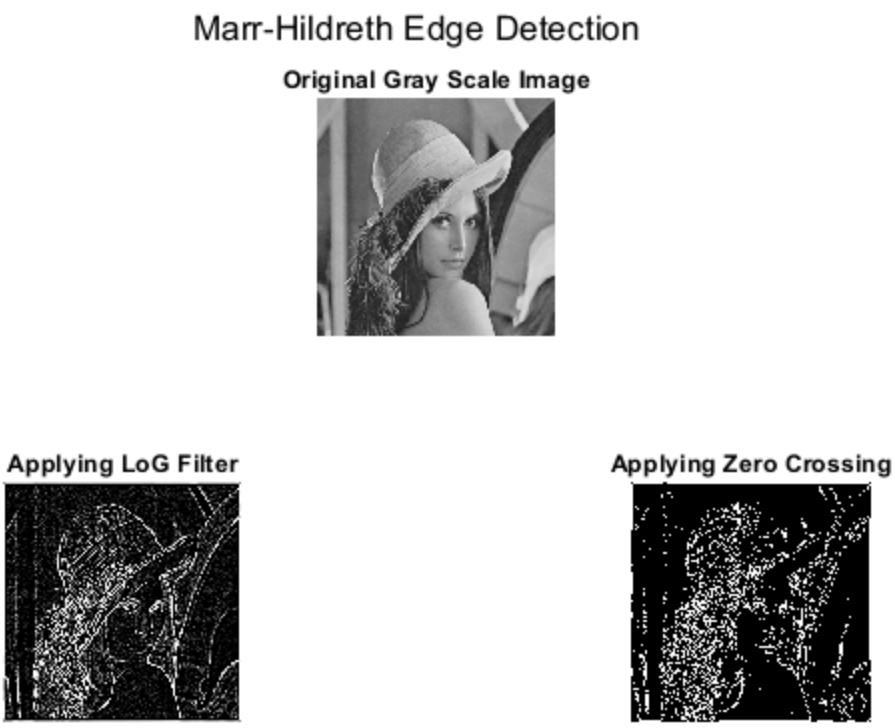
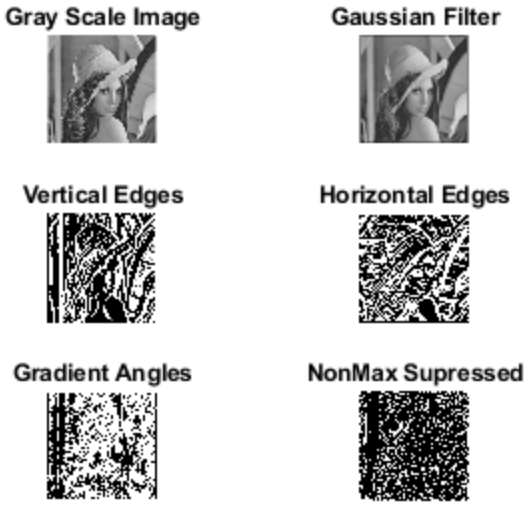
```
%NAME: Krishna Kodali
%INST: IIT Bhubaneswar
%DATE: 19/09/2020
%CATEGORY: BTech
%BRANCH: Computer Science
%Roll Number: 17CS01008

% Assignment-02
% Spatial Filtering
%Removing previous Buffer
clc;clear;close all;

%Question 1: Implement Maar edge detection and Canny edge detection on
 an input image

%Marr-Hiderith Edge Detection

MH_Image = imread("lena_gray_256.tif");

%Log Filter
Log_Filter = [0 0 -1 0 0;
              0 -1 -2 -1 0;
              -1 -2 16 -2 -1;
              0 -1 -2 -1 0;
              0 0 -1 0 0];
Neg_Log_Filter = Log_Filter.*(-1);

%Applying im2double
MH_Image = im2double(MH_Image);
LoG_Image = conv2(MH_Image, Neg_Log_Filter);

%Applying Zero Crossing
Zero_Crossing = zerocrosser(LoG_Image);
Zero_Crossing = Zero_Crossing > 100;

%Plots
sgtitle('Marr-Hildreth Edge Detection');
subplot(2,3,2)
imshow(MH_Image);
title("Original Gray Scale Image");
subplot(2,3,4)
imshow(LoG_Image)
title("Applying LoG Filter");
subplot(2,3,6)
imshow(Zero_Crossing);
title("Applying Zero Crossing");
```




```
%Canny Edge Detection
input_image = imread("lena_gray_256.tif");

%Gaussina Filter
Gaussian_Filter = [ 2 4 5 4 2;
         4 9 12 9 4;
         5 12 15 12 5;
         4 9 12 9 4;
         2 4 5 4 2 ];

% Applying Gaussina Filter
Gaussian_Filter = Gaussian_Filter.*(1 / 159);
after_Gaussian = conv2(input_image, Gaussian_Filter);

% x and y filters for Horizontal and Vertical Grading
x_filter = [ -1 0 1;
             -2 0 2;
             -1 0 1];
y_filter = [ 1 2 1;
             0 0 0;
             -1 -2 -1];
after_x_filter = conv2(after_Gaussian, x_filter, 'same');
after_y_filter = conv2(after_Gaussian, y_filter, 'same');

% Edge Orientations
edge_gradient = atan2(after_y_filter, after_x_filter);
```

```
edge_gradient = edge_gradient * 180 / pi;

[rows, cols] = size(after_Gaussian);

for i = 1 : rows
    for j = 1 : cols
        if edge_gradient(i, j) < 0
            edge_gradient(i, j) = 360 + edge_gradient(i, j);
        end
    end
end

% Getting edge directions as Angles
edge_angles = zeros([rows, cols]);
for i = 1 : rows
    for j = 1 : cols
        if ((edge_gradient(i, j) >= 0 ) && (edge_gradient(i, j) <
 22.5) || (edge_gradient(i, j) >= 157.5) && (edge_gradient(i, j) <
 202.5) || (edge_gradient(i, j) >= 337.5) && (edge_gradient(i, j) <=
 360))
            edge_angles(i, j) = 0;
        elseif ((edge_gradient(i, j) >= 22.5) && (edge_gradient(i, j)<
 67.5) || (edge_gradient(i, j) >= 202.5) && (edge_gradient(i, j) <
 247.5))
            edge_angles(i, j) = 45;
        elseif ((edge_gradient(i, j) >= 67.5 && edge_gradient(i, j)<
 112.5) || (edge_gradient(i, j) >= 247.5 && edge_gradient(i, j) <
 292.5))
            edge_angles(i, j) = 90;
        elseif ((edge_gradient(i, j) >= 112.5 && edge_gradient(i, j)<
 157.5) || (edge_gradient(i, j) >= 292.5 && edge_gradient(i, j) <
 337.5))
            edge_angles(i, j) = 135;
        end
    end
end

%Magnitide of Horizontal and Vertical edge filters
xy_filtered_magnitude = sqrt((after_x_filter.^2) +
 (after_y_filter.^2));
nmax_supr = zeros(rows, cols);
for i = 2 : rows - 1
    for j = 2 : cols - 1
        if edge_angles(i, j) == 0
            nmax_supr(i, j) = (xy_filtered_magnitude(i, j) ==
 max([xy_filtered_magnitude(i, j), xy_filtered_magnitude(i, j -
 1),xy_filtered_magnitude(i, j + 1)]));
        elseif edge_angles(i, j) == 45
            nmax_supr(i, j) = (xy_filtered_magnitude(i, j) ==
 max([xy_filtered_magnitude(i, j), xy_filtered_magnitude(i - 1, j +
 1), xy_filtered_magnitude(i + 1, j - 1)]));
        elseif edge_angles(i, j) == 90
```

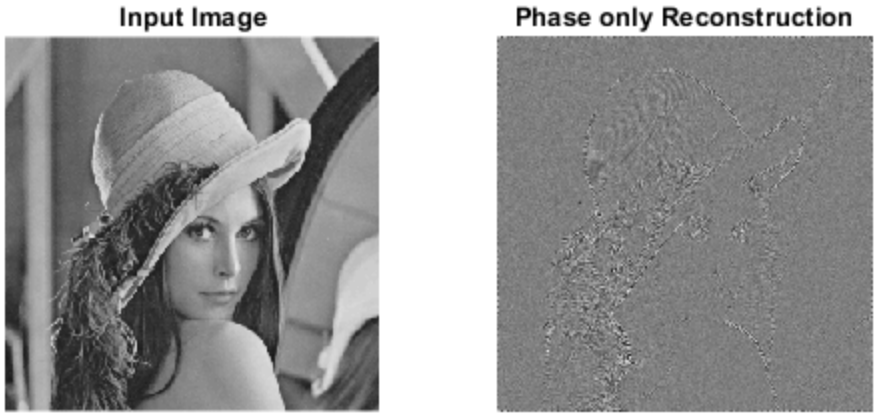
```
            nmax_supr(i, j) = (xy_filtered_magnitude(i, j)
 ==max([xy_filtered_magnitude(i, j), xy_filtered_magnitude(i - 1, j),
 xy_filtered_magnitude(i + 1, j)]));
        elseif edge_angles(i, j) == 135
            nmax_supr(i, j) = (xy_filtered_magnitude(i, j) ==
 max([xy_filtered_magnitude(i, j), xy_filtered_magnitude(i + 1, j +
 1), xy_filtered_magnitude(i - 1, j - 1)]));
        end
    end
end

nmax_supr = nmax_supr.*xy_filtered_magnitude;
% Thresholding values
% Values
t_low = 0.08 * max(max(nmax_supr));
t_high = 0.18 * max(max(nmax_supr));
result_image = zeros([rows, cols]);
for i = 1 : rows
    for j = 1 : cols
        if nmax_supr(i, j) < t_low
            result_image(i, j) = 0;
        elseif nmax_supr(i, j) > t_high
            result_image(i, j) = 1;
        elseif min([nmax_supr(i - 1, j - 1), nmax_supr(i - 1, j),
 nmax_supr(i - 1, j + 1), nmax_supr(i, j - 1), nmax_supr(i, j +
 1),nmax_supr(i + 1, j - 1), nmax_supr(i + 1, j), nmax_supr(i + 1, j +
 1)]) > t_high
            result_image(i, j) = 1;
        end
    end
end
result_image = uint8(result_image.*255);
figure;
subplot(1,1,1);
imshow(result_image);
title("Manual Method");
subplot(3, 2, 1);
imshow(input_image);
title('Gray Scale Image');
subplot(3, 2, 2);
imshow(after_Gaussian/255);
title('Gaussian Filter');
subplot(3, 2, 3);
imshow(after_x_filter);
title('Vertical Edges');
subplot(3, 2, 4);
imshow(after_y_filter);
title('Horizontal Edges');
subplot(3, 2, 5);
imshow(edge_angles);
title('Gradient Angles');
subplot(3, 2, 6);
imshow(nmax_supr);
title('NonMax Supressed');
```



3. Perform phase-only reconstruction using two images in Fourier domain.

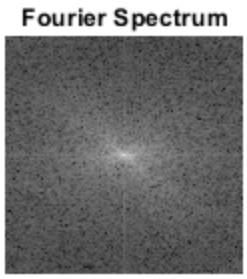
```
Image = imread("lena_gray_256.tif");
Image = double(Image);

[row,col]=size(Image);
%Finding 2d Fourier Transform
X=zeros(row,col);
Y=zeros(row,col);
for i=1:row
    X(i,:)=fft(Image(i,:));
end
for j=1:col
    Y(:,j)=fft(X(:,j));
end
%Making a shift
DFT_Image = fftshift(Y);
%Magnitude and Phase components
magnitude = abs(DFT_Image);
phase = atan2(imag(DFT_Image),real(DFT_Image));
Phase_Response = exp(1i*phase);
%Reconstruction from phase
for i=1:row
    X(i,:) = ifft(Phase_Response(i,:));
end
for j=1:col
    Y(:,j) = ifft(X(:,j));
end
Output_Image = real(Y);

%Plots
figure;
subplot(1, 2, 1);
imshow(uint8(Image));
```

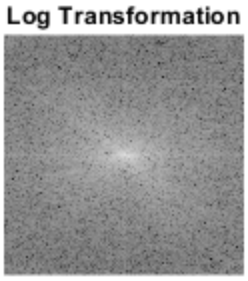
```
title("Input Image");
subplot(1, 2, 2);
imshow(mat2gray(Output_Image));
title("Phase only Reconstruction");
```




```
%4. Compute 2D fourier spectrum of an input image and centre the
 magnitude spectrum and display after log transformation.

%Reading the grayscale image
grayImage = imread('lena_gray_256.tif');

% Checking the dimensions of the image to verify the grayscale image.
if size(grayImage,3) == 3
  grayImage = grayImage(:, :, 2);
end

% Finding the fourier spectrum of the image.
Fourier = fft2(grayImage);
Spectrum = fftshift(log(1+abs(Fourier)));

% Applying log transformation of the Fourier spectrum.
logtrans = 2*log(1+(Spectrum));

% Plotting all the images using subplot
subplot(2, 3, 2);
imshow(grayImage, []);
```

```
title('Original Grayscale Image');
subplot(2, 3, 4);
imshow(Spectrum, []);
title('Fourier Spectrum');
subplot(2, 3, 6);
imshow(logtrans, []);
title('Log Transformation');

% Functions used previously in Marr-Hilderith Detector
function zero_crossed_image = zerocrosser(input_image)
    [rows, cols] = size(input_image);
    zero_crossed_image = zeros([rows, cols]);
    for i = 2 : rows - 1
        for j = 2 : cols - 1
            if input_image(i, j) > 0
                topleft = input_image(i - 1, j - 1);
                bottomright = input_image(i + 1, j + 1);

                topright = input_image(i - 1, j + 1);
                bottomleft = input_image(i + 1, j - 1);

                top = input_image(i - 1, j);
                bottom = input_image(i + 1, j);

                left = input_image(i, j - 1);
                right = input_image(i, j + 1);

                if (topleft >= 0 && bottomright < 0) || (topleft < 0
 && bottomright >= 0)
                    zero_crossed_image(i, j) = max(topleft,
 bottomright);
                elseif (topright >= 0 && bottomleft < 0) || (topright<
 0 && bottomleft >= 0)
                    zero_crossed_image(i, j) = max(topright,
 bottomleft);
                elseif (top >= 0 && bottom < 0) || (top < 0 && bottom
 >= 0)
                    zero_crossed_image(i, j) = max(top, bottom);
                elseif (left >= 0 && right < 0) || (left < 0 && right
 >= 0)
                    zero_crossed_image(i, j) = max(left, right);
                end
            end
        end
    end
 zero_crossed_image = im2uint8(zero_crossed_image);
end
```





Published with MATLAB® R2020a