GOKHALE EDUCATION SOCIETY'S

Above Self... Above Politics

N. B. MEHTA (V) SCIENCE COLLEGE

ACHARYA BHISE VIDYANAGAR, BORDI

TAL - DAHANU, DIST. PALGHAR, 401701, Tel. 02528 - 254357

DEPARTMENT OF INFORMATION TECHNOLOGY AND COMPUTER SCIENCE

Certificate

Class ____________ Year ___________

This is to certify that the work entered in this journal is the work of Shri/Kumari ____

Of _________________ division___________________ Roll No. ______________________

Uni. Exam No. ___________________ has satisfactorily completed the required number of practical and worked for the 1st term / 2nd term/ both the terms of the Year _______________ in the college laboratory as laid down by the university.

______________________	_____________________	____________________
Head of the Department	External Examiner	Internal Examiner Subject teacher

Date : / / 2022 Department of IT-CS

INDEX

Practical No: 01

Aim: Write program to demonstrate the following aspects of signal processing on suitable data.

A: Upsampling and Downsampling on Image/speech signal.

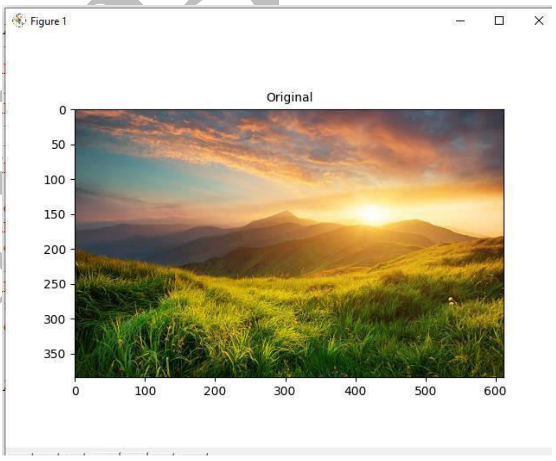
Upsampling:

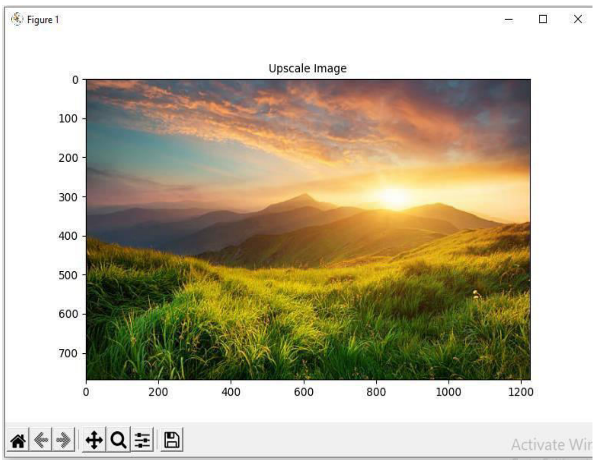
Upsampling is the increasing of the spatial resolution while keeping the 2D representation of an image. It is typically used for zooming in on a small region of an image, and for eliminating the pixilation effect that arises when a low-resolution image is displayed on a relatively large frame.

Program Code:

```
from PIL import Image
import pylab
import numpy
img=Image.open(r'C:\Users\ITLAB4-PC\Pictures\nature.jpg')
pylab.imshow(img)
pylab.show()
upScaleImg=img.resize((img.width*2,img.height*2),Image.Resampling.NEAREST)
pylab.figure(figsize=(10,10))
pylab.imshow(upScaleImg)
pylab.show()
```

Output:





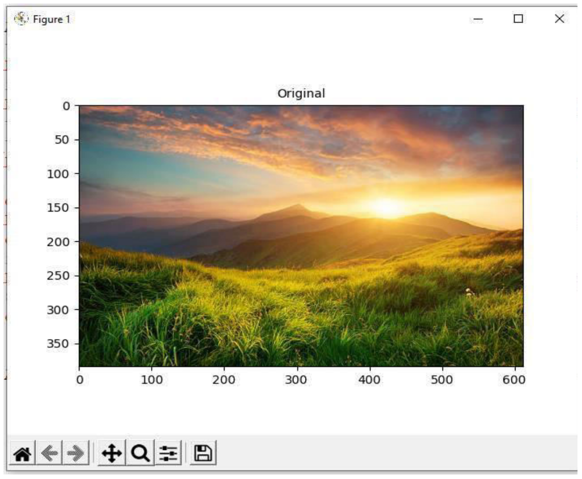
Downsampling:

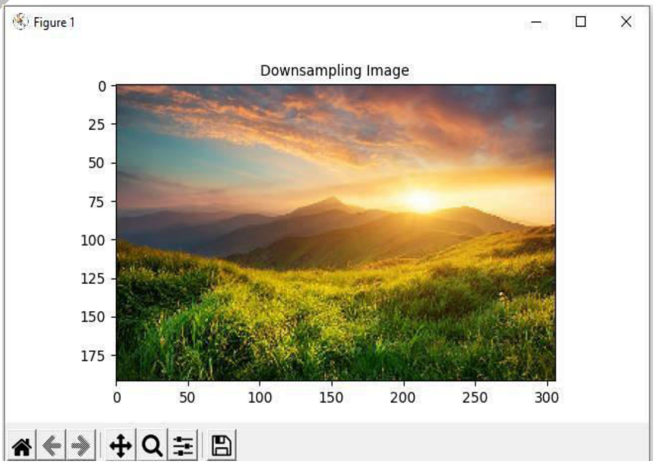
Downsampling is the reduction in spatial resolution while keeping the same two-dimensional (2D) representation. It is typically used to reduce the storage and/or transmission requirements of images. Upsampling is the increasing of the spatial resolution while keeping the 2D representation of an image.

Program Code:

```
from PIL import Image
import pylab
import numpy
img=Image.open(r'C:\Users\ITLAB4-PC\Pictures\nature.jpg')
pylab.imshow(img)
pylab.show()
downScaleImg=img.resize((img.width//2,img.height//2),Image.Resampling.NEAREST)
pylab.figure(figsize=(10,10))
pylab.imshow(downScaleImg)
pylab.show()
```

Output:





B: Fast Fourier Transform to compute DFT.

The Fast Fourier Transform (FFT) is commonly used to transform an image between the spatial and frequency domain. Unlike other domains such as Hough and Radon, the FFT method preserves all original data.

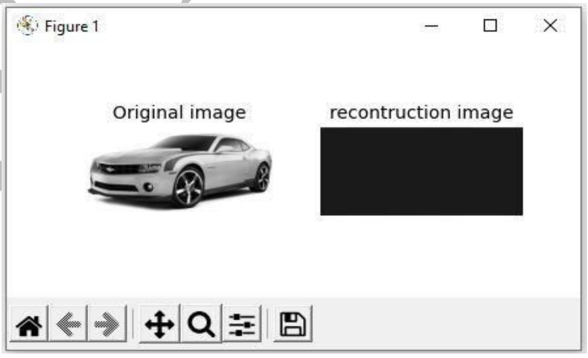
Program Code:

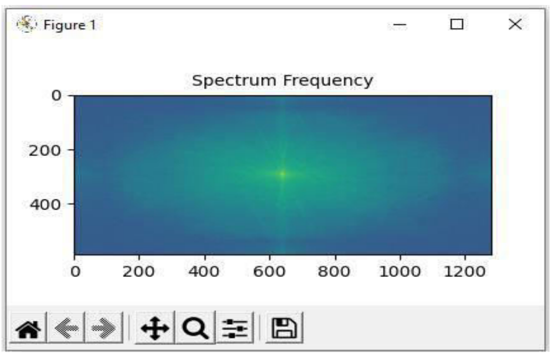
```
from PIL import Image
import numpy as np
import scipy.fftpack as fp
import pylab
im=np.array(Image.open(r'C:\Users\ITLAB4-PC\Desktop\Ravi\testing\car.png').convert('L'))
freq=fp.fft2(im)
im2=fp.ifft(freq).real
pylab.figure(figsize=(20,10))
pylab.subplot(121),pylab.imshow(im,cmap='gray'),pylab.axis('off')
pylab.title('Original image',size=20)
pylab.subplot(122),pylab.imshow(im2,cmap='gray'),pylab.axis('off')
pylab.subplot(122),pylab.imshow(im2,cmap='gray'),pylab.axis('off')
pylab.title('recontruction image',size=20)
pylab.show()
```

Plotting Spectrum Frequency

```
freq2=fp.fftshift(freq)
pylab.figure(figsize=(10,10))
pylab.imshow((20*np.log10(0.1+freq2)).astype(int))
pylab.show()
```

Output:





Practical No: 02

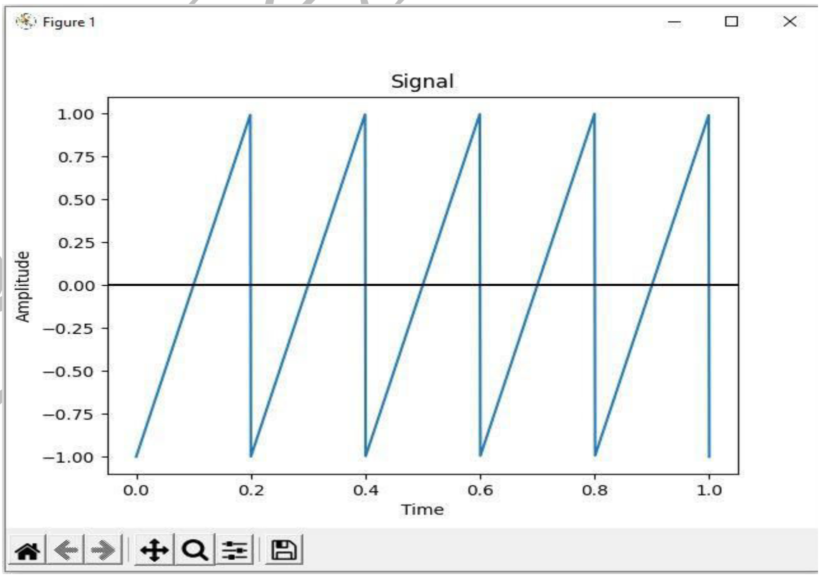
Aim: Write program to perform the following on signal

A: Create a triangle signal and plot a 3-period segment.

Program Code:

```
from scipy import  signal
import matplotlib.pyplot as plot
import numpy as np
t = np.linspace(0, 1, 1000, endpoint=True)
plot.plot(t, signal.sawtooth(2 * np.pi * 5 * t))
plot.xlabel('Time')
plot.ylabel('Amplitude')
plot.title('Signal')
plot.axhline(y=0, color='k')
plot.show()
```

Output:



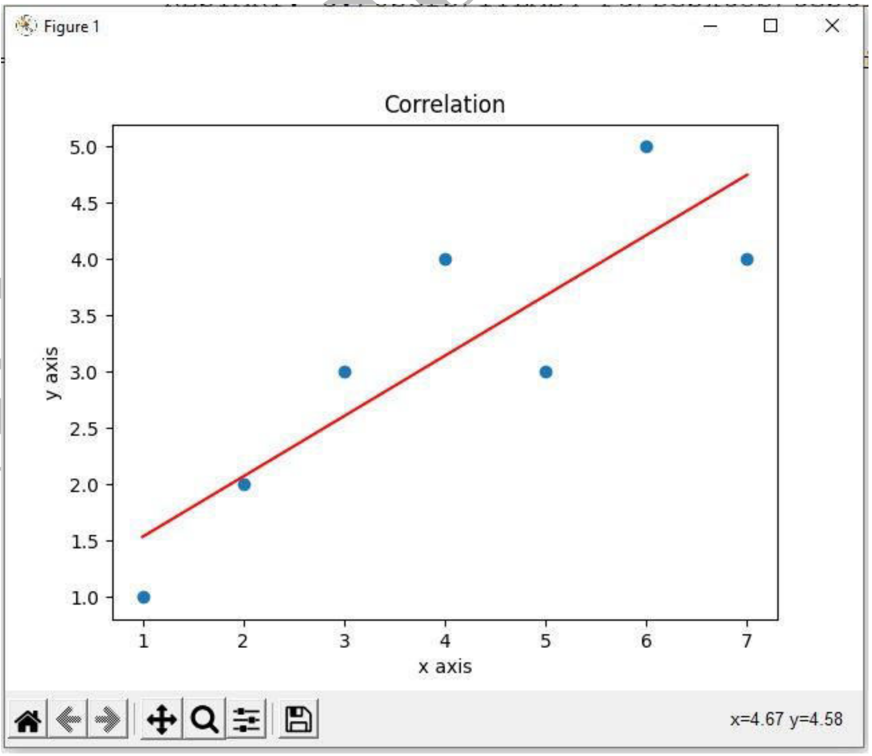
B: For a given signal, plot the segment and compute the correlation between them.

Correlation is a mathematical technique to see how close two things are related. In image processing terms, it is used to compute the response of a mask on an image.

Program Code:

```
import sklearn
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
y = pd.Series([1, 2, 3, 4, 3, 5, 4])
x = pd.Series([1, 2, 3, 4, 5, 6, 7])
correlation = y.corr(x)
plt.scatter(x, y)
plt.plot(np.unique(x), np.poly1d(np.polyfit(x, y, 1))(np.unique(x)), color='red')
plt.xlabel('x axis')
plt.ylabel('y axis')
plt.title('Correlation')
plt.show()
```

Output:



Practical No: 03

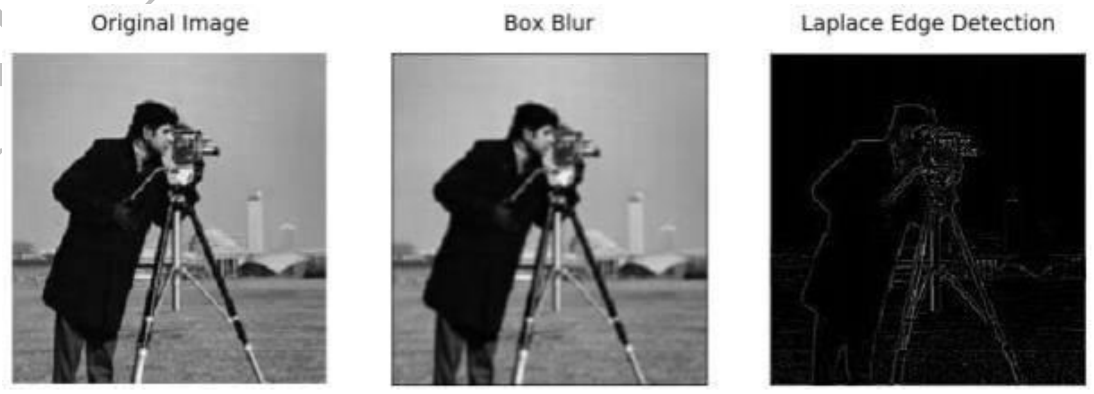
Aim: Write program to demonstrate the following aspects of signal on Sound/image data.

A: Convolution operation.
Convolution is a simple mathematical operation which is fundamental to many common image processing operators. Convolution provides a way of `multiplying together' two arrays of numbers, generally of different sizes, but of the same dimensionality, to produce a third array of numbers of the same dimensionality.

Program Code:

```
im = rgb2gray(imread('../images/cameraman.jpg')).astype(float)
 print(np.max(im))
 print(im.shape) blur_box_kernel = np.ones((3,3)) / 9
edge_laplace_kernel = np.array([[0,1,0],[1,-4,1],[0,1,0]])
im_blurred = signal.convolve2d(im, blur_box_kernel)
 im_edges = np.clip(signal.convolve2d(im, edge_laplace_kernel), 0, 1)
fig, axes = pylab.subplots(ncols=3, sharex=True, sharey=True, figsize=(18,6))
axes[0].imshow(im, cmap=pylab.cm.gray)
axes[0].set_title('Original Image', size=20)
axes[1].imshow(im_blurred, cmap=pylab.cm.gray)
axes[1].set_title('Box Blur', size=20)
axes[2].imshow(im_edges, cmap=pylab.cm.gray)
axes[2].set_title('Laplace Edge Detection', size=20)
for ax in axes:
    ax.axis('off')
pylab.show()
```

Output:



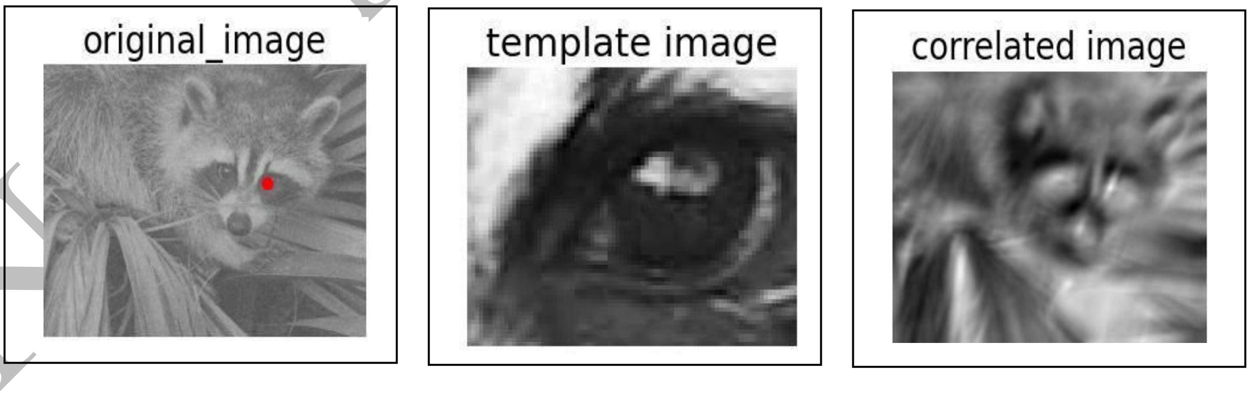
B: Template Matching.

Template matching is the process of moving the template over the entire image and calculating the similarity between the template and the covered window on the image.

Program Code:

```
from PIL import Image
import numpy as np
from scipy import misc
from scipy import signal
from scipy.signal import correlate2d
from matplotlib import pyplot
import pylab as pb
face_img=misc.face(gray=True)-misc.face(gray=True).mean()
temp_img=np.copy(face_img[300:365,670:750])
face_img=face_img+np.random.randn(*face_img.shape)*50
cor=signal.correlate2d(face_img,temp_img,boundary='symm',mode='same')
y,x=np.unravel_index(np.argmax(cor),cor.shape)
fig,(ax_original,ax_template,ax_correlation)=pb.subplots(3,1,figsize=(6,15))
ax_original.imshow(face_img,cmap='gray')
ax_original.set_title('original_image',size=20)
ax_original.set_axis_off()
ax_template.imshow(temp_img,cmap='gray')
ax_template.set_title('template image',size=20)
ax_template.set_axis_off()
ax_correlation.imshow(cor,cmap='gray')
ax_correlation.set_title('correlated image',size=20)
ax_correlation.set_axis_off()
ax_original.plot(x,y,'ro')
fig.show()
```

Output:



Practical No: 4

Aim: Write a program to implement various morphological images processing technique.

Erosion:

Erosion is the morphological operation that is performed to reduce the size of the foreground object. The boundary of the foreign object is slowly eroded. Erosion has many applications in image editing and transformations, and erosion shrinks the image pixels. Pixels on object boundaries are also removed.

Program code:

```
from PIL import Image
from skimage.io import imread
from skimage.morphology import binary_erosion, rectangle
from skimage.color import rgb2gray
import pylab
im=rgb2gray(imread(r'C:\Users\LJP_IT_LAB\Desktop\car.jpg' ))
im[im<=0.5]=0
im[im>0.5]=1
pylab.gray()
def plot_image(im,title=''):pylab.title(title,size=10)
pylab.subplot(1,3,1),plot_image(im, 'original')
pylab.imshow(im)
im1=binary_erosion(im,rectangle(1,5))
pylab.subplot(1,3,2),plot_image(im1,'erosion with recangular size (1,5)')
pylab.imshow(im1)
im1=binary_erosion(im,rectangle(1,15))
pylab.subplot(1,3,3),plot_image(im1,'erosion with rectangular sieze (1,15)')
pylab.imshow(im1)
pylab.show()
```

Output:

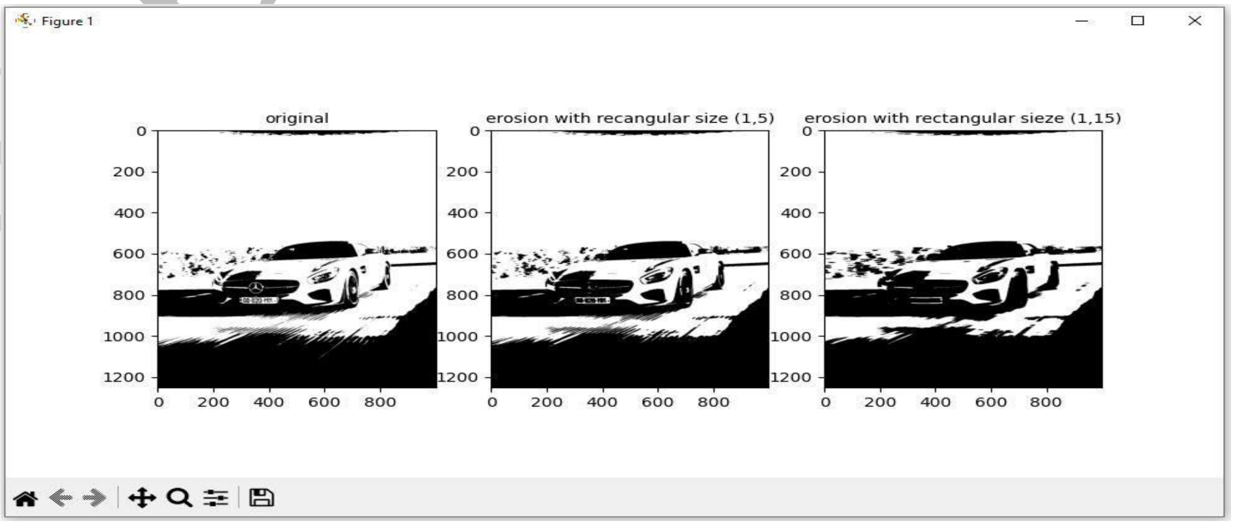


Figure 1
original
erosion with recangular size (1,5)
erosion with rectangular sieze (1,15)
0
200
400
600
800
1000
1200

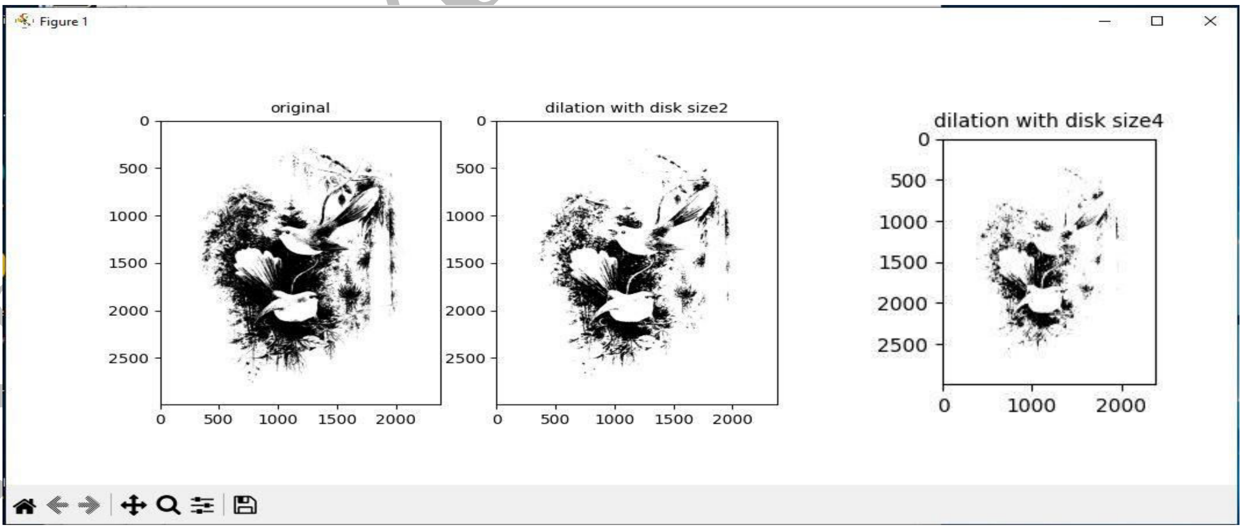
Dilation:

Dilation of an image is the process by which the object area in the image is increased. This process is used to accentuate features in the image. It increases the white region in the image or the size of the foreground object increases.

Program Code:

```
from PIL import Image
from skimage.morphology import binary_dilation, disk
from skimage.color import rgb2gray
from skimage.io import imread
import pylab
im = rgb2gray(imread(r'C:\Users\LJP_IT_LAB\Pictures\pic.jpg'))
im[im <= 0.5]=0
im[im > 0.5] = 1
pylab.gray()
def plot_image(im,title=''):pylab.title(title,size=10)
pylab.subplot(131),plot_image(im, 'original')
pylab.imshow(im)
for d in range(1,3):
   im1 = binary_dilation(im, disk(2*d))
   pylab.subplot(1,3,d+1), plot_image(im1, 'dilation with disk size' + str(2*d))
 pylab.imshow(im1)
 pylab.show()
```

Output:



Opening and Closing:

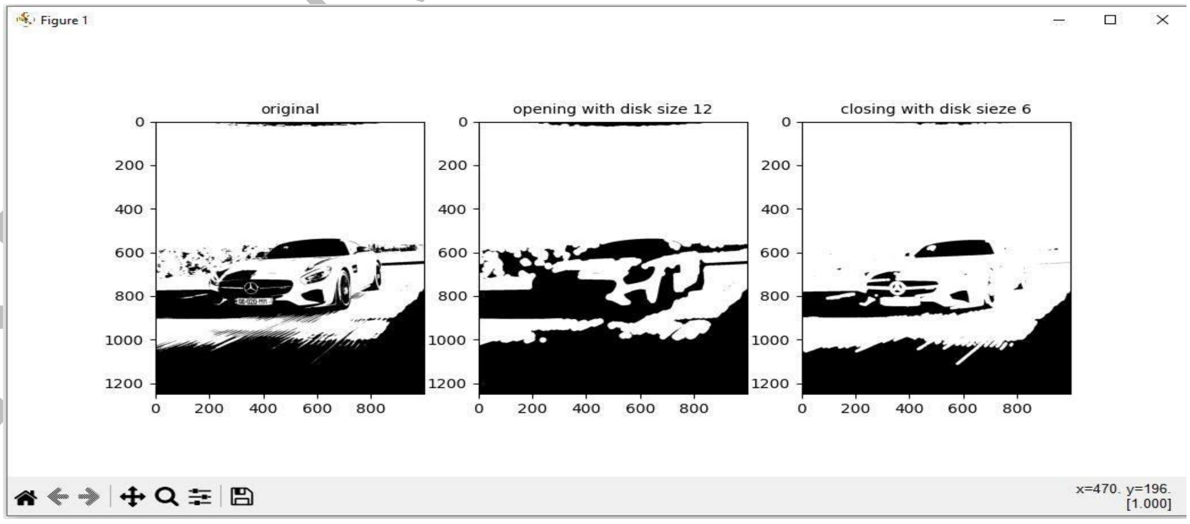
Opening is a morphological operation that can be expressed as a combination of first erosion and then dilation operation; it remove small object from a binary image.

Closing, to the contrary, is another morphological operation that can be expressed as combination of first dilation and then erosion operation; it removes small holes from a binary image.

Program Code:

```
from skimage.io import imread
from skimage.morphology import binary_erosion, disk, binary_dilation, binary_opening,
binary_closing
from skimage.color import rgb2gray
im=rgb2gray(imread(r'C:\Users\LJP_IT_LAB\Desktop\car.jpg' ))
im[im <= 0.5] = 0
im[im > 0.5] = 1
pylab.gray()
def plot_image(im,title=''):pylab.title(title,size=10)
pylab.subplot(1,3,1),plot_image(im, 'original')
pylab.imshow(im)
im1=binary_opening(im, disk(12))
pylab.subplot(1,3,2),plot_image(im1,’opening with disk size 12’)
pylab.imshow(im1)
im1=binary_dilation(im,disk(6))
pylab.subplot(1,3,3),plot_image(im1,’closing with disk sieze 6’)
pylab.imshow(im1)
pylab.show()
```

Output:



Practical No: 5

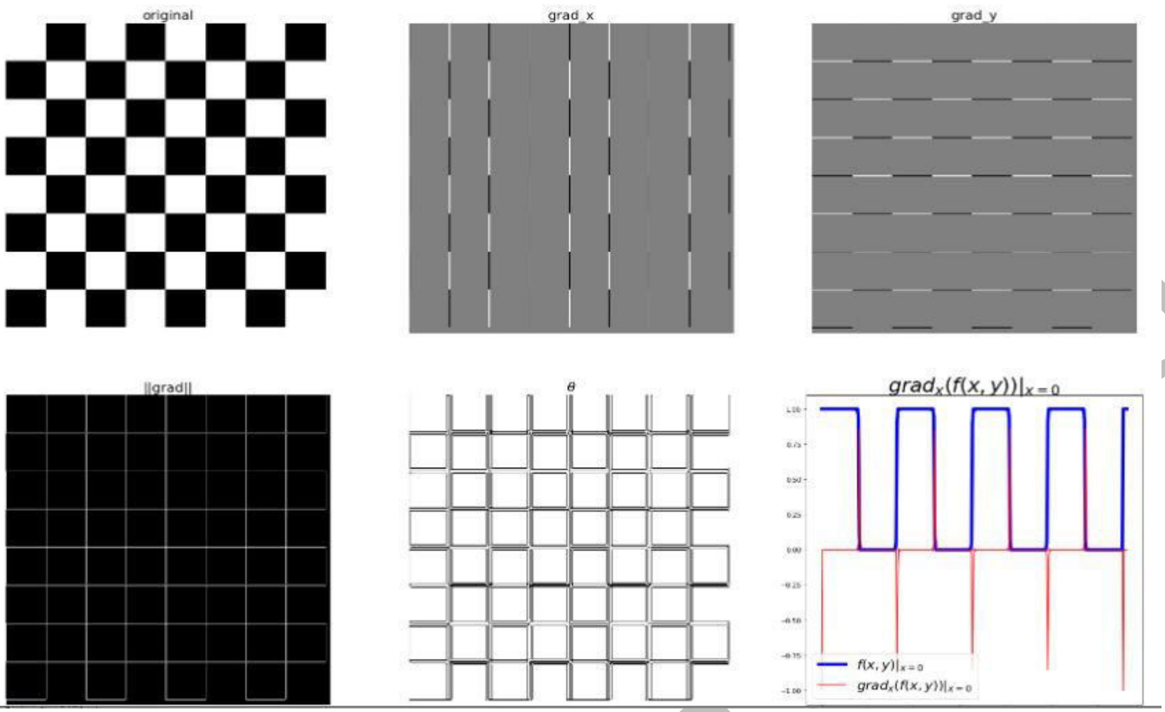
Aim: Write a program to apply various enhancements on images using image derivatives by implementing Gradient and Laplacian operations.

Gradient: The following code block shows how to compute the gradient with the convolution kernels shown previously, with the gray-scale chess image as input. It also plots how the image pixel values and x_component of the gradient vector changes with the y coordinates for the very first row in the image(x=0):

Program Code: A

```
import numpy as np
from scipy import signal, misc, ndimage
from skimage import filters, feature, img_as_float
from skimage.io import imread
from skimage.color import rgb2gray
from PIL import Image, ImageFilter
import matplotlib.pylab as pylab
def plot_image(image, title):
pylab.imshow(image), pylab.title(title, size=20), pylab.axis('off')
ker_x = [[-1, 1]]
ker_y = [[-1], [1]]
im = rgb2gray(imread('../images/chess.png'))
im_x = signal.convolve2d(im, ker_x, mode='same')
im_y = signal.convolve2d(im, ker_y, mode='same')
im_mag = np.sqrt(im_x**2 + im_y**2)
im_dir = np.arctan(im_y/im_x)
pylab.gray()
pylab.figure(figsize=(30,20))
pylab.subplot(231), plot_image(im, 'original'), pylab.subplot(232),
plot_image(im_x, 'grad_x')
pylab.subplot(233), plot_image(im_y, 'grad_y'), pylab.subplot(234),
plot_image(im_mag, '||grad||')
pylab.subplot(235), plot_image(im_dir, r'$\theta$'), pylab.subplot(236)
pylab.plot(range(im.shape[1]), im[0,:], 'b-', label=r'$f(x,y)|_{x=0}$', linewidth=5)
pylab.plot(range(im.shape[1]), im_x[0,:], 'r-', label=r'$grad_x (f(x,y))|_{x=0}$')
pylab.title(r'$grad_x (f(x,y))|_{x=0}$', size=30)
pylab.legend(prop={'size': 20}
pylab.show()
```

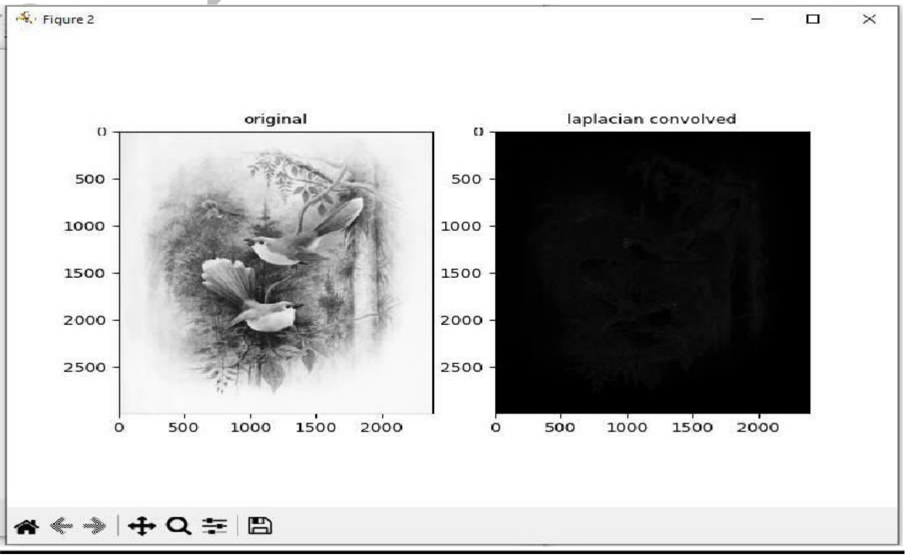
output:



Program Code B: Laplacian operations.

```
ker_laplacian = [[0,-1,0],[-1, 4, -1],[0,-1,0]]
im = rgb2gray(imread('../images/chess.png'))
im1 = np.clip(signal.convolve2d(im, ker_laplacian,
mode='same'),0,1)
pylab.gray()
pylab.figure(figsize=(20,10))
pylab.subplot(121), plot_image(im, 'original')
pylab.subplot(122), plot_image(im1, 'laplacian convolved')
pylab.show()
```

output:



Practical No: 6

Aim: Write a program to implement linear and nonlinear noise smoothing on suitable image or Sound signal.

A: Smoothing with Linear

Linear smoothing filters remove high-frequency components, and the sharp detail in the image is lost. For example, step changes will be blurred into gradual changes, and the ability to accurately localize a change will be sacri- ficed.

Program code:

```
import  pylab
import numpy as np
from PIL import Image, ImageFilter
from skimage import filters
i = 1
pylab.figure(figsize=(10,25))
for prop_noise in np.linspace(0.05,0.3,3):
   im = Image.open(r'C:\Users\LJP_IT_LAB\Desktop\Ravi Singh\Image Processing\Docs\nature.jpg')
   def plot_image(im,title=''):pylab.title(title, size=7)
   n = int(im.width * im.height * prop_noise)
   x, y = np.random.randint(0, im.width, n), np.random.randint(0, im.height, n)
   for (x,y) in zip(x,y):
      im.putpixel((x, y), ((0,0,0) if np.random.rand() < 0.5 else (255,255,255)))
   im.save(r'C:\Users\LJP_IT_LAB\Desktop\Ravi Singh\Image Processing\Docs\output' +
str(prop_noise) + '.jpg')
   pylab.subplot(6,2,i), plot_image(im, 'Original Image with ' +
   str(int(100*prop_noise)) + '% added noise')
   pylab.imshow(im)
   i += 1
   im1 = im.filter(ImageFilter.BLUR)
   pylab.subplot(6,2,i), plot_image(im1, 'Blurred Image')
   pylab.imshow(im1)
   i += 1
pylab.show()
```

Output:

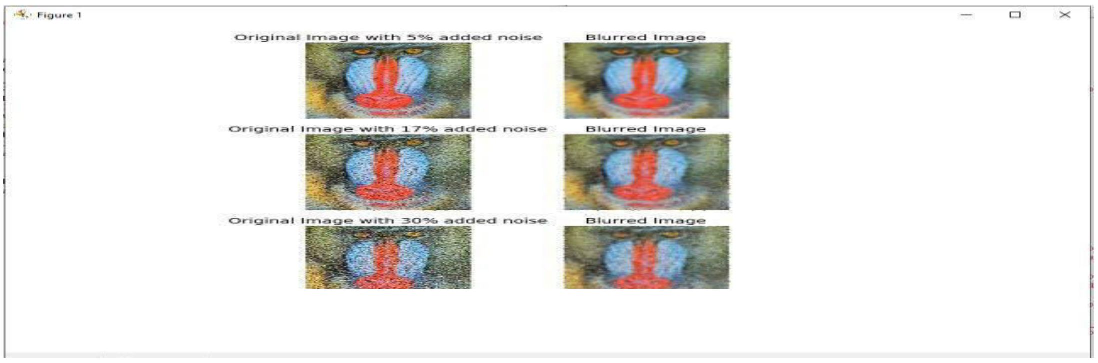


Figure 1
Original Image with 5% added noise
Blurred Image
Original Image with 17% added noise
Blurred Image
Original Image with 30% added noise
Blurred Image

B: Smoothing with Non-linear

Non-linear smoothing filters are a powerful weapon that should be in the arsenal of every image processing practitioner. They find application in several fields, from computer vision to astronomy, medical imaging, geology, digital art, photography and many more.

Program Code:

```
import  pylab
import numpy as np
from PIL import Image, ImageFilter
from skimage import filters
i = 1
pylab.figure(figsize=(25,35))
for prop_noise in np.linspace(0.05,0.3,3):
   im = Image.open(r'C:\Users\LJP_IT_LAB\Desktop\Ravi Singh\Image Processing\Docs\nature.jpg')
   def plot_image(im,title=''):pylab.title(title, size=7)
   n = int(im.width * im.height * prop_noise)
   x, y = np.random.randint(0, im.width, n), np.random.randint(0, im.height, n)
   for (x,y) in zip(x,y):
      im.putpixel((x, y), ((0,0,0) if np.random.rand() < 0.5 else (255,255,255)))
   im.save(r'C:\Users\LJP_IT_LAB\Desktop\Ravi Singh\Image Processing\Docs\output' +
str(prop_noise) + '.jpg')
   pylab.subplot(6,4,i)
   pylab.imshow(im)
   plot_image(im, 'Original Image with ' + str(int(100*prop_noise)) + '%added noise')
   i += 1
   for sz in [3,7,11]:
      im1 = im.filter(ImageFilter.MedianFilter(size=sz))
      pylab.subplot(6,4,i), plot_image(im1, 'Output (Median Filter size=' +  str(sz) + ')')
      pylab.imshow(im1)
      i += 1
pylab.show()
```

Output:

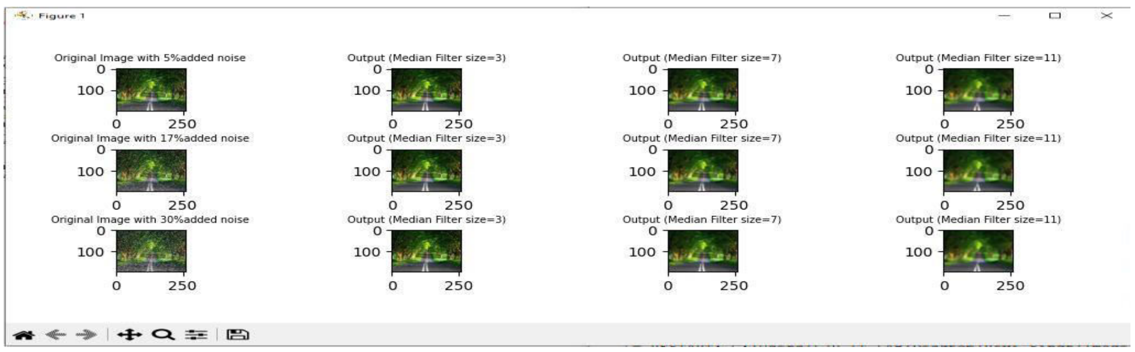


Figure 1
Original Image with 5%added noise
Output (Median Filter size=3)
Output (Median Filter size=7)
Output (Median Filter size=11)
Original Image with 17%added noise
Output (Median Filter size=3)
Output (Median Filter size=7)
Output (Median Filter size=11)
Original Image with 30%added noise
Output (Median Filter size=3)
Output (Median Filter size=7)
Output (Median Filter size=11)
0
100
0
250

Practical No: 7

Aim: Write program to implement point/pixel intensity transformations

A: Log and Power-law transformations.

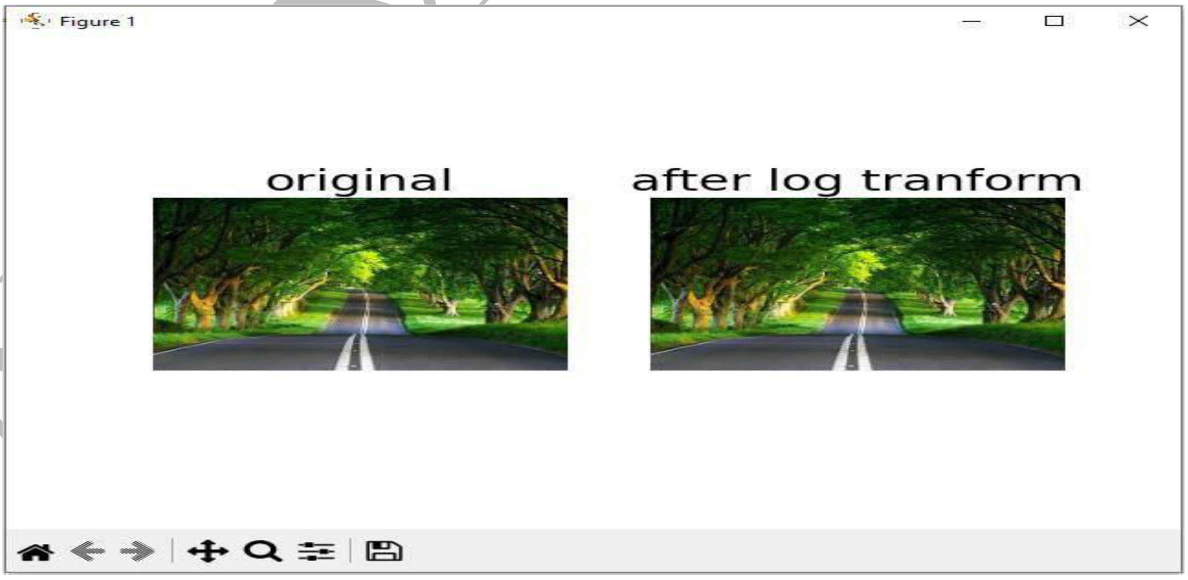
Log Transformation:

During log transformation, the dark pixels in an image are expanded as compare to the higher pixel values. The higher pixel values are kind of compressed in log transformation.

Program code:

```
from PIL import Image
import pylab

def  plot_image(image, title=''):
    pylab.title(title, size=20), pylab.imshow(image)
    pylab.axis('off')
im = Image.open(r'C:\Users\LJP_IT_LAB\Desktop\Ravi Singh\Image Processing\Docs\nature.jpg')
im1=im.point(lambda i: 255*np.log(1+i/255))
#im_r, im_g, im_b = im.split()
pylab.style.use('ggplot')
pylab.subplot(121), plot_image(im, 'original')
pylab.subplot(122), plot_image(im1, 'after log tranform')
pylab.imshow(im)
pylab.show()
```

Output:



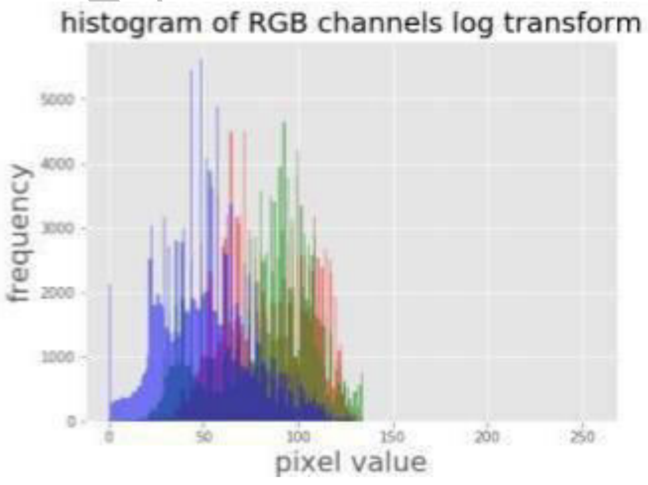
Power-Law Transformation:

This type of transformation is used for enhancing images for different type of display devices. The gamma of different display devices is different.

Program Code:

```
im = img_as_float(imread('../images/earthfromsky.jpg'))
gamma = 5
im1 = im**gamma
pylab.style.use('ggplot')
pylab.figure(figsize=(15,5))
pylab.subplot(121), plot_hist(im[...,0], im[...,1], im[...,2],'histogram for RGB channels (input)')
pylab.subplot(122), plot_hist(im1[...,0], im1[...,1], im1[...,2],'histogram for RGB channels (output)')
pylab.show()
```

Output:



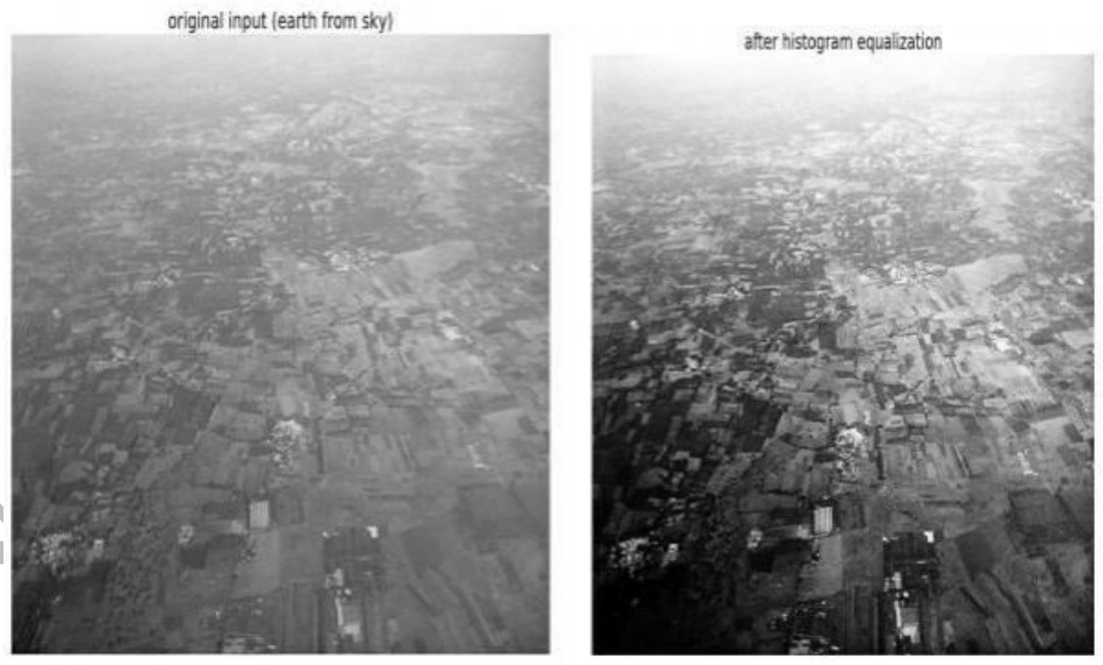
B: Histogram equalization with scikit-image

Histogram Equalization is a computer image processing technique used to improve contrast in images. It accomplishes this by effectively spreading out the most frequent intensity values

Program Code:

```
img = rgb2gray(imread('../images/earthfromsky.jpg'))
img_eq = exposure.equalize_hist(img)
img_adapteq = exposure.equalize_adapthist(img, clip_limit=0.03)
pylab.gray()
images = [img, img_eq, img_adapteq]
titles = ['original input (earth from sky)','after histogram equalization','after adaptive
histogramequalization']
for i in range(3):
pylab.figure(figsize=(20,10)), plot_image(images[i], titles[i])
pylab.figure(figsize=(15,5))
for i in range(3):
pylab.subplot(1,3,i+1), pylab.hist(images[i].ravel(), color='g'), pylab.title(titles[i], size=15)
pylab.show()
```

Output:

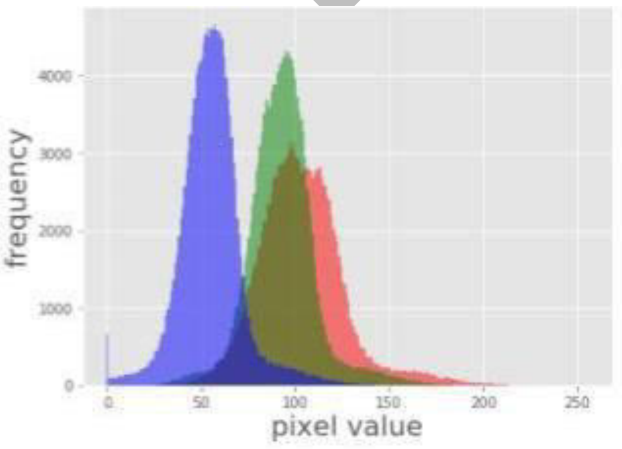


C: Contrast Adjustments

Contrast adjustment remaps image intensity values to the full display range of the data type. An image with good contrast has sharp differences between black and white.

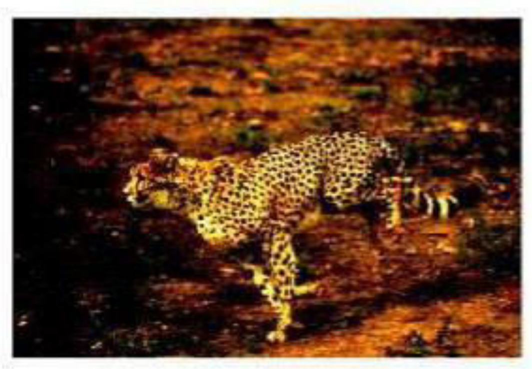
Program Code:

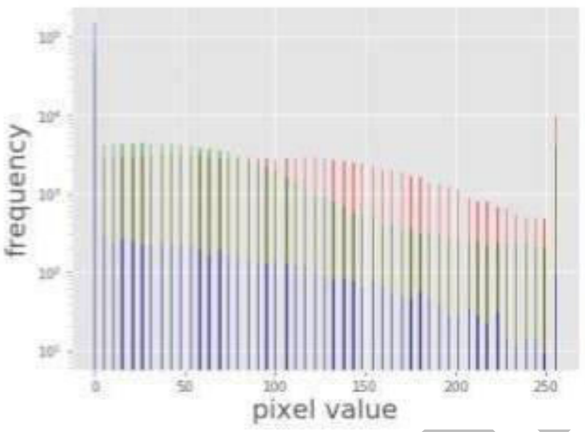
```
im = Image.open('../images/cheetah.png')
im_r, im_g, im_b, _ = im.split()
pylab.style.use('ggplot')
pylab.figure(figsize=(15,5))
pylab.subplot(121)
plot_image(im)
pylab.subplot(122)
plot_hist(im_r, im_g, im_b)
pylab.show()
```



```
 def contrast(c): return 0 if c < 70 else (255 if c > 150 else (255*c - 22950) / 48)
 # piece-wise linear
 functionim1 = im.point(contrast)
im_r, im_g, im_b, _ = im1.split()
pylab.style.use('ggplot')
pylab.figure(figsize=(15,5))
pylab.subplot(121)
plot_image(im1)
pylab.show()
```

Output:





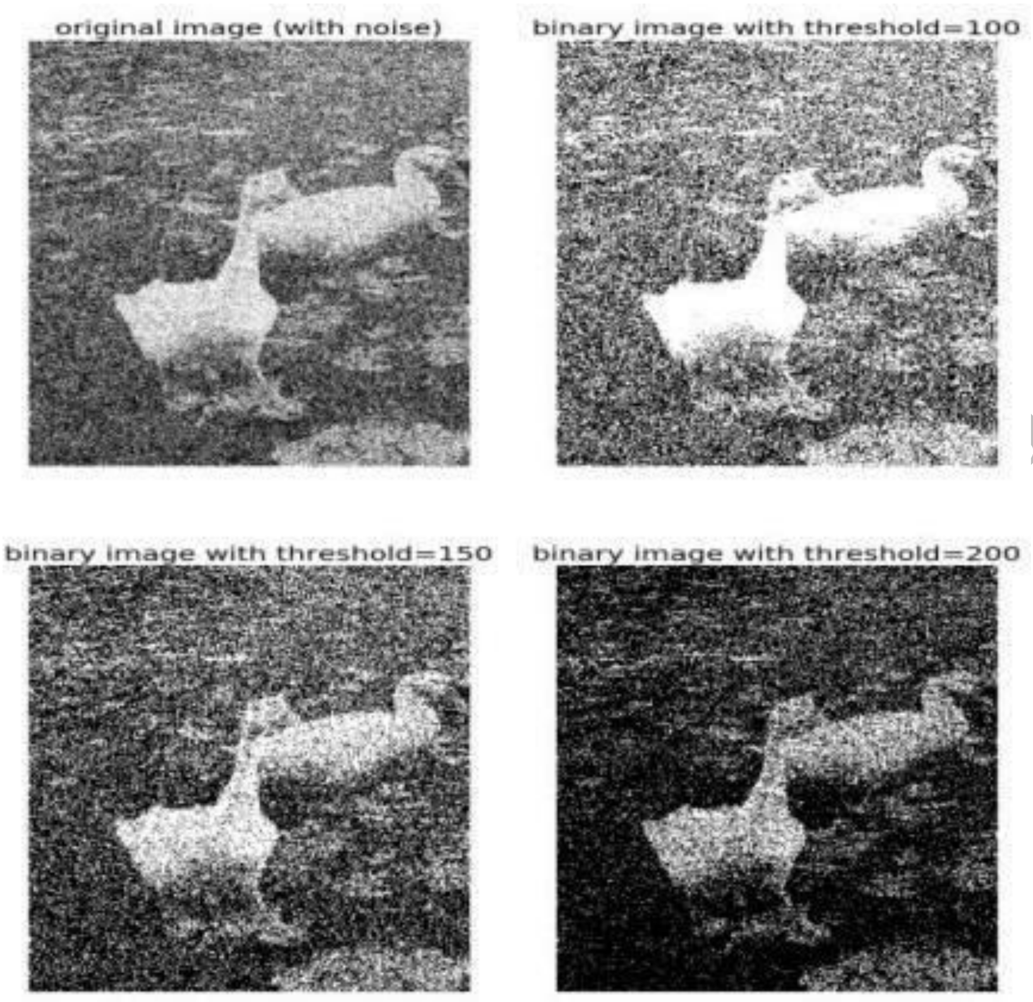
D: Half-toning

Halftone is the reprographic technique that simulates continuous-tone imagery through the use of dots, varying either in size or in spacing, thus generating a gradient-like effect.

Program Code:

```
im = Image.open('../images/swans.jpg').convert('L')
im = Image.fromarray(np.clip(im + np.random.randint(- 128, 128, (im.height, im.width)), 0,
255).astype(np.uint8))
pylab.figure(figsize=(12,18))
pylab.subplot(221), plot_image(im,'original image (with noise)')
th = [0, 50, 100, 150, 200]
for i in range(2, 5):
im1 = im.point(lambda x: x > th[i])
pylab.subplot(2,2,i), plot_image(im1,'binary image with threshold=' + str(th[i]))
pylab.show()
```

Output:


original image (with noise)
binary image with threshold=100
binary image with threshold=150
binary image with threshold=200

Practical No: 8

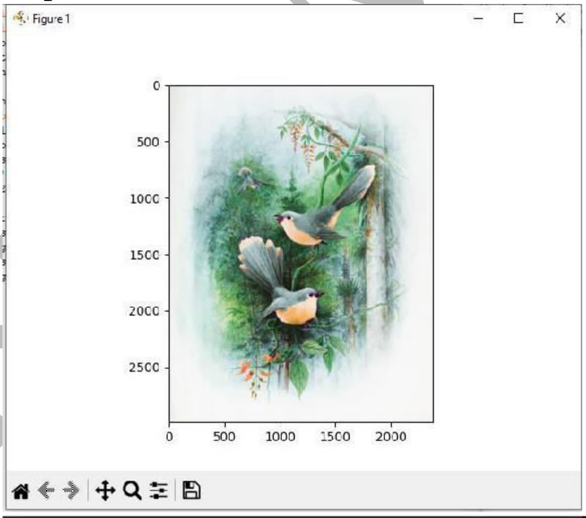
Aim: Write the program to extract image features by implementing methods like corner and blob detectors, HoG and Haar features

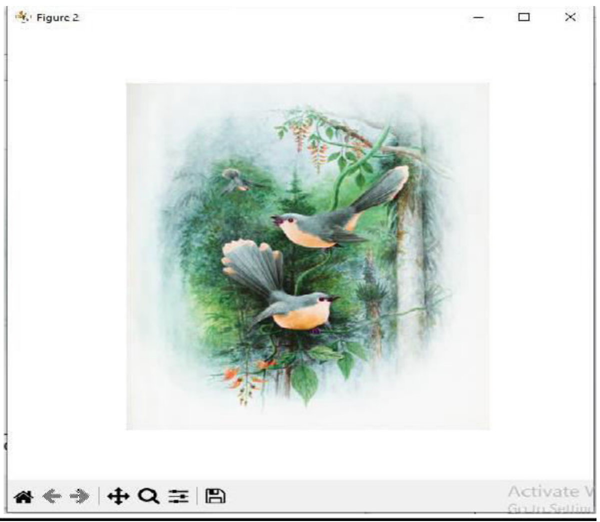
Harris Corner Detector With scikit-image : algorithm was developed to identify the internal corners of an image. The corners of an image are basically identified as the regions in which there are variations in large intensity of the gradient in all possible dimensions and directions. Corners extracted can be a part of the image features, which can be matched with features of other images, and can be used to extract accurate information. Harris Corner Detection is a method to extract the corners from the input image and to extract features from the input image.

Program Code: A

```
from matplotlib import pylab as pylab
from skimage.io import imread
from skimage.color import rgb2gray
from skimage.feature import corner_harris, corner_subpix, corner_peaks
from skimage.transform import warp, SimilarityTransform, AffineTransform, resize
import numpy as np
from skimage import data
from skimage.util import img_as_float
from skimage.exposure import rescale_intensity
from skimage.measure import ransac
image = imread(r'C:\Users\LJP_IT_LAB\Pictures\pic.jpg')
image_gray = rgb2gray(image)
coordinates = corner_harris(image_gray, k =0.001)
image[coordinates>0.01*coordinates.max()]=[255,0,0,255]
pylab.figure(figsize=(20,10))
pylab.imshow(image), pylab.axis('off'), pylab.show()
```

output:

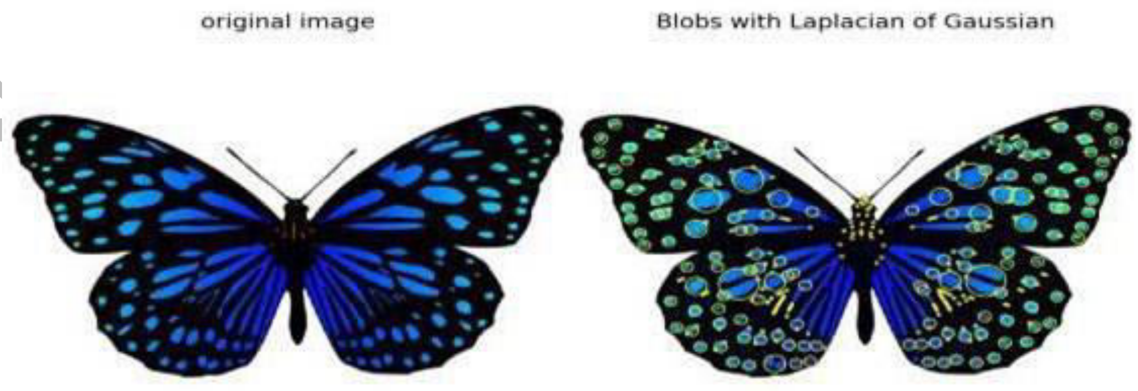


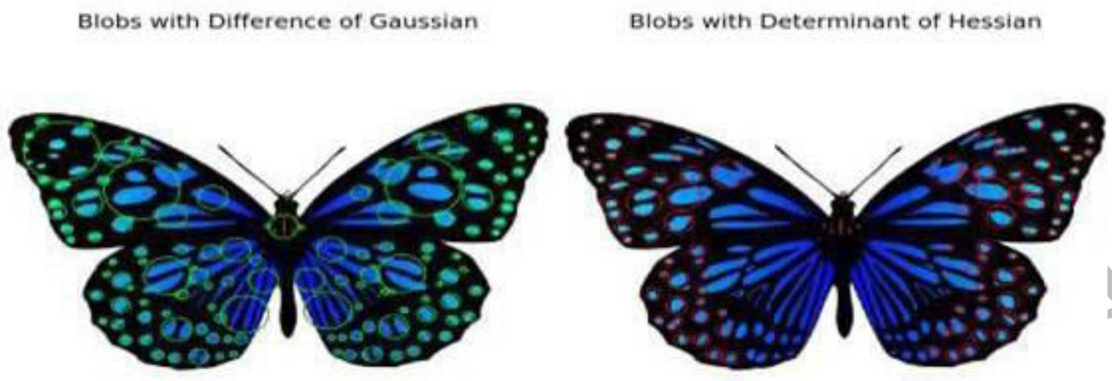


- **Blob detectors with LoG, DoG and DoH**

```
from numpy import sqrt
from skimage.feature import blob_dog, blob_log, blob_doh
im = imread('../images/butterfly.png')
im_gray = rgb2gray(im)
log_blobs = blob_log(im_gray, max_sigma=30, num_sigma=10, threshold=.1)
log_blobs[:, 2] = sqrt(2) * log_blobs[:, 2] # Compute radius in the
3rd column dog_blobs = blob_dog(im_gray, max_sigma=30,
threshold=0.1) dog_blobs[:, 2] = sqrt(2) * dog_blobs[:, 2]
doh_blobs = blob_doh(im_gray, max_sigma=30, threshold=0.005)
list_blobs = [log_blobs, dog_blobs, doh_blobs]
colors, titles = ['yellow','lime','red'], ['Laplacian of Gaussian', 'Difference of Gaussian',
'Determinant of Hessian'] sequence = zip(list_blobs, colors, titles)
fig, axes = pylab.subplots(2, 2,figsize=(20, 20),sharex=True, sharey=True)
axes = axes.ravel()
axes[0].imshow(im, interpolation='nearest')
axes[0].set_title('original image', size=30), axes[0].set_axis_off()

for idx, (blobs, color, title) in enumerate(sequence):
   axes[idx+1].imshow(im, interpolation='nearest')

   axes[idx+1].set_title('Blobs with ' + title, size=30)
   for blob in blobs:

      y, x, row = blob
      col = pylab.Circle((x, y), row, color=color, linewidth=2, fill=False)

      axes[idx+1].add_patch(col), axes[idx+1].set_axis_off()
pylab.tight_layout(), pylab.show()
```

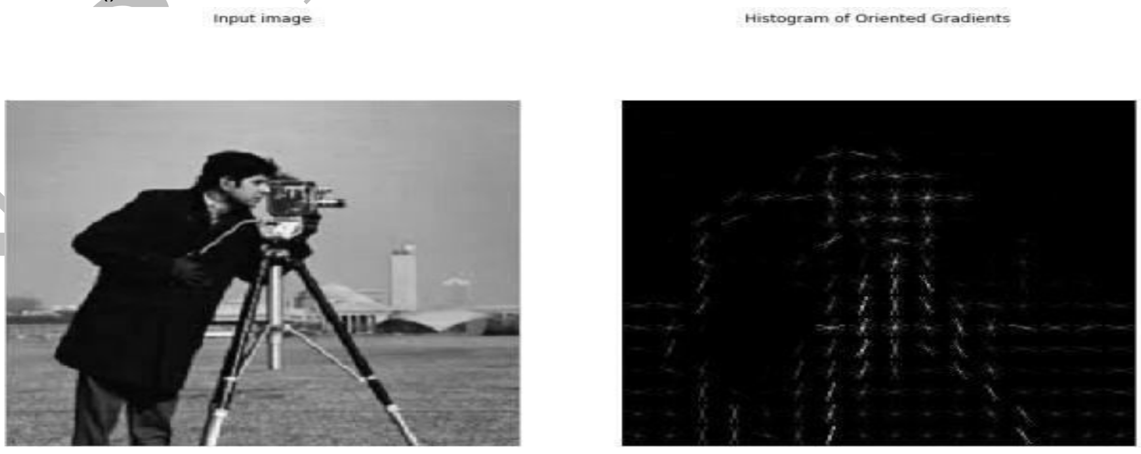
Output:




Blobs with Difference of Gaussian
Blobs with Determinant of Hessian

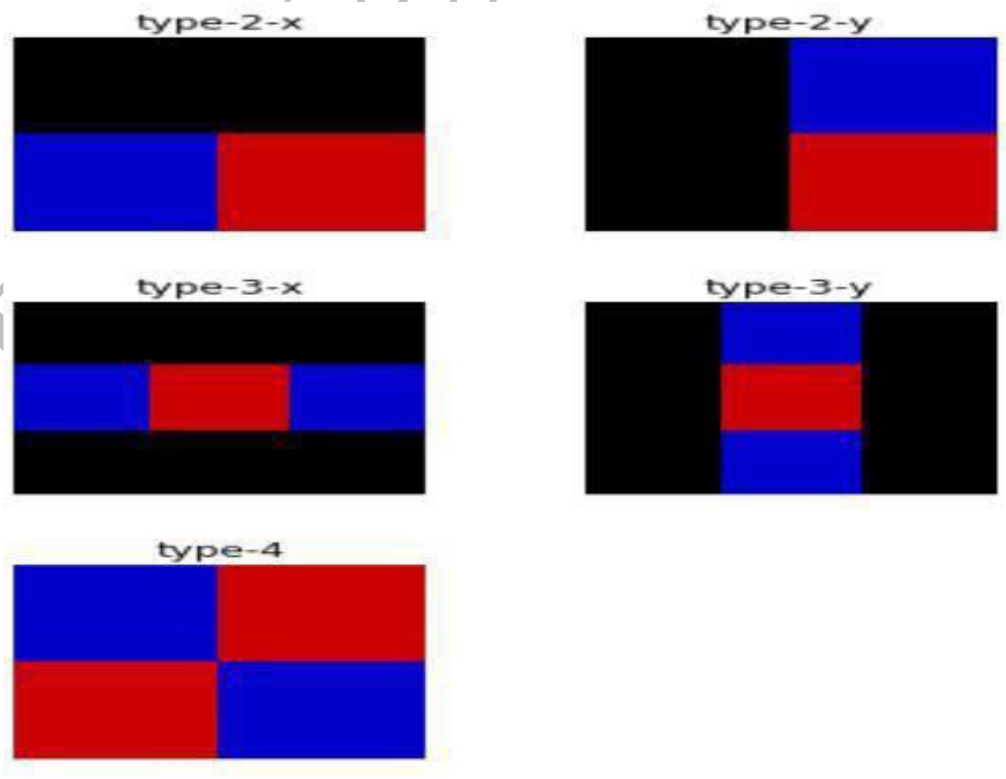
Compute HOG descriptors with scikit-image

```
from skimage.feature import hog
from skimage import exposure
image = rgb2gray(imread('../images/cameraman.jpg'))
fd,hog_image = hog(image, orientations=8, pixels_per_cell=(16, 16),
cells_per_block=(1, 1), visualize=True)

print(image.shape, len(fd))
# ((256L, 256L), 2048)
fig, (axes1, axes2) = pylab.subplots(1, 2,figsize=(15, 10),sharex=True, sharey=True)
axes1.axis('off'), axes1.imshow(image, cmap=pylab.cm.gray), axes1.set_title('Input
image')
hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))
axes2.axis('off'), axes2.imshow(hog_image_rescaled, cmap=pylab.cm.gray),
axes2.set_title('Histogram of Oriented Gradients')

pylab.show()
```



● Haar-like feature descriptor with scikit-image

```
from skimage.feature import haar_like_feature_coord

from skimage.feature import draw_haar_like_feature

images = [np.zeros((2, 2)), np.zeros((2, 2)), np.zeros((3, 3)),


np.zeros((3, 3)), np.zeros((2, 2))]
feature_types = ['type-2-x','type-2-y','type-3-x','type-3-y','type-4']

fig, axes = pylab.subplots(3, 2,figsize=(5,7))

for axes, img, feat_t in zip(np.ravel(axes), images, feature_types):

    coordinates, _ = haar_like_feature_coord(img.shape[0], img.shape[1], feat_t)

    haar_feature = draw_haar_like_feature(img, 0, 0, img.shape[0],img.shape[1], coordinates,
max n features=1, random_state=0, color_positive_block=(1.0, 0.0, 0.0),          _ _
color_negative_block=(0.0, 0.0, 1.0), alpha=0.8)
    axes.imshow(haar_feature), axes.set_title(feat_t), axes.set_axis_off()
#fig.suptitle('Different Haar-like feature descriptors')


pylab.axis('off'), pylab.tight_layout(), pylab.show()
```



Practical No: 9

Aim Write a program to apply various image enhancement using image derivatives by implementing smoothing, sharpening, and unsharp masking filters for generating suitable images for specific application requirements.

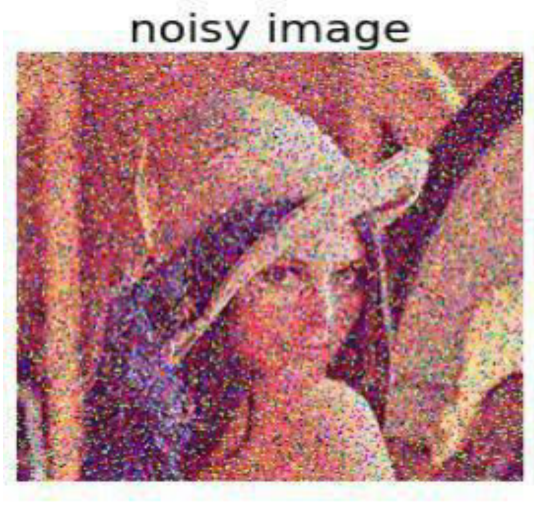
● Smoothing with scipy ndimage

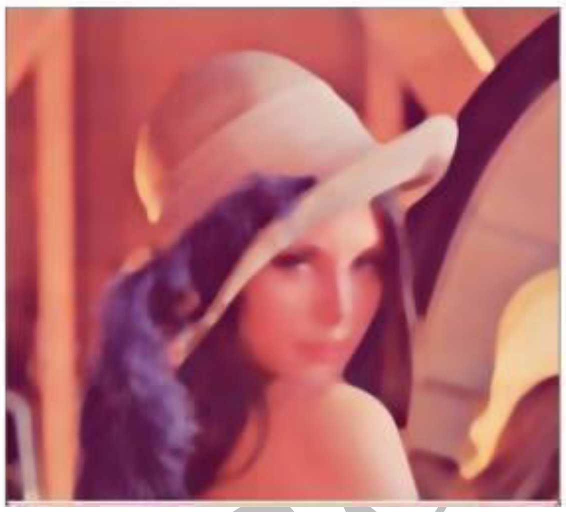
The scipy ndimage module provides a function named percentile_filter(), which is a generic version of the median filter.The following code block demonstrate how to use this filter.

Code:

```
lena = misc.imread('../images/lena.jpg')
# add salt-and-pepper noise
to the input image noise =
np.random.random(lena.shap
e) lena[noise > 0.9] = 255
lena[noise < 0.1] = 0
plot_image
(lena,'nois
y image')
pylab.sho
w()
fig = pylab.figure(figsize=(20,15))

i = 1
 for p in
range(25,
100, 25):
   for k in range(5, 25, 5):
      pylab.subplot(3,4,i)
      filtered = ndimage.percentile_filter(lena, percentile=p, size=(k,k,1))
      plot_image(filtered, str(p) + ' percentile, ' + str(k) + 'x' + str(k) + ' kernel')
      i += 1
 pylab.show()
```

output:



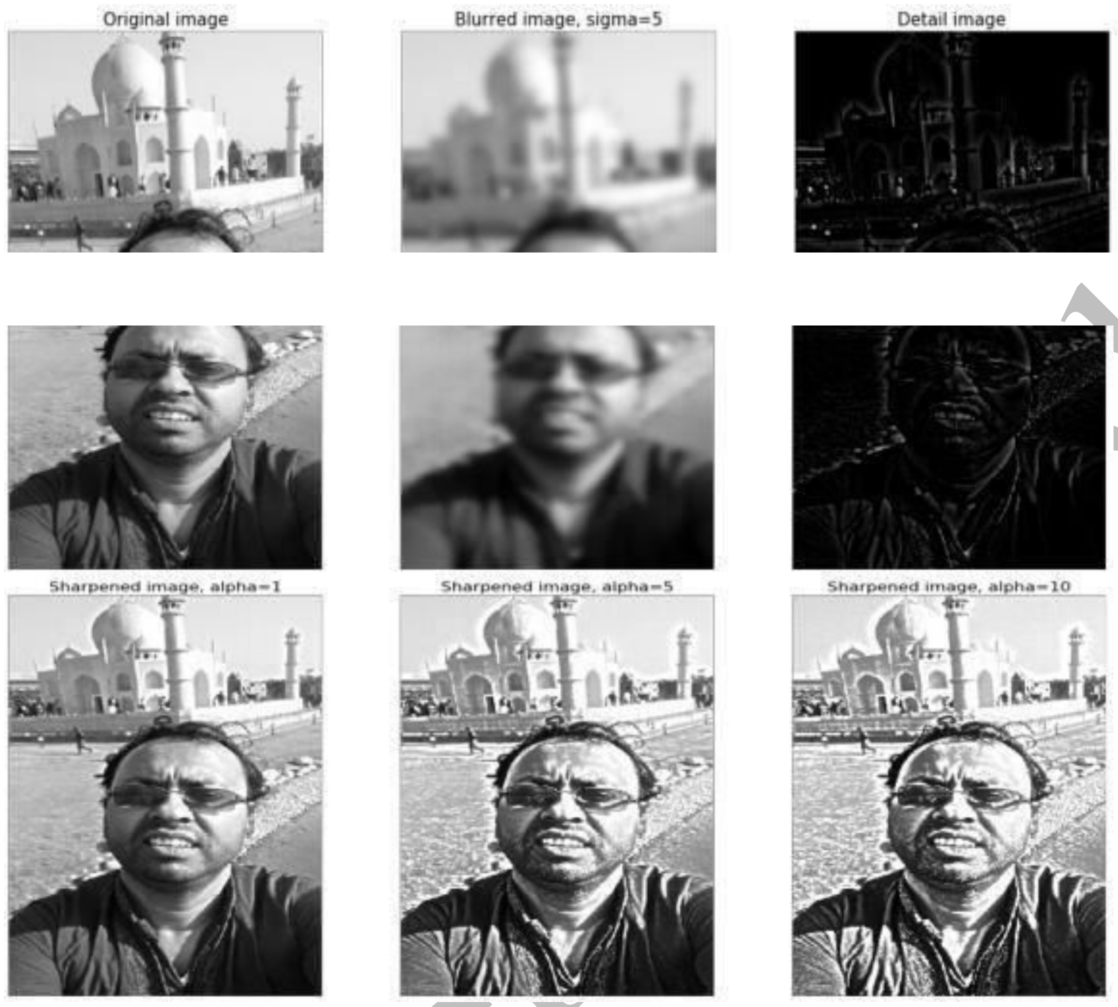


• **Sharpening with Laplacian**

```
from skimage.filters import laplace
im = rgb2gray(imread('../images/me8.jpg'))
im1 = np.clip(laplace(im) + im, 0, 1)
pylab.figure(figsize=(10,15))
pylab.subplot(211), plot_image(im,'original image')
pylab.subplot(212), plot_image(im1,'sharpened image')
pylab.tight_layout()
pylab.show()
```

- **Unsharp masking**

```
def rgb2gray(im):
    '''
    the input image is an RGB image
    with pixel values for each channel in [0,1]
    '''
    return np.clip(0.2989 * im[...,0] + 0.5870 * im[...,1] + 0.1140 * im[...,2], 0, 1)

im = rgb2gray(img_as_float(misc.imread('../images/me4.jpg')))
im_blurred = ndimage.gaussian_filter(im, 5)
im_detail = np.clip(im - im_blurred, 0, 1)
pylab.gray()
fig, axes = pylab.subplots(nrows=2, ncols=3, sharex=True, sharey=True, figsize=(15, 15))
axes = axes.ravel()

axes[0].set_title('Original image', size= 15), axes[0].imshow(im) axes[1].set_title('Blurred image, sigma=5',
size= 15), axes[1].imshow(im_blurred)axes[2].set_title('Detail image', size= 15), axes[2].imshow(im_detail)

alpha = [1, 5, 10]

for i in range(3):
    im_sharp = np.clip(im + alpha[i]*im_detail, 0, 1)
    axes[3+i].imshow(im_sharp), axes[3+i].set_title('Sharpened image, alpha=' + str(alpha[i]),
size= 15) for ax in axes:
    ax.axis('off')
fig.tight_layout()
pylab.show()
```


Original image
Blurred image, sigma=5
Detail image
Sharpened image, alpha=1
Sharpened image, alpha=5
Sharpened image, alpha=10

Practical No: 10

Aim: Write the program to apply segmentation for detecting lines, circles, and other shapes/objects. Also, implement edge-based and region-based segmentation.

A: Hough transform – detecting lines and circles

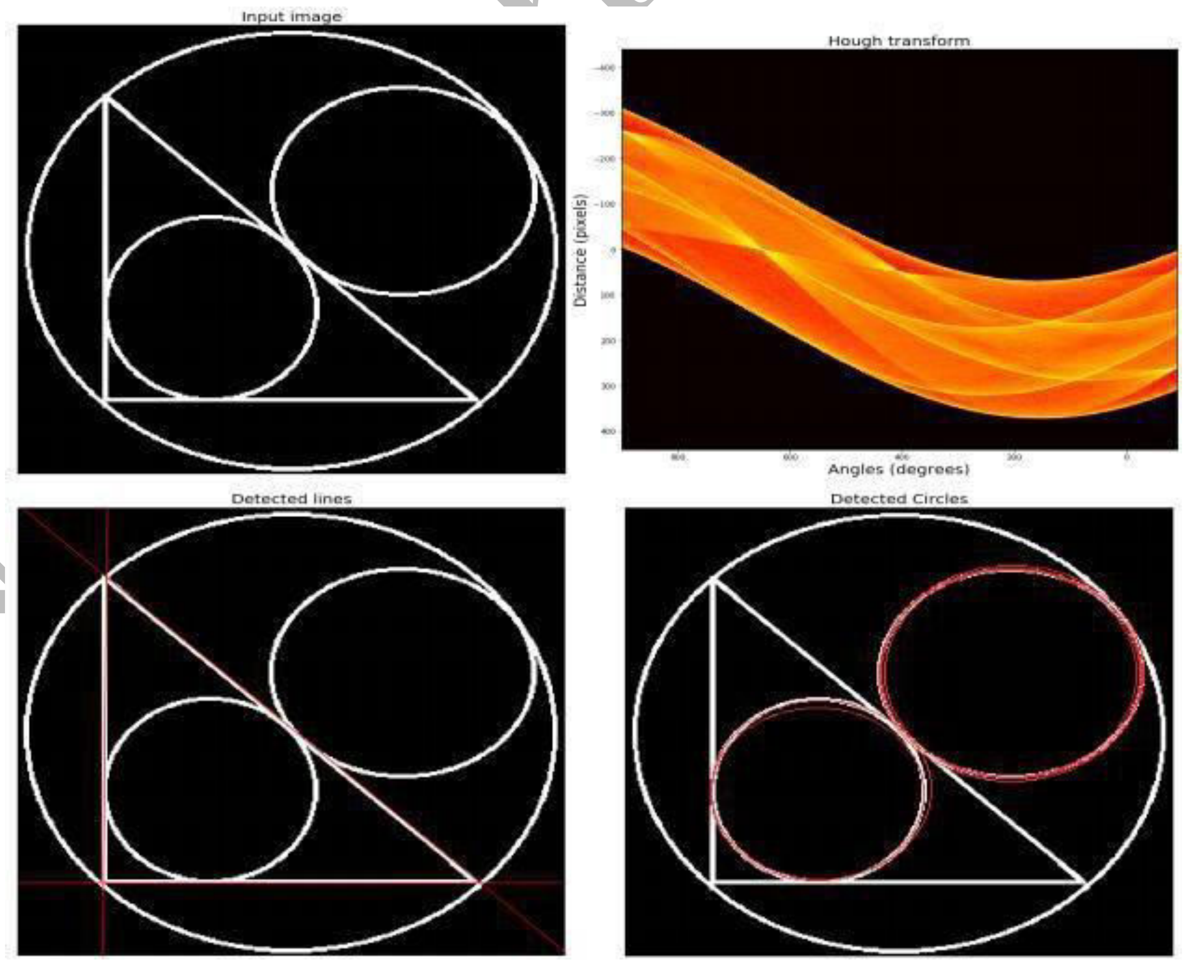
The Hough transform (HT) can be used to detect lines circles or • The Hough transform (HT) can be used to detect lines, circles or other parametric curves.

Program code:

```
image = rgb2gray(imread('../images/triangle_circle.png'))
fig, axes = plt.subplots(2, 2,figsize=(20, 20))
ax = axes.ravel()
ax[0].imshow(image, cmap=cm.gray)
ax[0].set_title('Input image', size=20)
ax[0].set_axis_off()
ax[1].imshow(np.log(1 + h),
extent=[10*np.rad2deg(theta[-1]), np.rad2deg(theta[0]), d[-1], d[0]],
cmap=cm.hot, aspect=1/1.5)
ax[1].set_title('Hough transform', size=20)
ax[1].set_xlabel('Angles (degrees)', size=20)
ax[1].set_ylabel('Distance (pixels)', size=20)
ax[1].axis('image')
ax[2].imshow(image, cmap=cm.gray)
for _, angle, dist in zip(*hough_line_peaks(h, theta, d)):
y0 = (dist - 0 * np.cos(angle)) / np.sin(angle)
y1 = (dist - image.shape[1] * np.cos(angle)) / np.sin(angle)
ax[2].plot((0, image.shape[1]), (y0, y1), '-r')
ax[2].set_xlim((0, image.shape[1]))
 ax[2].set_ylim((image.shape[0], 0))
ax[2].set_axis_off()
ax[2].set_title('Detected lines', size=20)
hough_radii = np.arange(50, 100, 2)
hough_res = hough_circle(image, hough_radii)
accums, cx, cy, radii = hough_circle_peaks(hough_res, hough_radii, total_num_peaks=6)
image = gray2rgb(image)
for center_y, center_x, radius in zip(cy, cx, radii):
circy, circx = circle_perimeter(center_y, center_x, radius)
image[circy, circx] = (0.9, 0.2, 0.2)
ax[3].imshow(image, cmap=plt.cm.gray)
ax[3].set_axis_off()
ax[3].set_title('Detected Circles', size=20)
```

```
plt.tight_layout()
plt.show()
image = rgb2gray(imread('../images/coins.png'))
fig, axes = plt.subplots(1, 2,figsize=(20, 10),sharex=True, sharey=True)
ax = axes.ravel()
ax[0].imshow(image, cmap=plt.cm.gray)
ax[0].set_axis_off()
ax[0].set_title('Original Image', size=20)
hough_radii = np.arange(65, 75, 1)
hough_res = hough_circle(image, hough_radii)
accums, cx, cy, radii = hough_circle_peaks(hough_res, hough_radii, total_num_peaks=4)
image = gray2rgb(image)
for center_y, center_x, radius in zip(cy, cx, radii):
circy, circx = circle_perimeter(center_y, center_x, radius)
image[circy, circx] = (1, 0, 0)
ax[1].imshow(image, cmap=plt.cm.gray)
ax[1].set_axis_off()
ax[1].set_title('Detected Circles', size=20)
plt.tight_layout()
plt.show()
```

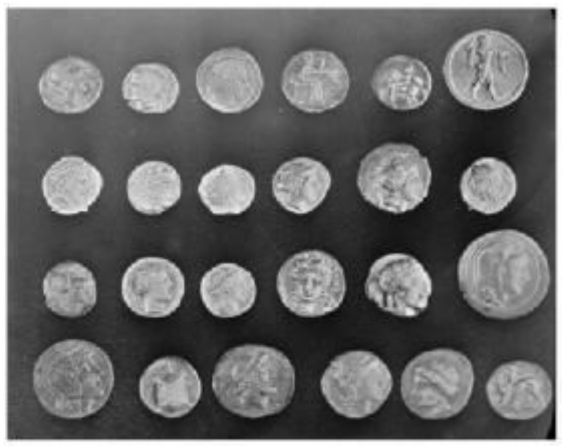
<u>Output:</u>

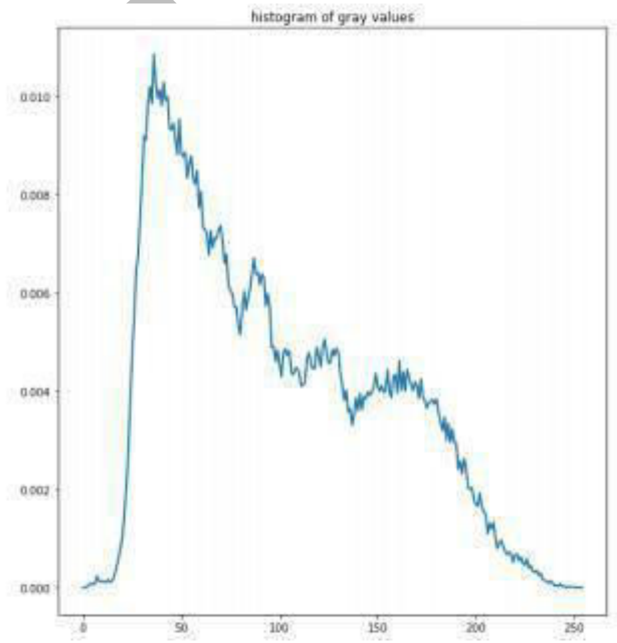


B: edge-based and region-based segmentation

Program Code:

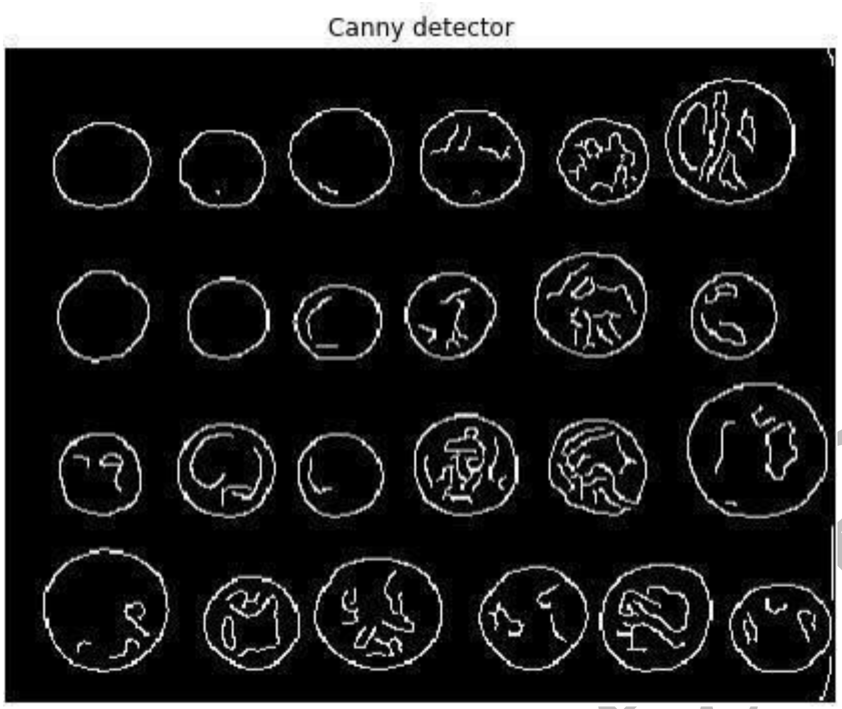
```
coins = data.coins()
hist = np.histogram(coins, bins=np.arange(0, 256), normed=True)
fig, axes = plt.subplots(1, 2,figsize=(20, 10))
axes[0].imshow(coins, cmap=plt.cm.gray, interpolation='nearest')
axes[0].axis('off')
axes[1].plot(hist[1][:-1], hist[0], lw=2)
axes[1].set_title('histogram of gray values')
plt.show()
```



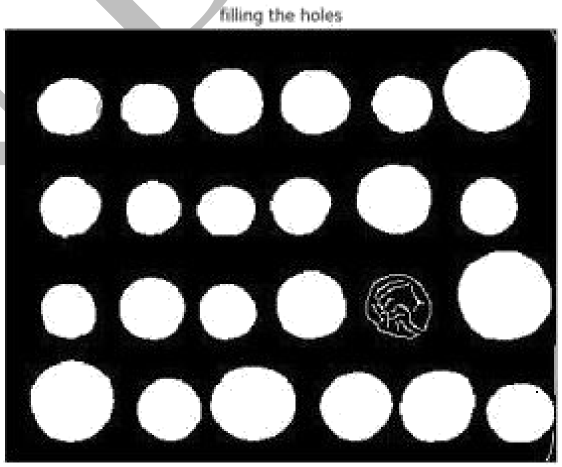


```
edges = canny(coins, sigma=2)
fig, ax = plt.subplots(figsize=( 10, 6))
ax.imshow(edges, cmap=plt.cm.gray, interpolation='nearest')
ax.set_title('Canny detector')
ax.axis('off')
```

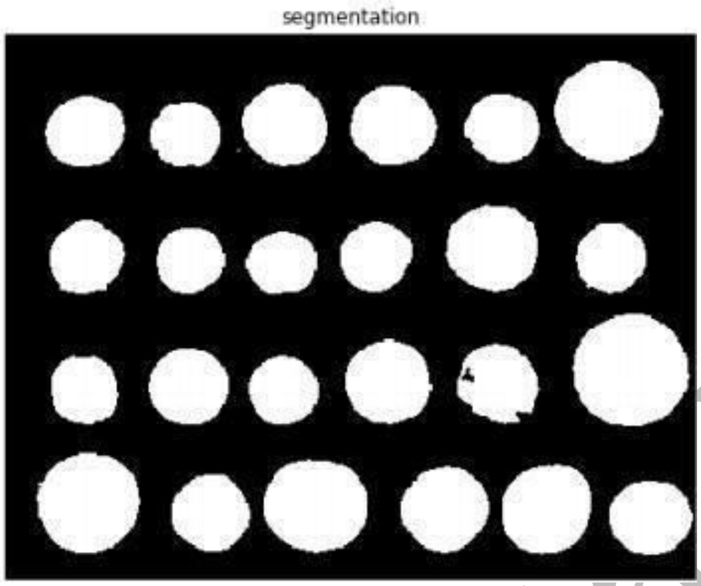
```
plt.show()
```



```
from scipy import ndimage as ndi
fill_coins = ndi.binary_fill_holes(edges)
fig, ax = plt.subplots(figsize=( 10, 6))
ax.imshow(fill_coins, cmap=plt.cm.gray, interpolation='nearest')
ax.set_title('filling the holes')
ax.axis('off')
plt.show()
```



```
ax.set_title('segmentation')

ax.axis('off')

plt.sho
```


segmentation