

|  |
| --- |
| **Title:** Implementation of edge detection operators   * Robert * Prewitt * Sobel * Laplacian |

**Objective:** To learn and understand different edge detection operators.

**Expected Outcome of Experiment:**

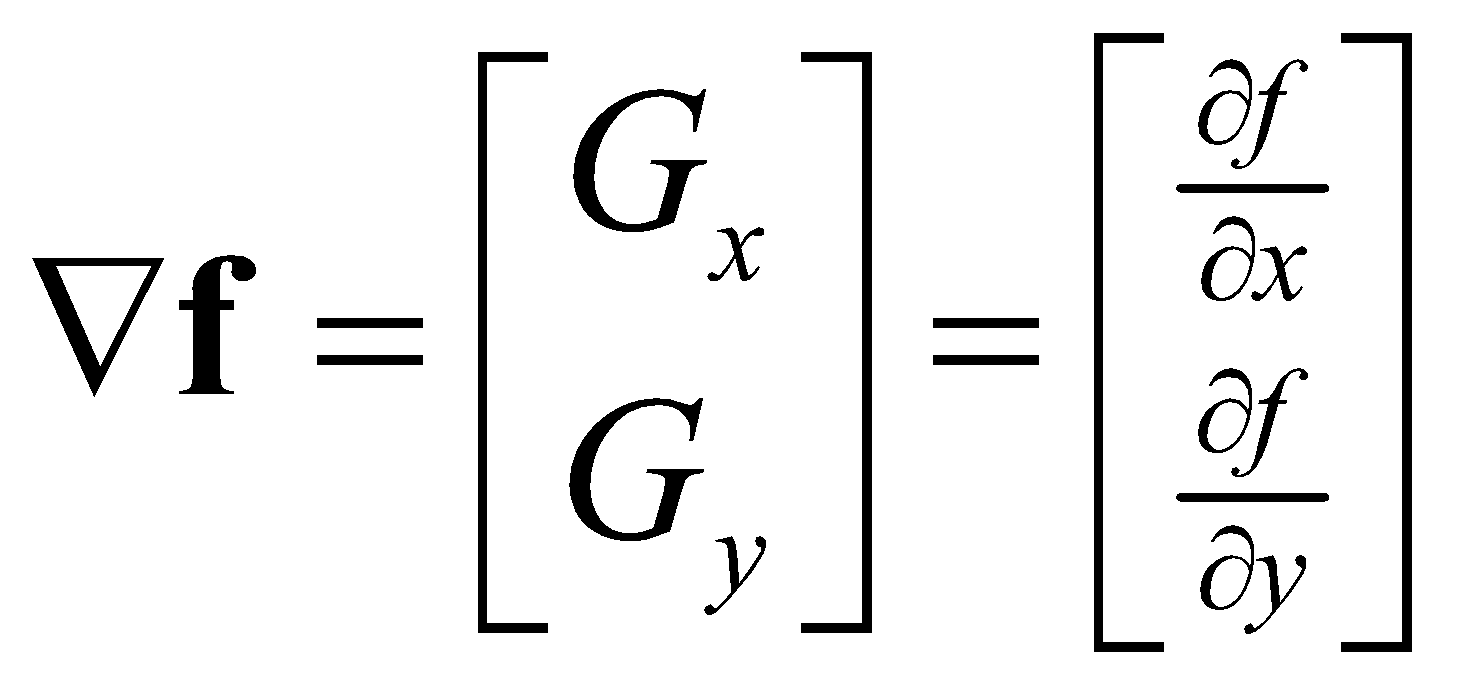
|  |  |
| --- | --- |
| **CO** | **Outcome** |
| **CO4** | Design & implement algorithms for digital image enhancement, segmentation & restoration. |

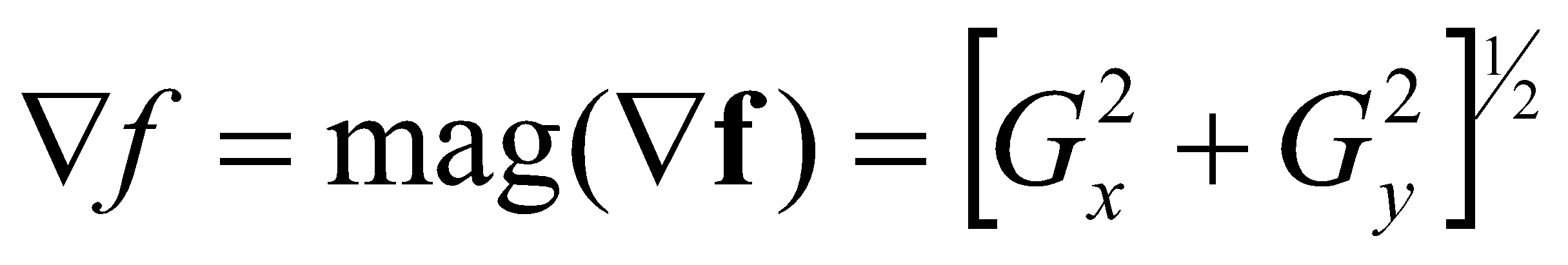
**Books/ Journals/ Websites referred:**

1. http://www.mathworks.com/support/
2. www.math.mtu.edu/~msgocken/intro/intro.html.
3. R. C.Gonsales R.E.Woods, “Digital Image Processing”, Second edition, Pearson Education
4. S.Jayaraman, S Esakkirajan, T Veerakumar “Digital Image Processing “Mc Graw Hill.
5. S.Sridhar,”Digital Image processing”, oxford university press, 1st edition."

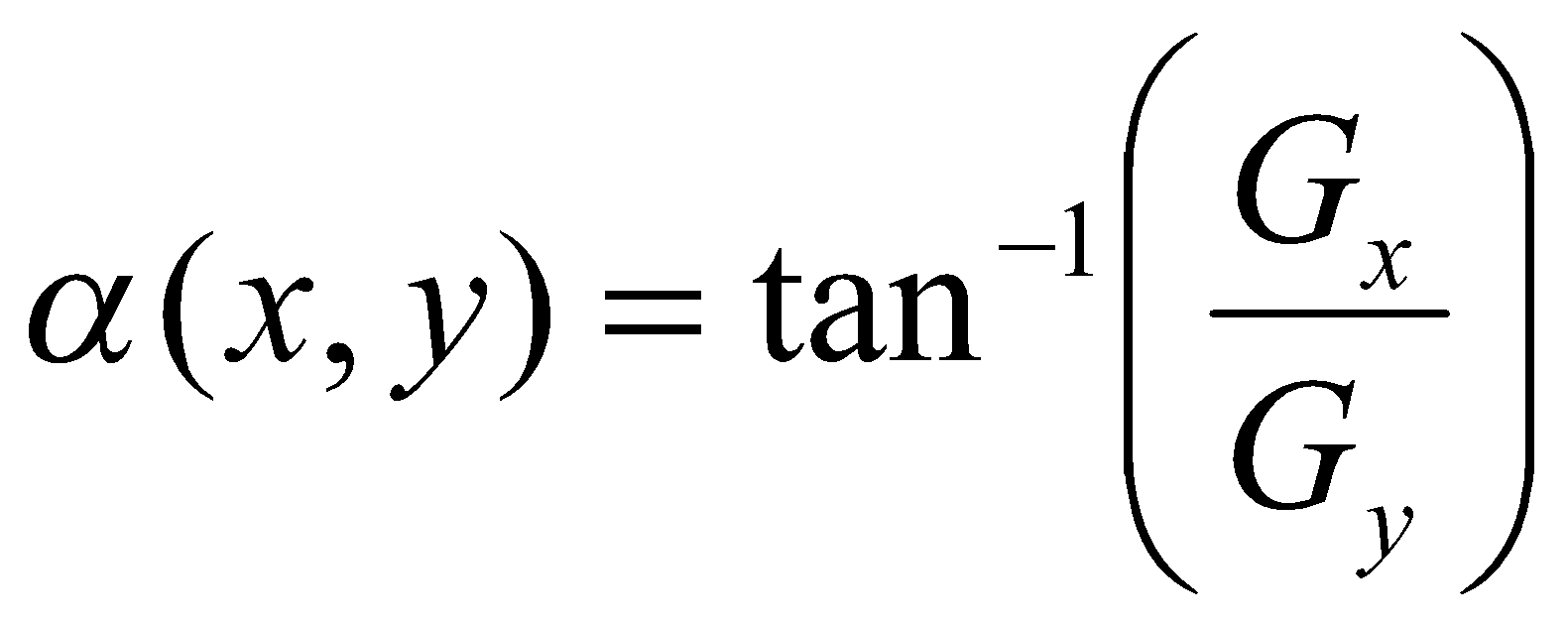
**Pre Lab/ Prior Concepts:**

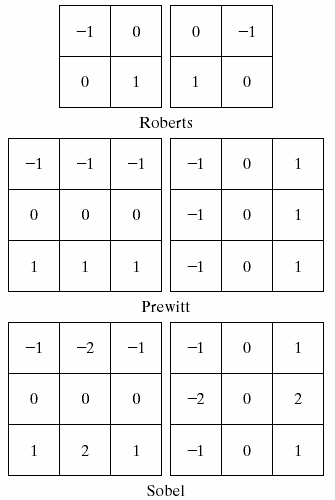
**Detection of Discontinuities using Gradient Operators**

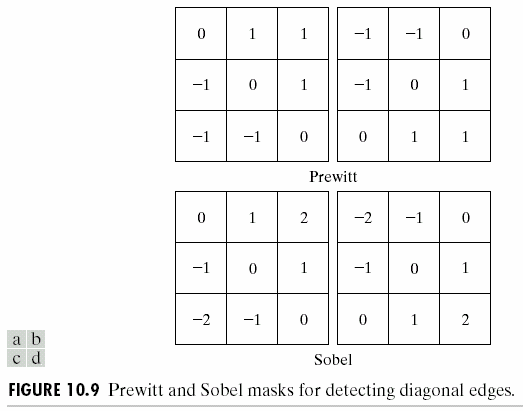
* **First-order derivatives:**
  + The gradient of an image *f*(*x*,*y*) at location (*x*,*y*) is defined as the vector:
  + The magnitude of this vector:

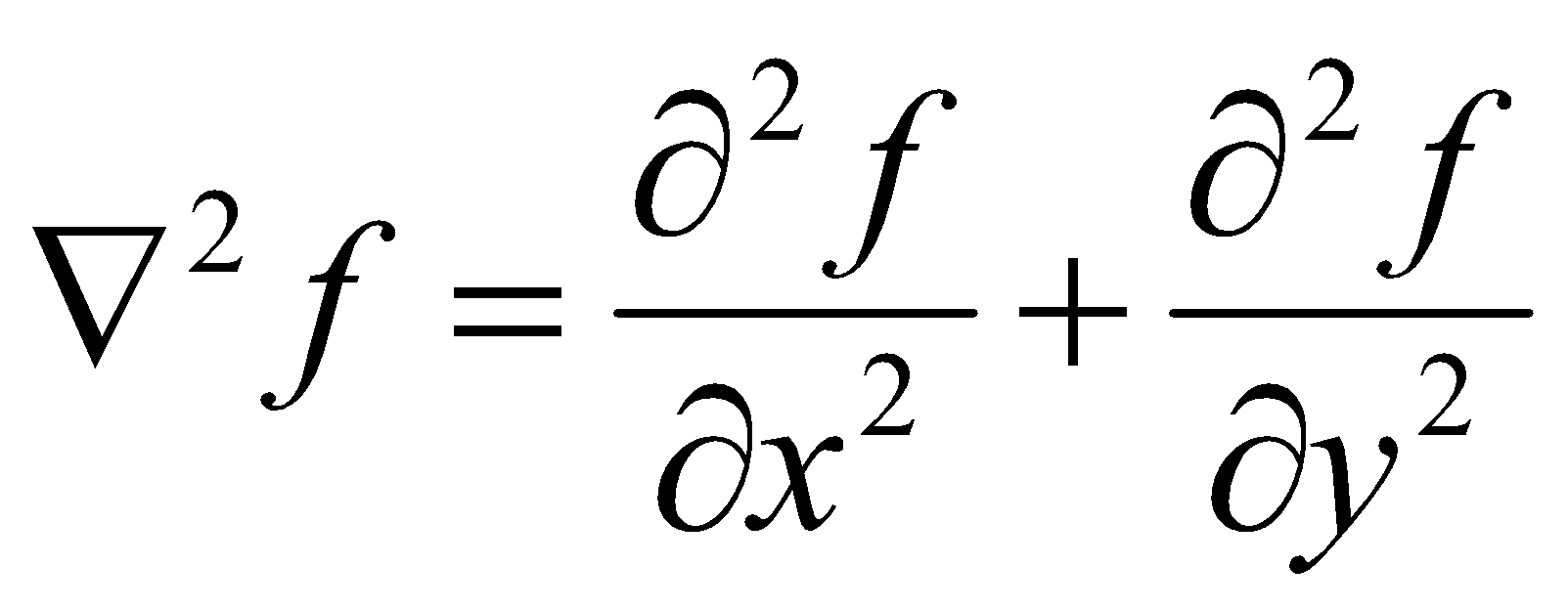


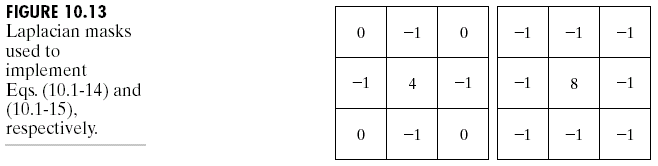
* + The direction of this vector:







* **Second-order derivatives: (The Laplacian)**
  + The Laplacian of an 2D function *f*(*x*,*y*) is defined as
  + Two forms in practice

****

The Laplacian of a Gaussian sometimes is called the Mexican hat function. It also can be computed by smoothing the image with the Gaussian smoothing mask, followed by application of the Laplacian mask.

**Algorithm:**

1. Select the masks needed and store them as matrices
2. Start from top left pixel and traverse horizontally row-wise till bottom right pixel
3. If pixel x is border pixel retain it as it is into the new matrix else form a matrix from the input image using x as centre. Now multiply mask and this matrix element wise.
4. Calculate the sum of all elements of this matrix
5. Assign this sum to centre pixel in the new matrix
6. Repeat this process till the mask is applied over the entire image.

**Implementation Details:**

1. **Robert**

x = imread('bfly.jpg');

subplot(2,3,1);

imshow(x,[]);

title('Original')

x = double(rgb2gray(x));

x = padarray(x, [1,1], 0, 'post');

subplot(2,3,2);

imshow(x,[]);

title('Grayscale (after padding)')

edgedetection = x;

[m,n] = size(x);

a = [1,0; 0,-1];

for i = 1:m-1

for j = 1:n-1

b = [x(i,j)\*a(1,1) x(i+1,j)\*a(2,1) x(i,j+1)\*a(1,2) x(i+1,j+1)\*a(2,2)];

edgedetection(i,j) = sum(b);

end

end

subplot(2,3,4);

edgedetection = uint8(edgedetection);

imshow(edgedetection);

title('Robert(x)');

a = [0,1; -1,0];

for i = 1:m-1

for j = 1:n-1

b = [x(i,j)\*a(1,1) x(i+1,j)\*a(2,1) x(i,j+1)\*a(1,2) x(i+1,j+1)\*a(2,2)];

edgedetection(i,j) = sum(b);

end

end

subplot(2,3,5);

edgedetection = uint8(edgedetection);

imshow(edgedetection);

title('Robert(y)');

a = [1,1; -1,-1];

for i = 1:m-1

for j = 1:n-1

b = [x(i,j)\*a(1,1) x(i+1,j)\*a(2,1) x(i,j+1)\*a(1,2) x(i+1,j+1)\*a(2,2)];

edgedetection(i,j) = sum(b);

end

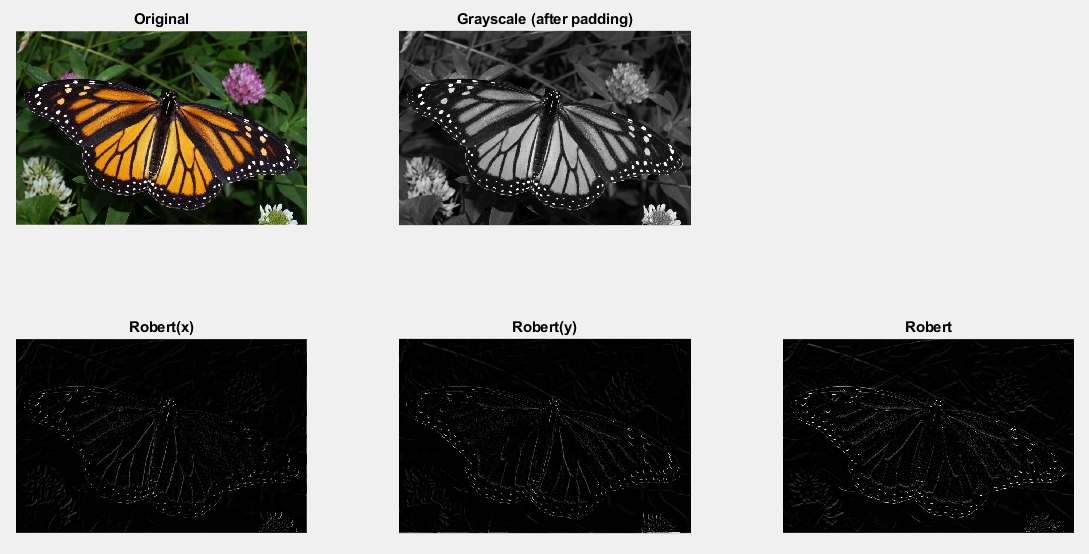
end

subplot(2,3,6);

edgedetection = uint8(edgedetection);

imshow(edgedetection);

title('Robert');



1. **Prewitt**

x = imread('bfly.jpg');

subplot(2,3,1);

imshow(x,[]);

title('Original')

x = double(rgb2gray(x));

x = padarray(x, [1,1], 0, 'both');

subplot(2,3,2);

imshow(x,[]);

title('Grayscale (after padding)')

edgedetection = x;

[m,n] = size(x);

a = [-1,-1,-1; 0,0,0; 1,1,1];

for i = 2:m-1

for j = 2:n-1

b = [x(i-1,j-1)\*a(1,1) x(i,j-1)\*a(2,1) x(i+1,j-1)\*a(3,1) x(i-1,j)\*a(1,2) x(i,j)\*a(2,2) x(i+1,j)\*a(3,2) x(i-1,j+1)\*a(1,3) x(i,j+1)\*a(2,3) x(i+1,j+1)\*a(3,3)];

edgedetection(i,j) = sum(b);

end

end

subplot(2,3,4);

edgedetection = uint8(edgedetection);

imshow(edgedetection);

title('Prewitt(x)');

a = [-1,0,1;-1,0,1;-1,0,1];

for i = 2:m-1

for j = 2:n-1

b = [x(i-1,j-1)\*a(1,1) x(i,j-1)\*a(2,1) x(i+1,j-1)\*a(3,1) x(i-1,j)\*a(1,2) x(i,j)\*a(2,2) x(i+1,j)\*a(3,2) x(i-1,j+1)\*a(1,3) x(i,j+1)\*a(2,3) x(i+1,j+1)\*a(3,3)];

edgedetection(i,j) = sum(b);

end

end

subplot(2,3,5);

edgedetection = uint8(edgedetection);

imshow(edgedetection);

title('Prewitt(y)');

a = [-2, -1, 0; -1, 0, 1; 0, 1, 2];

for i = 2:m-1

for j = 2:n-1

b = [x(i-1,j-1)\*a(1,1) x(i,j-1)\*a(2,1) x(i+1,j-1)\*a(3,1) x(i-1,j)\*a(1,2) x(i,j)\*a(2,2) x(i+1,j)\*a(3,2) x(i-1,j+1)\*a(1,3) x(i,j+1)\*a(2,3) x(i+1,j+1)\*a(3,3)];

edgedetection(i,j) = sum(b);

end

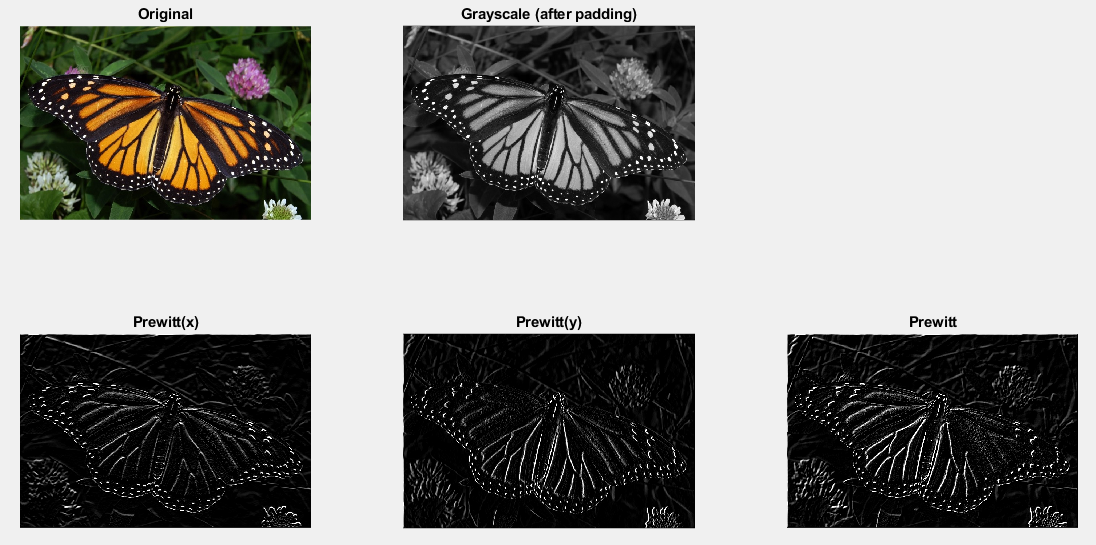
end

subplot(2,3,6);

edgedetection = uint8(edgedetection);

imshow(edgedetection);

title('Prewitt');



1. **Sobel**

x = imread('bfly.jpg');

subplot(2,3,1);

imshow(x,[]);

title('Original')

x = double(rgb2gray(x));

x = padarray(x, [1,1], 0, 'both');

subplot(2,3,2);

imshow(x,[]);

title('Grayscale (after padding)')

edgedetection = x;

[m,n] = size(x);

a = [-1,-2,-1; 0,0,0; 1,2,1];

for i = 2:m-1

for j = 2:n-1

b = [x(i-1,j-1)\*a(1,1) x(i,j-1)\*a(2,1) x(i+1,j-1)\*a(3,1) x(i-1,j)\*a(1,2) x(i,j)\*a(2,2) x(i+1,j)\*a(3,2) x(i-1,j+1)\*a(1,3) x(i,j+1)\*a(2,3) x(i+1,j+1)\*a(3,3)];

edgedetection(i,j) = sum(b);

end

end

subplot(2,3,4);

edgedetection = uint8(edgedetection);

imshow(edgedetection);

title('Sobel(x)');

a = [-1,0,1;-2,0,2;-1,0,1];

for i = 2:m-1

for j = 2:n-1

b = [x(i-1,j-1)\*a(1,1) x(i,j-1)\*a(2,1) x(i+1,j-1)\*a(3,1) x(i-1,j)\*a(1,2) x(i,j)\*a(2,2) x(i+1,j)\*a(3,2) x(i-1,j+1)\*a(1,3) x(i,j+1)\*a(2,3) x(i+1,j+1)\*a(3,3)];

edgedetection(i,j) = sum(b);

end

end

subplot(2,3,5);

edgedetection = uint8(edgedetection);

imshow(edgedetection);

title('Sobel(y)');

a = [-2, -2, 0; -2, 0, 2; 0, 2, 2];

for i = 2:m-1

for j = 2:n-1

b = [x(i-1,j-1)\*a(1,1) x(i,j-1)\*a(2,1) x(i+1,j-1)\*a(3,1) x(i-1,j)\*a(1,2) x(i,j)\*a(2,2) x(i+1,j)\*a(3,2) x(i-1,j+1)\*a(1,3) x(i,j+1)\*a(2,3) x(i+1,j+1)\*a(3,3)];

edgedetection(i,j) = sum(b);

end

end

subplot(2,3,6);

edgedetection = uint8(edgedetection);

imshow(edgedetection);

title('Sobel');



1. **Laplacian**

x = imread('bfly.jpg');

subplot(1,3,1);

imshow(x,[]);

title('Original')

x = double(rgb2gray(x));

x = padarray(x, [1,1], 0, 'both');

subplot(1,3,2);

imshow(x,[]);

title('Grayscale (after padding)')

edgedetection = x;

[m,n] = size(x);

a = [0,-1,0; -1,4,-1; 0,-1,0];

for i = 2:m-1

for j = 2:n-1

b = [x(i-1,j-1)\*a(1,1) x(i,j-1)\*a(2,1) x(i+1,j-1)\*a(3,1) x(i-1,j)\*a(1,2) x(i,j)\*a(2,2) x(i+1,j)\*a(3,2) x(i-1,j+1)\*a(1,3) x(i,j+1)\*a(2,3) x(i+1,j+1)\*a(3,3)];

edgedetection(i,j) = sum(b);

end

end

subplot(1,3,3);

edgedetection = uint8(edgedetection);

imshow(edgedetection);

title('Laplacian');



**Conclusion:-**

Thus, the various edge detection operators like Prewitt, Sobel, Robert, and Laplacian were understood and the same were implemented using MATLAB.

**Date: \_\_26/04/2019\_\_ Signature of faculty in-charge**

**Post Lab Descriptive Questions**

**1. Explain the need of edge linking.**

**Ans.**

Edge detectors yield pixels in an image lie on edges. The next step is to try to collect these pixels together into a set of edges. Thus, our aim is to replace many points on edges with a few edges themselves. Edge linking mainly finds its application in detecting objects in an image.

The practical problem may be much more difficult than the idealised case.

* Small pieces of edges may be missing,
* Small edge segments may appear to be present due to noise where there is no real edge, *etc*.

In general, edge linking methods can be classified into two categories:

**Local Edge Linkers**

-- where edge points are grouped to form edges by considering each point's relationship to any neighbouring edge points.

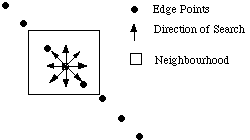
**Global Edge Linkers**

-- where all edge points in the image plane are considered at the same time and sets of edge points are sought according to some similarity constraint, such as points which share the same edge equation.

Most edge detectors yield information about the magnitude of the gradient at an edge point and, more importantly, the direction of the edge in the locality of the point.

This is obviously useful when deciding which edge points to link together since edge points in a neighbourhood which have similar gradients directions are likely to lie on the same edge.

Local edge linking methods usually start at some arbitrary edge point and consider points in a local neighbourhood for similarity of edge direction as shown in Fig. 29



If the points satisfy the similarity constraint then the points are added to the current edge set. The neighbourhoods based around the recently added edge points are then considered in turn and so on. If the points do not satisfy the constraint then we conclude we are at the end of the edge, and so the process stops. A new starting edge point is found which does not belong to any edge set found so far, and the process is repeated.

The algorithm terminates when all edge points have been linked to one edge or at least have been considered for linking once. Thus the basic process used by local edge linkers is that of tracking a sequence of edge points.

**2. Explain point detection and line detection.**

**Ans.**

Point Detection

Points and lines are high frequency components and thus, for point detection, we basically need a high pass filter. We make use of a threshold value so that our mask detects only points and not lines i.e. a point is said to be detected at the location at which the mask is centered only if the value after applying the mask at that location is greater than or equal to the threshold value.

Eg: -1 -1 -1

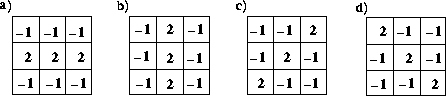
-1 8 -1

-1 -1 -1

Line Detection

While [edges](https://homepages.inf.ed.ac.uk/rbf/HIPR2/edgdetct.htm) (*i.e.* boundaries between regions with relatively distinct gray levels) are by far the most common type of discontinuity in an image, instances of thin lines in an image occur frequently enough that it is useful to have a separate mechanism for detecting them.

The line detection operator consists of a convolution kernel tuned to detect the presence of lines of a particular width *n*, at a particular orientation theta. Figure 1 shows a collection of four such kernels, which each respond to lines of single pixel width at the particular orientation shown.



Four line detection kernels which respond maximally to horizontal, vertical and oblique (+45 and - 45 degree) single pixel wide lines.

These masks above are tuned for light lines against a dark background, and would give a big negative response to dark lines against a light background. If you are only interested in detecting dark lines against a light background, then you should negate the mask values. Alternatively, you might be interested in either kind of line, in which case, you could take the absolute value of the convolution output. In the discussion and examples below, we will use the kernels above without an absolute value.

If Ri denotes the response of kernel *i*, we can apply each of these kernels across an image, and for any particular point, if Ri > Rj for all j!=i that point is more likely to contain a line whose orientation (and width) corresponds to that of kernel *i*. One usually thresholds Ri to eliminate weak lines corresponding to edges and other features with intensity gradients which have a different scale than the desired line width. In order to find complete lines, one must join together line fragments, *e.g.*, with an *edge tracking* operator.