CLASS : B.E. E &TC **SUBJECT: DIVP**

EXPT. NO. : 5 **DATE: 25/09/2020**

TITLE : TO PERFORM IMAGE ENHANCEMENT IN FREQUENCY DOMAIN

CO 1:	Apply the fundamentals of digital image processing to perform various operations on an image-enhancement in spatial domain/ frequency domain, image-restoration, image compression, video filtering and video compression on a given gray image. Examine the effect of varying the mask size and density of noise in an image and comment on the obtained results.
CO4:	Carry out experiments as an individual and in a team, comprehend and write a laboratory record and draw conclusions at a technical level.

AIM :

To implement the following filters using matlab

1. Smoothing Filter
2. Sharpening Filter.

SOFTWARES REQUIRED: Matlab 7.0 or above

THEORY:

5.1 Basics of DFT

Image enhancement approaches fall into two broad categories: spatial domain methods and frequency domain methods. The term *spatial domain* refers to the image plane itself, and approaches in this category are based on direct manipulation of pixels in an image.

Frequency domain processing techniques are based on modifying the Fourier transform of an image. The term *spatial domain* refers to the aggregate of pixels composing an image. Spatial domain methods are procedures that operate directly on these pixels.

$$F(u, v) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(ux/M + vy/N)}$$

$$f(x, y) \rightleftarrows F(u, v)$$

$$f(x, y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{j2\pi(ux/M + vy/N)}$$

Fig. 5.1: 2D DFT and IDFT

M, N : Image Size

x,y: Image pixel position

u,v: spatial frequency

Any signal can be represented as a linear combination of set of basic components which include Fourier Components (Sinusodial Pattern) and Fourier coefficients (Weighting factors assigned to the Fourier components). Spatial frequency is the frequency of the Fourier component

Real Part, Imaginary Part, Magnitude, Phase, Spectrum

Real part:	$R = \text{Real}(F)$
Imaginary part:	$I = \text{Imag}(F)$
Magnitude-phase representation:	$F(u,v) = \lvert F(u,v)\rvert e^{-j\phi(u,v)}$
Magnitude (spectrum):	$\lvert F(u,v)\rvert = \left[R^2(u,v) + I^2(u,v)\right]^{1/2}$
Phase (spectrum):	$\phi(u,v) = \tan^{-1}\left[\frac{I(u,v)}{R(u,v)}\right]$
Power Spectrum:	$P(u,v) = \lvert F(u,v)\rvert^2$

5 .2 Computation of 2D DFT

- To compute the 1D-DFT of a 1D signal $\mathbf{x}$ (as a vector):

$$\widetilde{\mathbf{x}} = \mathbf{F}_N \mathbf{x}$$

To compute the inverse 1D-DFT:

$$\mathbf{x} = \frac{1}{N}\mathbf{F}_N^{*}\widetilde{\mathbf{x}}$$

- To compute the 2D-DFT of an image $\mathbf{X}$ (as a matrix):

$$\widetilde{\mathbf{X}} = \mathbf{F}_N \mathbf{X} \mathbf{F}_N$$

To compute the inverse 2D-DFT:

$$\mathbf{X} = \frac{1}{N^2}\mathbf{F}_N^{*}\widetilde{\mathbf{X}}\mathbf{F}_N^{*}$$

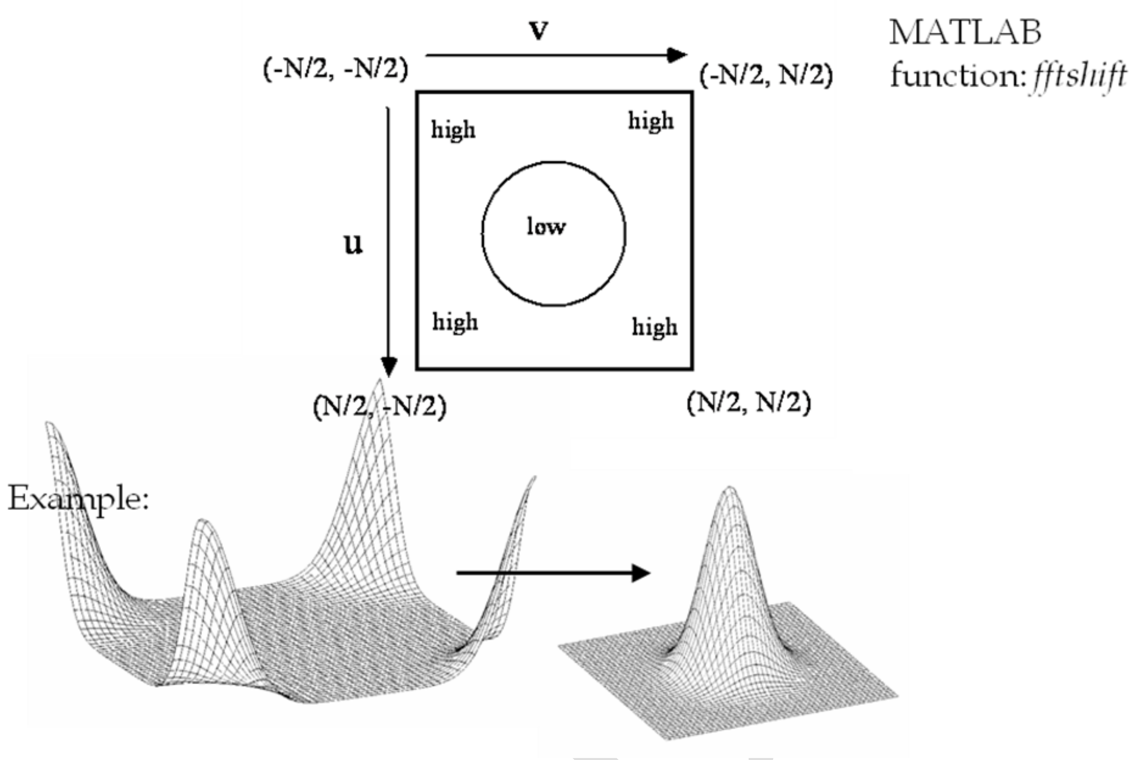


Fig. 5.2 : FFT Shift

To shift the low frequency at the centre, FFT shift is required. To see the minute details of frequencies log transform is used.

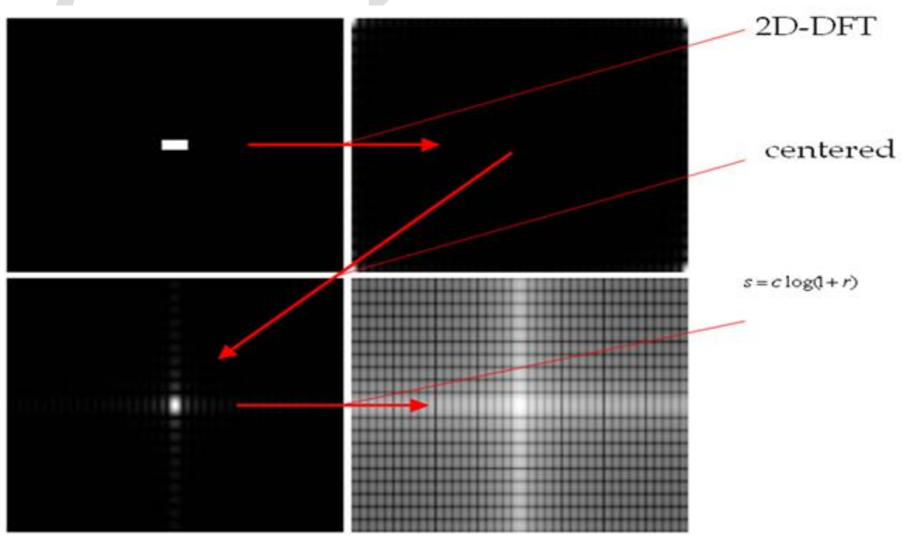


Fig. 5.3: Log magnitude visualization

5.3 Block diagram for frequency domain filtering

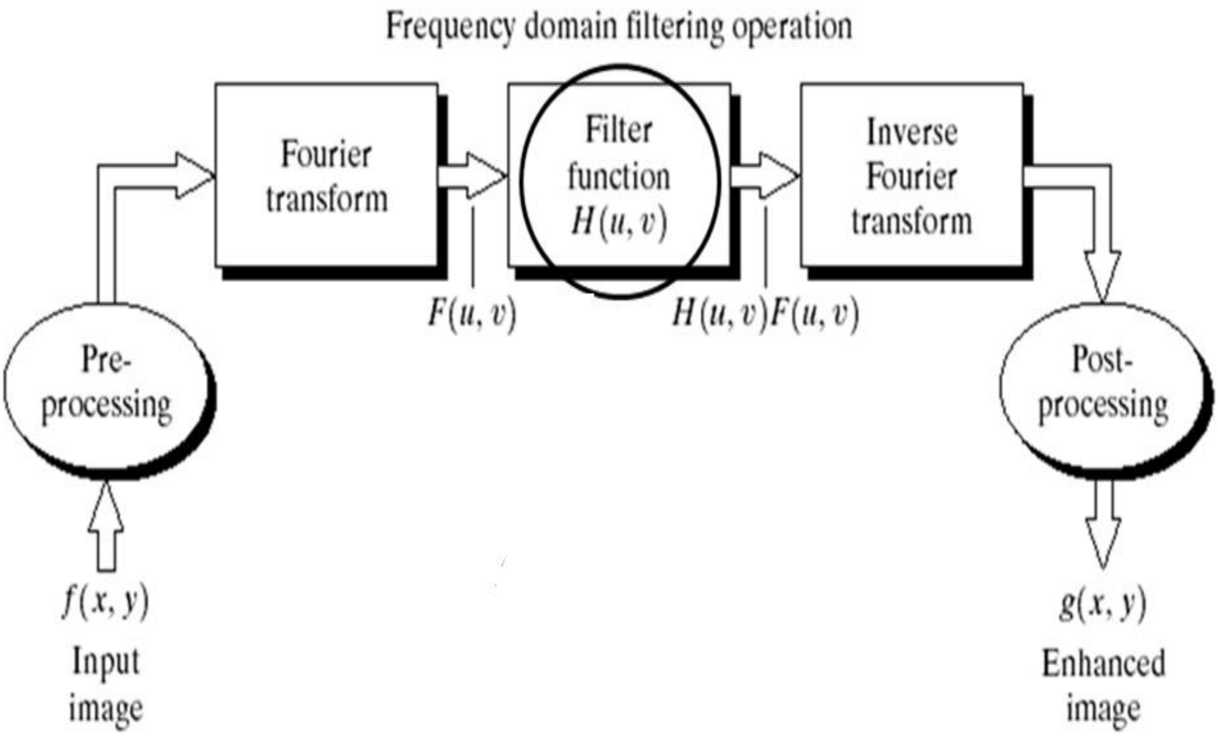


Fig. 5.4: Basic steps for filtering in the frequency domain

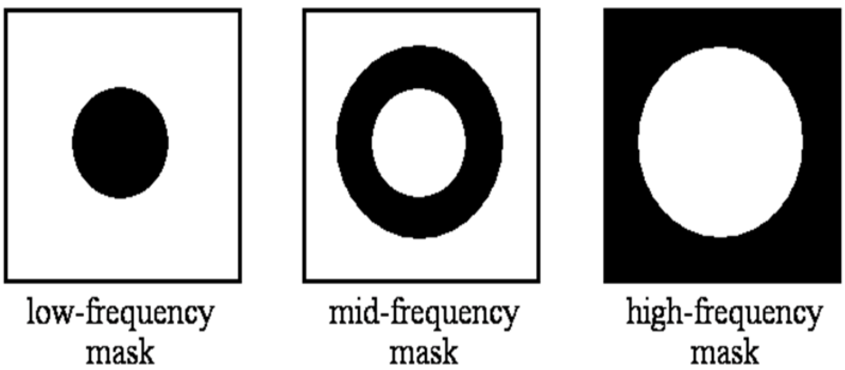
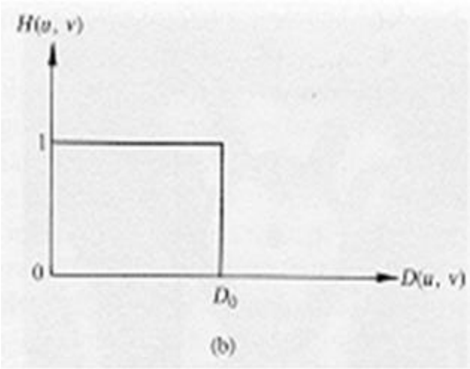
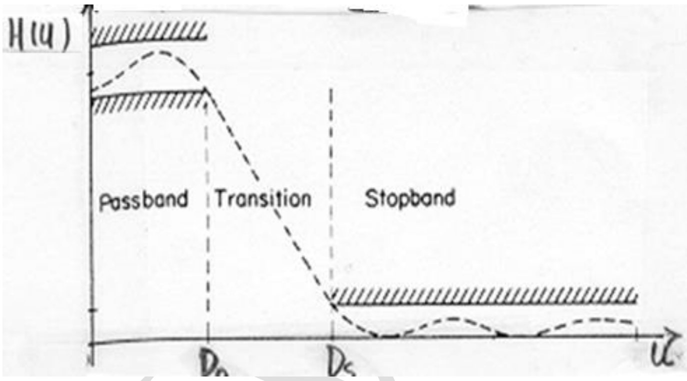


Fig 5.5: Ideal Low pass, Band pass and High pass filter masks

5.5 Low-Pass Filter:





(a) (b)

$$H(u,v) = \begin{cases} 1 & \text{if } 0 \le u \le D_0 \\ 0 & otherwise \end{cases}$$

Do :- Cutoff frequency

Fig 5.6 (a) Ideal and (b) Practical response of LPF

The ideal low-pass filter smoothes out the image, and is good for removing noise. The edges remain fairly sharp (better than mean filter).But it creates "ringing" artifacts around the edges. This is due to the sharp 0-1 transition in the filter and is called the *Gibbs phenomenon*.

$$H(u,v) = \begin{cases} 1 & \text{if } u^2 + v^2 \le D_0^{\,2} \\ 0 & otherwise \end{cases}$$

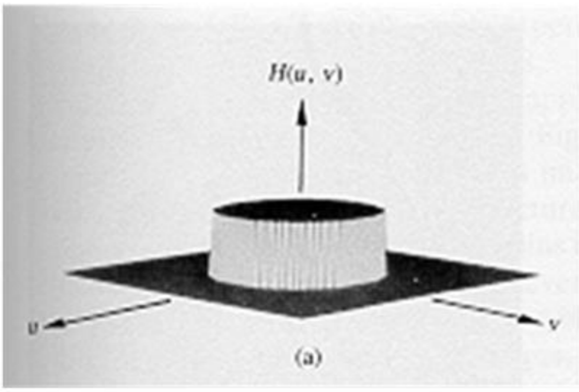
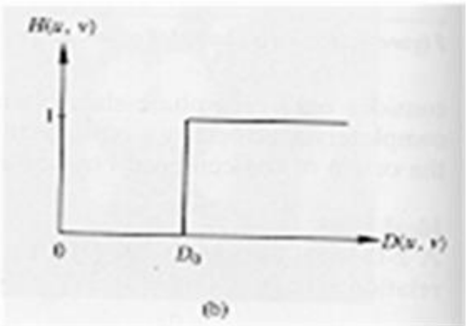
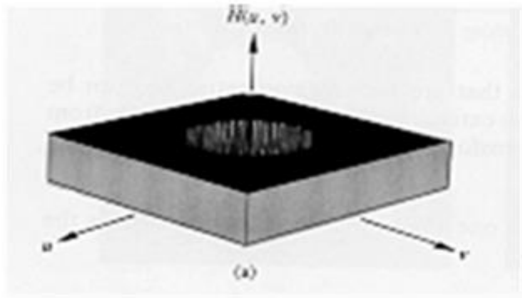


Fig 5.7: Cutoff frequencies lie on a circle

5.6 High-Pass Filter:

High pass filter preserves high frequencies and attenuates Lower frequencies**.**





$$H(u,v) = \begin{cases} 1 & \text{if} u \geq D_0 \\ 0 & otherwise \end{cases}$$

$$H(u,v) = \begin{cases} 1 & \text{if} u^2 + v^2 \geq {D_0}^2 \\ 0 & otherwise \end{cases}$$

Fig 5.6 (a) Ideal and (b) Practical response of LPF

5.7 Algorithm:

1. START
2. Take the input image from the user
3. Take 2D Fourier Transform of the image.
4. Perform Frequency shift
5. Apply the appropriate filter as per the user's requirement
6. Perform Frequency Shift
7. Take the Inverse Fourier Transform of the filtered image
8. Display the new filtered image.
9. END

5.8 Conclusion:

In this experiment, I learnt about filtering in frequency domain. I also learned to take the Fourier Transform, Frequency Domain, Applying the filters, and taking the Inverse Fourier Transform of these images. I implemented the code for High Pass and Low Pass Filter using Python and have observed how the different filters Sharpen and Blur the image respectively.

5.9 References:

i. Gonzalez R, Woods R, "Digital image processing", Pearson Prentice Hall, 2008.

ii. Gonzalez R, Woods R, Steven E, "Digital Image Processing Using MATLAB®", McGraw Hill Education, 2010.

iii. Jayaraman S, Esakkirajan S and Veerakumar T, "Digital Image Processing" Tata McGraw Hill, 2010

iv. Joshi, Madhuri A. "Digital Image Processing: an algorithm approach", PHI Learning Pvt. Ltd., 2006.

v. Pictures taken from: http://www.imageprocessingplace.com/root_files_V3/image_databases.html

____

(Course Teacher)

CLASS	**:**	**B.E (E &TC)**	**COURSE**	**:**	**DIVP**
AY	**:**	**2020-21 (SEM- I)**	**DATE**	**:**	**25/09/2020**
EXPT. NO.	**:**	**5**	**CLASS & ROLL NO**	**:**	**BE VIII 42410**
TITLE	**:**	**TO PERFORM IMAGE ENHANCEMENT IN FREQUENCY DOMAIN**			

I. LOW-PASS FILTER CODE:

```
# -*- coding: utf-8 -*-
import cv2
import numpy as np

def low_pass(img):
    #filter design
    Do = 50 #value of Cutoff freq Do
    ham = np.hamming(500)[:,None] # 1D hamming
    ham2d = np.sqrt(np.dot(ham, ham.T)) ** Do # expand to 2D hamming

    #Transforming the image to freq domain, output has 2 parts: real and imaginary
    f = cv2.dft(img.astype(np.float32), flags=cv2.DFT_COMPLEX_OUTPUT)
    #Since the center of image does not coincide with the origin
    #we have to handle this problem with np.fft.fftshift() function
    #What this function does is just divide an image into four small images
    #and then rearrange them such that it becomes symmetric about the center.
    f_shifted = np.fft.fftshift(f)
    f_complex = f_shifted[:,:,0]*1j + f_shifted[:,:,1]

    #applying the filter to the image
    f_filtered = ham2d * f_complex

    #taking inverse FT
    f_filtered_shifted = np.fft.fftshift(f_filtered)

    cv2.imshow('output',img)
    cv2.waitKey(30000)#30 secs
    cv2.destroyAllWindows()

    inv_img = np.fft.ifft2(f_filtered_shifted)
    filtered_img = np.abs(inv_img)
    filtered_img -= filtered_img.min()
    #expand the result such that all values are between 0 and 255
    filtered_img = filtered_img*255 / filtered_img.max()
    #convert back to uint8
    filtered_img = filtered_img.astype(np.uint8)
    final_img=cv2.hconcat([img,filtered_img])
    display_img(final_img)
```

```
def display_img(img):

    cv2.imshow('output',img)
    cv2.waitKey(30000)#30 secs
    cv2.destroyAllWindows()

def main():
    path = "C:/Users/Danish/Desktop/DIVP/presentation/einstein.jpg"
    img = cv2.imread(path,0) # gray-scale image
    img = img[:500, :500] # crop to 500 x 500
    low_pass(img)

main()
```

II. <u>HIGH-PASS FILTER CODE:</u>

```
# -*- coding: utf-8 -*-
import cv2
import numpy as np
# Image read
path = "C:/Users/Danish/Desktop/DIVP/presentation/einstein.jpg"
img = cv2.imread(path, 0)
img = cv2.resize(img, (500, 500))
cv2.imshow('img', img)
cv2.waitKey(0)
size = img.shape[0]
# Cut off Frequency
Do = 50
# High pass Filter using Distance Matrix
def FilterDesign(img, size, Do):
    # D is distance Matrix
    D = np.zeros([size, size], dtype=np.uint32)
    # H is Filter
    H = np.zeros([size, size], dtype=np.uint8)
    r = img.shape[0] // 2
    c = img.shape[1] // 2
    # Distance Vector
    for u in range(0, size):
        for v in range(0, size):
            D[u, v] = abs(u - r) + abs(v - c)
    # Using Cut off frequncy applying 0 and 255 in H to make a High Pass Filter and center = 1
    for i in range(size):
        for j in range(size):
            if D[i, j] > Do:
                H[i, j] = 255
            else:
```

```
                H[i, j] = 0
    return H
# High Pass Filter
H = FilterDesign(img, size, Do)
cv2.imshow('Rectangular High Pass Filter', H)
cv2.waitKey(0)
# Applying fft and shift
input = np.fft.fftshift(np.fft.fft2(img))
# Multiplying image with Low Pass Filter
out = input*H
# Taking Inverse Fourier of image
out = np.abs(np.fft.ifft2(np.fft.ifftshift(out)))
out = np.uint8(cv2.normalize(out, None, 0, 255, cv2.NORM_MINMAX, -1))
# Gradient image after applying High pass filter
cv2.imshow('High Pass Filtered Image', out)
cv2.waitKey(30000)#30 secs
cv2.destroyAllWindows()
```

III. LOW_PASS FILTER RESULTS

Input Image

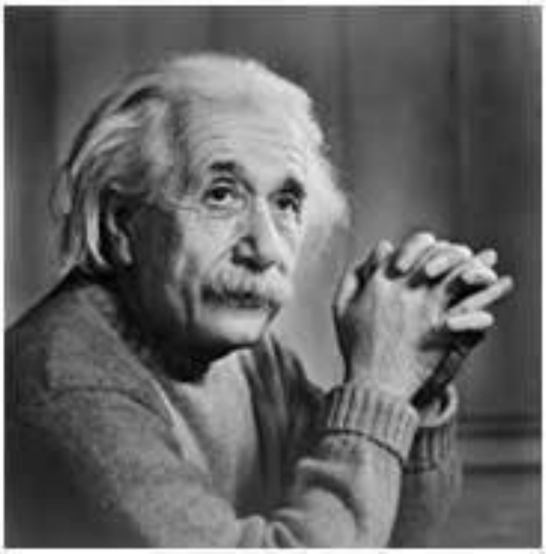
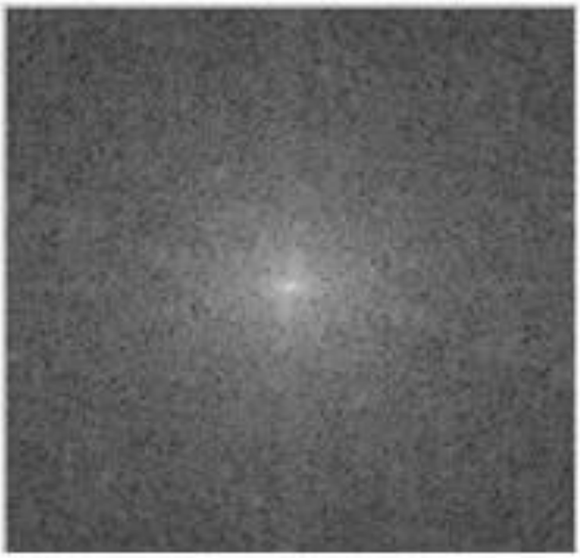
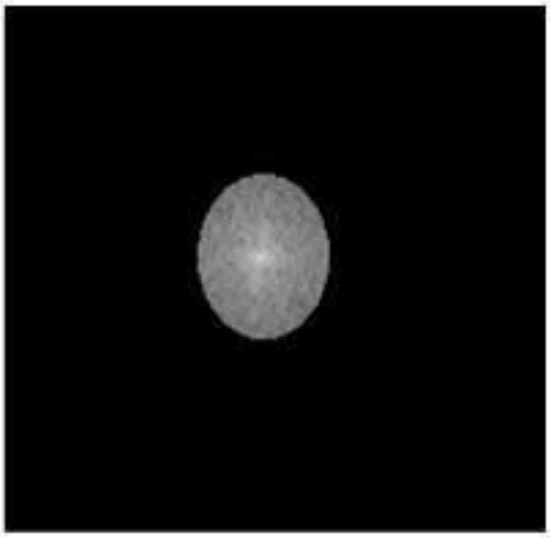
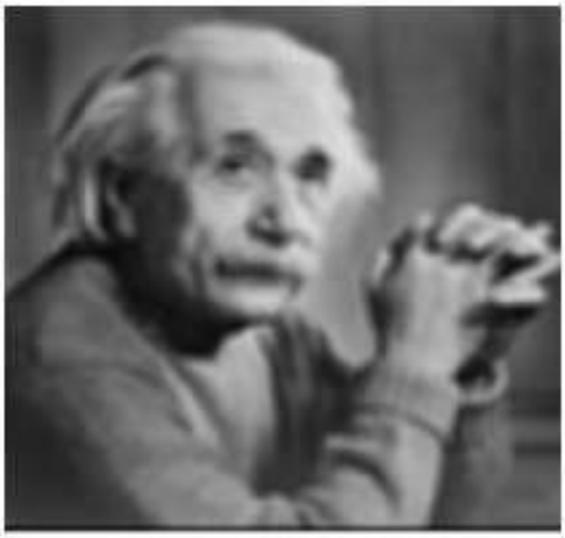


Image in frequency domain



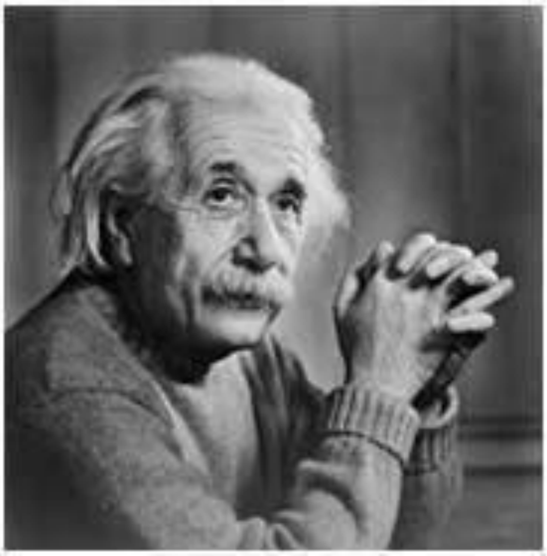
After shifting and Applying LPF over the image Resultant image after taking IDFT

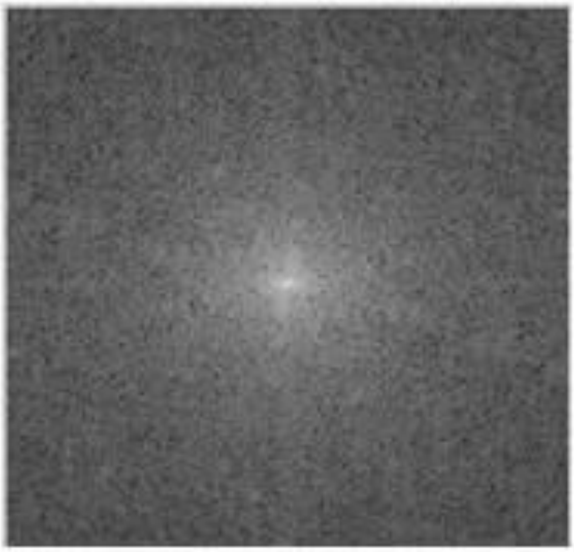




IV. HIGH_PASS FILTER RESULTS

Input Image Image in frequency domain





After shifting and Applying HPF over the image   Resultant image after taking IDFT