Subject: IMAGE PROCESSING

Branch: AIR/AIML/AIDS

Semester: 6th Sem

LAB

Introduction

We will cover the following topics Digital image fundamentals, Transformation in Spatial Domain, Transformation in Frequency Domain, Restoration and Reconstruction, Color Image Processing, Wavelets and Multi-resolution Processing, Image compression, Morphological Processing, Segmentation.

Objective

Introduction to the basic concept and methodology of Digital image Processing like image acquisition, enhancement, color image processing, restoration, compression, morphological processing, segmentation and its use in current systems that handle visual information, including television, photographs, x-rays etc. and further study and research in this field.

Scope

Support visual communication

Facilitate inspection, diagnosis of complex systems like Human body

Manufacturing

Entertainment

Keep record, history

Managing multimedia information

Security, monitoring, watermarking, etc.

Instructions

- Write and execute all programs in Python.
- Do not use standard Python functions whenever specified. Write program yourself without using standard functions.

References:

[1] R. C. Gonzalez, R. E. Woods and S. L. Eddins, Digital Image Processing using MATLAB. Pearson Education, Inc., 2004 (Library Dewey number 621.368 GON).

[2] A. K. Jain, Fundamentals of Digital Image Processing. Prentice-Hall, Inc., 1989 (Library Dewey number 621.368 JAI).

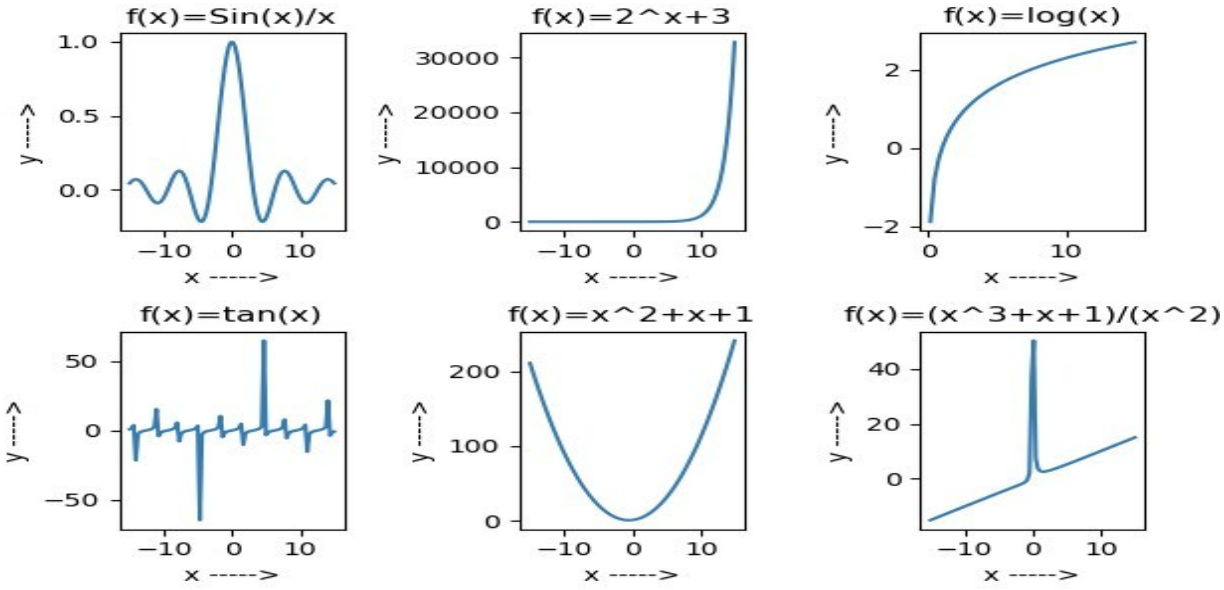
List of Experiments

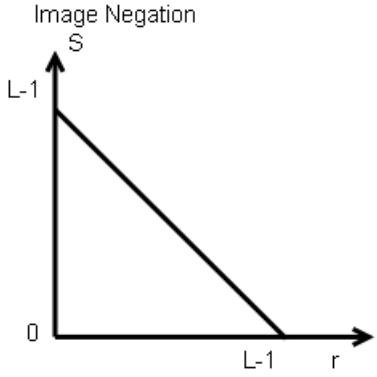
	b. Thresholding of an image	
3	**Contrast Stretching of an image**	
4	**Bit Plane Slicing**	
5	**Histogram Equalization**	
6	**Zooming by interpolation and replication**	
7	**Filtering in spatial domain** a. Low Pass Filtering b. High Pass Filtering c. Median filtering	
8	**To implement median filtering in spatial domain after adding salt and pepper noise**	
9	**Edge Detection using derivative filter mask** a. Canny b. Sobel	
10	**Mini Project on Image Processing Applications**	

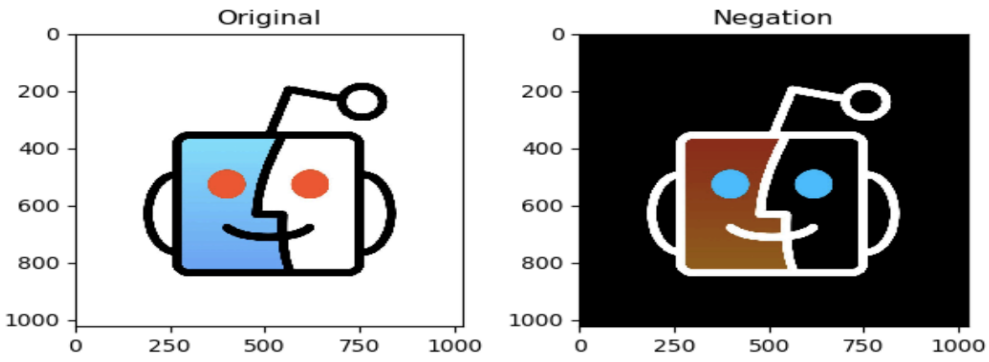
Experiment No. 1	**To study various signals in python.**
Aim	To study and plot the various basic signals: unit step, sine, cosine, exponential signal, square wave, impulse function etc.
Tool	Python
Theory	**Unit Step** The unit step function u(t) is basically a mathematical function that is defined by: $$u(t) = \begin{cases} 0, & t < 0 \\ 1, & t \geq 0 \end{cases}$$ **Unit Impluse**

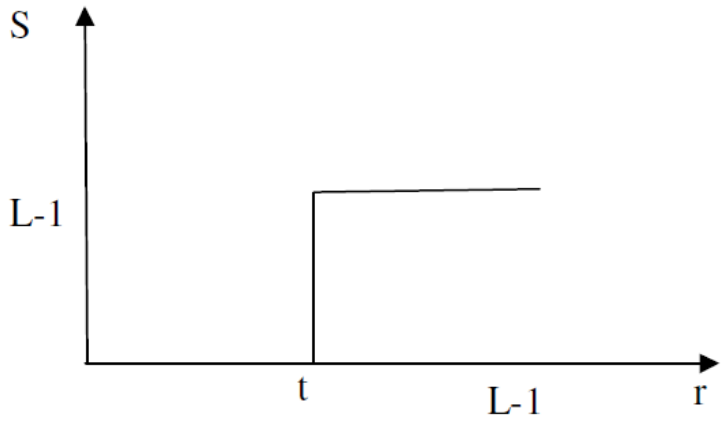
	it impulse function is not a function in the strict sense mean while it is very useful in the mathematical analysis of signals and specially signal which has discontinuities or sudden changes. It is mathematically defined as: $$\delta(t) = \begin{cases} 1, & t = 0 \\ 0, & t \neq 0 \end{cases}$$ **Sine Wave** A sine wave or sinusoid is a mathematical curve that describes a smooth periodic oscillation. A sine wave is a continuous wave. It is named after the function sine, of which it is the graph. Its most basic form as a function of time (t) is: $$x(t) = ASin(wt + \varphi)$$ **Exponential Signal** This function, also denoted as is called the "natural exponential function", or simply "the exponential function". Since any exponential function can be written in terms of the natural exponential as. It is computationally and conceptually convenient to reduce the study of exponential functions to this particular one. The natural exponential is hence denoted by: $$x(t) = e^t$$
Code	import numpy as np import cv2 import random import matplotlib.pyplot as plt x = np.linspace(-10, 10, 100) y = np.sin(x)/x plt.subplot(2, 3, 1) plt.title('f(x)=Sin(x)/x') plt.xlabel('x ----->') plt.ylabel('y ----->') plt.plot(x,y,'-.') y = 2**x+3 plt.subplot(2, 3, 2) plt.title('f(x)=2^x+3') plt.xlabel('x ----->') plt.ylabel('y ----->')

	plt.plot(x,y) y = np.log(x) plt.subplot(2, 3, 3) plt.title('f(x)=log(x)') plt.xlabel('x ----->') plt.ylabel('y ----->') plt.plot(x,y) y = np.tan(x) plt.subplot(2, 3, 4) plt.title('f(x)=tan(x)') plt.xlabel('x ----->') plt.ylabel('y ----->') plt.plot(x,y) y = x**2+x+1 plt.subplot(2, 3, 5) plt.title('f(x)=x^2+x+1') plt.xlabel('x ----->') plt.ylabel('y ----->') plt.plot(x,y) y = (x**3+x+1)/(x**2) plt.subplot(2, 3, 6) plt.title('f(x)=(x^3+x+1)/(x^2)') plt.xlabel('x ----->') plt.ylabel('y ----->') plt.plot(x,y) plt.show()
Conclusion & Results	


f(x)=Sin(x)/x
f(x)=2^x+3
f(x)=log(x)
f(x)=tan(x)
f(x)=x^2+x+1
f(x)=(x^3+x+1)/(x^2)
y ----->
x ----->
1.0
0.5
0.0
30000
20000
10000
0
2
−2
50
−50
200
100
40
20
−10
10

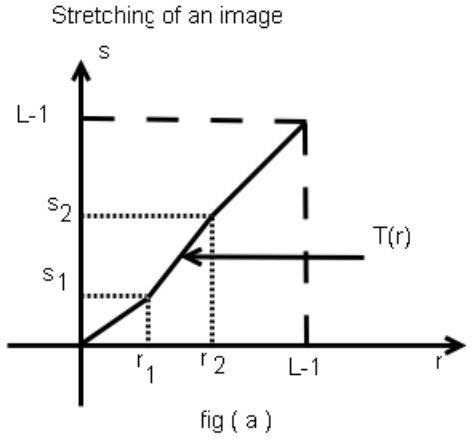
Experiment No. 2A	**Negation of an image**
Aim	To study image negative
Tool	Python
Theory	The negative of an image with gray levels in the range [0, L-1] is obtained by using the negative transformation given by the expression $S = L - 1 - r$ (1) This is according to the transformation $S = T(r)$ In above transformation (1), the intensity of the output image decreases as the intensity of the input increases. The type of processing is particularly suited for enhancing white or gray detail embedded in dark regions of an image especially when black areas are dominants in site. 
Code	*import numpy as np* *import cv2* *import random* *import matplotlib.pyplot as plt* *img = cv2.imread('test.jpg')* *5* *a = np.array(img.data)* *maximum = a.max()* *b = maximum-a* *plt.subplot(1,2,1)* *plt.title('ORIGINAL')* *plt.imshow(cv2.cvtColor(a, cv2.COLOR_BGR2RGB))* *plt.subplot(1,2,2)* *plt.title('NEGATION')* *plt.imshow(cv2.cvtColor(b, cv2.COLOR_BGR2RGB))* *plt.show()*

Conclusion & Results	

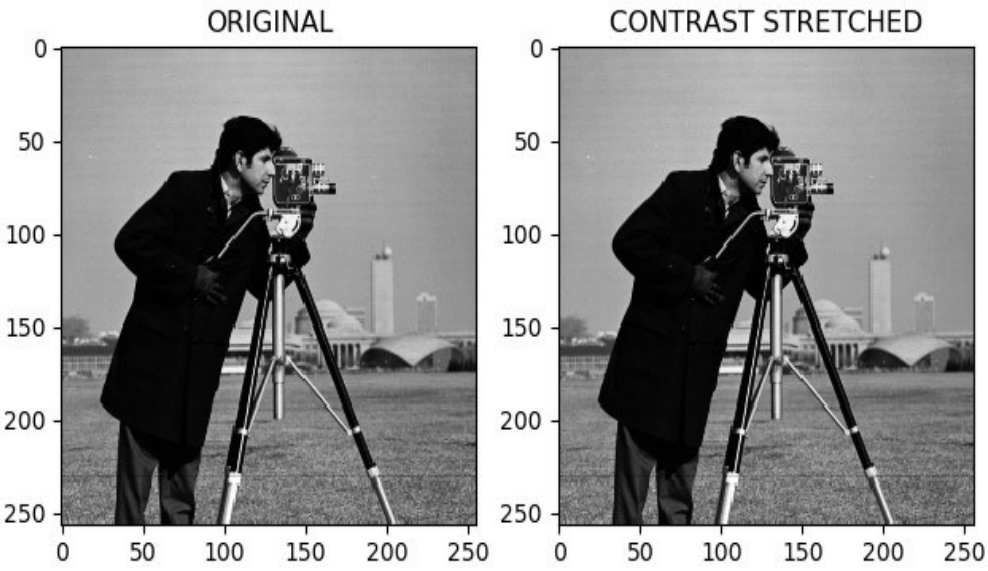
Experiment No. 2B	**Thresholding of an Image**
Aim	To study thresholding of the image
Tool	Python
Theory	Thresholding is a simple process to separate the interested object from the background. It gives the binary image. The formula for achieving thresholding is as follows: $s = 0 \; if \; r <= t$ $s = L - 1 \; if \; r > t$ 
Algorithm	1. Read input image 2. Enter thresholding value t 3. If image pixel is less than t replace it by zero. 4. If image pixel is > t replace it by 255 5. Display input image 6. Display threshold image 7. Write input image 8. Write threshold image

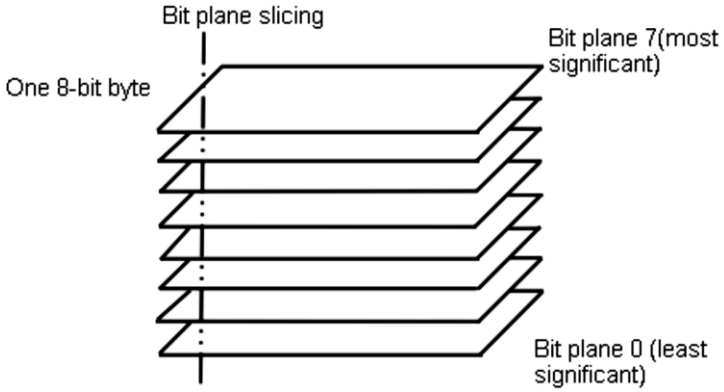
Code	import numpy as
	np import cv2
	import random
	import matplotlib.pyplot as plt
	img1 = cv2.imread('test.jpg', cv2.IMREAD_GRAYSCALE)
	c = np.array(img1.data)
	plt.subplot(1,2,1)
	plt.title('GRAYSCALE')
	plt.imshow(c, cmap='gray')
	maximum = c.max() minimum = c.min()
	threshold = (maximum + minimum)/2
	
	for i in range(c.shape[0]): for j in range(c.shape[1]):
	if c[i][j] >= threshold: c[i][j] = 255
	else:

	c[i][j] = 0 plt.subplot(1,2,2) plt.title('BINARY') plt.imshow(c, cmap='gray') plt.show()
Conclusion & Results	  (b) Grey image (a) Binary image

Experiment No. 3	***Contrast Stretching of an Image***
Aim	To study Contrast Stretching of an image
Tool	Python
Theory	Low contrast images can result from poor illumination, lack of dynamic range in the imaging sensor etc. The idea behind contrast stretching is to increase the dynamic range of the gray levels in the image being processed. The transformation function for contrast stretching is given by $T(r) = \alpha \times r \quad , r \leq r_1$ $T(r) = \beta \times (r - r_1) + s_1 \quad , r_1 < r \leq r_2$ $T(r) = \gamma \times (r - r_2) + s_2 \quad , r_2 < r \leq L - 1$  fig (a) The location of the points(r_1, s_1) & (r_2, s_2) control the shape of the Transformation function.
Algorithm	1. Read input image 2. Enter values r_1, s_1, r_2, s_2 3. Calculate α, β, and γ slopes. 3. if input pixel value is <= r_1 then o/p = $\alpha \times input$ 5. If input pixel is > $r_1$and <=r_2 then o/p = $\beta \times (r - r_1) + s_1$ 6. otherwise o/p = $\gamma \times (r - r_2) + s_2$ 7. Display i/p image 8. Display o/p image.
Code	img = cv2.imread('cameraman.jpg', cv2.IMREAD_GRAYSCALE) plt.subplot(1,2,1) plt.title('ORIGINAL') plt.imshow(img, cmap='gray') b = img.shape

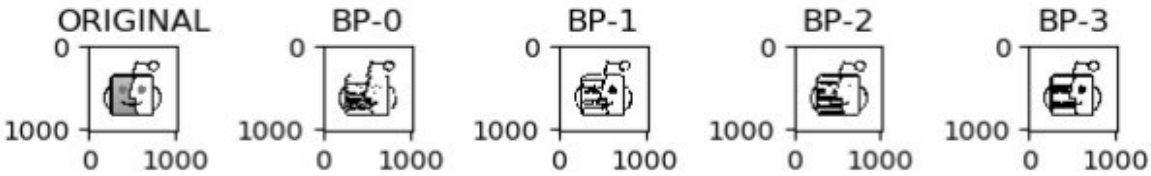
	contrastedImg = np.zeros((b[0],b[1]))
	print("Enter value of r1,r2,s1 and s2 : ")
	r1 = int(input())
	r2 = int(input()) s1 = int(input()) s2 = int(input()) alpha = s1/r1
	beta = (s2-s1)/(r2-r1) gamma = (15-s2)/(15-r2)
	print(alpha,beta,gamma) """
	s=alpha*r (if r<=r1)
	s=beta*(r-r1)+s1 (if r1<r<=r2) s=gamma*(r-r2)+s2 (if r>r2) """
	for i in range(b[0]): for j in range(b[1]):
	if img[i][j]<=r1: contrastedImg[i][j]=alpha*img[i][j]
	elif img[i][j]<=r2: contrastedImg[i][j]=beta*(img[i][j]-r1)+s1
	else:
	contrastedImg[i][j]=gamma*(img[i][j]-r2)+s2
	
	plt.subplot(1,2,2) plt.title('CONTRAST STRETCHED')
	plt.imshow(contrastedImg, cmap='gray') plt.show()

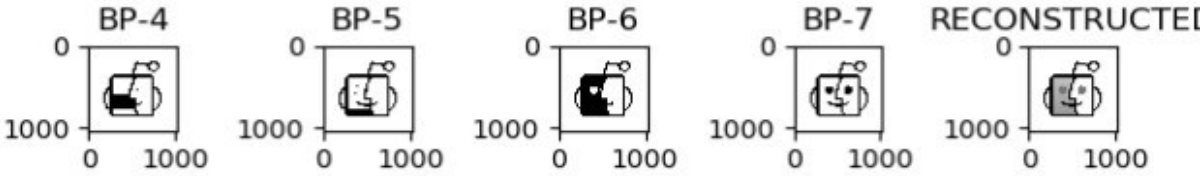
Conclusion & Results	

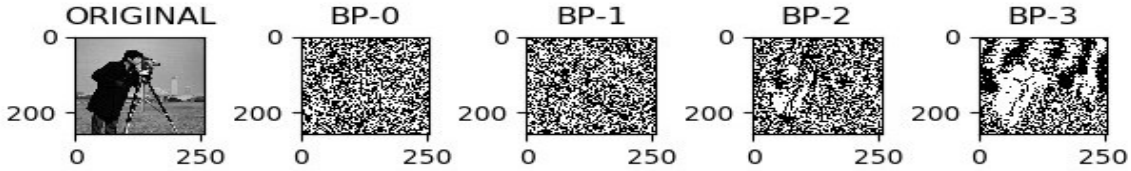
Experiment No. 4	**Bit Plane Slicing**
Aim	To study Bit Plane Slicing
Tool	Python
Theory	This transformation involves determining the number of usually significant bits in an image. In case of an 8 bit image each pixel is represented by 8 bits. Imagine that the image is composed of eight 1 bit planes ranging from bit plane 0 for the least significant bit to bit plane 7 for the most significant bit. Plane 0 contains all the lowest order bits in the bytes comprising the pixels in the image & plane 7 contains all the high order bits. The higher order bits contain usually significant data and the other bit planes contribute to more subtle details in the image. Separating a digital image into its bit planes is useful for analyzing the relative importance played by each bit of the image. 
Algorithm	1.Read i/p image 2. Use bit and operation to extract each bit 3. Do the step 2 for every pixel. 4. Display the original image and the biplanes formed by bits extracted
Code	import numpy as np import cv2 import random import matplotlib.pyplot as plt img1 = cv2.imread('test.jpg', cv2.IMREAD_GRAYSCALE) c = np.array(img1.data) plt.subplot(2,5,1)

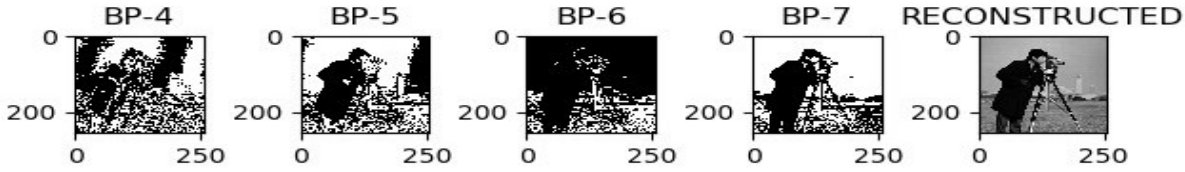
	plt.title('ORIGINAL') plt.imshow(c,cmap='gray')
	finalImage = [[0 for i in range(c.shape[1])]for j in
	range(c.shape[0])] for k in range(8):
	bitPlane = []
	for i in range(c.shape[0]): a = []
	for j in range(c.shape[1]): if c[i][j]% 2 == 1:
	a.append(255) else:
	a.append(0) c[i][j] = c[i][j]/2
	bitPlane.append(a) img = np.array(bitPlane) for i in
	range(c.shape[0]):
	for j in range(c.shape[1]): if img[i][j] == 255:
	finalImage[i][j] = finalImage[i][j] + np.power(2, k)
	plt.subplot(2,5,k+2)
	plt.title('BP-'+str(k)) plt.imshow(img,cmap='gray')
	
	fimg = np.array(finalImage) plt.subplot(2,5,10)
	plt.imshow(fimg,cmap='gray') plt.title('RECONSTRUCTED')
	plt.show()

Conclusion & Results	


ORIGINAL
BP-0
BP-1
BP-2
BP-3
0
1000
0
1000


BP-4
BP-5
BP-6
BP-7
RECONSTRUCTEI
0
1000
0
1000

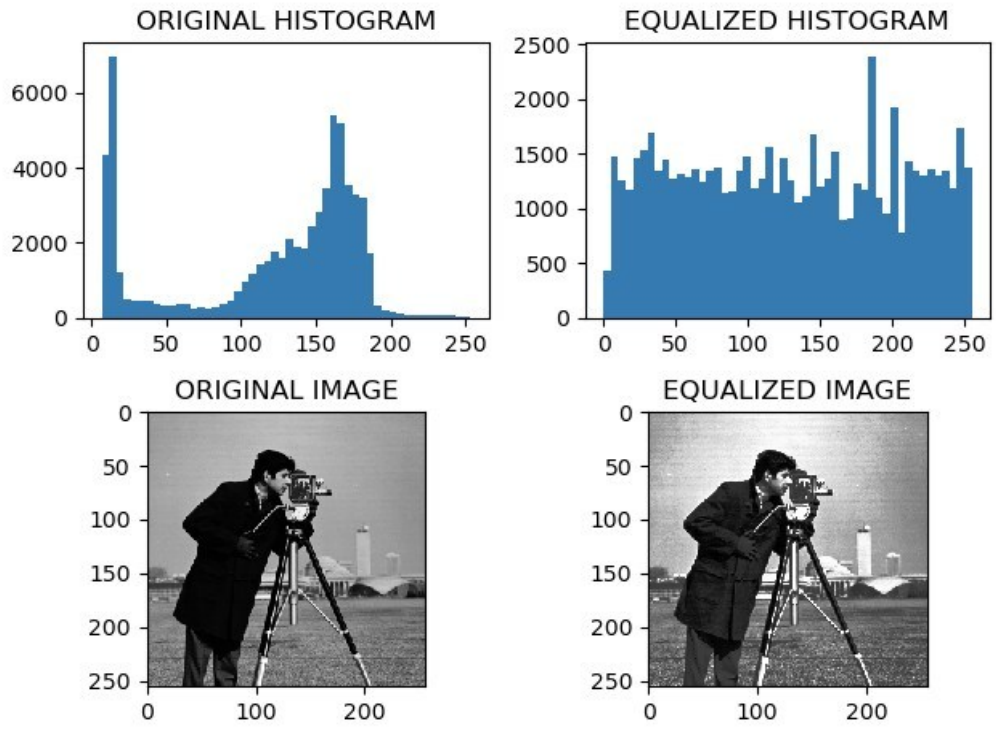

ORIGINAL
BP-0
BP-1
BP-2
BP-3
0
200
0
250


BP-4
BP-5
BP-6
BP-7
RECONSTRUCTED
0
200
0
250

Experiment No. 5	**Histogram Equalization**
Aim	To implement histogram equalization.
Tool	Python
Theory	Histogram of a digital image with gray levels in range [0, L-1] is a discrete function $h(r_k) = n_k$ where r_k kth gray level and n_k = no. of pixels of an image having gray level r_k. In histogram there are 3 possibilities as follows, 1. For a dark image the components of histogram on the low (dark) side. 2. For a bright image the component are on high (bright) side & 3. For an image with low contrast they are in the middle of gray side. Histogram equalization is done to spread their component uniformly over the gray scale as far as possible. This is obtained by function $S_k = \sum_{i=0}^{k} h_i/n \qquad k = 0,1,2, \ldots \ldots i-1$ Thus processed image is obtained by mapping each pixel with level r_k into a corresponding pixel with level S_k in o/p image. This transformation is called Histogram equalization.
Algorithm	1. Read the i/p image & its size. 2. Obtain the gray level values of each pixel & divide them by total number of gray level values. 3. Implement the function S_k 4. Plot the equalized histogram and original histogram. 5. Display the original and the new image.

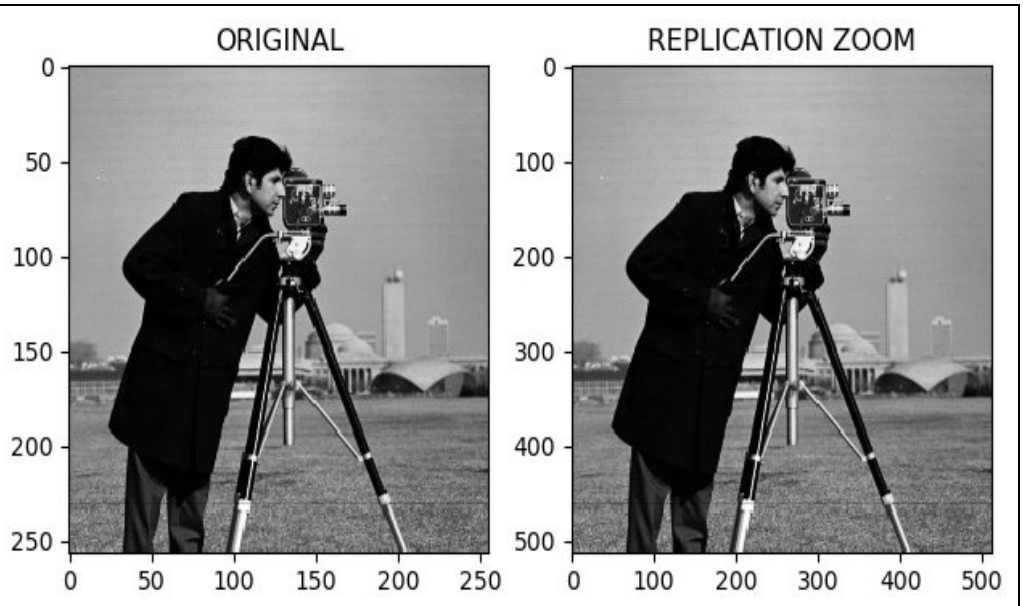
Code	import numpy as np import cv2
	import random
	import matplotlib.pyplot as plt
	
	img = cv2.imread('cameraman.jpg', cv2.IMREAD_GRAYSCALE)
	img = np.array(img)
	flat = img.flatten() plt.subplot(2,2,1)
	plt.title('ORIGINAL HISTOGRAM')
	plt.hist(flat, bins=50)
	def get_histogram(image, bins): histogram = np.zeros(bins) for pixel in image:
	histogram[pixel] += 1 return histogram
	hist = get_histogram(flat, 256)
	# plt.plot(hist)
	
	def cumsum(a):
	
	a = iter(a)
	b = [next(a)] for i in a:

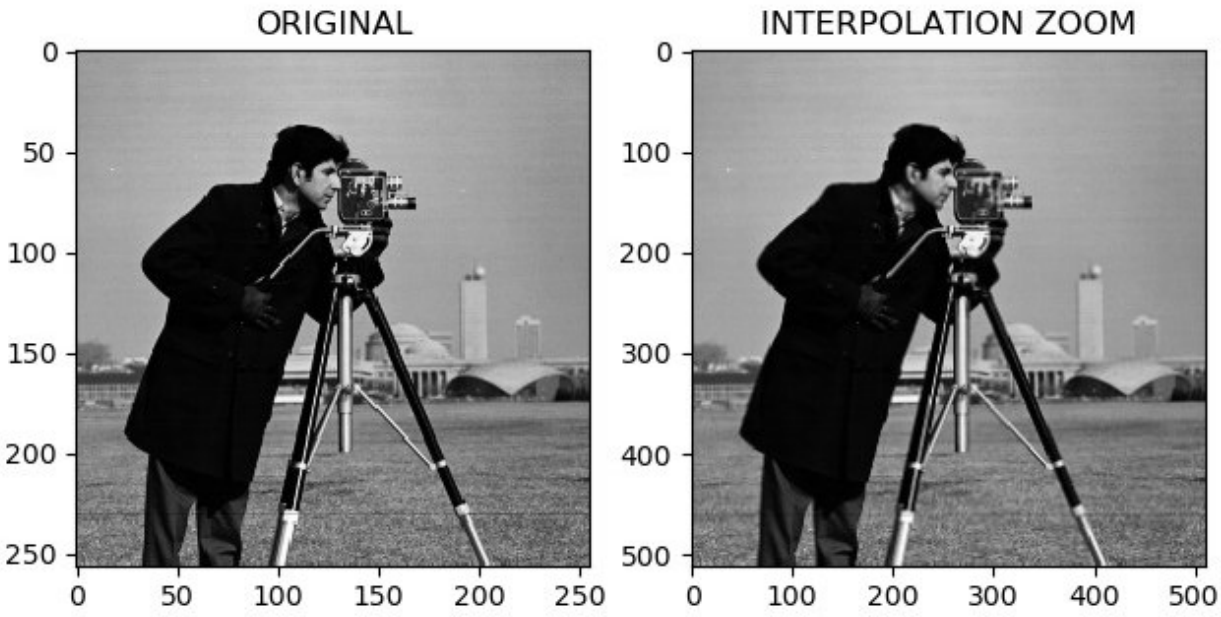
	b.append(b[-1] + i) return np.array(b)
	cs = cumsum(hist)
	# plt.plot(cs)
	nj = (cs - cs.min()) * 255 N = cs.max() - cs.min() cs = nj / N
	# plt.plot(cs) img_new = cs[flat] plt.subplot(2,2,2)
	plt.title('EQUALIZED HISTOGRAM')
	plt.hist(img_new, bins=50)
	img_new = np.reshape(img_new, img.shape)
	# print(img_new) plt.subplot(2,2,3) plt.title('ORIGINAL IMAGE')
	plt.imshow(img, cmap='gray') plt.subplot(2,2,4)
	plt.title('EQUALIZED IMAGE')
	plt.imshow(img_new, cmap='gray') plt.show()

Conclusion & Results	

Experiment No. 6	**Zooming by interpolation and replication**
Aim	To implement the magnification by replication and interpolation
Tool	Python
Theory	Zooming can be done in two ways. *1) Replication:* In replication we simply replicate each pixel and then replicate each row. Hence image of size n x n is zoomed to 2nx2n. Zooming by replication gives the final image a patchy look since clusters of grey levels are formed. This can be substantially reduced by using a better method of zooming known as interpolation. *2) Interpolation:* In this method instead of replicating each pixel, average of two adjacent pixels along the rows is taken and placed between two pixels. The same operation is then performed along the columns. The patchiness that was present in the replicated image is much less in the interpolated image.
Algorithm	*Replication:* 1. Read i/p image. 2. Replicate each pixel 3. Replicate each row 4. Display o/p image *Interpolation:* 1. Read i/p image 2. Average of two adjacent pixels along the rows is taken and placed between two pixels. 3. Do the same along columns 4. Display o/p image.
Code	*# REPLICATION* *img = cv2.imread('cameraman.jpg', cv2.IMREAD_GRAYSCALE)* *img = np.array(img)* *plt.subplot(1, 2, 1) plt.title('ORIGINAL') plt.imshow(img, cmap='gray') print(img)* *# ROW WISE ZOOMING* *for i in range(0, img.shape[0]*2, 2): x = img[:,i:i+1]* *y = x.flatten()* *x = np.insert(img, i, y, 1) img = x* *# COLUMN WISE ZOOMING* *for i in range(0, img.shape[0]*2, 2): x = img[i:i+1,:]* *y = x.flatten()* *x = np.insert(img, i, y, 0) img = x*

```
print(img) plt.subplot(1, 2, 2)

plt.title('REPLICATION ZOOM')
plt.imshow(img, cmap='gray') plt.show()

# INTERPOLATION
img = cv2.imread('cameraman.jpg', cv2.IMREAD_GRAYSCALE)
img = np.array(img)
plt.subplot(1, 2, 1) plt.title('ORIGINAL') plt.imshow(img,
cmap='gray')
#print(img)

# ROW WISE ZOOMING
for i in range(1, 2*img.shape[0]-1, 2): x = img[:,i-1:i]
x = x.flatten() y = img[:,i:i+1] y = y.flatten()
z = np.add(x,y,dtype=np.int64) x = np.insert(img, i, z/2, 1) img =
x

# COLUMN WISE ZOOMING
for i in range(1, 2*img.shape[0]-1, 2): x = img[i - 1:i, :]
x = x.flatten()
y = img[i:i + 1, :] y = y.flatten()
z = np.add(x,y,dtype=np.int64) x = np.insert(img, i, z/2, 0) img =
x

#print(img) plt.subplot(1, 2, 2)
plt.title('INTERPOLATION ZOOM')
plt.imshow(img, cmap='gray') plt.show()
```

Conclusion & Results	



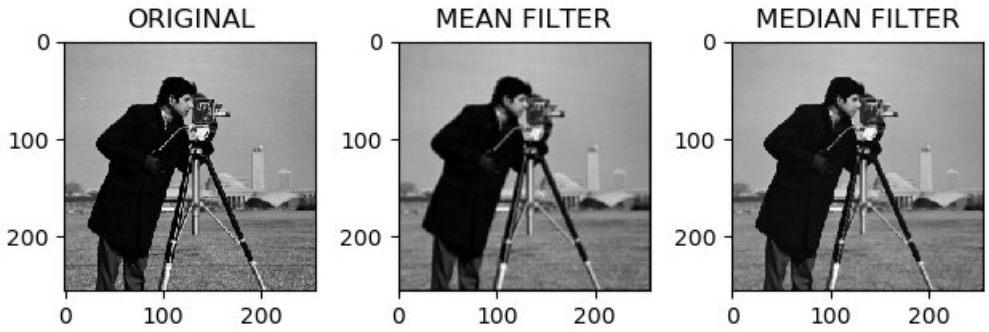
<table>
<tr><td>Experiment No. 7</td><td>Filtering in spatial domain: Low pass filtering</td></tr>
<tr><td>AIM</td><td>To implement low pass filtering in spatial domain</td></tr>
<tr><td>TOOL</td><td>Python</td></tr>
<tr><td>THEROY</td><td>Low pass filtering as the name suggests removes the high frequency content from the image. It is used to remove noise present in the image. Mask for the low pass filter is :

<table><tr><td>1/9</td><td>1/9</td><td>1/9</td></tr><tr><td>1/9</td><td>1/9</td><td>1/9</td></tr><tr><td>1/9</td><td>1/9</td><td>1/9</td></tr></table>
One important thing to note from the spatial response is that all the coefficients are positive. We could also use 5 x 5 or 7 x 7 mask as per our requirement. We place a 3 x 3 mask on the image. We start from the left hand top corner. We cannot work with the borders and hence are normally left as they are. We then multiply each component of the image with the corresponding value of the mask. Add these values to get the response. Replace the center pixel of the o/p image with these response. We now shift the mask towards the right till we reach the end of the line and then move it downwards.</td></tr>
<tr><td>ALGORITHM</td><td>1. Read I/p image
2. Ignore the border pixel
3. Apply low pass mask to each and every pixel.
4. Display the o/p image</td></tr>
<tr><td>CODE</td><td>import numpy as np
import cv2
import random
import matplotlib.pyplot as plt

img = cv2.imread('cameraman.jpg', cv2.IMREAD_GRAYSCALE)
img = np.array(img)
plt.subplot(1, 3, 1) plt.title('ORIGINAL') plt.imshow(img, cmap='gray')

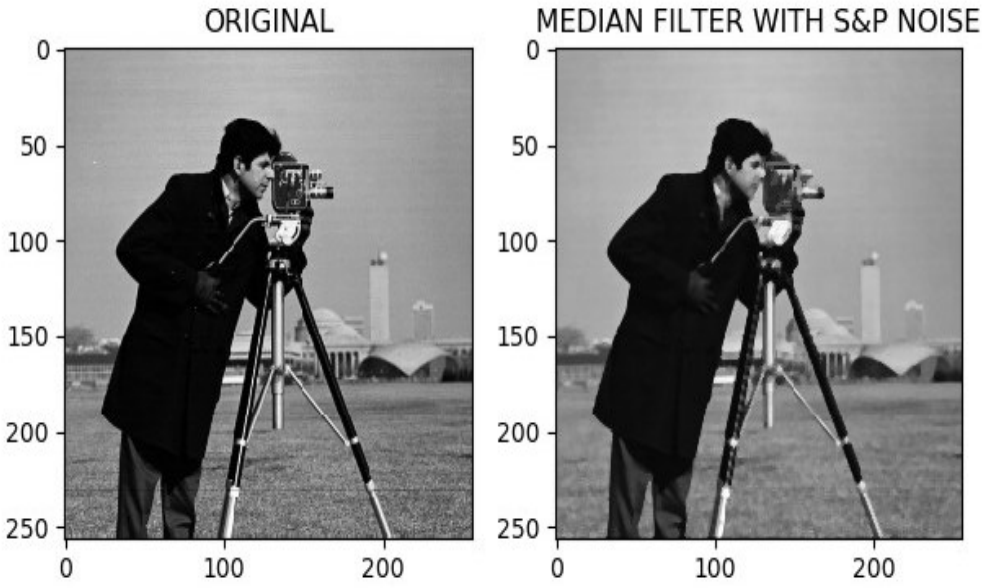
temp = [[0 for i in range(img.shape[1])]for j in range(img.shape[0])] img = np.pad(img, pad_width=1, mode='constant', constant_values=0) print(img)
meanFilter = [[1,1,1],[1,1,1],[1,1,1]]

for i in range(img.shape[0]-2): for j in range(img.shape[1]-2): temp[i][j]=np.sum(img[i:i+3,j:j+3])/9</td></tr>
</table>

	temp = np.array(temp) plt.subplot(1, 3, 2) plt.title('MEAN FILTER') plt.imshow(temp, cmap='gray') for i in range(img.shape[0]-2): for j in range(img.shape[1]-2): temp[i][j]=np.median(img[i:i+3,j:j+3]) temp = np.array(temp) plt.subplot(1, 3, 3) plt.title('MEDIAN FILTER') plt.imshow(temp, cmap='gray') plt.show()
Conclusion & Results	

Experiment No. 8	**To implement median filtering in spatial domain after adding salt and pepper noise**
Aim	**To implement median filtering in spatial domain after adding salt and pepper noise**
Tool	Python
Theory	Median filtering is a signal processing technique developed by Tukey that is useful for noise suppression in images. Here the input pixel is replaced by the median of the pixels contained in the window around the pixel. The median filter disregards extreme values and does not allow them to influence the selection of a pixel value which is truly representative of the neighborhood.

Code	import numpy as np import cv2
	import random
	import matplotlib.pyplot as plt
	
	img = cv2.imread('cameraman.jpg', cv2.IMREAD_GRAYSCALE)
	img = np.array(img)
	plt.subplot(1, 2, 1) plt.title('ORIGINAL')
	plt.imshow(img, cmap='gray')
	
	temp = [[0 for i in range(img.shape[1])]for j in
	range(img.shape[0])] img = np.pad(img, pad_width=1,
	mode='constant', constant_values=0)
	# print(img)
	
	def sp_noise(image,prob):
	output = np.zeros(image.shape,np.uint8) thres = 1 - prob
	for i in range(image.shape[0]): for j in range(image.shape[1]):
	rdn = random.random() if rdn < prob:
	output[i][j] = 0 elif rdn > thres:
	output[i][j] = 255 else:
	output[i][j] = image[i][j]
	
	return output
	
	img = sp_noise(img, 0.01) temp = np.array(temp)
	for i in range(img.shape[0]-2): for j in range(img.shape[1]-2):
	temp[i][j]=np.median(img[i:i+3,j:j+3])
	
	temp = np.array(temp) plt.subplot(1, 2, 2)
	plt.title('MEDIAN FILTER WITH S&P NOISE')

	plt.imshow(temp, cmap='gray') plt.show()
Conclusion & Results	

<table>
<tr><td>Experiment No. 9</td><td>Edge Detection</td></tr>
<tr><td>Aim</td><td>To implement Image segmentation using edge detection technique.</td></tr>
<tr><td>Tool</td><td>Python</td></tr>
<tr><td>Theory</td><td>Image segmentation can be achieved in two ways,
1. Segmentation based on discontinuities of intensity.
2. Segmentation based on similarities in intensity edge detection form an important part. An edge can be defined as a set of disconnected pixels that form a boundary between 2 disjoint regions. Edge detection is achieved through various masks.</td></tr>
<tr><td>Algorithm</td><td>1. Read i/p image & its size
2. Apply Prewitt, Sobel & Laplacian edge masks on i/p image
3. Display i/p image & edge detected image.
Use Prewitt mask
M1
-1 -1 -1
0 0 0
1 1 1
M2
1 0 1
1 0 1
1 0 1
Sobel mask
M1
1 2 1
0 0 0
-1 -2 -1
M2
1 0 -1
2 0 -2
1 0 -1
Laplacian mask
M1
0 -1 0
-1 4 -1
0 -1 0</td></tr>
<tr><td>Code</td><td>import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('cameraman.jpg',
cv2.IMREAD_GRAYSCALE)
img = np.array(img)</td></tr>
</table>

```
plt.subplot(1, 3, 1)
plt.title('ORIGINAL')
plt.imshow(img, cmap='gray')

sobelX = [[0 for i in range(img.shape[1])]
for j in range(img.shape[0])] sobelY = [[0 for i in
range(img.shape[1])]for j in range(img.shape[0])] prewittX = [[0
for i in range(img.shape[1])]for j in range(img.shape[0])] prewittY
= [[0 for i in range(img.shape[1])]for j in range(img.shape[0])]
laplacian = [[0 for i in range(img.shape[1])]for j in
range(img.shape[0])] img = np.pad(img, pad_width=1,
mode='constant', constant_values=0)
# print(img)
sobelGx = [[-1,0,-1],[-2,0,2],[-1,0,1]]
sobelGy = [[-1,-2,-1],[0,0,0],[1,2,1]]
prewittGx = [[-1,0,1],[-1,0,1],[-1,0,1]]
prewittGy = [[-1,-1,-1],[0,0,0],[1,1,1]]
laplacianG = [[0,-1,0],[-1,4,-1],[0,-1,0]]
for i in range(img.shape[0]-2): for j in range(img.shape[1]-2):
sobelX[i][j]=np.sum(np.multiply(sobelGx,img[i:i+3,j:j+3]))
sobelY[i][j] = np.sum(np.multiply(sobelGy, img[i:i + 3, j:j + 3]))
prewittX[i][j] = np.sum(np.multiply(prewittGx, img[i:i + 3, j:j +
3]))

prewittY[i][j] = np.sum(np.multiply(prewittGy, img[i:i + 3, j:j +
3])) laplacian[i][j] = np.sum(np.multiply(laplacianG, img[i:i + 3,
j:j + 3]))
plt.subplot(1, 3, 2) plt.title('SOBEL Gx') plt.imshow(sobelX,
cmap='gray') plt.subplot(1, 3, 3) plt.title('SOBEL Gy')
plt.imshow(sobelY, cmap='gray') plt.show()
plt.subplot(1, 3, 1) plt.title('ORIGINAL') plt.imshow(img,
cmap='gray') plt.subplot(1, 3, 2) plt.title('PREWITT Gx')
plt.imshow(prewittX, cmap='gray') plt.subplot(1, 3, 3)
plt.title('PREWITT Gy') plt.imshow(prewittY, cmap='gray')
plt.show()
plt.subplot(1, 2, 1) plt.title('ORIGINAL') plt.imshow(img,
cmap='gray') plt.subplot(1, 2, 2) plt.title('LAPLACIAN')
plt.imshow(laplacian, cmap='gray') plt.show()
```

Conclusion & Results