UNIVERSITY OF ENGINEERING AND TECHNOLOGY, TAXILA

ELECTRICAL ENGINEERING DEPARTMENT

Digital Image Processing
Lab Project
Complex Engineering Problem

For

7th Semester

Electrical Engineering Department

(B.Sc. Electrical Engineering)

Lab Instructor: Engr. Israr Ahmad Khan
Course Instructor: Prof. Dr. Gulistan Raja

Submitted By

Name	Registration Number
Javeria Moazzam	18-EE-003
Zarafshan Abbas	18-EE-036
Azib Farooq	18-EE-043

Table of Contents

1. Problem Statement 2
2. Aim and Objectives 2
3. Literature Review 2
   3.1. Edges 2
   3.2. Modeling Intensity Changes 3
   3.3. Noise in Images 3
   3.4. Edge Detection Techniques 4
   3.5. Applications of Image Edge Detection 6
4. Software Used 6
5. Proposed Design 6
   5.1. Canny Edge Detection Algorithm 7
   5.2. MATLAB Code 8
6. Results Interpretation and Investigation 12
   6.1. Output of Edge Detector 12
   6.2. Comparison with Built-In Function 13
   6.3. Performance Evaluation 14
7. Conclusion 16
8. References 16

1. Problem Statement

Select and implement an algorithm that will efficiently detect edges in the images. The implementation should be robust enough to deal certain uncertainties due to presence of noise. Select the controlling parameters for your design as the optimal detector is the one which minimizes the probability of detecting false edges and maximizes real edges.

2. Aim and Objectives

The aim of this project is to implement a suitable algorithm for edge detection using MATLAB software. The main objectives include:

- Studying different edge detection algorithms and implement an appropriate algorithm to meet the given constraints.
- Image smoothing for noise reduction
- Detection of edge points
- Edge localization

3. Literature Review

Edge detection is a segmentation approach in image processing used to identify points in a digital image with discontinuities, simply to say, sharp changes in the image brightness frequently can be classified. The goal of edge detection is to produce a line drawing of a scene from an image of that scene. Important features can be extracted from the edges of an image (e.g., corners, lines, curves). These features are used by higher-level computer vision algorithms. Edge detection works on the principle of identifying places in an image where brightness differs suddenly or radically. Edge detection is usually a part of image preprocessing. Let's look into some of the important concepts in edge detection and some basic edge detection algorithms in literature.

3.1. Edges

Edges normally exist inside a picture on the border of two distinct regions. In Image Processing, an edge can be defined as a set of contiguous pixel positions where an abrupt change of intensity (gray or color) values occur. Edges represent boundaries between objects and background. Sometimes, the edge-pixel-sequence may be broken due to insufficient intensity difference.

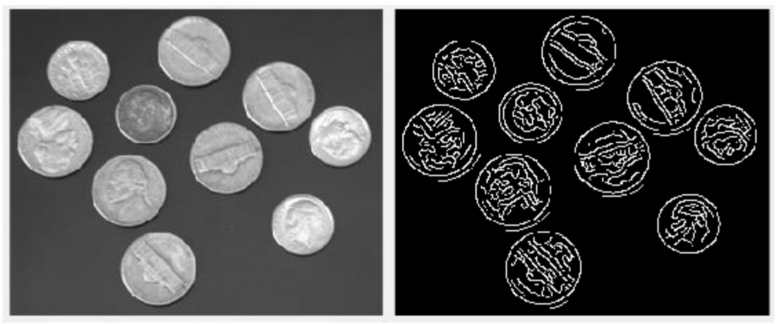


Figure 1: Example of image edges

3.2. Modeling Intensity Changes

Edges can be modeled according to their intensity profiles. They are discussed below:

1. **Step edge:** Step edge is characterized by a transition between two intensity levels occurring ideally over the distance of one pixel. The image intensity abruptly changes from one value to one side of the discontinuity to a different value on the opposite side.

2. **Ramp edge**: It is basically a step edge where the intensity change is not instantaneous but occurs over a finite distance. In this model, we no longer have a single “edge point” along the profile. Instead, an edge point now is any point contained in the ramp, and an edge segment would then be a set of such points that are connected

3. **Ridge edge:** The image intensity abruptly changes value but then returns to the starting value within some short distance generated usually by lines.

4. **Roof edge**: It is a ridge edge where the intensity change is not instantaneous but occurs over a finite distance (generated usually by the intersection of surfaces). Roof edges are models of lines through a region, with the base (width) of the edge being determined by the thickness and sharpness of the line.

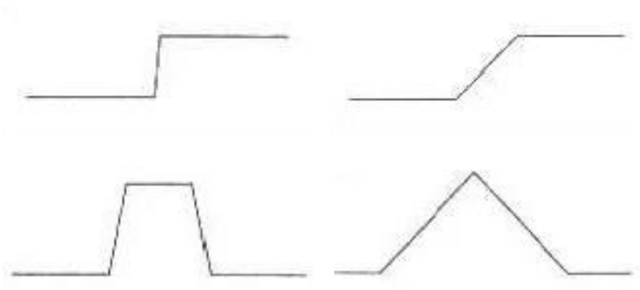


Figure 2 : Edge Profiles (a) Step Edge (b) Ramp Edge (c) Ridge Edge (d) Roof Edge

3.3. Noise in Images

Image noise is random variation of brightness or color information in the images captured. It is degradation in image signal caused by external sources. Mostly it is additive. We can represent noise in an image as following:

□(□,□)=□(□,□)+□(□,□)

Where:

A(x,y)= function of noisy image
B(x,y)= function of image noise
H(x,y)= function of original image

The presence of noise in an image can cause false edges to form during the edge detection process. This is because noise is also a abrupt variation in intensity in an image. Thus it is important to remove noise from an image before edge detection using suitable noise reduction filters.

There are various types of noises in image processing. Some of which are:

1. Salt and pepper noise
2. Speckle noise
3. Gaussian noise
4. Poisson noise

There are several techniques in image processing for removing noise from images. Linear smoothing filters such as the Gaussian filter are the most common type of noise reduction techniques. The noisy image is convolved with a smoothing (low pass) filter to produce the image with reduced noise.

3.4. Edge Detection Techniques

Many algorithms have been developed for edge detection. Here we will discuss some of the edge detectors that are most commonly used.

Sobel Operator

The sobel is one of the most commonly used edge detectors. It computes the gradient approximation of image intensity function for image edge detection. It uses two 3 x 3 kernels or masks which are convolved with the input image to calculate the vertical and horizontal derivative approximations respectively

$$\boldsymbol{Kx} = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, \quad \boldsymbol{Ky} = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

The advantages of sobel operator are that it is simple and time efficient. It can easily search for smooth edges. However, it has some limitations as well. The diagonal direction points are not preserved always and it is highly sensitive to noise. Also it is not very accurate in edge detection and the detected edges are thick and rough, so it does not give appropriate results

Prewitt Operator

This operator is almost similar to the sobel operator. It also detects vertical and horizontal edges of an image. It uses different kernels than that of sobel operator.

$$\boldsymbol{Kx} = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix}, \quad \boldsymbol{Ky} = \begin{bmatrix} +1 & +1 & +1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

It has better performance than sobel operator and it is the best operator to detect the orientation of an image. However the diagonal direction points are not preserved always.

Robert Operator

This gradient-based operator computes the sum of squares of the differences between diagonally adjacent pixels in an image through discrete differentiation. Then the gradient approximation is made. It uses the following 2 x 2 kernels

$$\boldsymbol{Kx} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \boldsymbol{Ky} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

The diagonal direction points are preserved in this kind of operator but it is very sensitive to noise.

Marr-Hildreth Operator or Laplacian of Gaussian (LoG)

It is a gaussian based operator which uses the Laplacian to take the second derivative of an image. It works on the zero-crossing method i.e when the second-order derivative crosses zero, then that particular location corresponds to a maximum level. It is called an edge location. Here the Gaussian operator reduces the noise and the Laplacian operator detects the sharp edges.

The Gaussian function is defined by the formula

$$G(x,y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp - \left(\frac{x^2+y^2}{2\sigma^2}\right)$$

Where σ is the standard deviation.

It is very easy to detect edges and their various orientations using this technique. However, this technique is very sensitive to noise and it generates false edges. Also, the localization error may be severe at curved edges.

Canny Operator

It is a gaussian-based operator in detecting edges. This operator is not susceptible to noise. It extracts image features without affecting or altering the feature. Canny edge detector have advanced algorithm derived from the previous work of Laplacian of Gaussian operator. It is widely used an optimal edge detection technique. It detects edges based on three criteria:

- Low error rate
- Edge points must be accurately localized
- There should be just one single edge response

This technique has good localization. It extracts image features without altering the features and is less sensitive to noise. However, it is time consuming.

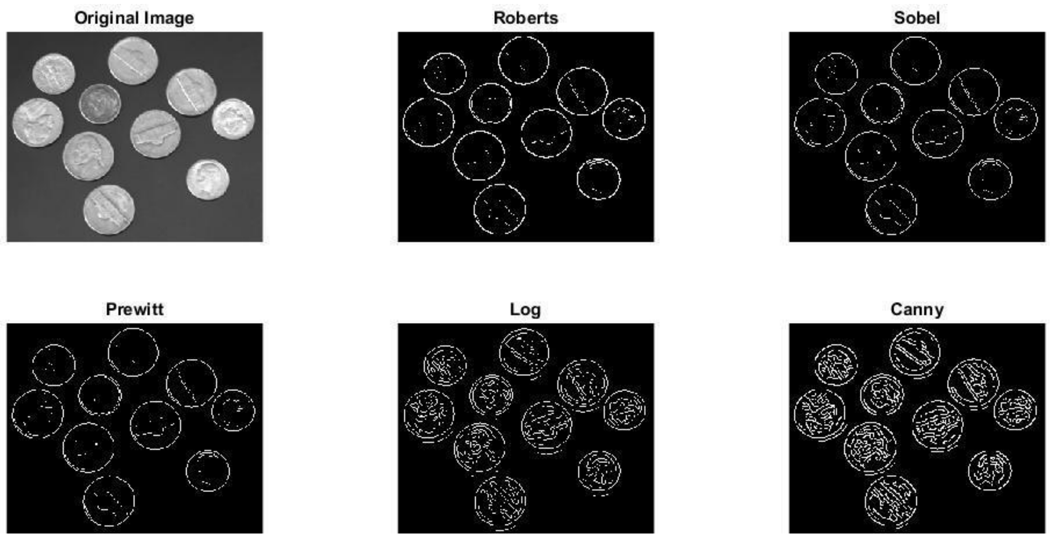


Figure 3: Comparison of Edge Detection Techniques

3.5. Applications of Image Edge Detection

Some real world applications of image edge detection include:

1. Medical imaging, study of anatomical structure
2. Object location in satellite images
3. Automatic traffic controlling systems
4. Face recognition
5. Fingerprint detection

4. Software Used

We have used **MATLAB R2020a** to implement this project

5. Proposed Design

After studying various edge detection techniques in literature, we conclude that the canny edge detection algorithm will best meet our requirements. The block diagram of our proposed design is show below:

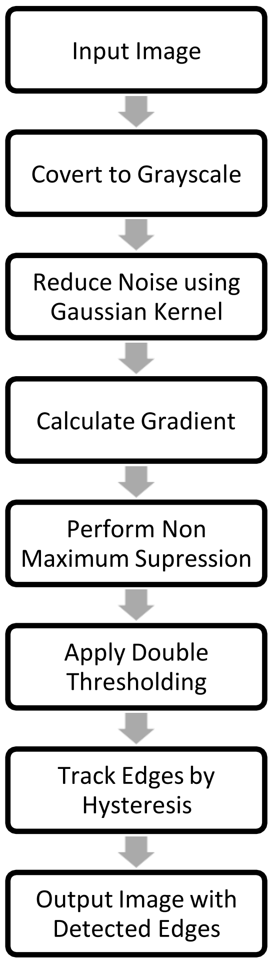

Input Image
Covert to Grayscale
Reduce Noise using Gaussian Kernel
Calculate Gradient
Perform Non Maximum Supression
Apply Double Thresholding
Track Edges by Hysteresis
Output Image with Detected Edges

Figure 4: Block Diagram for Canny Edge Detection Algorithm

5.1. Canny Edge Detection Algorithm

The Canny edge detector is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It has the following steps

Step 1: Input Image and Convert it to Grayscale

The first step is to input the image in which edges are to be detected. This algorithm is based on grayscale images so we need to convert the image to grayscale if it is an RGB image.

Step 2: Apply Gaussian Filter for Noise Reduction

The edge detection techniques are based on derivatives, so they are highly sensitive to noise. Thus noise reduction is a crucial step. Canny algorithm uses Gaussian filter to smooth the image. To do so, image convolution technique is applied with a Gaussian Kernel. Smaller the kernel, the less visible is the blur. We have used the built in function ‘imgaussfilt(image, sigma) with sigma equal to 1.4 for noise reduction.

Step 3: Compute Gradients

This step finds the intensity and direction of the edges by calculating the gradient of the image using edge detection operators. An image gradient is the two-dimensional gradient vector representing the directional change in intensity of an image. These intensity values provide the information necessary to determine the positions and polarities of edges. When the image is smoothed, the derivatives dx and dy with respect to x and y are calculated. It is done by convolving the image with Sobel kernels Kx and Ky, respectively:

$$\boldsymbol{Kx} = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, \quad \boldsymbol{Ky} = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Then, the magnitude and the direction of the gradient are calculated as follow:

$$|\boldsymbol{G}| = (\boldsymbol{dx}^2 + \boldsymbol{dy}^2)^{\frac{1}{2}}$$
$$\boldsymbol{\theta} = \arctan(\frac{\boldsymbol{dy}}{\boldsymbol{dx}})$$

Step 4: Non- Maximum Supression

Ideally, the final image should have thin edges. Thus, we must perform non-maximum suppression to thin out the edges. This step compares the edge strength of the current pixel with the edge strength of the pixel in the positive and negative gradient directions.

- Each pixel is iterated through
- If the edge strength of the current pixel is large compared to the other pixels in the mask with the same direction, the value will be preserved.
- Otherwise, it will be suppressed.

The result is the same image with thinner edges.

Step 5: Double Threshold

This step aims at identifying 3 kinds of pixels: strong, weak, and non-relevant.

- If an edge pixel’s gradient value is higher than the high threshold value, it is marked as a strong edge pixel.
- If an edge pixel’s gradient value is smaller than the high threshold value and larger than the low threshold value, it is marked as a weak edge pixel.
- If an edge pixel's gradient value is smaller than the low threshold value, it will be suppressed.

The result of this step is an image with only 2 pixel intensity values (strong and weak).

Step 6: Edge Detection by Hysteresis

Hysteresis is a way of linking the broken lines produced in the double threshold step. This is done by iterating over the pixels and checking if the current pixel is an edge.

- If it is an edge, check surrounding area for edges.
- If they have the same direction, mark them as an edge pixel.
- When there are no more changes to the image we stop

Based on the threshold results, the hysteresis consists of transforming weak pixels into strong ones, if and only if at least one of the pixels around the one being processed is a strong one

5.2. MATLAB Code

```
function [Edge_Detected_Image]=Edge_Detector(image)

%18-EE-3    Javeria Moazzam
%18-EE-36   Zarafshan Abbas
%18-EE-43   Azib Farooq

%DIP CEP: CANNY EDGE DETECTOR

%% STEP 1:

%Display original image
figure,
subplot(231)
imshow(image);
title('Original Image');

%Covert image from RGB to grayscale
if size(image,3)~=1
    image=rgb2gray(image);
end
image= double(image)/255;

%% STEP 2: NOISE REMOVAL
subplot(232)
image= imgaussfilt(image,1.4);
imshow(image)
title('Image after Noise Reduction');
```

```
%% STEP 3: Gradient Calculation
% Sobel Kernels
Kx= [-1 0 1; -2 0 2; -1 0 1];
Ky= [1 2 1; 0 0 0; -1 -2 -1];
K0 = {fliplr(flipud(Kx)), fliplr(flipud(Ky))};
[m n]= size(image);
padded_image = padarray((image), [1 1]);
output_K = zeros(m,n);
% Convolution
for num=1:2
    K= K0{1,num};
    for i = 1:size(padded_image,1)-2
        for j = 1:size(padded_image,2)-2
            Temp = (padded_image(i:i+2,j:j+2)).*K;
            output_K (i,j)  = sum(Temp(:));
        end
        if num==1
            dx= output_K;
        else
            dy= output_K;
        end
    end
end

% Magnitude and Direction
mag = sqrt((dx.^2 + dy.^2));
theta = atan2d(dy,dx);
subplot(233)
imshow(mag);
title('magnitude of gradient');
subplot(234)
imshow(theta);
title('directions of gradient');
[x,y]=size(theta);
%Adjustment for negative directions, making all directions positive
for i=1:x
    for j=1:y
        if (theta(i,j)<0)
            theta(i,j)=360+theta(i,j);
        end
    end
end
theta2=zeros(x, y);
%Adjusting directions to nearest 0, 45, 90, or 135 degree
for i = 1  : x
    for j = 1 : y
        if (((theta(i, j) >= 0 ) && (theta(i, j) < 22.5)) ||
((theta(i, j) >= 157.5) && (theta(i, j) < 202.5) )|| ((theta(i, j)
>= 337.5) && (theta(i, j) <= 360)))
            theta2(i, j) = 0;
        elseif (((theta(i, j) >= 22.5) && (theta(i, j) < 67.5)) ||
((theta(i, j) >= 202.5) && (theta(i, j) < 247.5)))
            theta2(i, j) = 45;
        elseif (((theta(i, j) >= 67.5 && theta(i, j) < 112.5)) ||
((theta(i, j) >= 247.5 && theta(i, j) < 292.5)))
            theta2(i, j) = 90;
        elseif (((theta(i, j) >= 112.5 && theta(i, j) < 157.5)) ||
((theta(i, j) >= 292.5 && theta(i, j) < 337.5)))
            theta2(i, j) = 135;
        end
    end
end
```

```
%% Step 4: Non Maximum Supression

NMS = zeros (x, y);

for i=2:x-1
    for j=2:y-1
        if (theta2(i,j)==0)
            NMS(i,j) = (mag(i,j) == max([mag(i,j), mag(i,j+1),
mag(i,j-1)]));
        elseif (theta2(i,j)==45)
            NMS(i,j) = (mag(i,j) == max([mag(i,j), mag(i+1,j-1),
mag(i-1,j+1)]));
        elseif (theta2(i,j)==90)
            NMS(i,j) = (mag(i,j) == max([mag(i,j), mag(i+1,j),
mag(i-1,j)]));
        elseif (theta2(i,j)==135)
            NMS(i,j) = (mag(i,j) == max([mag(i,j), mag(i+1,j+1),
mag(i-1,j-1)]));
        end
    end
end

NMS = NMS.*mag;
subplot(235)
imshow(NMS);
title('non-maximum suppression');

%% Step 5&6: Hysteresis Thresholding

T_Low = 0.05;
T_High = 0.12;

T_Low = T_Low * max(max(NMS));
T_High = T_High * max(max(NMS));
T_res = zeros (x, y);
for i = 1  : x
    for j = 1 : y
        if (NMS(i, j) < T_Low)
            T_res(i, j) = 0;
        elseif (NMS(i, j) > T_High)
            T_res(i, j) = 1;
        %Using 8-connected components
        elseif ( NMS(i+1,j)>T_High || NMS(i-1,j)>T_High ||
NMS(i,j+1)>T_High || NMS(i,j-1)>T_High || NMS(i-1, j-1)>T_High ||
NMS(i-1, j+1)>T_High || NMS(i+1, j+1)>T_High || NMS(i+1, j-
1)>T_High)
            T_res(i,j) = 1;
        end
    end
end

Edge_Detected_Image = uint8(T_res.*255);

%% Show final edge detection result
subplot(236)
imshow(Edge_Detected_Image);
title('Edge Detected Image');

end
```

Test Code

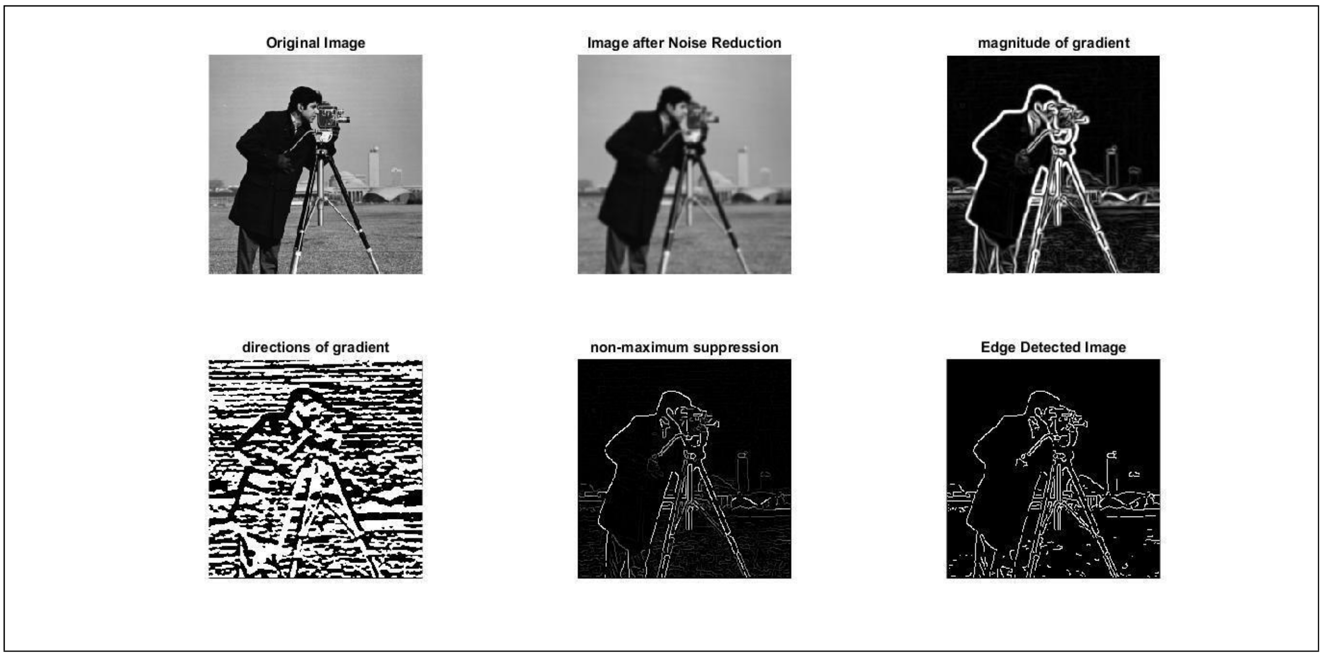
```
% CODE TO:
% TEST OUR EDGE DETECTOR FUNCTION
% COMPARE THE RESULTS OF OURFUNCTION WITH THAT OF BUILT IN FUNCTION
% EVALUATE PERFORMANCE OF OUR FUNCTION

%Read Image
image_noiseless=imread('cameraman.tif');
image_noisy=imnoise(image_noiseless,'speckle',0.002);

%Edge Detection using Algorithm
disp('For Noiseless Image');
tic
E_Im1=Edge_Detector(image_noiseless);
toc
disp('For Noisy Image');
tic
E_Im2=Edge_Detector(image_noisy);
toc

% Edge Detection using built-in function
E_Built_In1 = edge(image_noiseless, 'canny' , 0.15, 'both');
E_Built_In2 = edge(image_noisy, 'canny' , 0.15, 'both');

% Comaprison with Built in Function
figure;
subplot(131)
imshow(image_noiseless)
title('Noiseless Image');
subplot(1,3,2)
imshow(E_Im1);
title('Edges Detected using our Function');
subplot(1,3,3)
imshow(E_Built_In1);
title('Edges Detected using Built-in Function');

figure;
subplot(1,3,1)
imshow(image_noisy);
title('Noisy Image');
subplot(1,3,2)
imshow(E_Im2);
title('Edges Detected Using Our Function');
subplot(1,3,3)
imshow(E_Built_In2);
title('Edges Detected Using Built-In Function');

%Performance Evaluation

% Peak Signal to Noise Ratio
PSNR_Noiseless=psnr(E_Im1,image_noiseless)
PSNR_Noisy=psnr(E_Im2,image_noisy)

% Mean Square Error
mse_error_Noiseless = immse(E_Im1,image_noiseless)
mse_error_Noisy = immse(E_Im2,image_noisy)
```

6. Results Interpretation and Investigation

6.1. Output of Edge Detector

Following are the outputs obtained from the edge detector

(a) Noiseless Image



(b) Noisy Image

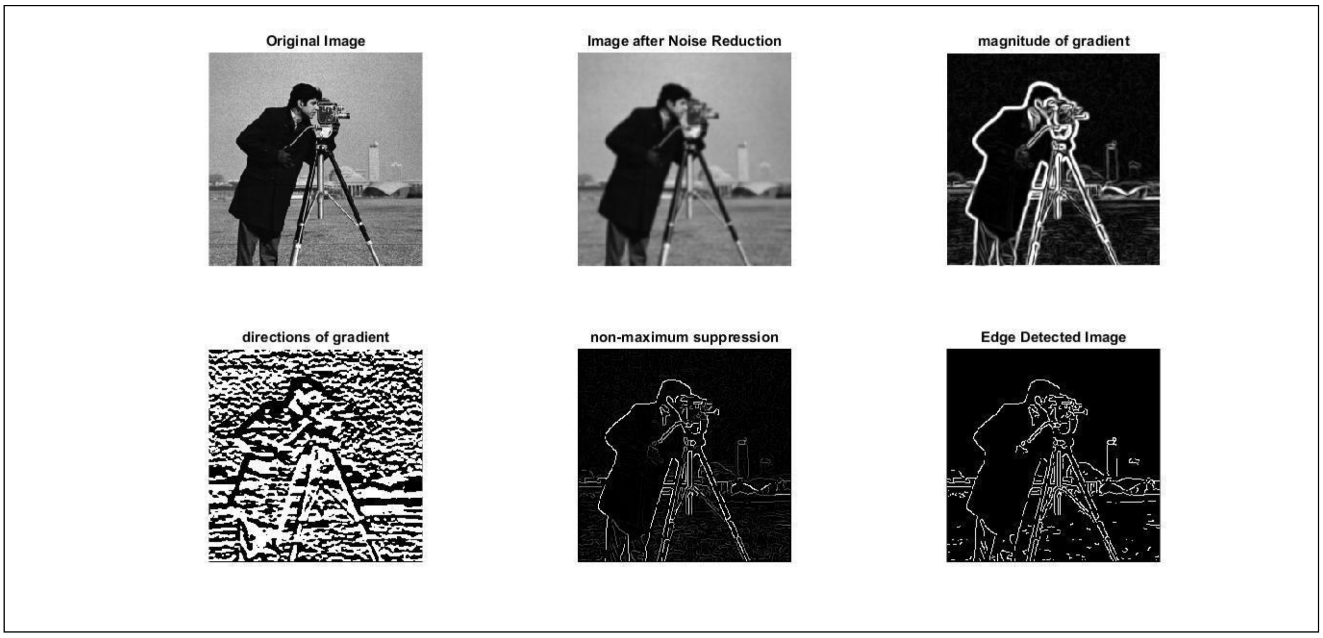


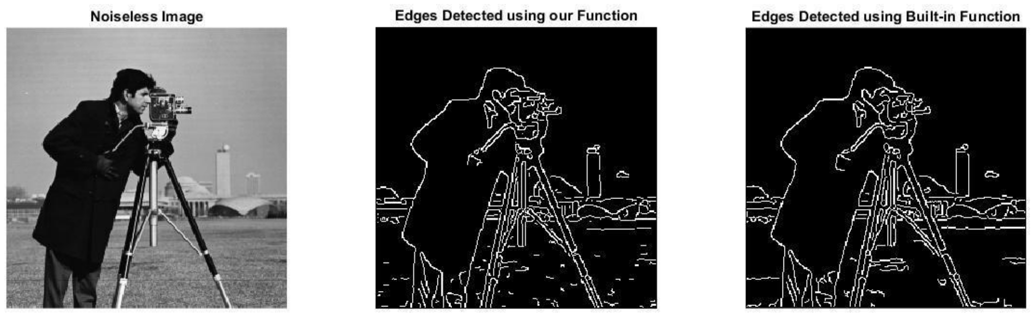
Figure 5: Edge Detector Results (a) Noiseless Image (b) Noisy Image

We see that the edges have been detected using our function. This proves the validity of our function. The results show that the function is working and has met all the required specifications of this project.

6.2. Comparison with Built-In Function

Next, we have compared the results of our function with that of the built in function of MATLAB.

(a) Noiseless Image


Noiseless Image
Edges Detected using our Function
Edges Detected using Built-in Function

(b) Noisy Image

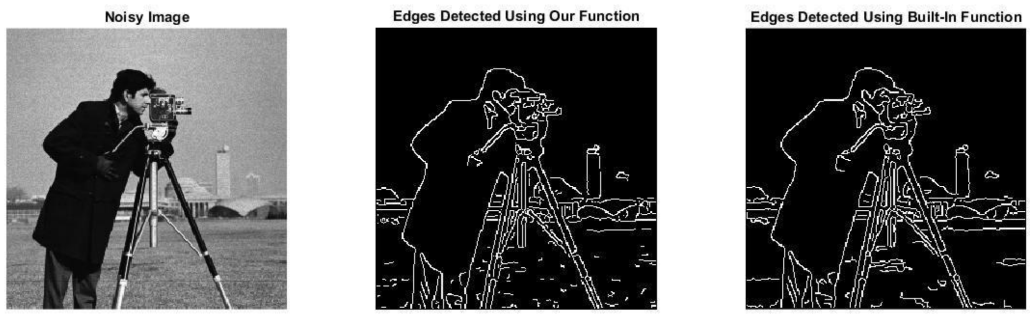

Noisy Image
Edges Detected Using Our Function
Edges Detected Using Built-In Function

Figure 6: Comparison (a) Noiseless Image (b) Noisy Image

The results show that the edge detected image using our function and the MATLAB's built in function are quite alike.

6.3. Performance Evaluation

The performance evaluation of edge detections algorithms are accomplished by detection of true edges, processing time, error ratio, and noise level etc. We have used the following three metrics for performance evaluation.

A. Mean Squared Error

MSE specifies the average difference of the pixels throughout the original ground truth image with edge detected image. The higher MSE indicates a greater difference between the original and processed image. The MSE should be less, when you are dealing with image restoration, reconstruction and compression. But for image edge detection, the Mean Squared Error should be higher to ensure it found more edge points on the image and it is capable of detecting weak edge points. The MSE calculated is:

```
mse_error_Noiseless =
   1.8849e+04
mse_error_Noisy =
   1.8854e+04
```

B. Processing Time

Processing Time should be as low as possible. But canny is time consuming operator so it takes more time to detect correct edges. These are the results obtained for processing time

```
For Noiseless Image
Elapsed time is 1.494295 seconds.
For Noisy Image
Elapsed time is 1.239507 seconds.
```

C. Peak Signal to Noise Ratio

Peak signal-to-noise ratio, is ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. Although a higher PSNR generally indicates that the reconstruction is of higher quality in image compression. But in some cases like edge detection PSNR should lesser to achieve proper results.

```
PSNR_Noiseless =
    5.3779
PSNR_Noisy =
    5.3767
```

Table: Performance Evaluation for noisy and noiseless image 'cameraman.tif'

Parameters	**Calculated Results**	
	Noiseless Image	**Noisy Image**
Mean Square Error	3.6880e+03	3.8884e+03
Processing Time	1.494295	1.239507
Peak Signal to Noise Ratio	12.4629	12.2330

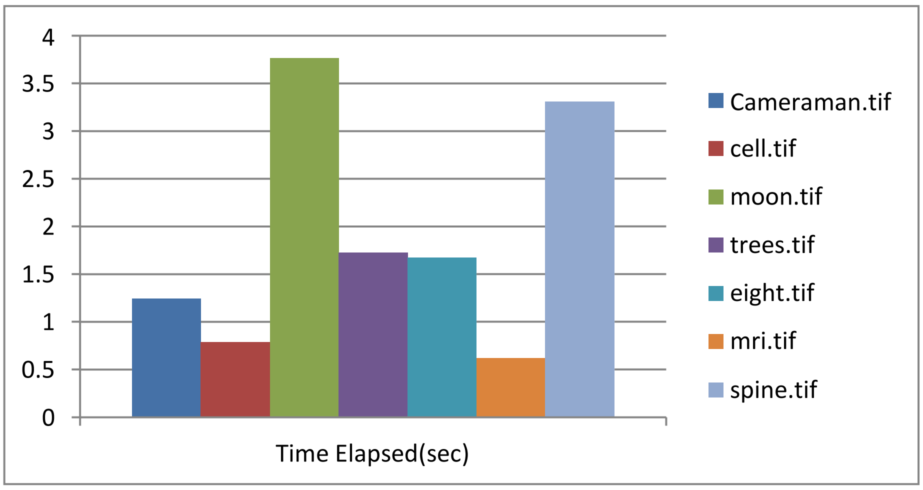


Figure 7: Time Elapsed for Different Images

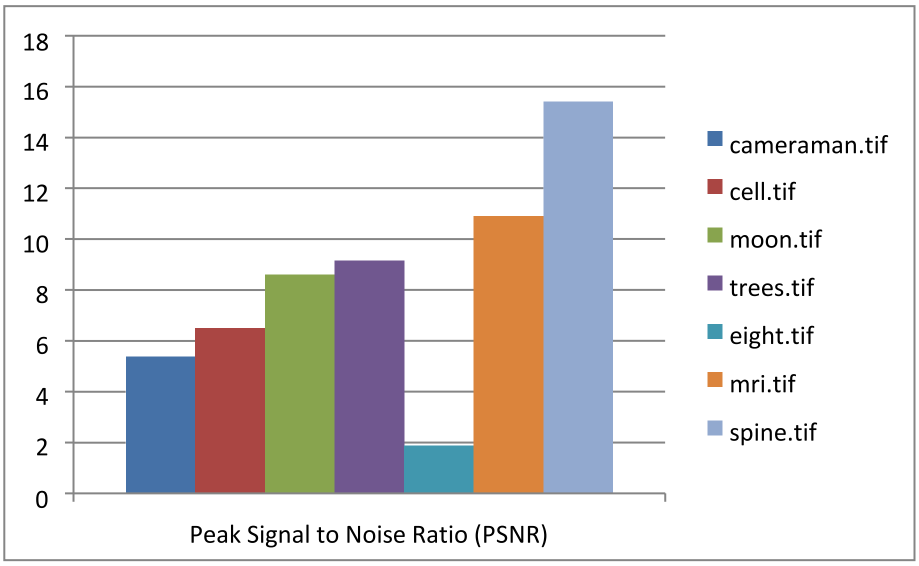


Figure 8: PSNR for Different Images

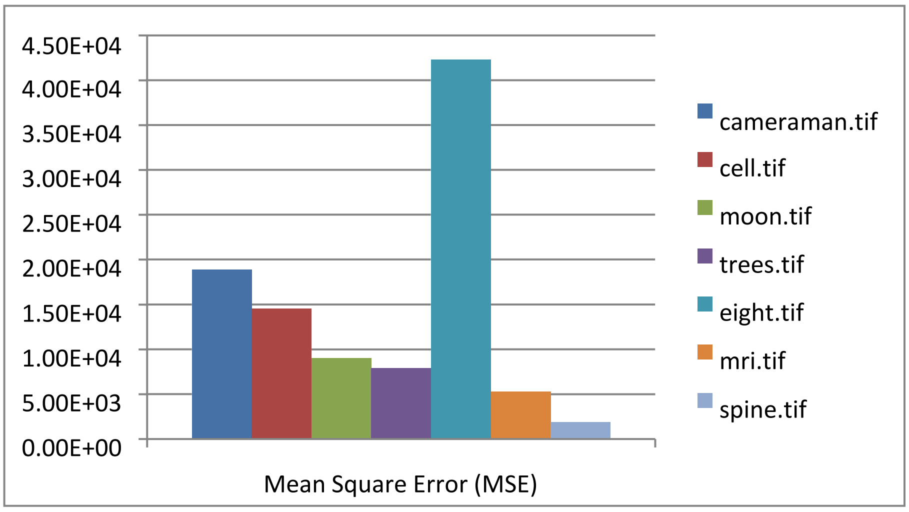


Figure 9: MSE for Different Images

7. Conclusion

Edge detection is an important step in many image processing and computer vision applications. Removal of noise before processing is necessary so we can get the accurate results. An edge detector has been successfully designed by us using canny edge detection algorithm. Our detector used Gaussian filter to smooth the noise first to prevent the detection of false edges. The results show that the edge detector designed by us meets all the design requirements and the results are close to the built in function for edge detection in MATLAB.

8. References

[1] Djemel Ziou, Salvatore Tabbone Edge detection techniques - an overview" International Journal of Pattern Recognition and Image Analysis,1998

[2] Wang and Jian-Qiu Jin, "An edge detection algorithm based on Canny operator," Intelligent Systems Design and Applications. ISDA 2007. 20-24 Oct. 2007 Page(s):623 – 628

[3] Ali Alshumrani, A.T. Papagiannakis 'An Improved Canny Edge Detection Application for Asphalt Concrete', October 2009

[4] G.Padmavathi, P.Subashini, and P.K.Lavanya, 'Performance evaluation of the various edge detectors and filters for the noisy IR images, Sensors, Signals, Visualization, Imaging, Simulation and Materials, pp. 199 – 203

[5] D. Poobathy, R. Manicka Chezian,"Edge Detection Operators: Peak Signal to Noise Ratio Based Comparison", IJIGSP, vol.6, no.10, pp.55-61, 2014.DOI: 10.5815/ijigsp.2014.10.07

[6] What is meant by edge detection in image processing? - R4 DN

[7] Image Processing Algorithms: Canny Edge Detector | by Alex Williams | smucs | Medium

[8] Canny Edge Detection — Computer Vision | by Sofiane Sahir | Towards Data Science