**Sobel filter:**

% Read Input Image

input\_image = imread('butterfly.jpg');

% Displaying Input Image

input\_image = uint8(input\_image);

figure, imshow(input\_image); title('Input Image');

% Convert the truecolor RGB image to the grayscale image

input\_image = rgb2gray(input\_image);

% Convert the image to double

input\_image = double(input\_image);

% Pre-allocate the filtered\_image matrix with zeros

filtered\_image = zeros(size(input\_image));

% Sobel Operator Mask

Mx = [-1 0 1; -2 0 2; -1 0 1];

My = [-1 -2 -1; 0 0 0; 1 2 1];

% Edge Detection Process

% When i = 1 and j = 1, then filtered\_image pixel

% position will be filtered\_image(2, 2)

% The mask is of 3x3, so we need to traverse

% to filtered\_image(size(input\_image, 1) - 2

%, size(input\_image, 2) - 2)

% Thus we are not considering the borders.

for i = 1:size(input\_image, 1) - 2

for j = 1:size(input\_image, 2) - 2

% Gradient approximations

Gx = sum(sum(Mx.\*input\_image(i:i+2, j:j+2)));

Gy = sum(sum(My.\*input\_image(i:i+2, j:j+2)));

% Calculate magnitude of vector

filtered\_image(i+1, j+1) = sqrt(Gx.^2 + Gy.^2);

end

end

% Displaying Filtered Image

filtered\_image = uint8(filtered\_image);

figure, imshow(filtered\_image); title('Filtered Image');

% Define a threshold value

thresholdValue = 100; % varies between [0 255]

output\_image = max(filtered\_image, thresholdValue);

output\_image(output\_image == round(thresholdValue)) = 0;

% Displaying Output Image

output\_image = im2bw(output\_image);

figure, imshow(output\_image); title('Edge Detected Image');

**Prewitt filter:**

% Read Input Image

input\_image = imread('butterfly.jpg');

% Displaying Input Image

input\_image = uint8(input\_image);

figure, imshow(input\_image); title('Input Image');

% Convert the truecolor RGB image to the grayscale image

input\_image = rgb2gray(input\_image);

% Convert the image to double

input\_image = double(input\_image);

% Pre-allocate the filtered\_image matrix with zeros

filtered\_image = zeros(size(input\_image));

% Prewitt Operator Mask

Mx = [-1 0 1; -1 0 1; -1 0 1];

My = [-1 -1 -1; 0 0 0; 1 1 1];

% Edge Detection Process

% When i = 1 and j = 1, then filtered\_image pixel

% position will be filtered\_image(2, 2)

% The mask is of 3x3, so we need to traverse

% to filtered\_image(size(input\_image, 1) - 2

%, size(input\_image, 2) - 2)

% Thus we are not considering the borders.

for i = 1:size(input\_image, 1) - 2

for j = 1:size(input\_image, 2) - 2

% Gradient approximations

Gx = sum(sum(Mx.\*input\_image(i:i+2, j:j+2)));

Gy = sum(sum(My.\*input\_image(i:i+2, j:j+2)));

% Calculate magnitude of vector

filtered\_image(i+1, j+1) = sqrt(Gx.^2 + Gy.^2);

end

end

% Displaying Filtered Image

filtered\_image = uint8(filtered\_image);

figure, imshow(filtered\_image); title('Filtered Image');

% Define a threshold value

thresholdValue = 100; % varies between [0 255]

output\_image = max(filtered\_image, thresholdValue);

output\_image(output\_image == round(thresholdValue)) = 0;

% Displaying Output Image

output\_image = im2bw(output\_image);

figure, imshow(output\_image); title('Edge Detected Image');

**Robert filter:**

% MATLAB Code | Robert Operator from Scratch

% Read Input Image

input\_image = imread('butterfly.jpg');

% Displaying Input Image

input\_image = uint8(input\_image);

figure, imshow(input\_image); title('Input Image');

% Convert the truecolor RGB image to the grayscale image

input\_image = rgb2gray(input\_image);

% Convert the image to double

input\_image = double(input\_image);

% Pre-allocate the filtered\_image matrix with zeros

filtered\_image = zeros(size(input\_image));

% Robert Operator Mask

Mx = [1 0; 0 -1];

My = [0 1; -1 0];

% Edge Detection Process

% When i = 1 and j = 1, then filtered\_image pixel

% position will be filtered\_image(1, 1)

% The mask is of 2x2, so we need to traverse

% to filtered\_image(size(input\_image, 1) - 1

%, size(input\_image, 2) - 1)

for i = 1:size(input\_image, 1) - 1

for j = 1:size(input\_image, 2) - 1

% Gradient approximations

Gx = sum(sum(Mx.\*input\_image(i:i+1, j:j+1)));

Gy = sum(sum(My.\*input\_image(i:i+1, j:j+1)));

% Calculate magnitude of vector

filtered\_image(i, j) = sqrt(Gx.^2 + Gy.^2);

end

end

% Displaying Filtered Image

filtered\_image = uint8(filtered\_image);

figure, imshow(filtered\_image); title('Filtered Image');

% Define a threshold value

thresholdValue = 100; % varies between [0 255]

output\_image = max(filtered\_image, thresholdValue);

output\_image(output\_image == round(thresholdValue)) = 0;

% Displaying Output Image

output\_image = im2bw(output\_image);

figure, imshow(output\_image); title('Edge Detected Image');