**1) Implement the following procedure, for smoothing the gray scale image using Fourier transformation algorithm.**

**a) Convert the gray scale image to the frequency Fourier transform (FFT)**

**b) Create a circular low pass filter (LPF).**

**c) Show inverse Fourier transform.**

**d) Generate the smoothened image.**

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


img = cv2.imread('lena.tif', 0)

dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)

dft_shift = np.fft.fftshift(dft)

magnitude_spectrum = 20 * np.log(cv2.magnitude(dft_shift[:, :, 0],
dft_shift[:, :, 1]))

rows, cols = img.shape
crow, ccol = int(rows / 2), int(cols / 2)
mask = np.ones((rows, cols, 2), np.uint8)
r = 50
center = [crow, ccol]
x, y = np.ogrid[:rows, :cols]
mask_area = (x - center[0]) ** 2 + (y - center[1]) ** 2 <= r*r
mask[mask_area] = 0

fshift = dft_shift * mask
fshift_mask_mag = 20 * np.log(cv2.magnitude(fshift[:, :, 0], fshift[:,
:, 1]))
f_ishift = np.fft.ifftshift(fshift)

img_back = cv2.idft(f_ishift)
img_back = cv2.magnitude(img_back[:, :, 0], img_back[:, :, 1])


fig = plt.figure(figsize=(12, 12))
ax1 = fig.add_subplot(2,2,1)
ax1.imshow(img, cmap='gray')
```

```
ax1.title.set_text('Input Image')
ax2 = fig.add_subplot(2,2,2)
ax2.imshow(magnitude_spectrum, cmap='gray')
ax2.title.set_text('FFT of image')
ax3 = fig.add_subplot(2,2,3)
ax3.imshow(fshift_mask_mag, cmap='gray')
ax3.title.set_text('FFT + Mask')
ax4 = fig.add_subplot(2,2,4)
ax4.imshow(img_back, cmap='gray')
ax4.title.set_text('After inverse FFT')
plt.show()
```

**2) Implement the following procedure for the Compression of gray scale image using bit plane slicing.**

**a) Convert the input image to the binary form.**

**b) Show the binary equivalent of the pixel, which is stored in the list [] (25000, 25499 1000, 8788).**

**c) Extract the bit-plane from the binary equivalent of image**

**d) For compression – apply stack for merging the top three MSB.**

**e) For compression – apply stack for merging the bottom three LSB.**

```
import numpy as np
import cv2
img = cv2.imread('img.jfif', 0)
cv2.imshow('input image', img)

list = []   #array
for i in range(img.shape[0]):   #two for loop - traverse the row and
column of image
    for j in range(img.shape[1]):
        list.append(np.binary_repr(img[i][j], width=8))
eight_bit_plane = (np.array([int(i[0]) for i in list], dtype = np.uint8)
* 128).reshape(img.shape[0], img.shape[1])
seven_bit_plane = (np.array([int(i[1]) for i in list], dtype = np.uint8)
* 64).reshape(img.shape[0], img.shape[1])
six_bit_plane = (np.array([int(i[2]) for i in list], dtype = np.uint8) *
32).reshape(img.shape[0], img.shape[1])
five_bit_plane = (np.array([int(i[3]) for i in list], dtype = np.uint8)
```

```
* 16).reshape(img.shape[0], img.shape[1])
four_bit_plane = (np.array([int(i[4]) for i in list], dtype = np.uint8)
* 8).reshape(img.shape[0], img.shape[1])
three_bit_plane = (np.array([int(i[5]) for i in list], dtype = np.uint8)
* 4).reshape(img.shape[0], img.shape[1])
two_bit_plane = (np.array([int(i[6]) for i in list], dtype = np.uint8) *
2).reshape(img.shape[0], img.shape[1])
one_bit_plane = (np.array([int(i[7]) for i in list], dtype = np.uint8) *
1).reshape(img.shape[0], img.shape[1])

cv2.imshow('bit plane 7', cv2.normalize(eight_bit_plane,
np.zeros(img.shape),0,255, cv2.NORM_MINMAX))
cv2.imshow('bit plane 6', cv2.normalize(seven_bit_plane,
np.zeros(img.shape),0,255, cv2.NORM_MINMAX))
cv2.imshow('bit plane 5', cv2.normalize(six_bit_plane,
np.zeros(img.shape),0,255, cv2.NORM_MINMAX))
cv2.imshow('bit plane 4', cv2.normalize(five_bit_plane,
np.zeros(img.shape),0,255, cv2.NORM_MINMAX))
cv2.imshow('bit plane 3', cv2.normalize(four_bit_plane,
np.zeros(img.shape),0,255, cv2.NORM_MINMAX))
cv2.imshow('bit plane 2', cv2.normalize(three_bit_plane,
np.zeros(img.shape),0,255, cv2.NORM_MINMAX))
cv2.imshow('bit plane 1', cv2.normalize(two_bit_plane,
np.zeros(img.shape),0,255, cv2.NORM_MINMAX))
cv2.imshow('bit plane 0', cv2.normalize(one_bit_plane,
np.zeros(img.shape),0,255, cv2.NORM_MINMAX))

top_three = eight_bit_plane + seven_bit_plane + six_bit_plane
bottom_three = one_bit_plane + two_bit_plane + three_bit_plane

cv2.imshow('Input image', img)
cv2.imshow('bit plane 8,7,6', top_three)
cv2.imshow('bit plane 1,2,3', bottom_three)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

**3) Using Gaussian function add noise to the gray scale image. Display the probability density function. For the noisy the image apply the following the filter.**

**a) Gaussian filter b) bi-lateral filter.**

**Noise**

```
import cv2
```

```
import numpy as np
from scipy.stats.kde import gaussian_kde
import matplotlib.pyplot as plt
from PIL import Image, ImageFilter

img=cv2.imread('images.jpg', 0)
img= img/255
x, y=img.shape
mean =0
var=0.05
sigma = np.sqrt(var)
n=np.random.normal(loc=mean, scale=sigma, size=(x,y))
g=img+n
plt.imshow(g,cmap='gray')

plt.savefig('foo.jpg')
```

**Probability function**

```
kde = gaussian_kde(n.reshape(int(x*y)))
dist_space=np.linspace(np.min(n), np.max(n), 100)
plt.plot(dist_space, kde(dist_space))
plt.title('Probability density function')
plt.xlabel('Noise pixel value')
plt.ylabel('Frequency')
plt.show()
```

**Remove noise**

```
img = cv2.imread('foo.jpg')
gblur=cv2.GaussianBlur(img, (5,5), 0)
bil = cv2.bilateralFilter(img, 9, 75, 75)
plt.subplot(221),plt.imshow(img),plt.title('Input
Image'),plt.axis("off")
plt.subplot(222),plt.imshow(gblur),plt.title('Gaussian
Filter'),plt.axis("off")
plt.subplot(223),plt.imshow(bil),plt.title('Bi-lateral
Filter'),plt.axis("off")
```

**4) Detect the edges of the gray scale image using sobel operator, and show the following.**

**a) Obtain the vertical line**

**b) Obtain the horizontal line**

**c) Combine the vertical and horizontal line, if the combined image does not provide the expected image, how do you solve it?**

**d) Plot the input image, sobel x, sobel y, combine x and y in 2x2. Write your observation**

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
# Read image
image = cv2.imread('/lena.png')

gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)

vertical = cv2.Sobel(gray,cv2.CV_64F,1,0,ksize=5)


horizontal = cv2.Sobel(gray,cv2.CV_64F,0,1,ksize=5)


final=vertical+horizontal


fig = plt.figure(figsize=(10, 10))

ax1=fig.add_subplot(2, 2, 1)
ax1.imshow(gray)
ax1.title.set_text('gray image')

ax2=fig.add_subplot(2, 2, 2)
ax2.imshow(vertical)
ax2.title.set_text('vertical')
```

```
ax3=fig.add_subplot(2, 2, 3)
ax3.imshow(horizontal)
ax3.title.set_text('horizontal')

ax4=fig.add_subplot(2, 2, 4)
ax4.imshow(final)
ax4.title.set_text('final image')
```

**5)**

**A) Create the image of 200 x 200, all the pixels in the images are black (0), Plot the histogram using plt.hist () function. Add white pixels to the same image 200x200 (number of white pixels is your choice. Plot the histogram for the new image. Compare both the histogram and write your observation.**

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = np.zeros((200, 200), np.uint8)
cv.imshow('img', img)
cv.waitKey(0)
plt.hist(img.ravel(), 256, [0, 256])

plt.show()

#Adding Equal no of 0's and 1's
one = cv.rectangle(img, (0,100), (200,200), (255), -1)
cv.imshow('one',one)
cv.waitKey(0)
plt.hist(one.ravel(), 256, [0, 256])
```

**b) For the RGB image, split the image to R, G and B, determine the pixel intensity of each colors. Show the histogram for each of (R, G, B) image using plt.hist () function.**

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('Colorimage.jfif', 1)
b,g,r=cv.split(img)
```

```
cv.imshow("img", img)
cv.imshow("img", b)
cv.imshow("img", g)
cv.imshow("img", r)
plt.hist(b.ravel(), 256, [0,256])
plt.hist(g.ravel(), 256, [0,256])
plt.hist(r.ravel(), 256, [0,256])
plt.show()
```

**6) Detect the edges of the image using the following procedure and show the output of each step.**

**a) Apply Gaussian filter to remove the noise**

```
import cv2
import numpy as np
from scipy.stats.kde import gaussian_kde
import matplotlib.pyplot as plt
from PIL import Image, ImageFilter
```

```
img = cv2.imread('foo.jpg')
gblur=cv2.GaussianBlur(img, (5,5), 0)
plt.imshow(gblur,cmap='gray')
```

**b) Find the intensity gradient of the image**

```
vertical = cv2.Sobel(gblur,cv2.CV_64F,1,0,ksize=5)


horizontal = cv2.Sobel(gblur,cv2.CV_64F,0,1,ksize=5)


final=vertical+horizontal
plt.imshow(final)
```

**c) Adopt the non-maximum suppression**

**d) Double threshold to determine potential edges.**

```
img = cv2.imread("test.jpeg")  # Read image

# Setting parameter values
t_lower = 50  # Lower Threshold
t_upper = 150  # Upper threshold

# Applying the Canny Edge filter
edge = cv2.Canny(img, t_lower, t_upper)

cv2.imshow('original', img)
cv2.imshow('edge', edge)
```

**7) For segmenting the features of the image apply the K-mean algorithm using following procedure.**
**a) Number of clusters are 5, 4, 3 (generate output for this number, write your observation)**
**b) Maximum iteration is 5.**
**c) Epsilon is 2.0**

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

img = cv2.imread("Gray.jpg")

img2 = img.reshape((-1,3))

img2 = np.float32(img2)

criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 5, 2.0)

k1 = 5
k2 = 4
k3 = 3
attempts = 3

ret1,label1,center1=cv2.kmeans(img2, k1, None, criteria, attempts,
cv2.KMEANS_PP_CENTERS)
```

```
ret2,label2,center2=cv2.kmeans(img2, k2, None, criteria, attempts,
cv2.KMEANS_PP_CENTERS)
ret3,label3,center3=cv2.kmeans(img2, k3, None, criteria, attempts,
cv2.KMEANS_PP_CENTERS)

center1 = np.uint8(center1)
center2 = np.uint8(center2)
center3 = np.uint8(center3)

res1 = center1[label1.flatten()]
res_1 = res1.reshape((img.shape))

res2 = center2[label2.flatten()]
res_2 = res2.reshape((img.shape))

res3 = center3[label3.flatten()]
res_3 = res3.reshape((img.shape))

titles = ['Original image', 'Segmentation image(k=5)', 'Segmentation
image(k=4)', 'Segmentation image(k=3)']
images = [img, res_1, res_2, res_3]

for i in range(4):
    plt.subplot(2, 2, i+1), plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]), plt.yticks([])

plt.show()
```

**8) Generate the salt and pepper noise using below procedure;**
**a) Create the blank image using numpy library,**
**b) Consider different pepper value is 0.09, 0.08 and 0.07.**
**c) Formula for salt is 1-pepper. c) For each pepper value (salt and pepper image) apply Gaussian filter b) bi-lateral filter. d) Write your Observation on output image of different value of pepper.**

```
import cv2
import numpy as np
from skimage import io, filters
from matplotlib import pyplot as plt

Oimg = cv2.imread('Gray.jpg', 0)
```

```
x, y = Oimg.shape
g = np.zeros((x,y), dtype=np.float32)

pepper1 = 0.09
pepper2 = 0.08
pepper3 = 0.07
salt1 = 1-pepper1
salt2 = 1-pepper2
salt3 = 1-pepper3

for i in range(x):
    for j in range(y):
        rdn = np.random.random()
        if rdn < pepper1:
            g[i][j] = 0
        elif rdn > salt1:
            g[i][j] = 1
        else:
            g[i][j] = Oimg[i][j]
cv2.imwrite('Image with noise 0.09.jpg', g)

for i in range(x):
    for j in range(y):
        rdn = np.random.random()
        if rdn < pepper2:
            g[i][j] = 0
        elif rdn > salt2:
            g[i][j] = 1
        else:
            g[i][j] = Oimg[i][j]
cv2.imwrite('Image with noise 0.08.jpg', g)

for i in range(x):
    for j in range(y):
        rdn = np.random.random()
        if rdn < pepper3:
            g[i][j] = 0
        elif rdn > salt3:
            g[i][j] = 1
        else:
            g[i][j] = Oimg[i][j]
cv2.imwrite('Image with noise 0.07.jpg', g)


img1=io.imread('Image with noise 0.09.jpg')
img2=io.imread('Image with noise 0.08.jpg')
```

```
img3=io.imread('Image with noise 0.07.jpg')


gauss_img1=cv2.GaussianBlur(img1,(5,5),0)
cv2.imwrite('gauss_img1.jpg',gauss_img1)

gauss_img2=cv2.GaussianBlur(img1,(5,5),0)
cv2.imwrite('gauss_img2.jpg',gauss_img2)

gauss_img3=cv2.GaussianBlur(img1,(5,5),0)
cv2.imwrite('gauss_img3.jpg',gauss_img3)

bilateral1 = cv2.bilateralFilter(img1, 15, 75, 75)
cv2.imwrite('bilateral1.jpg',bilateral1)

bilateral2 = cv2.bilateralFilter(img2, 15, 75, 75)
cv2.imwrite('bilateral2.jpg',bilateral2)

bilateral3 = cv2.bilateralFilter(img3, 15, 75, 75)
cv2.imwrite('bilateral3.jpg',bilateral3)
```

**9) Implement the following procedure of watershed algorithm for image segmentation.**

**a) Threshold the image using OTSU.**
**b) Apply Morphological opening to remove the noise.**
**c) Apply Morphological closing to remove the holes**
**d) Obtain the sure background image**
**e) Obtain the sure foreground using distance transform, print the maximum value of distance transform.**
**f) Finally, generate the watershed image.**
**g) Plot threshold, opening, sure background, distance transform, sure foreground, watershed image**

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from scipy import ndimage
from skimage import measure, color, io

img1 = cv2.imread("/lena.png")
img = cv2.cvtColor(img1,cv2.COLOR_BGR2GRAY)
ret1, thresh = cv2.threshold(img, 0, 255,
cv2.THRESH_BINARY+cv2.THRESH_OTSU)
```

```
kernel = np.ones((3,3),np.uint8)
opening = cv2.morphologyEx(thresh,cv2.MORPH_OPEN,kernel, iterations =
1)
from skimage.segmentation import clear_border
opening = clear_border(opening)
sure_bg = cv2.dilate(opening,kernel,iterations=2)
dist_transform = cv2.distanceTransform(opening,cv2.DIST_L2,3)
ret2, sure_fg =
cv2.threshold(dist_transform,0.001*dist_transform.max(),255,0)
print(dist_transform.max())
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg,sure_fg)
ret3, markers = cv2.connectedComponents(sure_fg)
markers = markers+10
markers[unknown==255] = 0
markers = cv2.watershed(img1,markers)
img1[markers == -1] = [255,0,255]
img2 = color.label2rgb(markers, bg_label=0)
fig=plt.figure(figsize=(4,4))
ax1=fig.add_subplot(3,3,1)
ax1.imshow(opening)
ax1.title.set_text('opening')
ax2=fig.add_subplot(3,3,2)
ax2.imshow(sure_bg)
ax2.title.set_text('sure_bg')
ax3=fig.add_subplot(3,3,3)
ax3.imshow(sure_fg)
ax3.title.set_text('sure_fg')
ax4=fig.add_subplot(3,3,4)
ax4.imshow(img2)
ax4.title.set_text("watershed")
ax4=fig.add_subplot(3,3,5)
ax4.imshow(thresh)
ax4.title.set_text("watershed")
```

**10. Implement the following procedure, for detecting the edges of gray scale image using Fourier transformation algorithm.**

**e) Convert the gray scale image to the frequency Fourier transform (FFT)**

**f) Create a circular high pass filter (HPF).**

**g) Show inverse Fourier transform.**

**h) Generate the smoothened image.**

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

img = cv2.imread('Gray.jpg', 0)

dft = cv2.dft(np.float32(img), flags=cv2.DFT_COMPLEX_OUTPUT)

dft_shift = np.fft.fftshift(dft)

magnitude_spectrum = 20 * np.log(cv2.magnitude(dft_shift[:, :, 0],
dft_shift[:, :, 1]))

rows, cols = img.shape
crow, ccol = int(rows / 2), int(cols / 2)
mask = np.ones((rows, cols, 2), np.uint8)
r = 200
center = [crow, ccol]
x, y = np.ogrid[:rows, :cols]
mask_area = (x - center[0]) ** 2 + (y - center[1]) ** 2 <= r*r
mask[mask_area] = 1

fshift = dft_shift * mask
fshift_mask_mag = 20 * np.log(cv2.magnitude(fshift[:, :, 0], fshift[:,
:, 1]))
f_ishift = np.fft.ifftshift(fshift)

img_back = cv2.idft(f_ishift)
img_back = cv2.magnitude(img_back[:, :, 0], img_back[:, :, 1])

fig = plt.figure(figsize=(12, 12))
ax1 = fig.add_subplot(2,2,1)
ax1.imshow(img, cmap='gray')
ax1.title.set_text('Input Image')
ax2 = fig.add_subplot(2,2,2)
ax2.imshow(magnitude_spectrum, cmap='gray')
ax2.title.set_text('FFT of image')
ax3 = fig.add_subplot(2,2,3)
ax3.imshow(fshift_mask_mag, cmap='gray')
ax3.title.set_text('FFT + Mask')
ax4 = fig.add_subplot(2,2,4)
ax4.imshow(img_back, cmap='gray')
ax4.title.set_text('After inverse FFT')
plt.show()
```