**Aim:** Simulate smoothing and sharpening operations on images using frequency domain filters.

**Theory:-**

* Smoothing and sharpening operations can also be performed in the frequency domain using filters such as the Gaussian filter and the Laplacian filter.

**Smoothing in Frequency Domain:-**

* To perform smoothing in the frequency domain, we apply a low-pass filter that attenuates high-frequency components. This helps to blur the image and reduce noise. One common approach is to use a Gaussian filter in the frequency domain.

**Sharpening in Frequency Domain:-**

* To perform sharpening in the frequency domain, we apply a high-pass filter that enhances high-frequency components. This amplifies the edges and fine details in the image. One common approach is to use a combination of a high-pass filter and the original image.

**Programm:-**

import cv2

import numpy as np

import matplotlib.pyplot as plt

def apply\_gaussian\_filter(*image*, *sigma*):

    image = np.float32(image)

    frequency\_domain = cv2.dft(image, *flags*=cv2.DFT\_COMPLEX\_OUTPUT)

    shifted\_frequency\_domain = np.fft.fftshift(frequency\_domain)

    rows, cols = image.shape

    crow, ccol = rows // 2, cols // 2

    mask = np.zeros((rows, cols, 2), np.float32)

    for i in range(rows):

        for j in range(cols):

            distance\_squared = (i - crow) \*\* 2 + (j - ccol) \*\* 2

            gaussian\_value = np.exp(-distance\_squared / (2 \* sigma\*\*2))

            mask[i, j] = [gaussian\_value, gaussian\_value]

    filtered\_frequency\_domain = shifted\_frequency\_domain \* mask

    shifted\_filtered\_frequency\_domain = np.fft.ifftshift(filtered\_frequency\_domain)

    filtered\_image = cv2.idft(

        shifted\_filtered\_frequency\_domain, *flags*=cv2.DFT\_SCALE | cv2.DFT\_REAL\_OUTPUT

    )

    filtered\_image = cv2.normalize(filtered\_image, None, 0, 255, cv2.NORM\_MINMAX)

    return np.uint8(filtered\_image)

def apply\_sharpening\_filter(*image*, *strength*):

    image = np.float32(image)

    frequency\_domain = cv2.dft(image, *flags*=cv2.DFT\_COMPLEX\_OUTPUT)

    shifted\_frequency\_domain = np.fft.fftshift(frequency\_domain)

rows, cols = image.shape

    crow, ccol = rows // 2, cols // 2

    mask = np.ones((rows, cols, 2), np.float32)

    mask[crow - strength : crow + strength, ccol - strength : ccol + strength] = 0

    filtered\_frequency\_domain = shifted\_frequency\_domain \* mask

    shifted\_filtered\_frequency\_domain = np.fft.ifftshift(filtered\_frequency\_domain)

    filtered\_image = cv2.idft(

        shifted\_filtered\_frequency\_domain, *flags*=cv2.DFT\_SCALE | cv2.DFT\_REAL\_OUTPUT

    )

    filtered\_image = cv2.normalize(filtered\_image, None, 0, 255, cv2.NORM\_MINMAX)

    return np.uint8(filtered\_image)

image\_path = "./Images.jpg"

input\_image = cv2.imread(image\_path, 0)

if input\_image is None:

    raise FileNotFoundError(

        f"The image '{image\_path}' could not be loaded. Check the file path."

    )

sigma = 20

smoothed\_image = apply\_gaussian\_filter(input\_image, sigma)

strength = 20

sharpened\_image = apply\_sharpening\_filter(input\_image, strength)

combined\_image = np.hstack((input\_image, smoothed\_image, sharpened\_image))

plt.imshow(combined\_image, *cmap*="gray")

plt.title("Original | Smoothed | Sharpened")

plt.axis("off")

plt.show()

smoothed\_path = "smoothed\_image.jpg"

sharpened\_path = "sharpened\_image.jpg"

cv2.imwrite(smoothed\_path, smoothed\_image)

cv2.imwrite(sharpened\_path, sharpened\_image)

print(f"Smoothed image saved at: {smoothed\_path}")

print(f"Sharpened image saved at: {sharpened\_path}")

**Output:-**

|  |  |  |
| --- | --- | --- |
| **Original Image** | **Sharpened Image** | **Smoothed Image** |
|  |  |  |

**Conclusion :-**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_