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

EXPT. NO. : 4 **DATE: 11/09/2020**

TITLE : TO PERFORM IMAGE ENHANCEMENT IN SPATIAL 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 recordand draw conclusions at a technical level.

AIM:

To implement the following filters using matlab

1. Box Filter
2. Weighted filter.
3. Median filter.
4. Laplacian filter.
5. High boost filter

SOFTWARES REQUIRED : Matlab 7.0. or above

THEORY:

4.1 Basics of filtering operation

Filtering refers to accepting or rejecting certain frequency components. Filtering creates a new pixel with co-ordinates equal to the co-ordinates of the centre of the neighborhood and whose value is the result of the filtering operation. The processed (filtered) image is generated as the centre of the filter mask visits each pixel in the input image.

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 ofpixels composing an image. Spatial domain methods are procedures that operate directly on these pixels. Spatial domain processes will be denoted by the expression

$$g(x, y) = T[f(x, y)]$$

where f(x, y) is the input image, g(x, y) is the processed image, and *T* is an operator on *f*, defined over some neighborhood of (x, y). Some neighborhood operations work with the values of the image pixels in the neighborhood *and* the corresponding values of a sub-image that has the same dimensions as the neighborhood. The sub-image is called a *filter, mask, kernel, template,* or *window.* The process of spatial filtering consists simply of moving the filter mask from point to point in an image. For *linear* spatial filtering the response is given by a sum of products of the filter coefficients and the corresponding image pixels in the area spanned by the filter mask.

When interest lies on the response, *R*, of an m*n mask at any point (x, y), and not on the mechanics of implementing mask convolution, it is common practice to simplify the notation by using the following expression:

$R = w1\ z1 + w2\ z2 + \ldots.. + w_{mn}z_{mn}$

$= \sum_{i=1}^{mn} w_i z_i$ where the w's are mask coefficients, the z's are the values of the image gray levels corresponding to those coefficients, and m*n is the total number of coefficients in the mask. The filtering operation is based conditionally on the values of the pixels in the neighborhood under consideration, and they do not explicitly use coefficients in the sum-of-products.

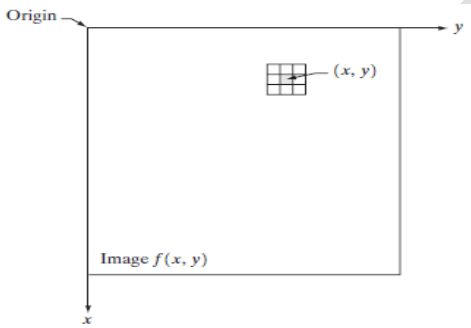

Fig: 4.1 (3*3 neighborhood about a point (x, y) in an image)

4.2 Types of spatial filtering:

4.2.1 Smoothing filters

Smoothing filters are used for blurring and for noise reduction. Blurring is used in preprocessing steps, such as removal of small details from an image prior to (large) object extraction, and bridging of small gaps in lines or curves. Noise reduction can be accomplished by blurring with a linear filter and also by nonlinear filtering.

4.2.2 Smoothing Linear Filters

The output of a smoothing, linear spatial filter is the average of the pixels contained in the neighborhood of the filter mask. These filters sometimes are called *averaging filters*. It replaces the value of every pixel in an image by the average of the gray levels in the neighborhood defined by the filter mask, this process results in an image with reduced "sharp" transitions in gray levels. Averaging filters have the undesirable side effect that they

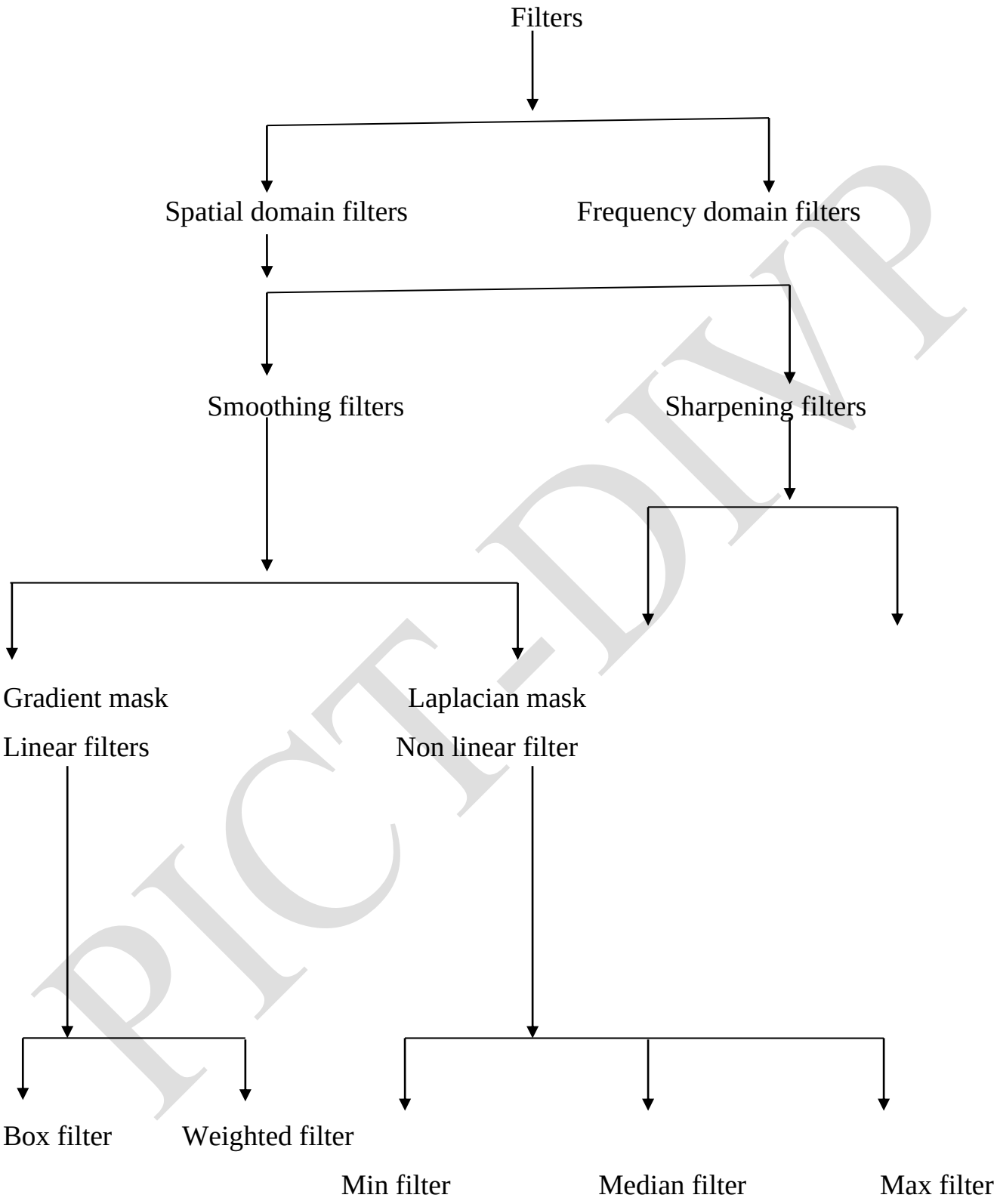
Filters
Spatial domain filters
Frequency domain filters
Smoothing filters
Sharpening filters
Gradient mask
Linear filters
Laplacian mask
Non linear filter
Box filter
Weighted filter
Min filter
Median filter
Max filter

Fig: 4.2 Filter Classifications

blur edges. A spatial averaging filter in which all coefficients are equal is sometimes called a *box filter*.

The second mask is *weighted average mask*, pixels are multiplied by different coefficients, thus giving more importance (weight) to some pixels at the expense of others. The basic strategy behind weighing the center point the highest and then reducing the value of the coefficients as a function of increasing distance from the origin is simply an attempt to reduce blurring in the smoothing process.

Box filter

$$\frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Weighted average filter

$$\frac{1}{16} \times \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Fig: 4.3Smoothing Filter Mask

4.2.3 Smoothing non-linear filters (Order-Statistics Filters)

Order-statistics filters are nonlinear spatial filters whose response is based on ordering (ranking) the pixels contained in the image area encompassed by the filter, and then replacing the value of the center pixel with the value determined by the ranking result. There are median filters, Max and Min filters.

Median filter

1	1	1
1	1	1
1	1	1

4.2.4 Sharpening Filters

The principal objective of sharpening is to highlight fine detail in an image or to enhance detail that has been blurred. Since averaging is analogous to integration, it is logical to conclude that sharpening could be accomplished by spatial differentiation. Image differentiation enhances edges and other discontinuities (such as noise) and deemphasizes areas with slowly varying gray-level values.

Comparing the response between first- and second-order derivatives:

First-order derivatives	Second-order derivatives
(1) First-order derivatives generally produce thicker edges in an image.	(1)Second-order derivatives have a stronger response to fine detail, such as thin lines and isolated points.
(2) First order derivatives generally have a stronger response to a gray-level step.	(2) Second-order derivatives produce a double response at step changes in gray level.

A basic definition of the first-order derivative of a one-dimensional function f(x) is the difference

∂f/∂x = f(x + 1) - f(x).

Similarly, we define a second-order derivative as the difference

∂^2f/ ∂x^2 = f(x + 1) + f(x - 1) - 2f(x)

4.3 Use of Second Derivatives for Enhancement–The Laplacian

The approach basically consists of defining a discrete formulation of the second-order derivative and then constructing a filter mask based on that formulation.We are interested in *isotropic* filters, isotropic filters are *rotation invariant* whose response is independent of the direction of the discontinuities in the image to which the filter is applied.

simplest isotropic derivativeoperator is the *Laplacian*, which, for a function (image) f(x, y) of two variables, is defined as

∂ ^2f = ∂^2f/ ∂x^2 + ∂^2f/ ∂y^2

Because derivatives of any order are linear operations, the Laplacian is a linear operator.

Because the Laplacian is a derivative operator, its use highlights gray-level discontinuities in an image and deemphasizes regions with slowly varying gray levels. This will tend to produce images that have grayish edge lines and other discontinuities, all superimposed on a dark, featureless background. Background features can be “recovered” while still preserving the sharpening effect of the Laplacian operation simply by adding the original and Laplacian images.

Mask :

-1	-1	-1
-1	8	-1
-1	-1	-1

$$g(x,y) = f(x,y) + \nabla^2 f(x,y) \quad (1)$$

$$g(x,y) = f(x,y) - \nabla^2 f(x,y) \quad (2)$$

Equation 1 is used when center coefficient is positive and Equation 2 is used when center coefficient is negative.

4.4 High boost filter

A process Generalization of unsharp masking is called *high boost filtering*.A high boost filtered,*f*, *is defined at any point (x,y)as*

$$f = Af(x,y) - \bar{f}(x,y)$$

The following masks are used

0	**-1**	**0**
-1	**A+4**	**-1**
0	**-1**	**0**

-1	**-1**	**-1**
-1	**A+8**	**-1**
-1	**-1**	**-1**

Note that,when A=1,high boost filtering becomes standard Laplacian sharpening.

4.5 Algorithm :

1. Start
2. Take the input image from the user
3. Display the various filters
4. Ask the user for the filter to be used
5. Display the histogram of the input image

6. Get the dimensions of the image
7. Pass the image and its dimensions to the respective functions
8. The function returns the new image
9. Display the histogram of the new image
10. Stop

4.6 Conclusion:

In this experiment I learnt about the different Filters in the spatial domain and their applications on image. I implemented filters were namely Box, Weighted, Median, High Boost and Laplacian on the image.

4.7 References:

i. GonzalezR, Woods R, "Digital image processing", Pearson Prentice Hall, 2008.
ii. GonzalezR, 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**	**:**	**11/09/2020**
EXPT. NO.	**:**	**4**	**CLASS & ROLL NO**	**:**	**BE VIII 42410**
TITLE	**:**	**TO PERFORM IMAGE ENHANCEMENT IN SPATIAL**			

I. CODE:

Min, Max, Median Filter Code:

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

def padding(originalImg, padSize,rows,columns):
    padImg = np.zeros((rows+2*padSize, columns+2*padSize), dtype=np.uint8)
    # Using Splicing
    padImg[padSize:rows+padSize, padSize:columns+padSize] = originalImg
    return padImg

def MinFilter(padImg, size, rows, columns):
    output = np.zeros((rows, columns), dtype=np.uint8)
    for i in range(0, rows):
        for j in range(0, columns):
            # using Splicing
            portion = padImg[i:i+size, j:j+size]

            # Converting Matrix to Array
            array1 = portion.flatten()
            Minv = np.min(array1)
            output[i][j] = Minv
    return output

def MaxFilter(padImg, size, rows, columns):
    output = np.zeros((rows, columns), dtype=np.uint8)
    for i in range(0, rows):
        for j in range(0, columns):
            # using Splicing
            portion = padImg[i:i+size, j:j+size]
            # Converting Matrix to Array
            array1 = portion.flatten()
            Minv = np.max(array1)
            output[i][j] = Minv
    return output
```

```
def MedianFilter(padImg, size,rows, columns):
    output = np.zeros((rows, columns), dtype=np.uint8)
    for i in range(0, rows):
        for j in range(0, columns):
            # using Splicing
            portion = padImg[i:i+size, j:j+size]
            # Converting Matrix to Array
            array1 = portion.flatten()
            medianv = np.lib.median(array1)
            output[i][j] = medianv
    return output

def inbuiltMedianFilter(img,n):
    blurImg = cv2.medianBlur(img,n)
    return blurImg

def inbuiltMinFilter(img,n):
    size = (n, n)
    shape = cv2.MORPH_RECT
    kernel = cv2.getStructuringElement(shape, size)
    minImg = cv2.erode(img, kernel)
    return minImg

def inbuiltMaxFilter(img,n):
    size = (n,n)
    shape = cv2.MORPH_RECT
    kernel = cv2.getStructuringElement(shape, size)
    minImg = cv2.dilate(img, kernel)
    return minImg


def hist_plot(histimg1,histimg2,histimg3):
  #plot histogram
  plt.subplot(131),plt.hist(histimg1.ravel(),255,[0,255]),plt.title('Original')
  plt.xticks([]), plt.yticks([])
  plt.subplot(132),plt.hist(histimg2.ravel(),255,[0,255]),plt.title('Manual Filter')
  plt.xticks([]), plt.yticks([])
  plt.subplot(133),plt.hist(histimg3.ravel(),255,[0,255]),plt.title('Inbuilt Filter')
  plt.xticks([]), plt.yticks([])
  plt.show()

def main():
    path="C:/Users/Danish/Desktop/DIVP/Images/saltimage.jpg"
    img = cv2.imread(path,0)

    size=3 #mask size
```

```
    pad_size= size//2
    rows = img.shape[0]
    columns = img.shape[1]
    padImg = padding(img, pad_size,rows,columns)

    op=input("1. Median Filter\n2. Min Filter\n3. Max Filter\n")
    if op == 1:
        img2=MedianFilter(padImg, size, rows, columns)
        img3=inbuiltMedianFilter(img,size)
    elif op == 1:
        img2=MinFilter(img, size, rows, columns)
        img3=inbuiltMinFilter(img,size)
    else:
        img2=MaxFilter(padImg, size, rows, columns)
        img3=inbuiltMaxFilter(img,size)

    hist_plot(img,img2, img3)
    out=cv2.hconcat([img,img2,img3])
    cv2.imshow('output',out)
    cv2.waitKey(300000)
    cv2.destroyAllWindows()

main()
```

Box, Weighted Average, Laplacian and High Boost Filter Code:

```
# -*- coding: utf-8 -*-
import cv2
import numpy as np
from matplotlib import pyplot as plt


def manual_laplacian_filter(img):
    mask = (np.array([-1, -1, -1,-1 ,8 ,-1,-1 ,-1 ,-1])).reshape(3,3)
    man_img = cv2.filter2D(img,-1,mask)
    return(man_img)

def manual_highboost_filter(img,k):
    mask = (np.array([-1, -1, -1,-1 ,k+8 ,-1,-1 ,-1 ,-1])).reshape(3,3)
    man_img = cv2.filter2D(img,-1,mask)
    return(man_img)

def manual_avg_filter(img):
    mask = np.ones((5,5),np.float32)/25 #9
    man_img = cv2.filter2D(img,-1,mask)
    return(man_img)
```

```
def manual_weighted_avg_filter(img):
    mask = (np.array([1, 2, 1,2 ,4 ,2,1 ,2 ,1])).reshape(3,3)
    mask=mask/16
    man_img = cv2.filter2D(img,-1,mask)
    return(man_img)


def inbuilt_avg_filter(img):
    blur_img = cv2.blur(img,(5,5))
    return(blur_img)

def hist_plot(histimg1,histimg2,histimg3):
   #plot histogram
   plt.subplot(131),plt.hist(histimg1.ravel(),255,[0,255]),plt.title('Original')
   plt.xticks([]), plt.yticks([])
   plt.subplot(132),plt.hist(histimg2.ravel(),255,[0,255]),plt.title('Manual Averaging')
   plt.xticks([]), plt.yticks([])
   plt.subplot(133),plt.hist(histimg3.ravel(),255,[0,255]),plt.title('Inbuilt Averaging')
   plt.xticks([]), plt.yticks([])
   plt.show()

def main():
    path="C:/Users/Danish/Desktop/DIVP/Images/lenna.jpg"
    img = cv2.imread(path,0)
    img2=manual_avg_filter(img)
   # img3=inbuilt_avg_filter(img)
    img3=manual_weighted_avg_filter(img)
    img4=manual_laplacian_filter(img)
    img5=manual_highboost_filter(img, 3) #A=3
    hist_plot(img, img2, img3)
    out=cv2.hconcat([img,img2,img3,img4,img5])
    cv2.imshow('output',out)
    cv2.waitKey(30000)
    cv2.destroyAllWindows()

main()
```

II. RESULTS:

Input Image

Input Image | Box Filter | Weighted Average | Laplacian Filter | High Boost Filter