

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

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

**Expected Outcome of Experiment :**

| **Course Outcome** | **Description** |
| --- | --- |
| **CO-4** | 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 :**

Image segmentation can be achieved in two ways,

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

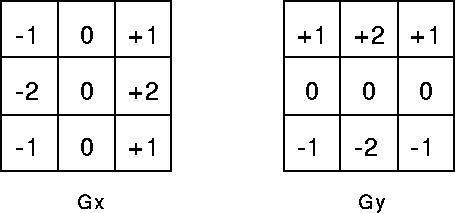
Edge information in an image is found by looking at the relationship a pixel has with its neighbourhoods. If a pixel’s gray-level value is similar to those around it, there is probably not an edge at that point. If a pixel’s has neighbors with widely varying gray levels, it may present an edge point.

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.

1. 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°.



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:



Typically, an approximate magnitude is computed using:

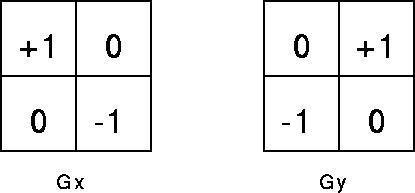


which is much faster to compute. The angle of orientation of the edge (relative to the pixel grid) giving rise to the spatial gradient is given by:



1. **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.



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:



An approximate magnitude is computed using:

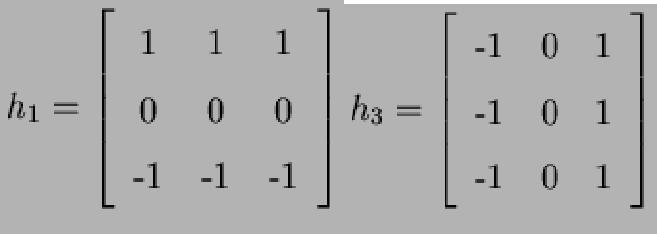


Which is much faster to compute? The angle of orientation of the edge giving rise to the spatial gradient (relative to the pixel grid orientation) is given by:



1. **Prewitt’s operator**

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



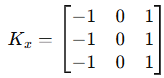
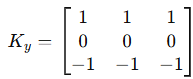
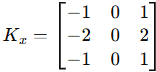
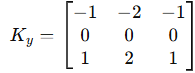
**Write Algorithm and MATLAB commands used :**

### 1. Edge Detection Algorithm (Roberts, Prewitt, and Sobel Operators)

This algorithm detects edges in an image using three popular edge detection filters: **Roberts**, **Prewitt**, and **Sobel**. These filters are applied to both the X and Y directions to detect horizontal and vertical edges.

#### Steps:

1. **Image Preprocessing**:  
   * **Input**: Load the input image using imread() and convert it to grayscale using rgb2gray().
   * **Padding**: The image is padded with zeros around its borders to handle edge cases when applying filters.
2. **Define Filters**:  
   * Define 3 filters for each edge detection technique:  
     + **Roberts**:  
       - 
       - 

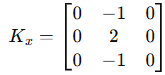
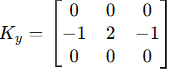
* + - **Prewitt**:  
      * 
      * 
    - **Sobel**:  
      * 
      * 

1. **Edge Detection (Filter Application)**:  
   * Loop through the entire image and apply the edge detection filters:  
     + **Roberts**: Convolve the image with both the **Roberts X** and **Roberts Y** filters and calculate the magnitude of the gradient:  
        Magnitude=(Gx2+Gy2)\text{Magnitude} = \sqrt{(G\_x^2 + G\_y^2)}
     + **Prewitt**: Similarly, convolve the image with the **Prewitt X** and **Prewitt Y** filters and compute the gradient magnitude.
     + **Sobel**: Apply the **Sobel X** and **Sobel Y** filters, and calculate the magnitude of the gradient.
2. **Display Results**:  
   * Display the results of the filters in a subplot. The outputs will include:  
     + **X and Y gradients** for each filter.
     + **Magnitude** of gradients (final edge detection result).

### 2. Laplacian Edge Detection Algorithm

This algorithm detects edges using the **Laplacian** operator, which is a second-order derivative operator. It highlights regions of rapid intensity change, often corresponding to edges in an image.

#### Steps:

1. **Image Preprocessing**:  
   * **Input**: Load the input image and convert it to grayscale.
   * **Padding**: Similar to the edge detection, the image is padded with zeros to handle edge pixels while applying the Laplacian filters.
2. **Define Laplacian Filters**:  
   * Define two Laplacian filters:  
     + **Laplacian X-axis**:  
       - 
     + **Laplacian Y-axis**:  
       - 
3. **Edge Detection (Apply Laplacian Filter)**:  
   * Loop through the entire padded image and apply the Laplacian filters for both X and Y axes:  
     + Convolve the image with the **Laplacian X** and **Laplacian Y** filters to compute the gradients in both directions.
     + Sum the gradients along both axes for the final Laplacian output:  
        Total=Sx+Sy\text{Total} = S\_x + S\_y
   * where SxS\_x and SyS\_y are the responses of the image after applying the filters.
4. **Display Results**:  
   * Display the original image, the X and Y responses of the Laplacian filter, and the final Laplacian edge-detected image.

**Commands:**

imread

im2gray

length

zeros

conv2

sqrt

uint8

imshow

subplot

title

rgb2gray

input

padarray

sum

double

* **Implementation Steps with Screenshots :**

**MATLAB Code :**

img = imread(input('Enter image filename (with extension): ', 's'));

gray = double(rgb2gray(img));

[rows, cols] = size(gray);

padded = zeros(rows + 2, cols + 2);

padded(2:end-1, 2:end-1) = gray;

gray = padded;

roberts\_kx = [1 0; 0 -1];

roberts\_ky = [0 1; -1 0];

prewitt\_kx = [-1 0 1; -1 0 1; -1 0 1];

prewitt\_ky = [1 1 1; 0 0 0; -1 -1 -1];

sobel\_kx = [-1 0 1; -2 0 2; -1 0 1];

sobel\_ky = [-1 -2 -1; 0 0 0; 1 2 1];

roberts\_x = zeros(rows, cols);

roberts\_y = zeros(rows, cols);

roberts\_mag = zeros(rows, cols);

prewitt\_x = zeros(rows, cols);

prewitt\_y = zeros(rows, cols);

prewitt\_mag = zeros(rows, cols);

sobel\_x = zeros(rows, cols);

sobel\_y = zeros(rows, cols);

sobel\_mag = zeros(rows, cols);

for i = 2:rows+1

for j = 2:cols+1

r = gray(i-1:i, j-1:j);

gx = sum(sum(r .\* roberts\_kx));

gy = sum(sum(r .\* roberts\_ky));

roberts\_x(i-1,j-1) = gx;

roberts\_y(i-1,j-1) = gy;

roberts\_mag(i-1,j-1) = sqrt(gx^2 + gy^2);

p = gray(i-1:i+1, j-1:j+1);

gx = sum(sum(p .\* prewitt\_kx));

gy = sum(sum(p .\* prewitt\_ky));

prewitt\_x(i-1,j-1) = gx;

prewitt\_y(i-1,j-1) = gy;

prewitt\_mag(i-1,j-1) = sqrt(gx^2 + gy^2);

gx = sum(sum(p .\* sobel\_kx));

gy = sum(sum(p .\* sobel\_ky));

sobel\_x(i-1,j-1) = gx;

sobel\_y(i-1,j-1) = gy;

sobel\_mag(i-1,j-1) = sqrt(gx^2 + gy^2);

end

end

figure;

subplot(3,4,1), imshow(uint8(roberts\_x)), title('Roberts X');

subplot(3,4,2), imshow(uint8(roberts\_y)), title('Roberts Y');

subplot(3,4,3), imshow(uint8(roberts\_mag)), title('Roberts Mag');

subplot(3,4,4), imshow(uint8(gray(2:end-1,2:end-1))), title('Original');

subplot(3,4,5), imshow(uint8(prewitt\_x)), title('Prewitt X');

subplot(3,4,6), imshow(uint8(prewitt\_y)), title('Prewitt Y');

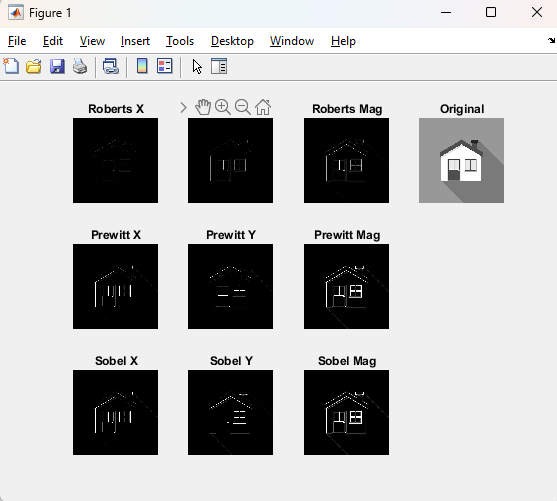
subplot(3,4,7), imshow(uint8(prewitt\_mag)), title('Prewitt Mag');

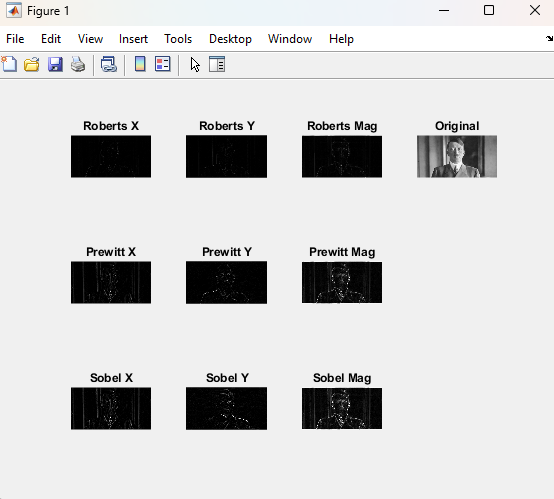
subplot(3,4,9), imshow(uint8(sobel\_x)), title('Sobel X');

subplot(3,4,10), imshow(uint8(sobel\_y)), title('Sobel Y');

subplot(3,4,11), imshow(uint8(sobel\_mag)), title('Sobel Mag');

**OUTPUT:**

****

****

**Laplacian:**  
img\_name = input('Enter image filename (with extension): ', 's');

img = imread(img\_name);

gray = double(rgb2gray(img));

[rows, cols] = size(gray);

padded = zeros(rows + 2, cols + 2);

for i = 1:rows

for j = 1:cols

padded(i+1, j+1) = gray(i, j);

end

end

laplacian\_x = [0 -1 0; 0 2 0; 0 -1 0];

laplacian\_y = [0 0 0; -1 2 -1; 0 0 0];

img\_x = zeros(rows, cols);

img\_y = zeros(rows, cols);

img\_total = zeros(rows, cols);

for i = 2:rows+1

for j = 2:cols+1

sx = 0;

sy = 0;

for ki = -1:1

for kj = -1:1

sx = sx + padded(i+ki, j+kj) \* laplacian\_x(ki+2, kj+2);

sy = sy + padded(i+ki, j+kj) \* laplacian\_y(ki+2, kj+2);

end

end

img\_x(i-1,j-1) = sx;

img\_y(i-1,j-1) = sy;

img\_total(i-1,j-1) = sx + sy;

end

end

figure;

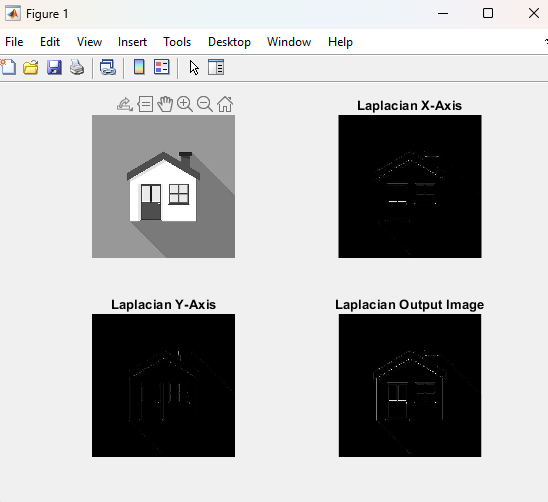
subplot(2,2,1); imshow(uint8(gray)); title('Original Image');

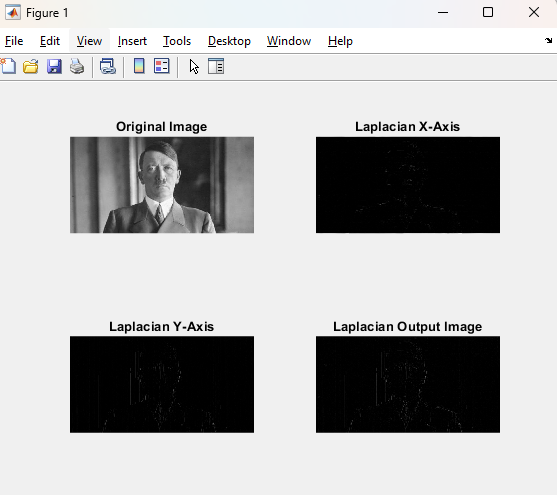
subplot(2,2,2); imshow(uint8(img\_x)); title('Laplacian X-Axis');

subplot(2,2,3); imshow(uint8(img\_y)); title('Laplacian Y-Axis');

subplot(2,2,4); imshow(uint8(img\_total)); title('Laplacian Output Image');

**OUTPUT:**

****

****

**Conclusion:** Manual implementation of Roberts, Prewitt, Sobel, and Laplacian operators detects edges by identifying intensity changes, enhancing image structure clearly.

* **Post-Lab Questions :**

1. **Explain the need of a LOG operator.**

The **Laplacian of Gaussian (LOG)** operator is used for **edge detection** in image processing. It combines **Gaussian smoothing** (to reduce noise) and the **Laplacian operator** (to detect regions of rapid intensity change). The need arises because raw Laplacian is sensitive to noise—so LOG improves accuracy by first smoothing the image, making it effective for detecting fine details and boundaries.

1. **Explain the technique of thresholding for segmentation**

**Thresholding** is a simple segmentation technique that converts a grayscale image into a binary image. It works by selecting a **threshold value (T)**:

* Pixels > T → assigned to foreground (e.g., white)
* Pixels ≤ T → assigned to background (e.g., black)

It’s useful when object and background have distinct intensity ranges. Variants include **global thresholding**, **adaptive thresholding**, and **Otsu’s method** (automatic threshold selection).

**Date: 04/04/2025 Signature of faculty In-charge**