```
[ ]: def scale_boost(xsize, f2):
    scale_boost = xsize / f2.shape[1]
    return np.array([
        [scale_boost, 0, 0],
```

```
[0, scale_boost, 0],  
[0, 0, 1],  
])
```

Cargamos el gif e iteramos por los valores de xshrink:

```
[ ]: frames = []  
specs = []  
  
f = plt.imread('../assets/barbara.gif')  
  
for xshrink in range(0, 600, 5):  
    xsize = f.shape[1]  
    t1 = scale_shrink(xshrink, xsize)  
    f2 = centered_affine_transform(t1, f)  
    t2 = scale_boost(xsize, f2)  
    f3 = centered_affine_transform(t2, f2)  
    frames.append(f3[:, :, 0])  
    Fd = np.log(1 + np.abs(np.fft.fftshift(np.fft.fft2(frames[-1]))))  
    specs.append(np.uint8(256 * Fd / Fd.max()))
```

Creamos un video con los frames:

```
[ ]: write_video('aliasing_pics.avi', frames)
```

Creamos un video con los espectros de frecuencia de las imágenes de cada frame:

```
[ ]: write_video('aliasing_specs.avi', specs)
```

3. Ejercicio 3

3.1. Enunciado

Algunas cámaras digitales utilizan un optical low pass filter (OLPF) Investigar para que se utiliza. Todas las cámaras digitales lo tienen? Buscar ejemplos.

3.2. Respuesta

El Optical Low Pass Filter (OLPF) es un filtro optico que trabaja sobre la luz de la escena antes de que esta llegue al sensor de las camaras digitales para evitar o minimizar los efectos del aliasing. Un filtro pasa bajo lo que hace es dejar pasar la informacion de baja frecuencia y bloquear la de alta frecuencia; esto es justamente lo que hace el OLPF.

Es importante el uso del este tipo de filtros dado que una vez que llega la imagen, si esta tiene aliasing ya no se puede recuperar la imagen original.

Esto ademas de sus ventajas tiene desventajas tales como que se aplica siempre independientemente si la escena lo necesita o no y que al filtrar las altas frecuencias se pierde la nitidez de la imagen. Por esta ultima desventaja, cada filtro OLPF se diseña con el compromiso de minimizar los efectos mas visibles del aliasing sin que afecte demasiado la nitidez.

Es por esto que no todos los filtros OLPF son iguales ni afectan a los sensores de la misma manera, de hecho hay distintas prioridades como la eliminacion del efecto de Moire para lo cual se usan filtros muy agresivos o la priorizacion de la nitidez para lo cual usan un filtro suave o directamente no lo usan.

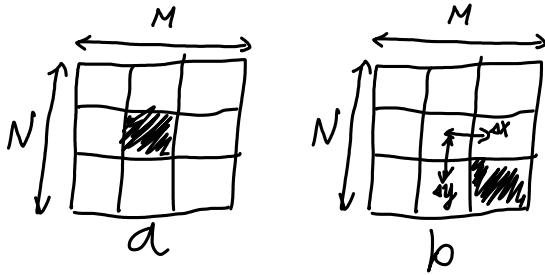
La Nikon D5200 utiliza un filtro OLPF fuerte para reducir el moiré, lo que causa una ligera pérdida de nitidez en su sensor de 24 Mp.

La Nikon D5300 elimina el filtro para priorizar la nitidez, pero aumenta el riesgo de aliasing. En las gamas altas, la Nikon D800 tiene filtro OLPF, mientras que la D800e no, lo que resulta en una diferencia notable de nitidez en sus sensores de 36 Mp.

En Canon, la EOS 700D y la EOS 7D comparten sensor, pero la 7D tiene un filtro OLPF más agresivo, causando una leve pérdida de contraste frente

a la 700D.

Sean a y b dos imágenes | b sea a desplazada un Δx y un Δy



I_i es la 2D-DFT de la imagen i

Por la propiedad de desplazamiento en Fase:

$$I_b = I_a e^{-2\pi i \left(\frac{u\Delta x}{M} + \frac{v\Delta y}{N} \right)}$$

Al conjugar I_b :

$$I_b^{(c)} = I_a^{(c)} e^{2\pi i \left(\frac{u\Delta x}{M} + \frac{v\Delta y}{N} \right)}$$

Entonces:

$$\beta(u, v) = \frac{I_a \otimes I_b^{(c)}}{|I_a \otimes I_b^{(c)}|} = \frac{I_a \otimes I_a^{(c)} \cdot e^{2\pi i \left(\frac{u\Delta x}{M} + \frac{v\Delta y}{N} \right)}}{|I_a \otimes I_a^{(c)} \cdot e^{2\pi i \left(\frac{u\Delta x}{M} + \frac{v\Delta y}{N} \right)}|}$$

$$= \frac{I_a \otimes I_a^{(c)}}{|I_a \otimes I_a^{(c)}|} \cdot e^{2\pi i \left(\frac{u\Delta x}{M} + \frac{v\Delta y}{N} \right)} = e^{2\pi i \left(\frac{u\Delta x}{M} + \frac{v\Delta y}{N} \right)}$$

$$|e^{2\pi i \cdot n}| = 1 \forall n \in \mathbb{Z}$$

$$r(x, y) = \text{IDFT} \left[e^{2\pi i \left(\frac{u\Delta x}{M} + \frac{v\Delta y}{N} \right)} \right]$$

$$r(x, y) = \delta(x + \Delta x, y + \Delta y)$$

Al ser una δ , $r(x, y) = 0 \forall x, y$ salvo en $(-\Delta x, -\Delta y)$

images_test

September 4, 2024

1 Ejercicio 4.2

```
[ ]: import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from scipy import fftpack

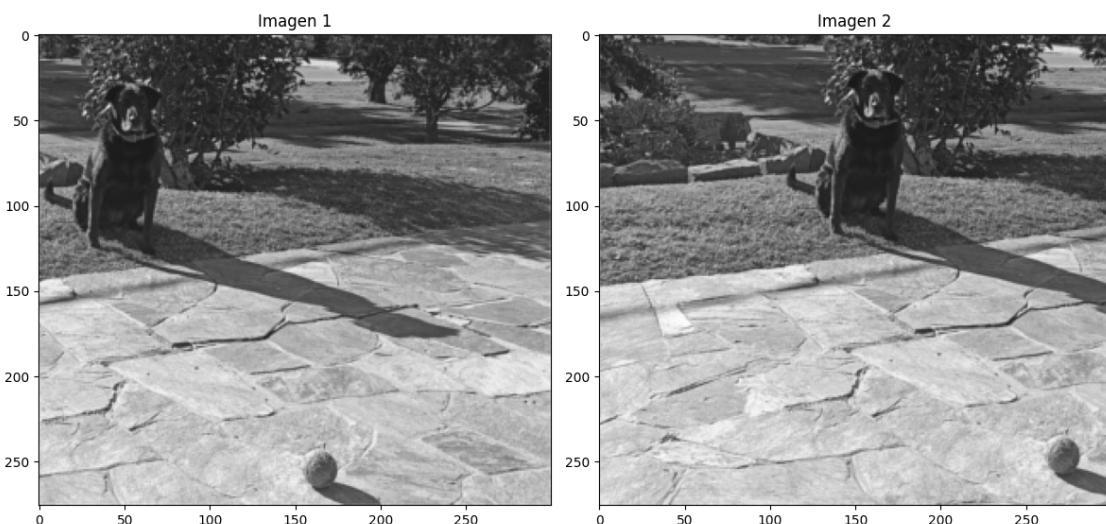
[ ]: imga = np.array(Image.open('../assets/dog1ss.jpg').convert("L"))
imgb = np.array(Image.open('../assets/dog2ss.jpg').convert('L'))

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.imshow(imga, cmap='gray', vmin=0, vmax=255)
plt.title("Imagen 1")

plt.subplot(1, 2, 2)
plt.imshow(imgb, cmap='gray', vmin=0, vmax=255)
plt.title("Imagen 2")

plt.tight_layout()
plt.show()
```



```
[ ]: # 2D-DFT
fft_a = fftpack.fft2(imga)
fft_b = fftpack.fft2(imgb)
# Conjugate
fft_b_c = np.conj(fft_b)
# Hadamart Product
result = np.multiply(fft_a, fft_b_c)
# Normalize
result_normalized = (result - result.min()) / (result.max() - result.min()) * 255
# R(u,v)
epsilon = 1e-10
R_u_v = np.divide(result, result_normalized + epsilon)
# DFT^-1
dft_inverse = fftpack.ifft(R_u_v)
# El DFT-1 me muestra el desplazamiento de cada pixel
```

(276, 300)

```
[ ]: # Tomar la magnitud
dft_magnitude = np.abs(dft_inverse)

# Encontrar el indice del valor máximo
max_idx = np.unravel_index(np.argmax(dft_magnitude), dft_magnitude.shape)

# Desplazamiento en x y y
deltay, deltax = max_idx

print(f"Desplazamiento en x: {deltax}")
print(f"Desplazamiento en y: {deltay}")
```

Desplazamiento en x: 193

Desplazamiento en y: 3