Batch: B1 Roll No: 16010121045

Experiment No. 9

Title: Implement following Edge detection operators (Prewitt, Sobel, Robert, and Laplacian).

Objective: To learn and understand different edge detection operators.

Expected Outcome of Experiment:

CO	Outcome
CO4	Evaluate extracted analyzed information for synthesis of digital signals.

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:

Image segmentation can be achieved in two ways,

1. Segmentation based on discontinuities of intensity.
2. Segmentation based on similarities in intensity.

Edge information in an image is found by looking at the relationship a pixel has with its

neighbourhoods. If a pixel's gray-level value is similar to those around it, there is probably not an edge at that point. If a pixel's has neighbors with widely varying gray levels, it may present an edge point.

Edge Detection Methods:

Many are implemented with convolution mask and based on discrete approximations to differential operators. Differential operations measure the rate of change in the image brightness function.Some operators return orientation information. Other only return information about the existence of an edge at each point.

Sobel Operator

The operator consists of a pair of 3×3 convolution kernels as shown in Figure 1. One kernel is simply the other rotated by 90°.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

These kernels are designed to respond maximally to edges running vertically and horizontally relative to the pixel grid, one kernel for each of the two perpendicular orientations. The kernels can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation (call these *Gx* and *Gy*). These can then be combined together to find the absolute magnitude of the gradient at each point and the orientation of that gradient. The gradient magnitude is given by:

$$|G| = \sqrt{Gx^2 + Gy^2}$$

Typically, an approximate magnitude is computed using:

$$|G| = |Gx| + |Gy|$$

which is much faster to compute. The angle of orientation of the edge (relative to the pixel grid) giving rise to the spatial gradient is given by:

$$\theta = \arctan(Gy/Gx)$$

Robert's cross operator:

The Roberts Cross operator performs a simple, quick to compute, 2-D spatial gradient measurement on an image. Pixel values at each point in the output represent the estimated absolute magnitude of the spatial gradient of the input image at that point. The operator consists of a pair of 2×2 convolution kernels as shown in Figure. One kernel is simply the other rotated by 90°. This is very similar to the Sobel operator.

+1	0
0	-1

Gx

0	+1
-1	0

Gy

These kernels are designed to respond maximally to edges running at 45° to the pixel grid, one kernel for each of the two perpendicular orientations. The kernels can be applied separately to the input image, to produce separate measurements of the gradient component in each orientation (call these *Gx* and *Gy*). These can then be combined together to find the absolute magnitude of the gradient at each point and the orientation of that gradient. The gradient magnitude is given by:

$$|G| = \sqrt{Gx^2 + Gy^2}$$

An approximate magnitude is computed using:

$$|G| = |Gx| + |Gy|$$

Which is much faster to compute? The angle of orientation of the edge giving rise to the spatial gradient (relative to the pixel grid orientation) is given by:

$$\theta = \arctan(Gy/Gx) - 3\pi/4$$

Prewitt's operator:

Prewitt operator is similar to the Sobel operator and is used for detecting vertical and horizontal edges in images.

$$h_1 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \quad h_3 = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Laplacian of Gaussian:

The Laplacian is a 2-D isotropic measure of the 2nd spatial derivative of an image. The Laplacian of an image highlights regions of rapid intensity change and is therefore often used for edge detection. The Laplacian is often applied to an image that has first been smoothed with something approximating a Gaussian Smoothing filter in order to reduce its sensitivity to noise. The operator normally takes a single gray level image as input and produces another gray level image as output.

The Laplacian *L(x,y)* of an image with pixel intensity values *I(x,y)* is given by:

$$L(x,y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Since the input image is represented as a set of discrete pixels, we have to find a discrete convolution kernel that can approximate the second derivatives in the definition of the Laplacian. Three commonly used small kernels are shown in Fig below

0	1	0
1	−4	1
0	1	0

1	1	1
1	−8	1
1	1	1

−1	2	−1
2	−4	2
−1	2	−1

Because these kernels are approximating a second derivative measurement on the image, they are very sensitive to noise. To counter this, the image is often Gaussian Smoothed before applying the Laplacian filter. This pre-processing step reduces the high frequency noise components prior to the differentiation step.

Implementation Details:

Write Algorithm and Matlab commands used:

Robert:

```
figure(1);
img = imread("example.jpg");
[width, height, ~] = size(img);
subplot(1,3,1);
imshow(img);
title("Original Image");

gray_img = rgb2gray(img); % Convert the image to grayscale

A = zeros(width+2, height+2);
for i = 1:1:width
   for j = 1:1:height
      A(i+1, j+1) = gray_img(i, j);
   end
end

w1 = [1 0 0 -1];
w2 = [0 1 -1 0];
for i = 2:1:width+1
   for j = 2:1:height+1
      gx = w1(1)*A(i, j) + w1(2)*A(i, j+1) + w1(3)*A(i+1, j) + w1(4)*A(i+1, j+1);
      gy = w2(1)*A(i, j) + w2(2)*A(i, j+1) + w2(3)*A(i+1, j) + w2(4)*A(i+1, j+1);
      C(i, j) = sqrt(gx.^2 + gy.^2);
   end
end

for i = 1:1:width
   for j = 1:1:height
      B(i, j) = C(i+1, j+1);
   end
end

subplot(1,3,2);
imshow(uint8(gray_img));
title("Grayscale Image");
```

```
subplot(1,3,3);
imshow(uint8(B));
title("Robert Image");
```

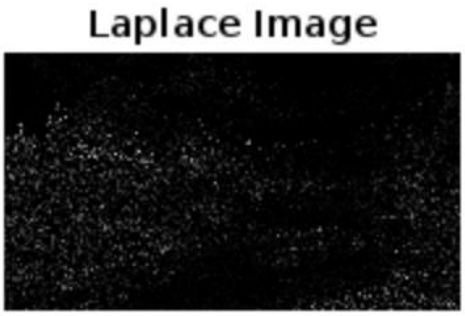
Prewitt:

```
figure(2);
img = imread("example.jpg");
[width, height, dim] = size(img);

subplot(1,3,1);
imshow(img);
title("Original Image");

gray_img = rgb2gray(img); % Convert the image to grayscale

A = zeros(width+2, height+2);
for i = 1:1:width
   for j = 1:1:height
      A(i+1, j+1) = gray_img(i, j);
   end
end

w1 = [1 1 1 0 0 0 -1 -1 -1];
w2 = [-1 0 1 -1 0 1 -1 0 1];
for i = 2:1:width+1
   for j = 2:1:height+1
      gx = w1(1)*A(i-1,j-1) + w1(2)*A(i-1,j) + w1(3)*A(i-1,j+1) + w1(4)*A(i,j-1) + w1(5)*A(i,j) + w1(6)*A(i,j+1) + w1(7)*A(i+1,j-1) +
w1(8)*A(i+1,j) + w1(9)*A(i+1,j+1);
      gy = w2(1)*A(i-1,j-1) + w2(2)*A(i-1,j) + w2(3)*A(i-1,j+1) + w2(4)*A(i,j-1) + w2(5)*A(i,j) + w2(6)*A(i,j+1) + w2(7)*A(i+1,j-1) +
w2(8)*A(i+1,j) + w2(9)*A(i+1,j+1);
      C(i,j) = sqrt(gx.^2 + gy.^2);
   end
end

for i = 1:1:width
   for j = 1:1:height
```

```
      B(i,j) = C(i+1,j+1);
    end
end

subplot(1,3,2);
imshow(gray_img);
title("Grayscale Image");

subplot(1,3,3);
imshow(uint8(B));
title("Prewitt Image");
```

Laplace:

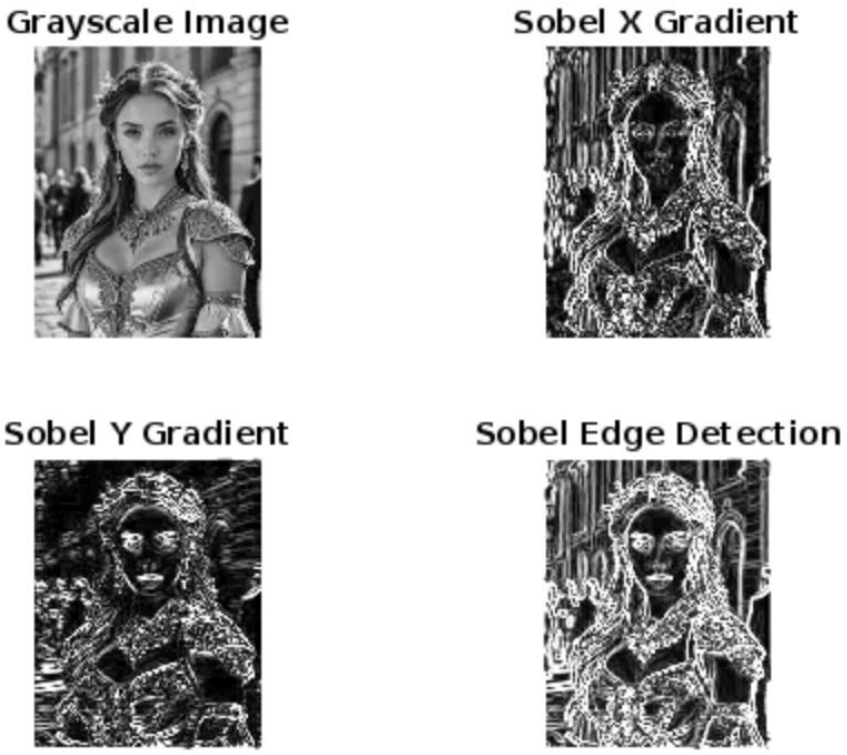
```
figure(4);
img = imread("example.jpg");
[width, height, dim] = size(img);

if dim == 3
   gray_img = rgb2gray(img); % Convert the image to grayscale if it's not already grayscale
else
   gray_img = img;
end

A = zeros(width+2, height+2);
for i = 1:1:width
   for j = 1:1:height
      A(i+1, j+1) = gray_img(i, j);
   end
end

w1 = [0 1 0 1 -4 1 0 1 0];
for i = 2:1:width+1
   for j = 2:1:height+1
      gx = w1(1)*A(i-1,j-1) + w1(2)*A(i-1,j) + w1(3)*A(i-1,j+1) + w1(4)*A(i,j-1) + w1(5)*A(i,j) + w1(6)*A(i,j+1) + w1(7)*A(i+1,j-1) +
w1(8)*A(i+1,j) + w1(9)*A(i+1,j+1);
      C(i,j) = gx;
   end
end

for i = 1:1:width
   for j = 1:1:height
      B(i,j) = C(i+1,j+1);
   end
end

subplot(1,2,1);
imshow(gray_img);
title("Grayscale Image");

subplot(1,2,2);
imshow(uint8(B));
title("Laplace Image");
```



Sobel:

```
% Read the input image
inputImage = imread('images.jpg');

% Convert the image to grayscale
grayImage = rgb2gray(inputImage);

% Compute Sobel x gradient
sobelX = [-1 0 1; -2 0 2; -1 0 1]; % Sobel filter for x-direction
sobelXGradient = conv2(double(grayImage), sobelX, 'same');

% Compute Sobel y gradient
sobelY = [-1 -2 -1; 0 0 0; 1 2 1]; % Sobel filter for y-direction
sobelYGradient = conv2(double(grayImage), sobelY, 'same');

% Compute edge magnitude using Sobel x and y gradients
edgeMagnitude = sqrt(sobelXGradient.^2 + sobelYGradient.^2);

% Display results
figure;
subplot(2, 2, 1);
imshow(grayImage);
title('Grayscale Image');
```

```
subplot(2, 2, 2);
imshow(uint8(abs(sobelXGradient)), []);
title('Sobel X Gradient');

subplot(2, 2, 3);
imshow(uint8(abs(sobelYGradient)), []);
title('Sobel Y Gradient');

subplot(2, 2, 4);
imshow(uint8(edgeMagnitude), []);
title('Sobel Edge Detection');
```



Conclusion:- Completed the given lab assignment and perform the various edge detection operators.

Date: _____

Post Lab Descriptive Questions

1. Explain the need of LOG operator.

The LOG (Laplacian of Gaussian) operator is needed primarily for edge detection in

images. Unlike other operators such as the gradient or Sobel operators, the LOG operator enhances edges regardless of their orientation. It achieves this by first smoothing the image with a Gaussian filter to reduce noise, then computing the Laplacian operator to detect regions of rapid intensity change, which typically correspond to edges. This combined approach makes the LOG operator effective in detecting edges accurately, especially in images with varying levels of noise.

2. Explain the technique of thresholding for segmentation.

Thresholding is a technique used in image processing for segmentation, which involves dividing an image into meaningful regions or objects. In thresholding, a threshold value is selected, and pixels in the image are categorized as either foreground or background based on whether their intensity values are above or below this threshold. This process essentially creates a binary image where pixels with intensity values above the threshold are set to one color (often white) representing the foreground, while pixels below the threshold are set to another color (often black) representing the background. Thresholding can be performed using various methods such as global thresholding, where a single threshold is applied to the entire image, or adaptive thresholding, where the threshold value is calculated locally for different regions of the image. Overall, thresholding is a fundamental technique for segmenting objects in images by simplifying their representation into binary form based on intensity levels.