Homework 2

Edge Detection

Name – Gowri Kurthkoti Sridhara Rao

Introduction

Edge detection is includes a variety of mathematical methods that aim at identifying edges, curves in a digital image at which the image brightness changes sharply or, more formally, has discontinuities. In this report three different types of edge detection methods are explored. The methods used are Sobel filtering, Marr-Hildreth edge detection, Canny edge detector.

1. Sobel Filtering

Sobel filtering is carried out in following steps:

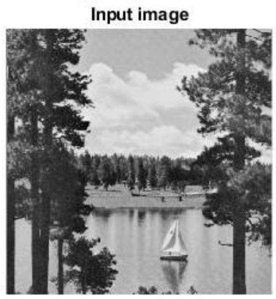
i. Image is first converted to gray scale image
ii. The image is then mean filtered using a mean filtering mask
iii. To find the edges in x direction, Gx mask is convolved with the mean filtered image
iv. To find the edges in y direction, Gy mask is convolved with the mean filtered image
v. Using the convolved images in both x and y direction, their magnitude is found
vi. In the magnitude image, the edges are highlighted

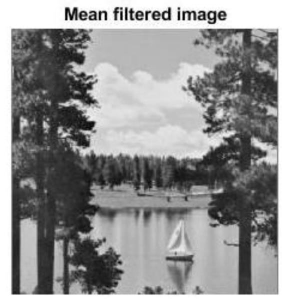
Code:

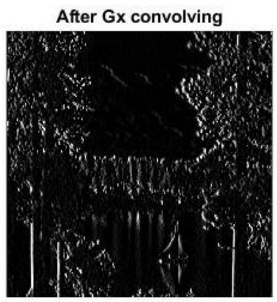
```
%% %% Colour to gray
img_rgb=img_1;
[row,col,d]=size(img_rgb);
if(d==3)
    img_gray=0.21*img_rgb(:,:,1)+0.72*img_rgb(:,:,2)+0.07*img_rgb(:,:,3);
else
   img_gray=img_rgb;
end
%% Mean filter and Sobel filter
mean_filter= [1 2 1;2 4 2;1 2 1]./16;
Gx = [1 0 -1;2 0 -2;1 0 -1];
Gy = [-1 -2 -1;0 0 0;1 2 1];

filt_img = convd2(img_gray,mean_filter,3);
x_img = convd2(filt_img,Gx,3);
y_img = convd2(filt_img,Gy,3);
out_img =(x_img.^2)+(y_img.^2);
out_img = (out_img).^(1/2);
```

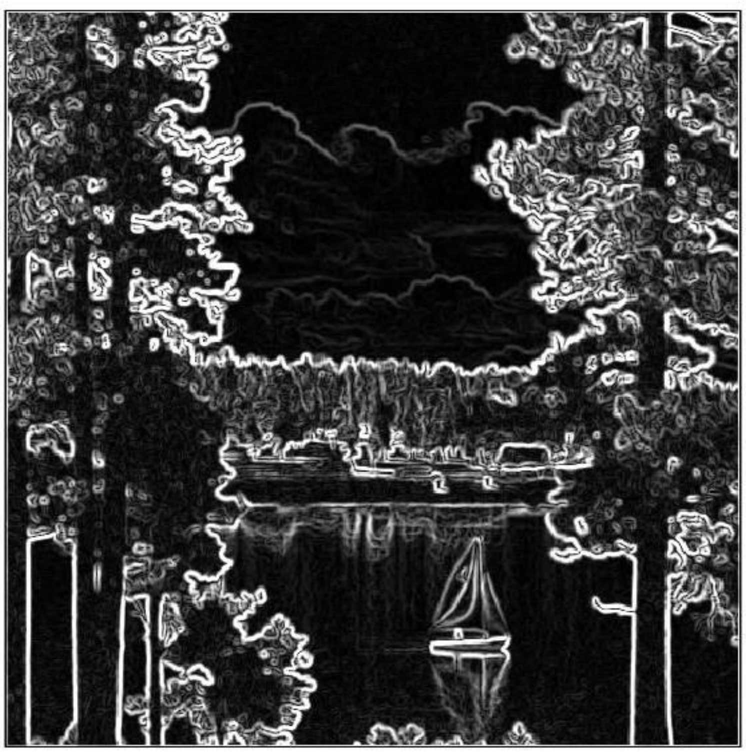
Results :

1. Image 1

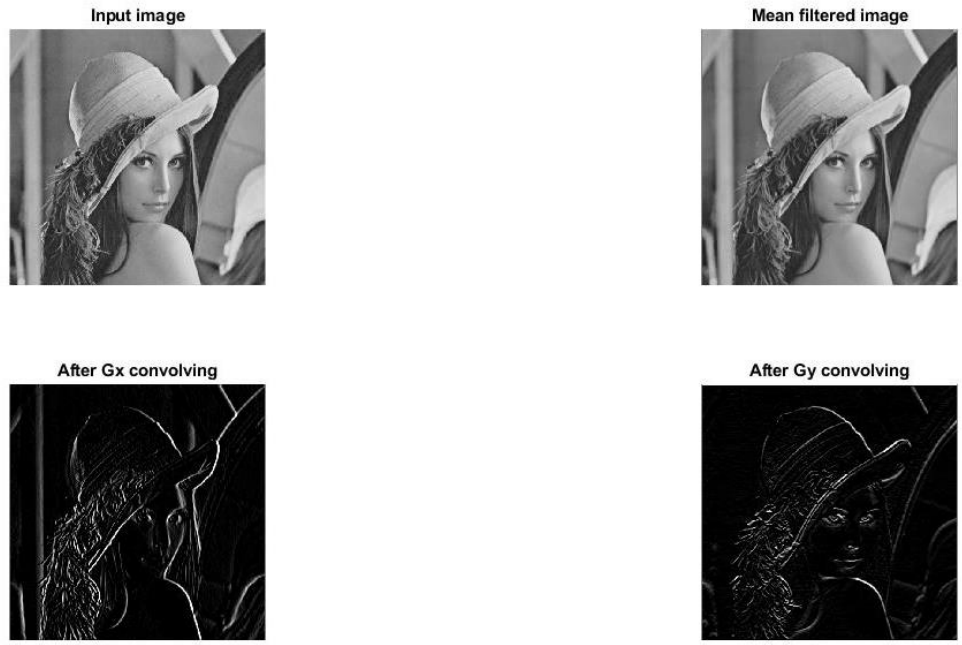








2. Image 2


Input image
Mean filtered image
After Gx convolving
After Gy convolving

3. Image 3

Input image

Mean filtered image

After Gx convolving

After Gy convolving

2. Marr – Hildreth edge detection

Marr – Hildreth edge detection detects edges using the following steps:

i. Input image is first converted to gray scale image
ii. Laplacian of Gaussian filter is created using the sigma value which determines the smoothening factor
iii. Image is convolved with the Laplacian of Gaussian (LoG) filter.
iv. In the resulting image zero crossings are found to determine the edges

Code:

LoG filter function

```
function [nLoG] = LoG(sigma)
%LoG This function takes input of sigma value and creates a normalized
Laplacian of
%Gaussian filter
%   Sigma value is used to determine the kernel size and for that kernel,
%   LoG filter equation is implemented. Normalization of this LoG filter
is
%   done and provided as output.
k=ceil(sigma)*5;
k = (k-1)/2; %Determining kernel size
[x,y] = meshgrid(-k:k,-k:k); %Initializing kernel

a = (x.^2+y.^2-2*sigma^2)/sigma^4;
b = exp(-( x.^2 + y.^2 )/(2*sigma^2));
b= b/sum(b(:));

LoG=a.*b; %LoG filter
nLoG=LoG-mean2(LoG); %Normalized LoG filter
end
```

Zero-crossing function

```
function [Edges] = zero_cross(LoG_img,slope)
%zero_cross Input: LoG filtered image, slope; Output: Detected edges
%   This function inputs the LoG filtered image and the slope of the image
%   and traverses throughout the image and finds all the zero crossings
%   that are greater than the slope and returns the edges.
[rows,cols]=size(LoG_img);
Edges = zeros(rows,cols);
for i=2:rows-1
    for j=2:cols
        if(LoG_img(i,j)>0)
            if(LoG_img(i,j+1)>=0 && LoG_img(i,j-1)<0 || (LoG_img(i,j+1)<0
&& LoG_img(i,j-1)>=0) && abs(LoG_img(i,j+1)-LoG_img(i,j-1))> slope)
                Edges(i,j)=LoG_img(i,j+1);
            elseif (LoG_img(i+1,j)>=0 && LoG_img(i-1,j)<0) ||
(LoG_img(i+1,j)<0 && LoG_img(i-1,j)>=0 && abs(LoG_img(i+1,j)- LoG_img(i-
1,j))>slope)
                Edges(i,j)=LoG_img(i,j+1);
            elseif (LoG_img(i+1,j+1)>=0 && LoG_img(i-1,j-1))<0 ||
(LoG_img(i+1,j+1)<0 && LoG_img(i-1,j-1)>=0 && abs(LoG_img(i+1,j+1)-
LoG_img(i-1,j-1))>slope)
```

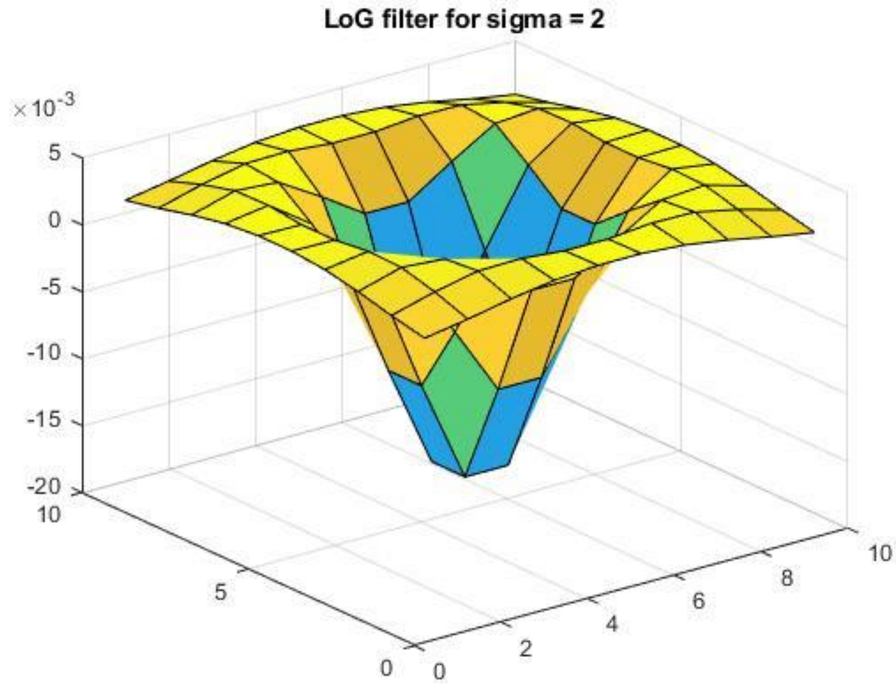
```
                Edges(i,j)=LoG_img(i,j+1);
            elseif (LoG_img(i-1,j+1)>=0 && LoG_img(i+1,j-1))<0 ||
(LoG_img(i-1,j+1)<0 && LoG_img(i+1,j-1)>=0 && abs(LoG_img(i+1,j-1)-
LoG_img(i-1,j-1))>slope)
                Edges(i,j)=LoG_img(i,j+1);
            end
        end
    end
end
end
```

Main code

```
% LoG filter definition
sigma=2;
log_filt=LoG(sigma);
%disp(log_filt);
figure(4);
surf(log_filt);title("LoG filter for sigma = 2");

%% Convolving
LoG_img = zeros(size(img_gray));
K=size(log_filt,1);
mid=round(K/2-1);
img_gray = double(img_gray);
for i=1:size(img_gray,1)-K-1
    for j=1:size(img_gray,2)-K-1
        LoG_img(i+mid,j+mid) = sum(sum(log_filt.*img_gray(i:i+(K-
1),j:j+(K-1))));
    end
end
figure, imshow(LoG_img); title("LoG filtered image");

%% Slope
slope = 0.5*mean(abs(LoG_img(:)));
%% Zero cross
detectedEdges = zero_cross(LoG_img,slope);
figure, imshow(detectedEdges); title("Edge detected imges");
```

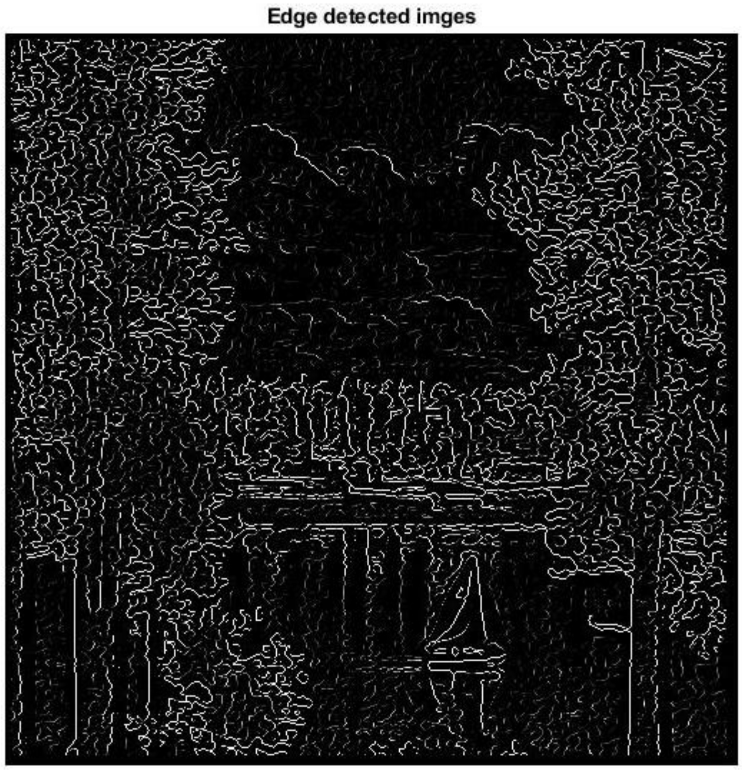
Result

The LoG filter used was formed using sigma = 2. The resulting LoG (Mexican hat filter) used is as follows:



1. **Image 1**
   - LoG filtered image

- Edge detected image



2. **Image 2**
   - LoG filtered image

- Edge detected Images

Edge detected imges

- **Image 3**
- LoG filtered Images

LoG filtered image

- Edge detected Images

3. Canny Edge detection

Canny Edge detection is another type of edge detection algorithm which follows different steps. Canny edge detection has thinner edges and more noise suppression. The steps used are as follows:

i. Image is first converted to a gray scale image
ii. Gray image is then convolved with a gauss filter with sigma = 1.4(for this project)
iii. Filtered image is then convolved with Sobel masks in both x and y directions
iv. Magnitude and angle of the Sobel filtered image is computed
v. Non-maximum suppression is carried out on magnitude and angle of the image
vi. Double threshold is applied on the non-max suppressed image
vii. Hysteresis is performed on the thresholded image to find the true edges and eliminate false edges.

Code:

Gauss Filter

```
function [gauss_matrix] = gauss(sigma)
%gauss Creats a gauss matrix using the sigma value
%   Sigma value is used to determine the kernel size of the gaussian
filter
%   A gauss matrix is created using the gauss equation.
k=ceil(sigma)*5;
k = (k-1)/2;
```

```
[x,y] = meshgrid(-k:k,-k:k);
normal = 1/(2*pi*sigma^2);
a = exp(-(x.^2+y.^2)/2*sigma^2);

gauss_matrix = a.*normal;
end
```

Non-max suppression

```
function [nms_img] = non_max(mag_img,theta)
%non_max : Non maximum suppression suppresses the non-edges based on
the
%angles
%   This function takes input of magnitude image and angles and
provides
%   non-maximum supressed image.
[rows,cols]=size(mag_img);
nms_img = zeros(rows,cols);

for i=2:rows-1
    for j=2:cols-1
        q=255;
        r=255;
        %angle 0
        if(0 <= theta(i,j) <22.5) || (157.5 <= theta(i,j) <= 180)
            q = mag_img(i,j+1);
            r = mag_img(i,j-1);
        %angle 45
        elseif(22.5<=theta(i,j)< 67.5)
            q=mag_img(i-1,j-1);
            r=mag_img(i+1,j+1);
        % angle 90
        elseif (67.5 <=theta(i,j) <112.5)
            q = mag_img(i+1,j);
            r = mag_img(i-1,j);
        %angle 135
        elseif (112.5 <= theta(i,j) <157.5)
            q = mag_img(i+1,j-1);
            r = mag_img(i-1,j+1);
        end

        if(mag_img(i,j) >= q) && (mag_img(i,j) >=r)
            nms_img(i,j) = mag_img(i,j);
        else
            nms_img(i,j) = 0;
        end
    end
end

end
```

Double Thresholding

```
function [res,strong,weak] =
threshold(img,lowThreshRatio,highThreshRatio)
%threshold Using low and high threshold ratio it outputs image with
strong
%and weak pixels only.
%   The function performs thresholding using two threshold and returns
an
%   image with strong and weak pixels and sets all the pixels below low
%   threshold to 0.
highThresh= max(max(img)) * highThreshRatio;
% disp(highThresh);
lowThresh = highThresh * lowThreshRatio;
% disp(lowThresh);

[rows,cols]=size(img);
res=zeros(rows,cols);
weak =25;
strong = 255;

for i=1:rows
    for j=1:cols
        if(img(i,j) >= highThresh)
            res(i,j)=strong;
        elseif(img(i,j) <lowThresh)
            res(i,j)=0;
        elseif(img(i,j)<=highThresh && img(i,j) >= lowThresh)
            res(i,j)=weak;
        end
    end
end
end
```

Hysteresis

```
function [out_img] = hysteresis(img,weak,strong)
%hysteresis : Performs histeresis on the image considering the weak and
%strong pixel levels
%   This function iterates through the image and finds if the weak
pixels
%   are connected to the strong pixels.
[rows,cols]=size(img);
out_img=img
for i=2:rows-1
    for j=2:cols-1
        if(img(i,j) == weak)
            if((img(i+1,j-1) == strong) || (img(i+1,j) == strong) ||
(img(i+1,j+1)==strong) || (img(i,j-1)==strong) || (img(i,j+1) ==
strong) || (img(i-1,j-1)==strong) || (img(i-1,j)==strong) || (img(i-
1,j+1)==strong))
                out_img(i,j)= strong;
            else
                out_img(i,j)=0;
            end
        end
```

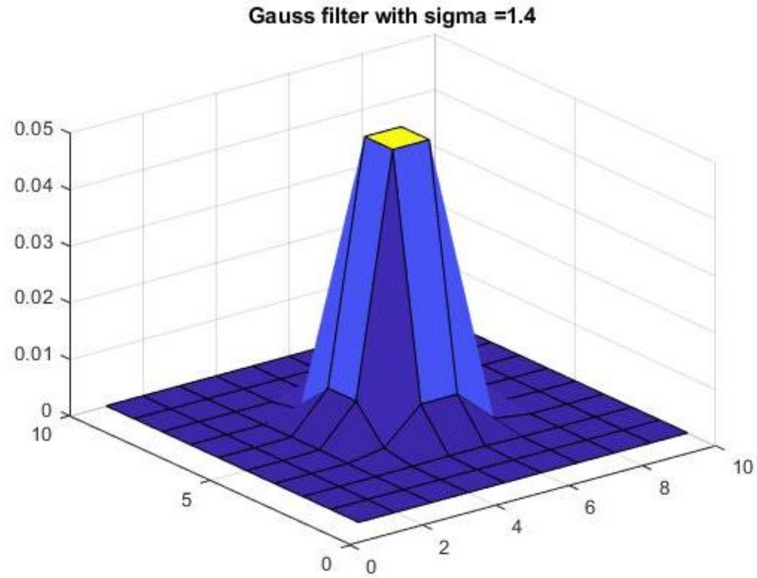
```
    end
end
end
```

Main

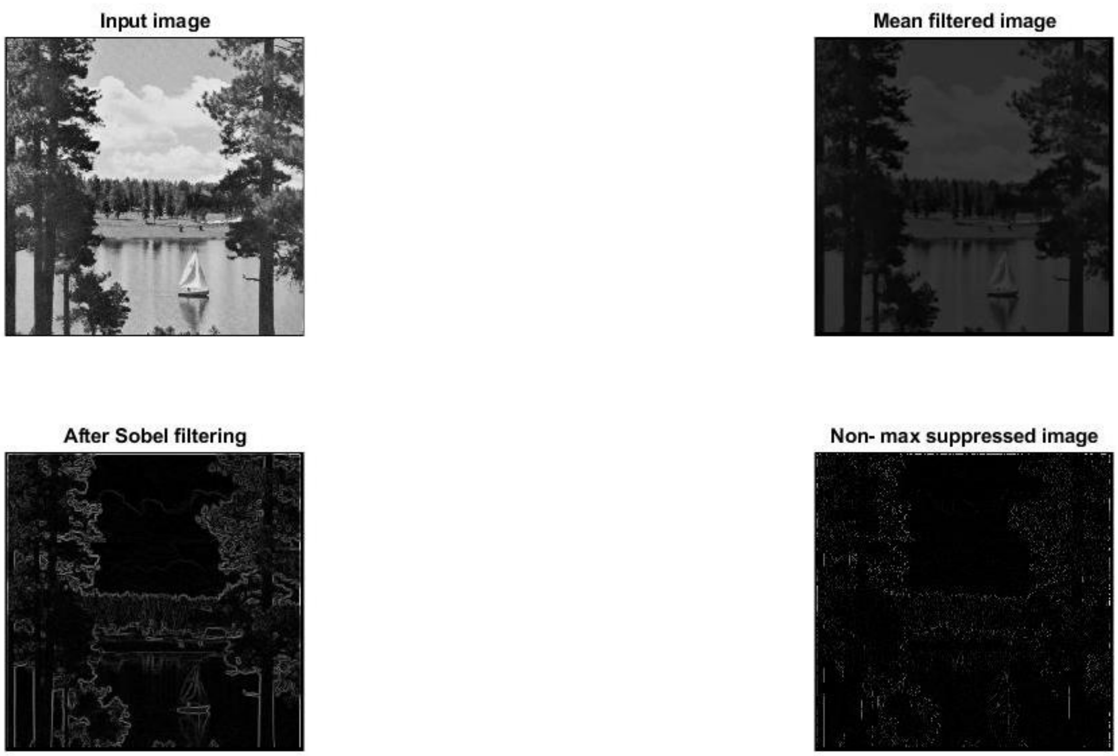
```
%% Gaussian filtering
sigma = 1.4;
gauss_matrix = gauss(sigma);
filt_img = convolve2(img_gray,gauss_matrix);

%% Sobel filtering

Gx = [1 0 -1;2 0 -2;1 0 -1];
Gy = [-1 -2 -1;0 0 0;1 2 1];

x_img = convd2(filt_img,Gx);

y_img = convd2(filt_img,Gy);

mag_img =(x_img.^2)+(y_img.^2);
mag_img = (mag_img).^(1/2);
mag_img = mag_img./255;

theta = atan2d(y_img,x_img);

%% Converting all angles to positive angles
for i=1:size(theta,1)
    for j=1:size(theta,2)
        if(theta(i,j)<0)
            theta(i,j) = theta(i,j)+180;
        end
    end
end

%% Non max supression
nms_img = non_max(mag_img,theta);
figure; imshow(nms_img);
%% Threshold
[thresh_img,strong,weak] = threshold(nms_img,0.03,0.09);
thresh1_img=uint8(thresh_img);
figure, imshow(thresh1_img);
%% Hysterisis
out_img = hysteresis(thresh_img,weak,strong);
out_img = out_img(3:size(out_img,1)-2,3:size(out_img,2)-2);
figure, imshow(out_img);
```

Results:

Gauss filter used for the smoothening is as follows:



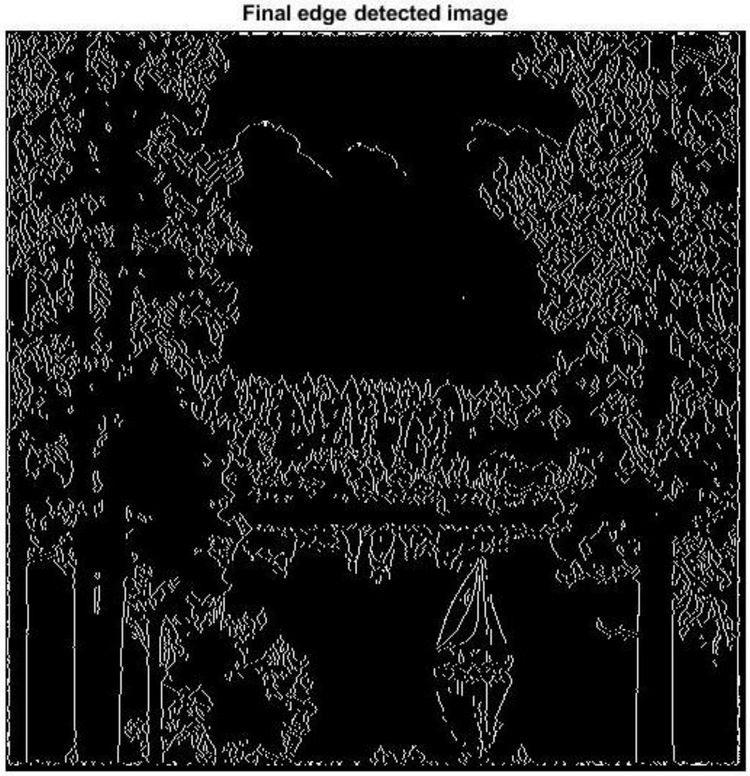
1. Image 1



Edge Detection

Double thresholded image

Final edge detected image



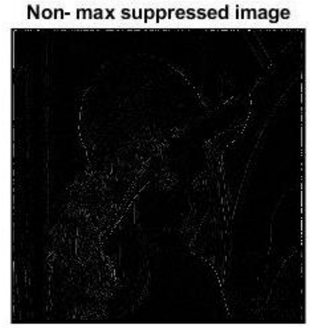
2. Image 2

Input image

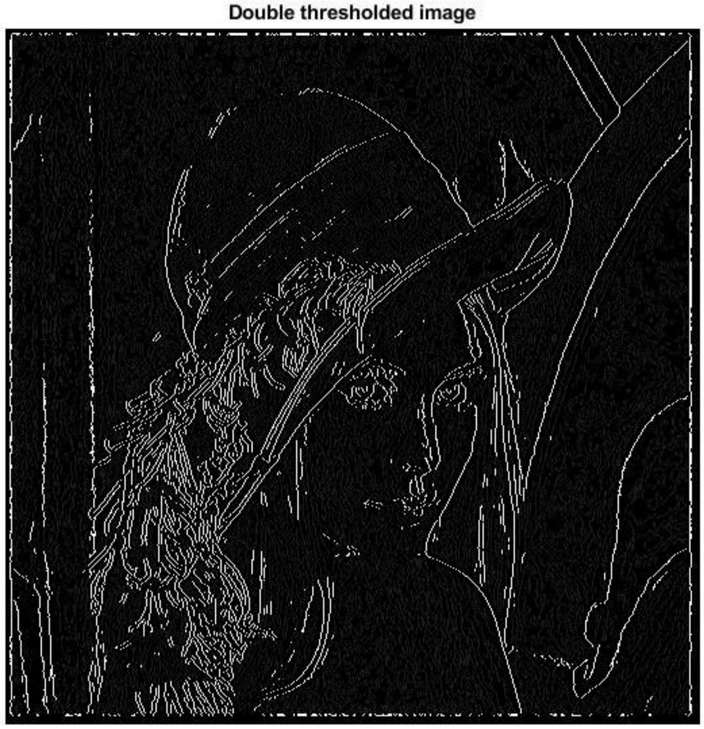
Mean filtered image

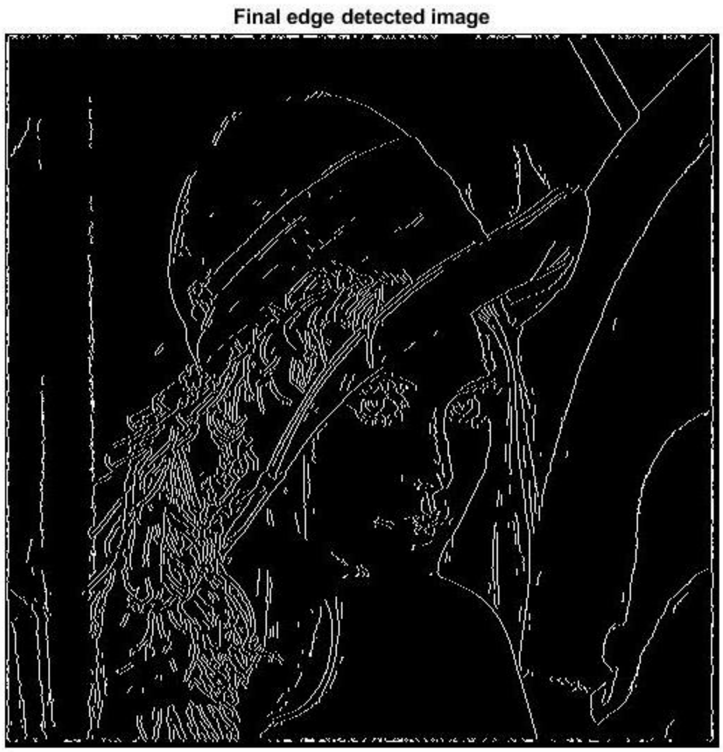
After Sobel filtering

Non- max suppressed image

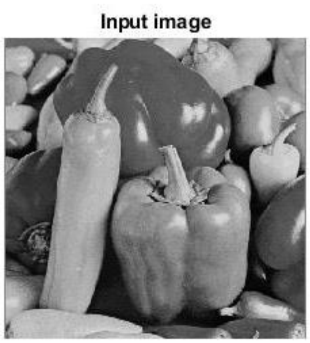


Double thresholded image




Final edge detected image

3. Image 3


Input image


Mean filtered image


After Sobel filtering

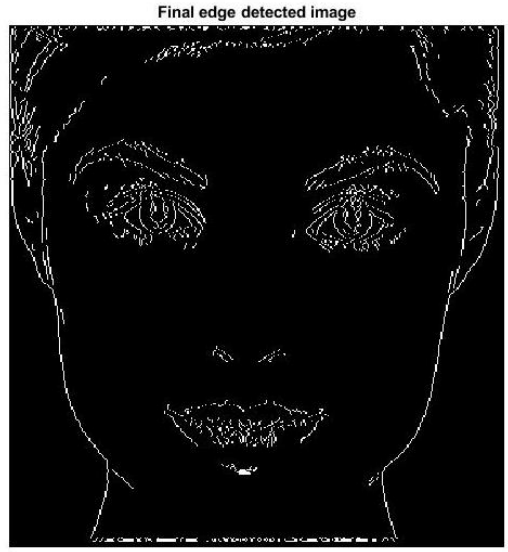

Non- max suppressed image

Double thresholded image

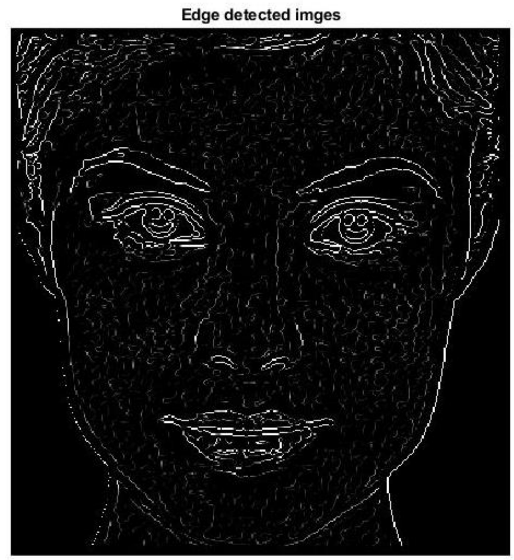
Final edge detected image

To check edge detection code scalability, the algorithm was tested on another test image. A .png image was chosen as a .jpg/.jpeg image did not provide satisfactory results.

Canny Edge detection



Marr-Hildreth edge detection



Conclusion:

This project presents implementation and results of different edge detection algorithms like Sobel, Marr-Hildreth and Canny edge detection.

References

1. https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123
2. https://purdue.brightspace.com/d2l/le/content/386398/viewContent/7401258/View