Course Project

Project title: Implementing Canny Edge Detection Using Matlab

Name of the Student: MD TAUHIDUL ISLAM
Student Id: 2016521460553

Project Supervisor: Professor 刘艳丽

Grading Results: _____

Comments: _____

_____

_____

Time for Submitting the Report: 2019/06/19

Software Engineering

Student: MD TAUHIDUL ISLAM **Supervisor:** Professor 刘艳丽

Abstract: Edges of an image are considered a type of crucial information that can be extracted by applying detectors with different methodology. This report presents a brief study of the fundamental concepts of the Canny edge detection operation and self-implemented canny edge algorithm and its test result.

Keywords: GRAYSCALE CONVERSION, GAUSSIAN BLUR, INTENSITY GRADIENTS, NON-MAXIMUM SUPPRESSION, DOUBLE THRESHOLDING & HYSTERESIS.

Experiment environment:

1. Matlab 2018a

Experiment content (step, code, process & test result):

INTRODUCTION

Canny edge detection is a image processing method used to detect edges in an image while suppressing noise. The main steps are as follows:

- **Step 1 - Choose test image & Grayscale Conversion**
- **Step 2- Smooth image with a Gaussian Blur**
- **Step 3- Determine the Intensity Gradients**
- **Step 5- Non-Maximum Suppression**
- **Step 6- Double Thresholding**
- **Step 7- Edge Tracking by Hysteresis**
- **Step 8- Cleaning Up**

ALGORITHM STEPS, PROCESS & IMPLEMENTATION:

❖ STEP 1 - Choose test image& Grayscale Conversion:

Here, we Choose ‘canny.bmp’ as our test image and we read the image in variable ‘im’.

In MATLAB the intensity values of the pixels are 8 bit and range from 0 to 255.

Implementation in MATLAB:

- **Read test image & Grayscale Conversion**

```
1. im=imread(‘canny.bmp’);
2. im = rgb2gray(im);
3. figure(1);imshow(im);title('Original Image');
4. figure(2); imshow(im);title('B/W Image');
```

- **Test result:**

B/W Image

❖ Step 2- Smooth image with a Gaussian:

➢ Process:

The first step of canny edge detection is to filter out any noise in the original image before trying to locate and detect any edges. The Gaussian filter is used to blur and remove unwanted detail and noise. By calculating a suitable 5 X 5 mask, the Gaussian smoothing can be performed using standard convolution method. A convolution mask is much smaller than the actual image. As a result, the mask is slid over the image, calculating every square of pixels at a time.

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2}\right); 1 \leq i, j \leq (2k+1)$$

Here is an example of a 5×5 Gaussian filter, used to create the adjacent image, with sigma = 1.4. (The asterisk denotes a convolution operation.)

$$\mathbf{B} = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * \mathbf{A}.$$

It is important to understand that the selection of the size of the Gaussian kernel will affect the performance of the detector. The larger the size is, the lower the detector's sensitivity to noise. Additionally, the localization error to detect the edge will slightly increase with the increase of the Gaussian filter kernel size. A 5×5 is a good size for most cases, but this will also vary depending on specific situations.
In my code I use sigma value as 1.4.

- **Code:**

```
1. im = double(imgaussfilt(im,sigma));
2. figure(3); imshow(im,[]);title('Gaussian Filter');
```

- **Test result:**

❖ Step 3- Determine the Intensity Gradients:

➢ Process:

After smoothing the image and eliminating the noise, the next step is to find the edge strength by taking the gradient of the image. The Sobel operator performs a 2-D spatial gradient measurement on an image.

The sobel filter uses two 3 x 3 kernels. One for changes in the horizontal direction, and one for changes in the vertical direction. The two kernels are convolved with the original image to calculate the approximations of the derivatives.

If we define Gx and Gy as two images that contain the horizontal and vertical derivative approximations respectively, the computations are:

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

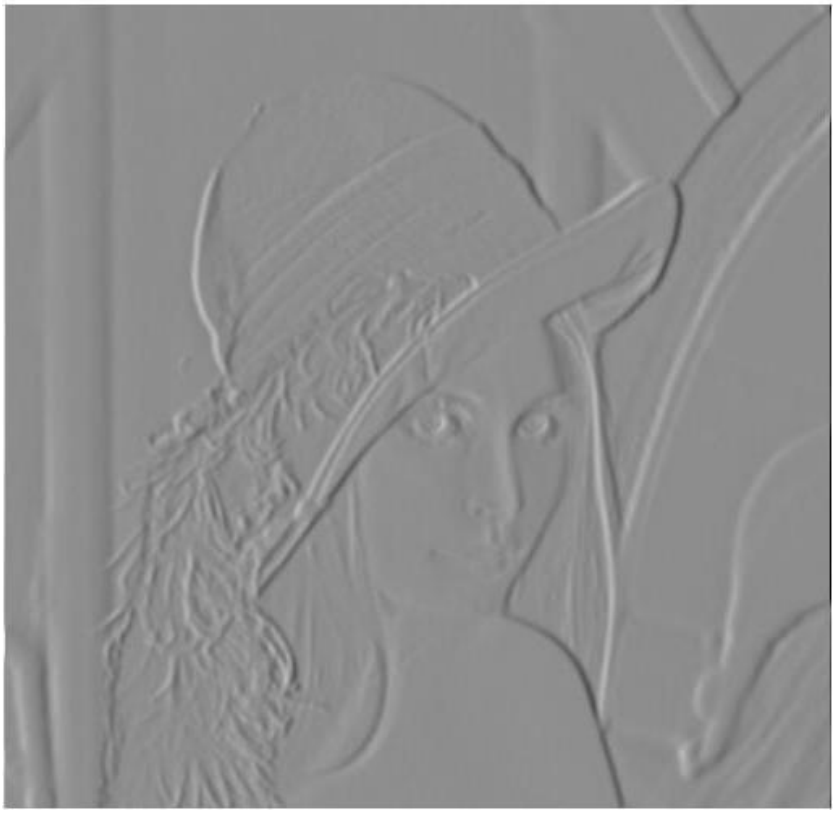
Gy

➢ Code:

```
1. Gx =[-1 0 1;-2 0 2;-1 0 1];
2. Gx =imfilter(im,Gx,sigma);

3. Gy =[1 2 1;0 0 0;-1 -2 -1];
4. Gy =imfilter(im,Gy,sigma);

5. figure(4); imshow(Gx, []);title('Gx Sobel Filter');

6. figure(5); imshow(Gy, []);title('Gy Sobel Filter');
```

- **Test Result:**

Gx Sobel Filter



Gy Sobel Filter

- **Then, calculate the magnitude and angle of the directional gradients:**

From the value for the first derivative in the horizontal direction (Gx) and the vertical direction (Gy). From this the edge gradient and direction can be determined:

$$|G| = \sqrt{{G_x}^2 + {G_y}^2}$$

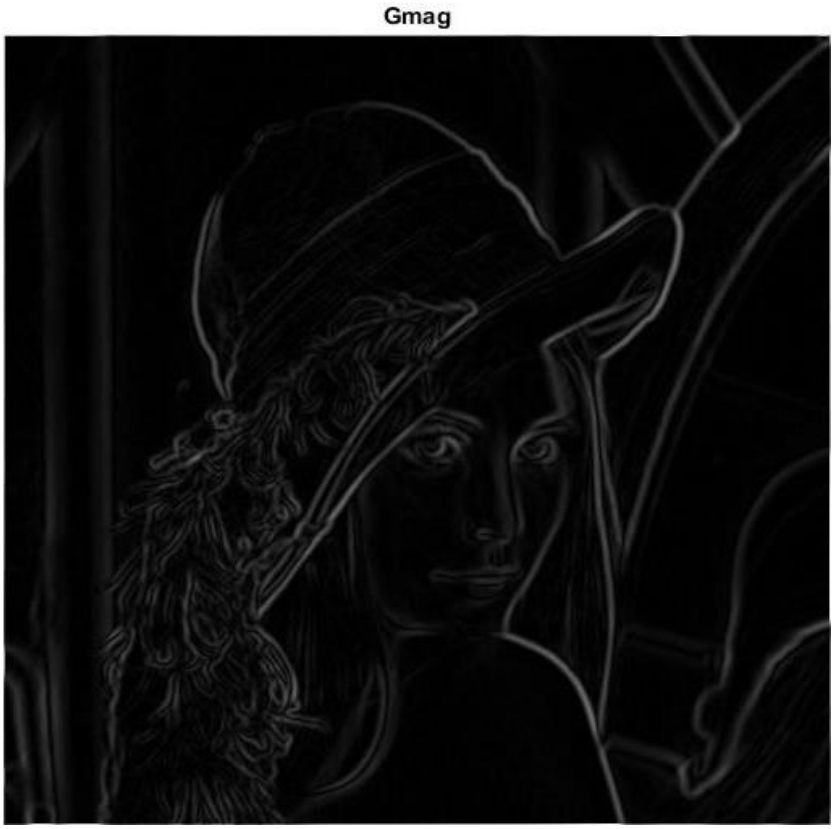
$$\angle G = arctan(G_y/G_x)$$

where G can be computed using the hypot function and arctan is the arctangent function with two arguments. The edge direction angle is rounded to one of four angles representing vertical, horizontal and the two diagonals (0°, 45°, 90° and 135°).

- **Code:**

```
1. Gmag = sqrt(Gx.^2 + Gy.^2);
2. angle = atan2(Gy,Gx)*180/pi;
3. figure(6); imshow(Gmag,[]);title('Gmag');
```

- ✓ **Test Result:**



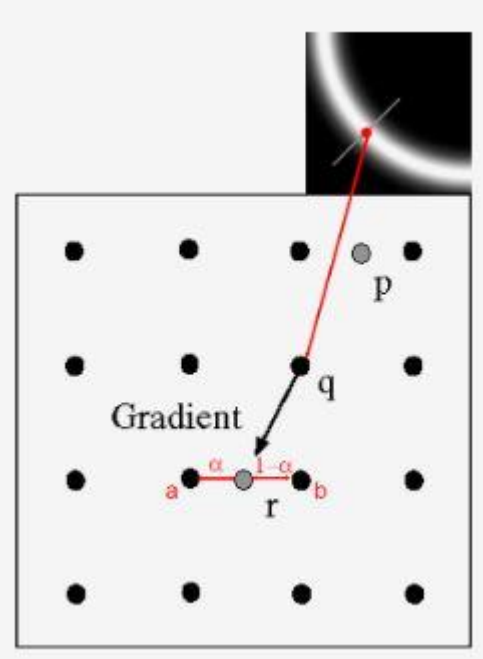
❖ Step 5- Non-Maximum Suppression:

➢ Process:

After the edge directions are known, non-maximum suppression is applied. Nonmaximum suppression is used to trace along the gradient in the edge direction and compare the value perpendicular to the gradient. Two perpendicular pixel values are compared with the value in the edge direction. If their value is lower than the pixel on the edge then they are suppressed i.e. their pixel value is changed to 0, else the higher pixel value is set as the edge and the other two suppressed with a pixel value of 0.

the gradient direction can be defined as follows:

1. [angle=0]: the edge point's intensity is greater than the intensities in the west and east directions,
2. [angle=0]: the edge point's intensity is greater than the intensities in the west and east directions,
3. [angle=135]: the edge's intensity is greater than the intensities in the north west and south east directions,
4. [angle=45]: the edge's intensity is greater than the intensities in the north east and south west directions.



Non maximum suppression can be achieved by interpolating the pixels for greater accuracy:

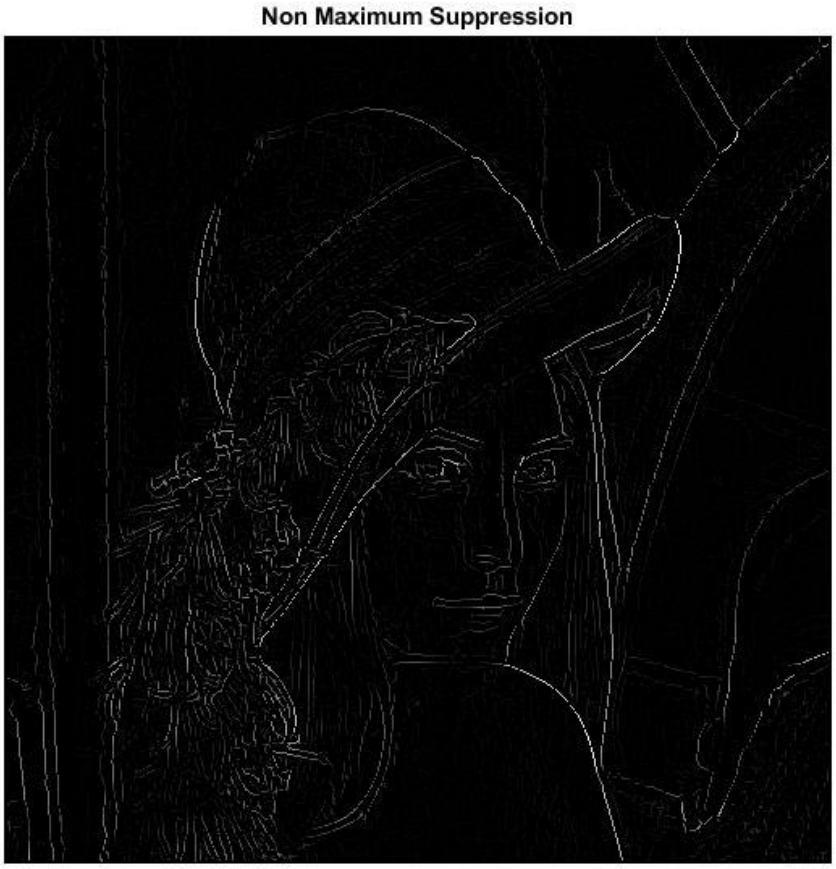
$r=\alpha b+(1-\alpha)a$

- **Code:**

```
% Perform non-maximum suppression using interpolation
    [h,w] = size(im);
    X=[-1,0,+1 ;-1,0,+1 ;-1,0,+1];
    Y=[-1,-1,-1 ;0,0,0 ;+1,+1,+1];
    output = zeros(h,w);
    x = [0 1];
    for i=2:h-1 % row
        for j=2:w-1 % col
            if (angle(i,j)>=0 && angle(i,j)<=45) || ...
                    (angle(i,j)<-135 && angle(i,j)>=-180)
                yBot = [Gmag(i,j+1) Gmag(i+1,j+1)];
                yTop = [Gmag(i,j-1) Gmag(i-1,j-1)];
                x_est = abs(Gy(i,j)/Gmag(i,j)); % y
                if (Gmag(i,j) >= ((yBot(2)-yBot(1))*x_est+yBot(1)) && ...
                    Gmag(i,j) >= ((yTop(2)-yTop(1))*x_est+yTop(1))) % interpolation
                    output(i,j)= Gmag(i,j);
                else
                    output(i,j)=0;
                end
            elseif (angle(i,j)>45 && angle(i,j)<=90) || ...
                    (angle(i,j)<-90 && angle(i,j)>=-135)
                yBot = [Gmag(i+1,j) Gmag(i+1,j+1)];
                yTop = [Gmag(i-1,j) Gmag(i-1,j-1)];
                x_est = abs(Gx(i,j)/Gmag(i,j));
                if (Gmag(i,j) >= ((yBot(2)-yBot(1))*x_est+yBot(1)) && ...
                    Gmag(i,j) >= ((yTop(2)-yTop(1))*x_est+yTop(1)))
                    output(i,j)= Gmag(i,j);
                else
                    output(i,j)=0;
                end
            elseif (angle(i,j)>90 && angle(i,j)<=135) || ...
                    (angle(i,j)<-45 && angle(i,j)>=-90)
                yBot = [Gmag(i+1,j) Gmag(i+1,j-1)];
                yTop = [Gmag(i-1,j) Gmag(i-1,j+1)];
                x_est = abs(Gx(i,j)/Gmag(i,j));
                if (Gmag(i,j) >= ((yBot(2)-yBot(1))*x_est+yBot(1)) && ...
                    Gmag(i,j) >= ((yTop(2)-yTop(1))*x_est+yTop(1)))
                    output(i,j)= Gmag(i,j);
                else
                    output(i,j)=0;
                end
```

```
            elseif (angle(i,j)>135 && angle(i,j)<=180) || ...
                    (angle(i,j)<0 && angle(i,j)>=-45)
                yBot = [Gmag(i,j-1) Gmag(i+1,j-1)];
                yTop = [Gmag(i,j+1) Gmag(i-1,j+1)];
                x_est = abs(Gx(i,j)/Gmag(i,j));
                if (Gmag(i,j) >= ((yBot(2)-yBot(1))*x_est+yBot(1)) && ...
                    Gmag(i,j) >= ((yTop(2)-yTop(1))*x_est+yTop(1)))
                    output(i,j)= Gmag(i,j);
                else
                    output(i,j)=0;
                end
            end
        end
    end

Gmag = output/max(max(output));
    figure(7); imshow(Gmag,[]);title('Non Maximum Suppression');
```

- ✓ **Test Result:**



❖ Step 6- Double Thresholding

➢ Process:

After application of non-maximum suppression, remaining edge pixels provide a more accurate representation of real edges in an image. However, some edge pixels remain that are caused by noise and color variation. In order to account for these spurious responses, it is essential to filter out edge pixels with a weak gradient value and preserve edge pixels with a high gradient value. This is accomplished by selecting high and low threshold values. 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 value is smaller than the low threshold value, it will be suppressed. The two threshold values are empirically determined and their definition will depend on the content of a given input image.

In my implementation, I normalized all the values such that they will only range from 0 to 1. Pixels with a high value are most likely to be edges. For example, you might choose the high threshold to be 0.7, this means that all pixels with a value larger than 0.7 will be a strong edge. You might also choose a low threshold of 0.3, this means that all pixels less than it is not an edge and you would set it to 0. The values in between 0.3 and 0.7 would be weak edges, in other words, we do not know if these are actual edges or not edges at all.

In my implementation I choose a threshold ratio instead of a specific value and multiple that by the max pixel value in the image. As for the low threshold, I chose a low threshold ratio and multiplied it by the high threshold value.

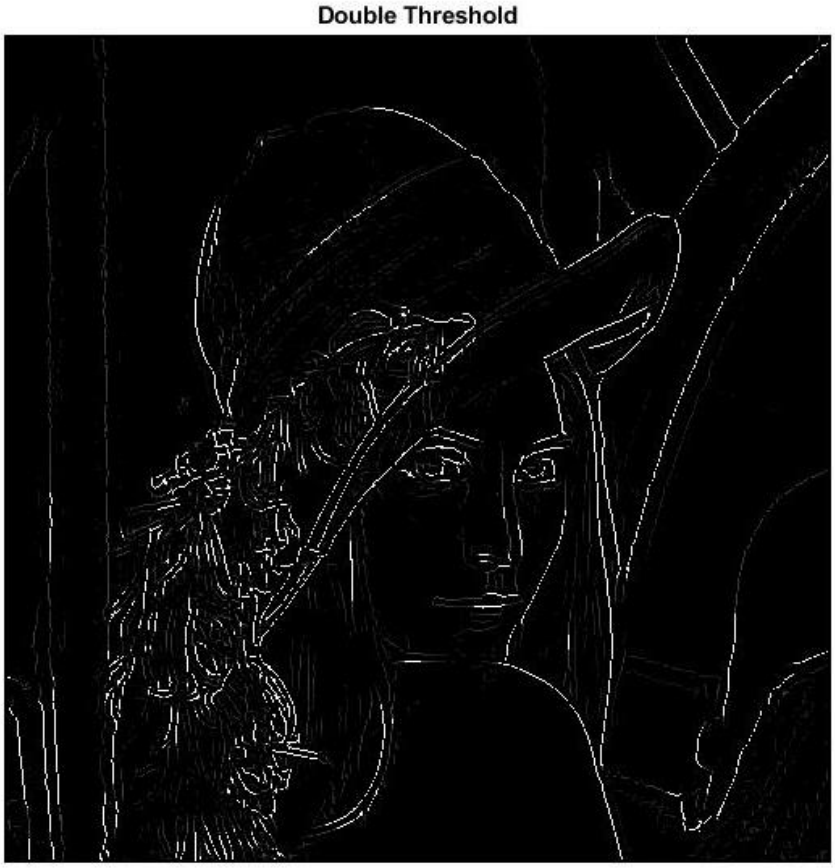
```
highThreshold = max(max(Gmag))*highThresholdRatio;
lowThreshold = highThreshold*lowThresholdRatio;
```

➢ Code:

```
% Perform double thresholding
    highThreshold = max(max(Gmag))*highThresholdRatio;
    lowThreshold = highThreshold*lowThresholdRatio;
    strongEdgesRow = zeros(1,h*w); % Keep track of the strong edge row index
    strongEdgesCol = zeros(1,h*w); % Keep track of the strong edge col index
    weakEdgesRow = zeros(1,h*w);    % Keep track of the weak edge row index
    weakEdgesCol = zeros(1,h*w);    % Keep track of the weak edge col index
    strongIndex = 1;
    weakIndex = 1;
    for i=2:h-1 % row
        for j=2:w-1 % col
            if Gmag(i,j) > highThreshold        % Strong edge
```

```
                Gmag(i,j) = 1;
                strongEdgesRow(strongIndex) = i;
                strongEdgesCol(strongIndex) = j;
                strongIndex = strongIndex + 1;
            elseif Gmag(i,j) < lowThreshold % No edge
                Gmag(i,j) = 0;
            else                                    % Weak edge
                weakEdgesRow(weakIndex) = i;
                weakEdgesCol(weakIndex) = j;
                weakIndex = weakIndex + 1;
            end
        end
    end
    figure(8); imshow(Gmag,[]);title('Double Threshold');
```

- ✓ **Test Result:**



❖ Step 7- Edge Tracking by Hysteresis

➢ Process:

So far, the strong edge pixels should certainly be involved in the final edge image, as they are extracted from the true edges in the image. However, there will be some debate on the weak edge pixels, as these pixels can either be extracted from the true edge, or the noise/color variations. To achieve an accurate result, the weak edges caused by the latter reasons should be removed. Usually a weak edge pixel caused from true edges will be connected to a strong edge pixel while noise responses are unconnected. To track the edge connection, blob analysis is applied by looking at a weak edge pixel and its 8-connected neighborhood pixels. As long as there is one strong edge pixel that is involved in the blob, that weak edge point can be identified as one that should be preserved.
To speed up this process, my specified a recursive FindConnectedWeaEdges function to keeps track of the weak and strong edges that way I can recursively iterate through the strong edges and see if there are connected weak edges instead of having to iterate through every pixel in the image. Here,

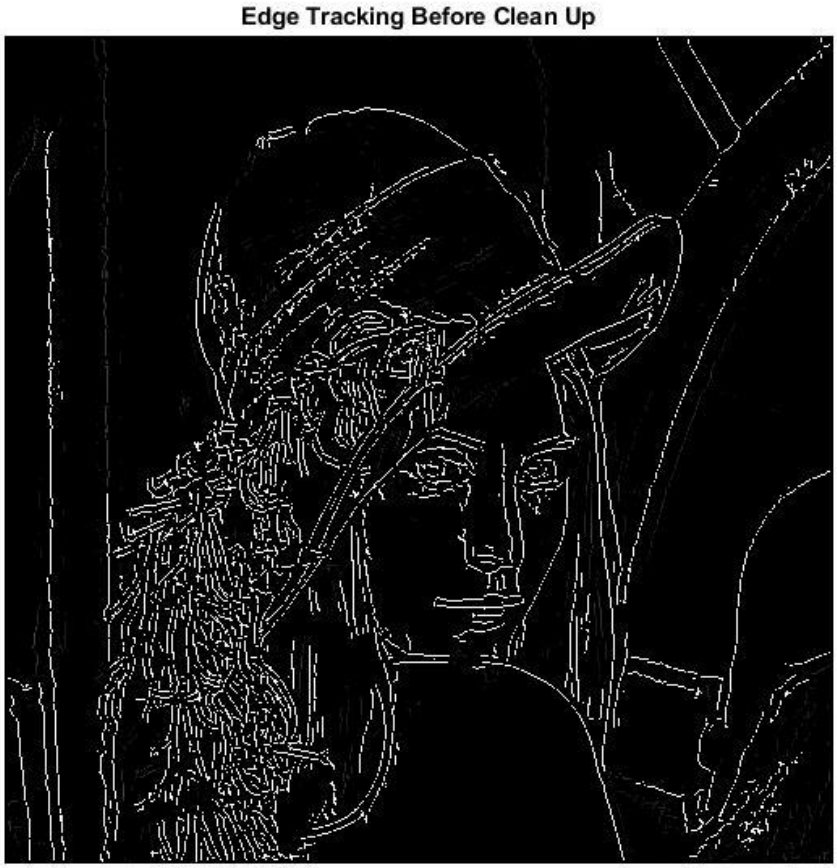
```
% Find weak edges that are connected to strong edges and set them to 1
function[Gmag] = FindConnectedWeakEdges(Gmag, row, col)
    for i = -3:1:3
        for j = -3:1:3
            if (row+i > 0) && (col+j > 0) && (row+i < size(Gmag,1)) && ...
                    (col+j < size(Gmag,2)) % Make sure we are not out of bounds
                if (Gmag(row+i,col+j) > 0) && (Gmag(row+i,col+j) < 1)
                    Gmag(row+i,col+j) = 1;
                    Gmag = FindConnectedWeakEdges(Gmag, row+i, col+j);
                end
            end
        end
    end
end
```

I used this function to identify the weak edges through the Column and row of the image.

➢ Code:

```
% Perform edge tracking by hysteresis
set(0,'RecursionLimit',10000)
for i=1:strongIndex-1
    % Find the weak edges that are connected to strong edges and set
    % them to 1
    Gmag = FindConnectedWeakEdges(Gmag, strongEdgesRow(i),...
        strongEdgesCol(i));
end
figure(9); imshow(Gmag,[]);title('Edge Tracking Before Clean Up');
```

✓ Test Result:



❖ Step 8- Cleaning Up

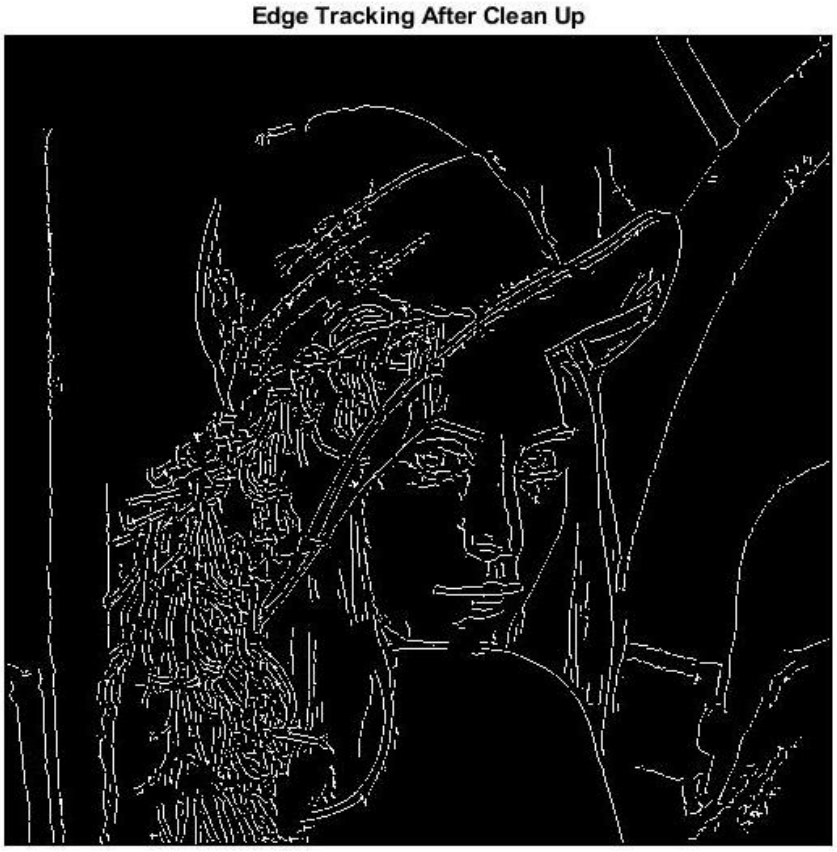
➢ Process:

Finally, I filter all other weak edges that are not basically edges and is noise instead. I set them to 0.

➢ Code:

```
% Remove the remaining weak edges that are not actually edges
    % and is noise instead
    for i=1:weakIndex-1
        if Gmag(weakEdgesRow(i),weakEdgesCol(i)) ~= 1
            Gmag(weakEdgesRow(i),weakEdgesCol(i)) = 0;
        end
    end
    figure(10); imshow(Gmag,[]);title('Edge Tracking After Clean Up');
```

✓ **Test Result:**

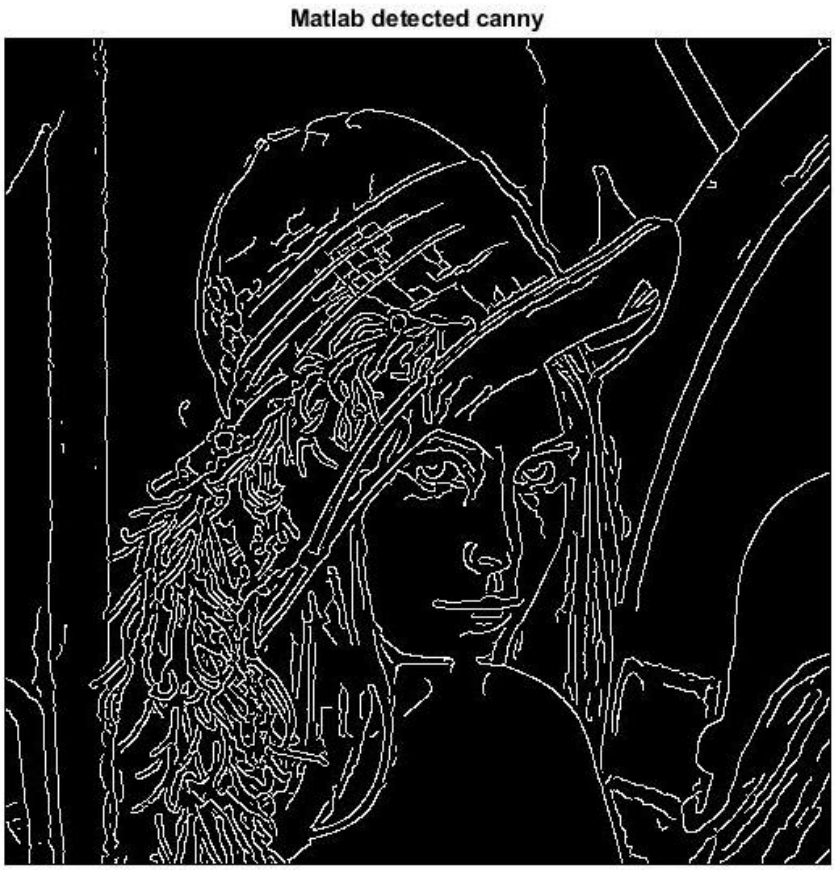


❖ Compare with Matlab detected Test Result:

➢ Edge function code:

```
1. im=imread(‘'Test_Photos\canny.bmp');
2. im=rgb2gray(im);
3. canny=edge(im,’canny’);
4. figure();imshow(canny,[]);title('Matlab detected canny');
```

✓ **Test Result:**



❖ Comparing Table

Test Parameters	Malab Edge Detection Function(edge(image,'canny')	My implementation	Comment
1. Noise reduction	It reduces the noise efficiently.	Noise reduction rate good enough	Comparing two, I need to improve in noise reduction with different filter mask and sigma ratio
2. Clarity in edge tracking	Edge tracking is very clear and visible.	Edge tracking is also clear but a bit noisy and thick.	Need to improve in thresholding with different parameter and also need to apply different hysteresis ratio to detect weak edges that make the edge thick

Comparing with above two figures, I can say that my implementation works fair enough to track edges following canny edge algorithm.
I hope to improve my result by further better implementation techniques.

❖ Conclusion:

Edge detection applying the Canny edge detection algorithm was discussed. The various steps in performing Canny's algorithm i.e. gaussian, sobel, non-maximum suppression and hysteresis were explained in detail.

References:
1. Irwin Sobel, 2014, History and Definition of the Sobel Operator
2. https://reference.wolfram.com/language/ref/GaussianFilter.html
3. https://www.encyclopedia.com/science-and-technology/mathematics/mathematics/normal-distribution#3
4. Linda G. Shapiro and George C. Stockman, Computer Vision, Upper Saddle River: Prentice–Hall, 2001
5. CSE576: Computer Vision: Chapter 7 (PDF). University of Washington. 2000. pp. 9–10.
6. R. Deriche, Using Canny's criteria to derive a recursively implemented optimal edge detector, Int. J. Computer Vision, Vol. 1, pp. 167–187, April 1987.
7. Zhou, P., Ye, W., & Wang, Q. (2011). An Improved Canny Algorithm for Edge Detection. Journal of Computational Information Systems, 7(5), 1516-1523
8. https://www.mathworks.com/discovery/edge-detection.html

ACKNOWLEDGMENTS

At last I would like to thank my friends and course instructor. Their suggestion and direction helped me to complete this project.
I would like to thank the reviewer of this report. I hope that it will be helpful for them. I Will wait for their valuable suggestion. if anyone have any suggestion and question, you can send me an email. That is all!!
Email: tauhidscu16@gmail.com

MatlabR2018A is used to do this project.