# Image Processing Programming Assignment #2

In this assignment, our goal is to implement some image segmentation techniques. There are two tasks in total, which are, trying various gradient filters on the images and choose one between the implementation of the Canny edge detector, Hough transforms, or watershed algorithm for segmentation. The code is done with Matlab programming language and is released on https://github.com/KennethYapWL/NCTU-IP-2020/tree/main/Project%202 , and the references are listed on the last page of this report.

Those methods implemented are:

1. Gradient Filters
   → Prewitt filter
   → Sobel filter
   → LoG filter
2. Canny Edge Detection
3. Thresholding
4. Conversion between color models (RGB, YCbCr, HSI, HSV)
5. And some preprocessing techniques used in the previous project.

This report is organized as follow:

→ Section 1 (Experimental Results): This part will show all the results of six different images after doing edge detection.
→ Section 2 (Observations and Discussions): This part will give brief descriptions of the other experiment I have tried, also the further observations of section 1 will be discussed in this section.
→ Section 3 (Code Analysis): This part will show all the code used in this assignment.
→ Section 4 (References): This part will list all the articles I have read.

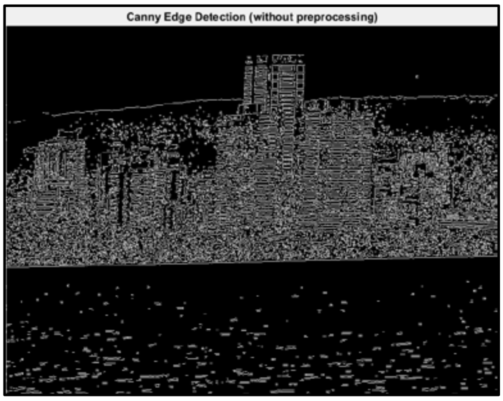
## Section 1: Experimental Results

This section is focused on showing the results of the edge detection applied to six different images. 4 of the images are from the previous project, and the other two images are from Google.

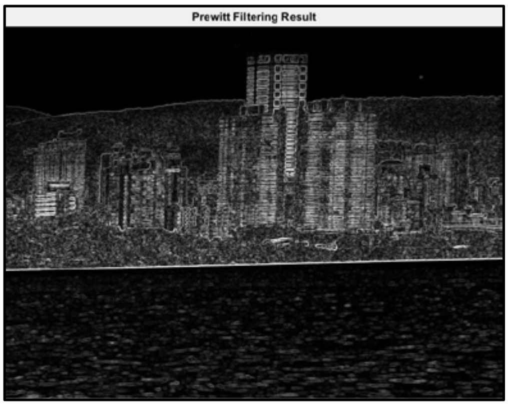
Note that the value of the threshold in Prewitt, Sobel, and LoG is between 0 to 1. For example, 0.5 means threshold value of 255 * 0.5 = 127.5.

A) The first image (p1im1.png)

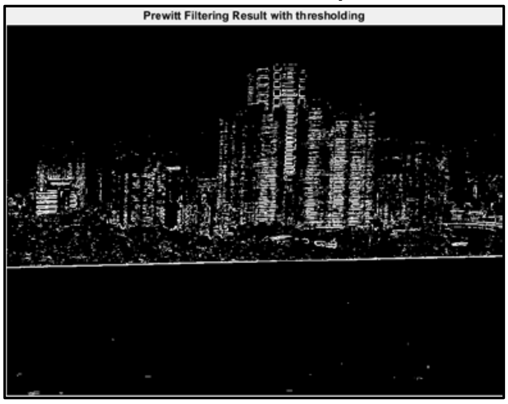

**Original Image**



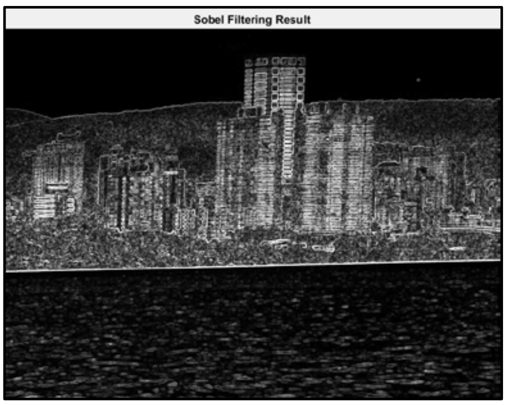

**Canny (sigma=0.15, low threshold=50, high threshold=90)**



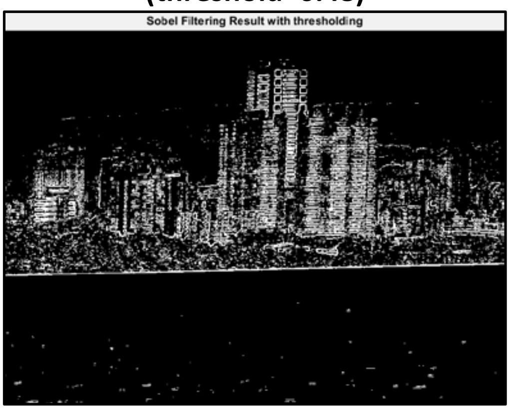

**Prewitt**



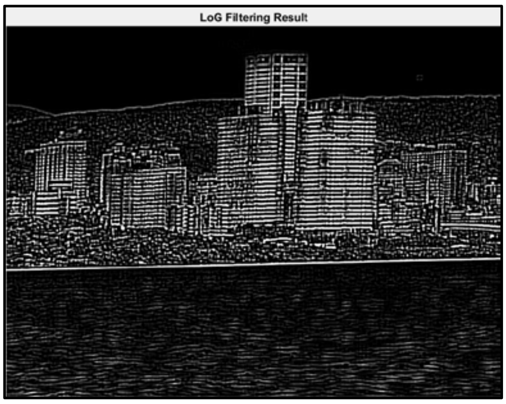

**Prewitt with thresholding (threshold=0.48)**

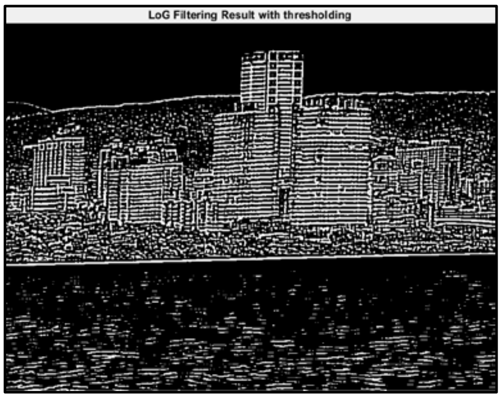



**Sobel**




**Sobel with thresholding (threshold=0.49)**



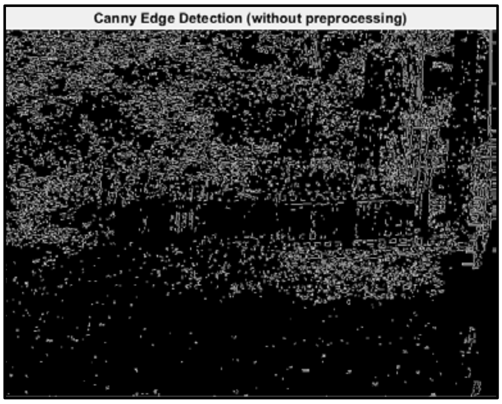

**LoG (sigma=2.75)** **LoG with thresholding (0.25)**

We could see that the result of Sobel and Prewitt are quite similar to each other, but Sobel is letting more pixels as edge points. LoG is more able to catch the edge information on the 'ocean' part and the 'mountain' part in the image.

The result of Canny edge detection is quite different to the result of those gradient filters. Notice that some of the edge point of the Sobel, Prewitt and LoG doesn't connect to another edge point, while each Canny edge point seems to have connected to form edge lines.

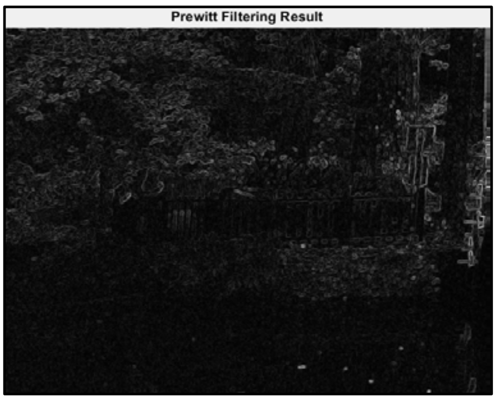
More details can be found on the file "Sol_p1im1.m" in the provided GitHub.

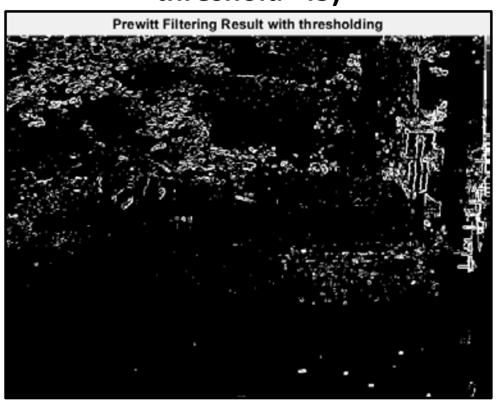
B) The second image (p1im2.png)

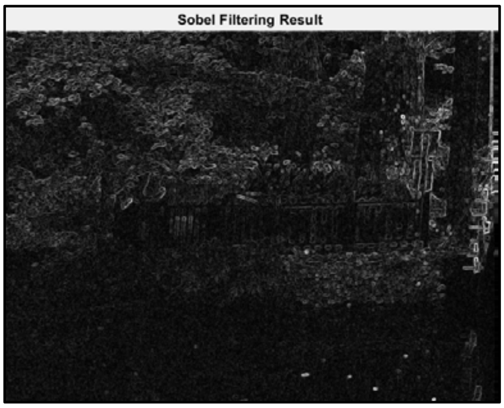

**Original Image**

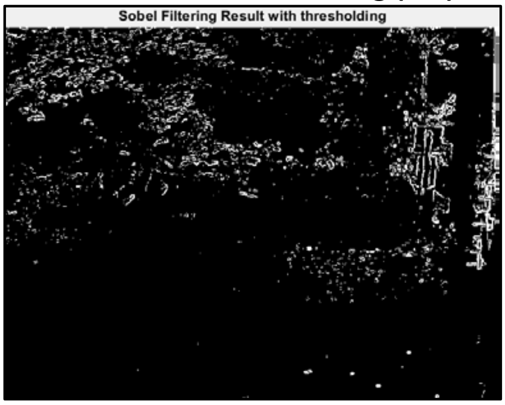
**Canny (sigma=0.08, low threshold=30, high threshold=45)**



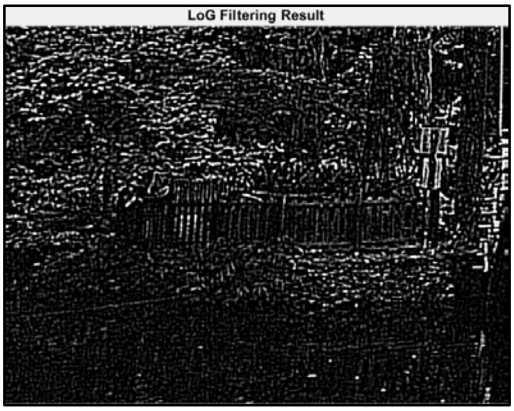



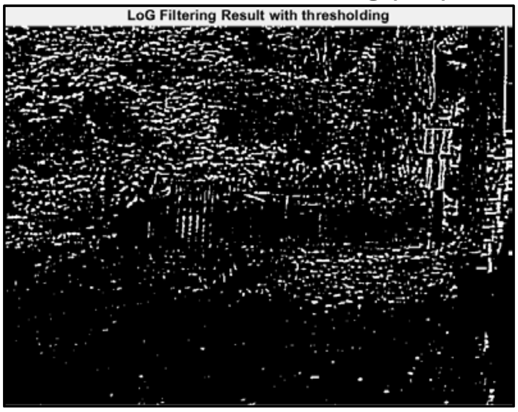
**Prewitt** | **Prewitt with thresholding (0.2)**




**Sobel** | **Sobel with thresholding (0.3)**

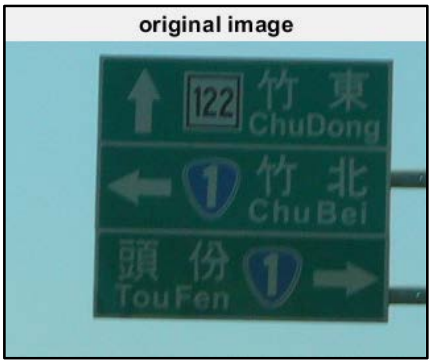


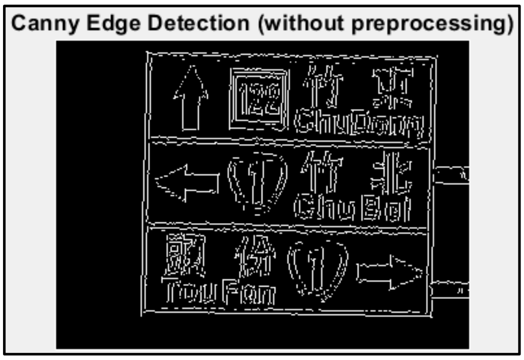

**LoG(sigma=2.3)** | **LoG with thresholding (0.4)**

More details can be found on the file “Sol_p1im2.m” in the provided GitHub.

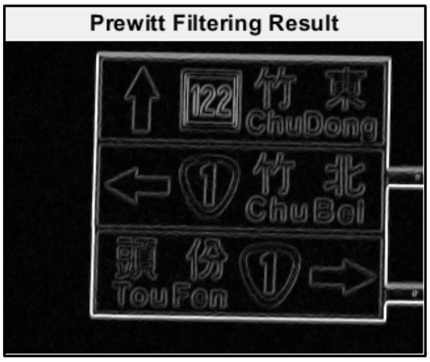
C) The third image (p1im4.png)

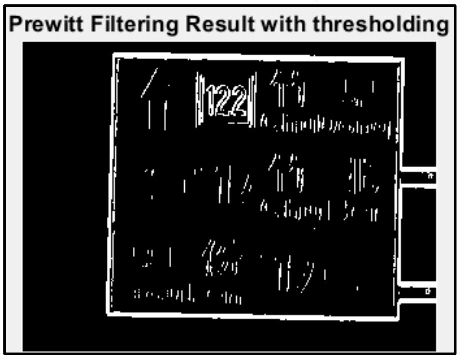



**Original Image**

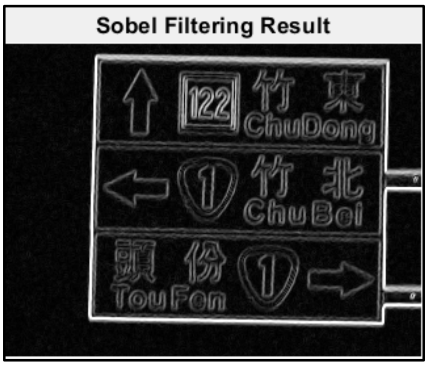


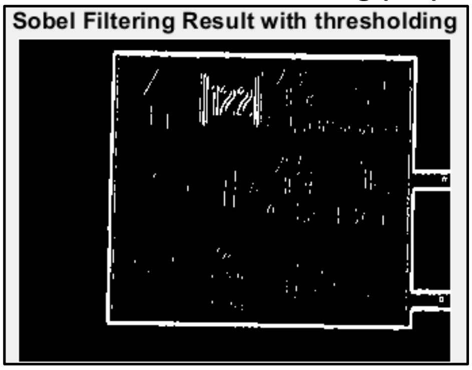

**Canny (sigma=0.00001, low threshold=20, high threshold=50)**



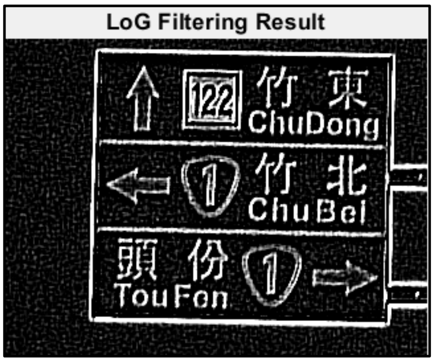



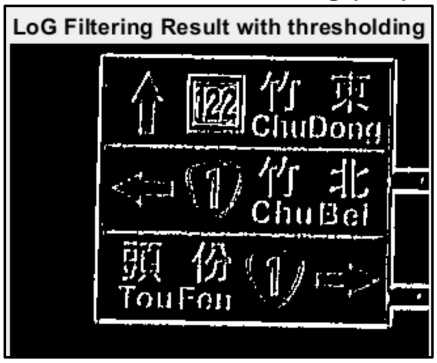
**Prewitt** **Prewitt with thresholding (0.3)**




**Sobel** **Sobel with thresholding (0.5)**

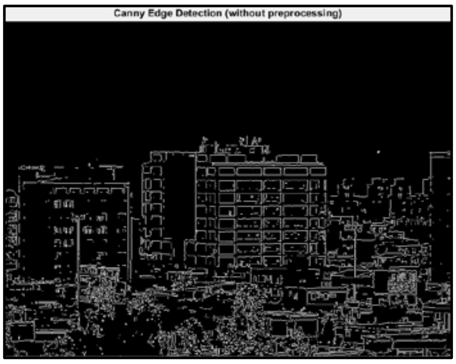



**LoG(sigma=1.71)** **LoG with thresholding (0.75)**

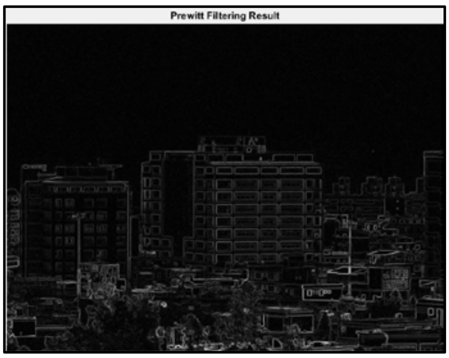
More details can be found on the file “Sol_p1im4.m” in the provided GitHub.

D) The forth image (p1im5.png)

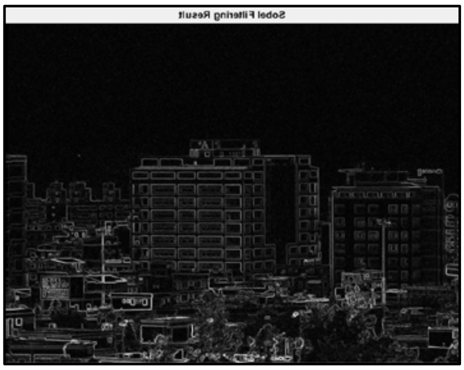

**Original Image**

**Canny**
**(sigma=0.0005, low threshold=20, high threshold=50)**



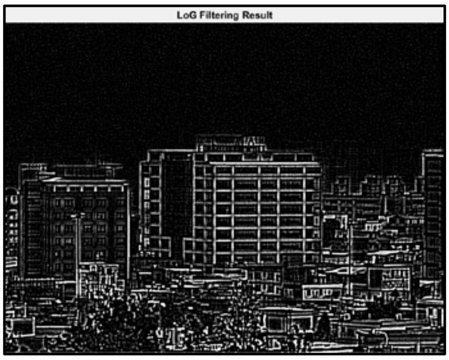


**Prewitt** **Prewitt with thresholding (0.2)**



**Sobel**


**Sobel with thresholding (0.2)**



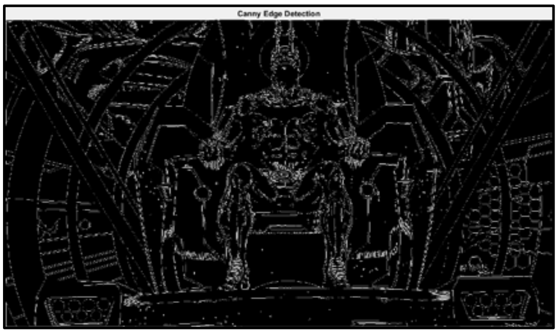

**LoG(sigma=2.2)**


**LoG with thresholding (0.35)**

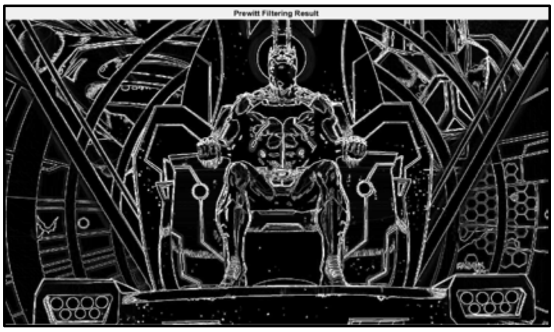
More details can be found on the file “Sol_p1im5.m” in the provided GitHub.

E) The fifth image [From Google] (batman_god_of_knowledge.png)

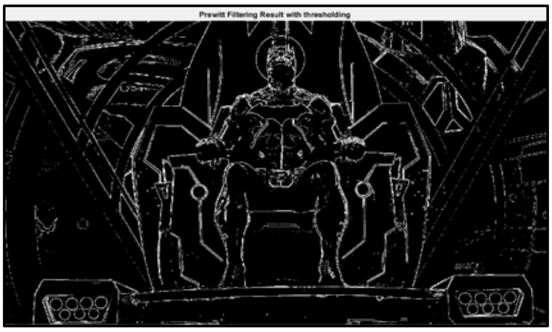

**Original Image**



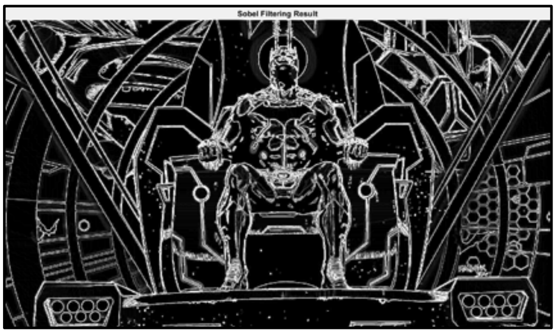

**Canny (sigma=0.8, low threshold=40, high threshold=80)**



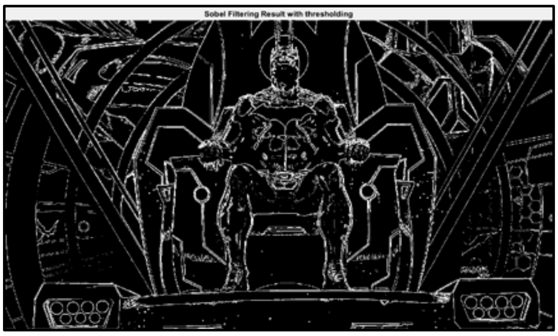

**Prewitt**



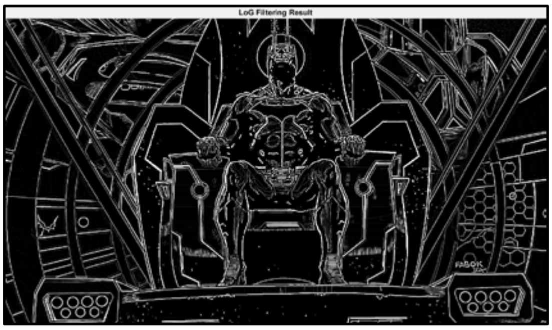

**Prewitt with thresholding (0.9)**



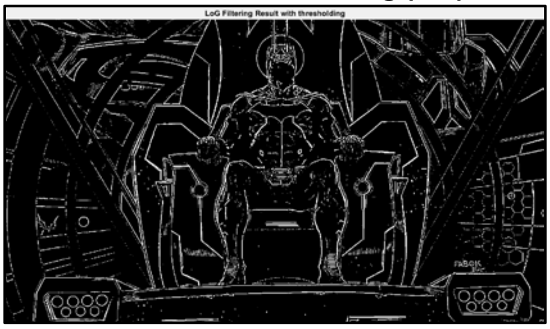

**Sobel**




**Sobel with thresholding (0.9)**




**LoG(sigma=3)**

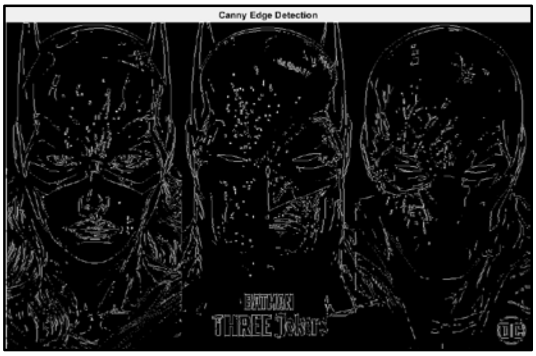



**LoG with thresholding (0.8)**

More details can be found on the file “Sol_ batman_god_of_knowledge.m” in the provided GitHub.

F) The sixth image [From Google] (batman_three_jokers.png)

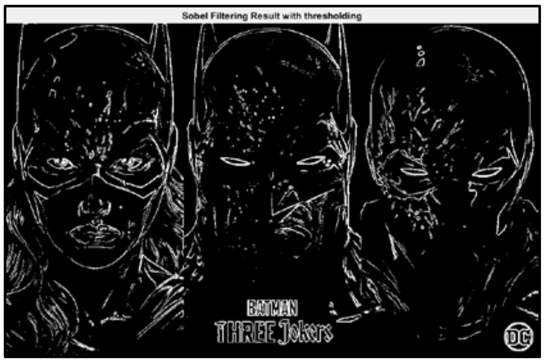
**Original Image**




**Canny**
**(sigma=0.9, low threshold=40, high threshold=90)**


**Prewitt**


**Prewitt with thresholding (0.8)**


**Sobel** **Sobel with thresholding (0.9)**



**LoG(sigma=3)** **LoG with thresholding (0.6)**

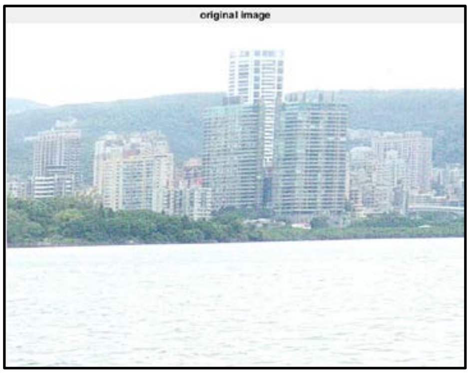
More details can be found on the file "Sol_ batman_three_jokers.m" in the provided GitHub.

## Section 2: Observations and Discussions

1. *How pre-processing affects edge detection ???*

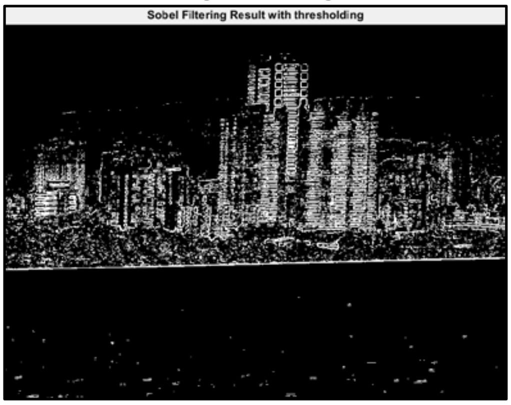
   As assignment asked, I have done some experiments about how pre-processing could affects edge detection. This section will only show one of them, like previous project, the image (p1im1.png) is first pre-processed by the following steps:

   1) Piecewise Linear Stretching, with settings, r1=120, s1=70, r2=250, s2=190. Figure 1.A.3 shows the transform function of these settings.
   2) Adaptive Local Noise Reduction filtering, with a filter size of 3.
   3) Color Correction by adjusting the HSV components, with H and S remain unchanged, and V subtract 40 (V = V - 40).

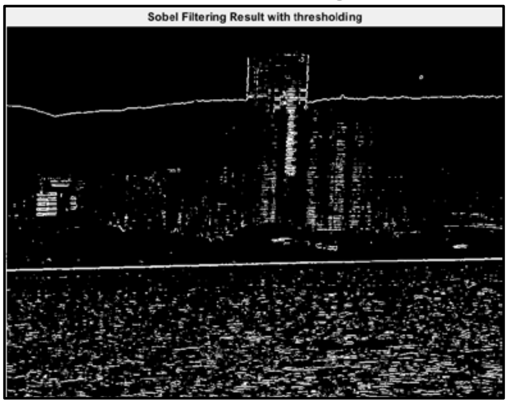



**Original Image**

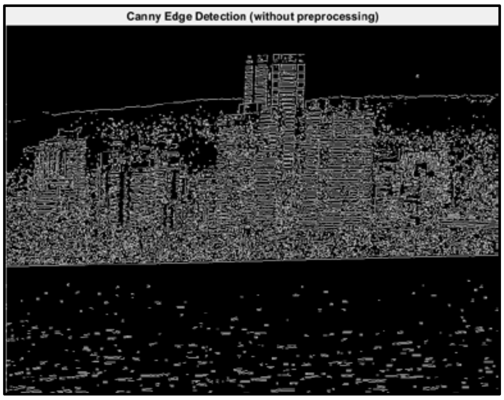

**Enhanced Image**



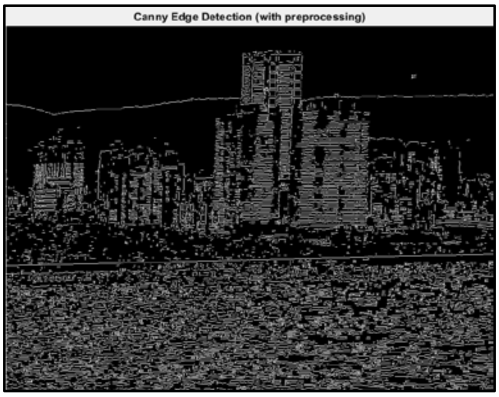

**Sobel on Original**



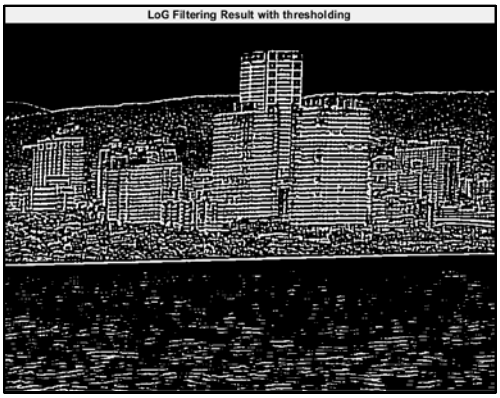

**Sobel on Enhanced**

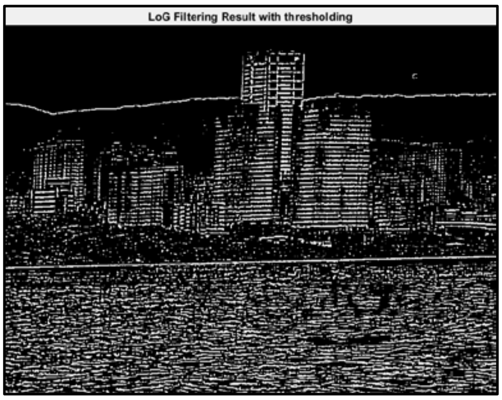



**Canny on Original**




**Canny on Enhanced**



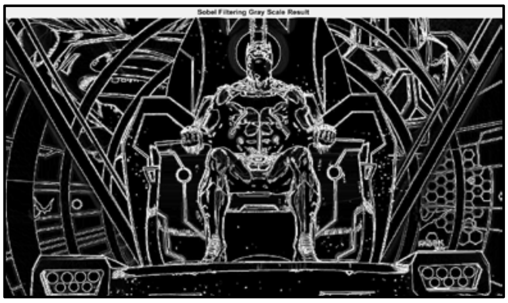

**LoG on Original** **LoG on Enhanced**

As the results shown, the experiment found that contrast adjustment or color correction usually will change the focus point of edge detection process. For example, LoG on original image had more edge candidate points in “mountain” and “building” while LoG on enhanced image is focus on the “ocean” part.

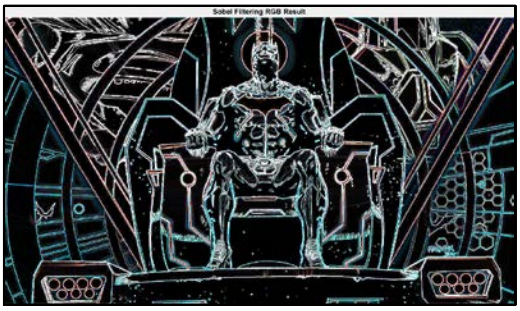
Besides, the experiment also found that, the smoothing prevent edge detection process affected from noises. For the four images supplied from previous project, similar pre-processing are done, and the results of edge detection on each image are achieved.

More details can be found on the files “Sol_p1im1.m”, “Sol_p1im2.m”, “Sol_p1im4.m”, and “Sol_p1im5.m”in the provided GitHub.

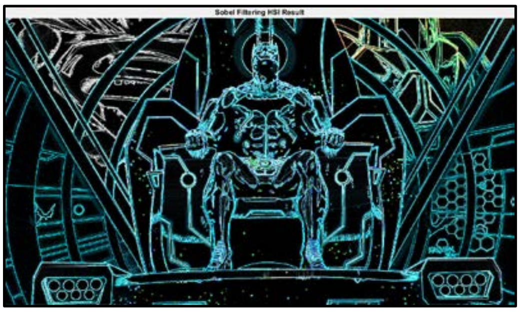
2. *Gradient filtering on different color model.*
   It is interesting to know what outcome will be produced by applying gradient filters on different color model. This experiment not just doing gradient filters on gray scale images, but also do that on other color model such as each component of RGB, Intensity component of HSI, Y component of YCbCr. The experiment below shows the result of using Sobel filter on different color models.



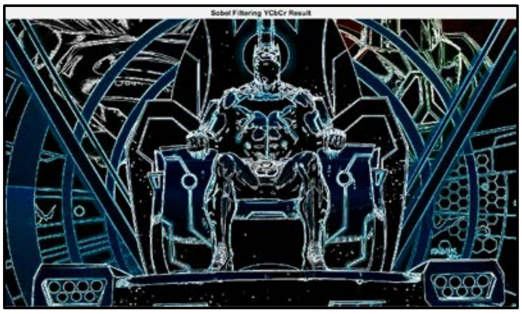
**Original Image** **Sobel on Gray Scale**



Sobel on RGB



Sobel on HSI



Sobel on YCbCr

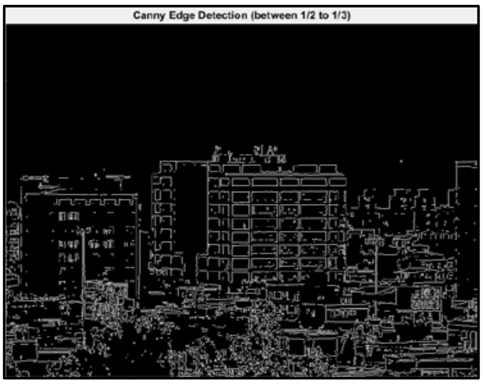
Notice that the result on YCbCr is closest to the original image. Edge detection can also produce a 3 channel output, one of the common color model choice they have used is YCbCr.

More details can be found on the file "sobel_on_different_color_model.m" in the provided GitHub.

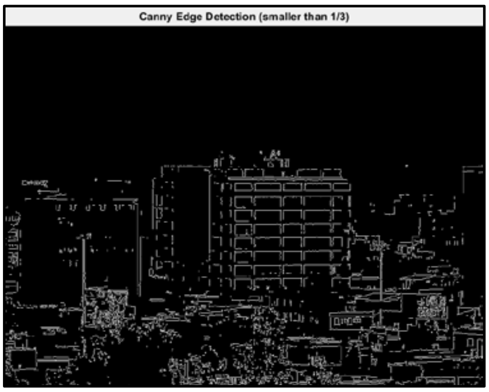
3. *<u>What is the common ratio between higher threshold and lower threshold in canny edge detection.</u>*

   As stated in ref[1], the thresholds used in canny edge detection is usually on the ratio of $\frac{\mathbf{lower\_threshold}}{\boldsymbol{higher\_threshold}} \in [\frac{1}{3},\frac{1}{2}]$ . A simple experiment had been done to see what happen if we didn't follow this ratio.

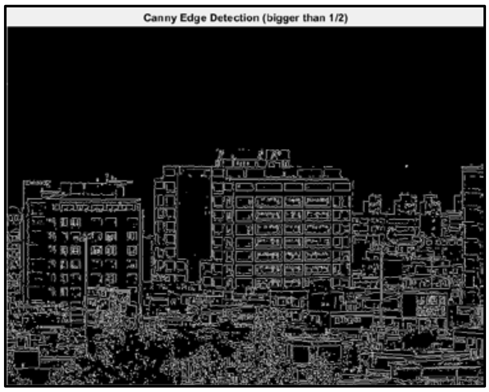

Original Image




Canny with ratio in [1/3,1/2]




**Canny with ratio smaller than 1/3**




**Canny with ratio greater than 1/2**

From the result above, by comparing with the Canny Edge with ratio in [1/3, 1/2], the result of Canny Edge with ratio smaller than 1/3 lost some edge candidate points, while the Canny Edge with ratio greater than 1/2 had too much edge candidates, which some of them might not suitable to form an edge.

More details can be found on the file “ratio_between_thresholds.m” in the provided GitHub.

## Section 3: Code Analysis

Since the content in files “Sol_p1im1.m”, “Sol_p1im2.m”, “Sol_p1im4.m”, “Sol_p1im5.m”, “Sol_ batman_god_of_knowledge.m”, and “Sol_ batman_three_jokers.m” are already described in Section 1, these files will not be shown in this section. (There are almost more than 200 lines of code in each of the files)

Note that the explanations of each part of the code are shown in the comment (font color in light green).

1. Gradient filter.

→ Prewitt filtering

```
function transImgMatr = prewitt_filtering(imgMatr, direction)
    % --------------------------------
    % This function perform the edge detection with prewitt filtering
    % ref: https://www.geeksforgeeks.org/matlab-image-edge-detection-using-prewitt-operator-from-scratch/
    % return transformed image matrix
    % --------------------------------

    imgMatr = double(imgMatr);

    hg = size(imgMatr,1); % get the height of image
    wd = size(imgMatr,2); % get the width of image

    % zero padding on image matrix
    filtSize = 3;
    shift = double(floor(filtSize / 2));
    paddMatr = padarray(imgMatr,[shift shift],0);

    % prewitt filter
    prewittx = [-1,0,1; -1,0,1; -1,0,1];
    prewitty = [-1,-1,-1; 0,0,0; 1,1,1];

    % create the matrix with width and height
    % exacly same as the input image to store the transformed pixel values
    gx = zeros(hg,wd);
    gy = zeros(hg,wd);
    for row = 1 : hg
        for col = 1 : wd
            % x direction
            mul = paddMatr(row : row + (filtSize - 1),col : col + (filtSize - 1)).* prewittx;
            gx(row,col) = sum(mul(:));

            % y direction
            mul = paddMatr(row : row + (filtSize - 1),col : col + (filtSize - 1)).* prewitty;
            gy(row,col) = sum(mul(:));

            switch direction
                case 'vertical'
                    gx(row,col) = 0; % remove gx
                case 'horizontal'
                    gy(row,col) = 0; % remove gy
            end

        end
    end

    grad = sqrt((gx .^2) + (gy .^2));
    transImgMatr = uint8(grad);

end
```

→ Sobel filtering

```
function transImgMatr = sobel_filtering(imgMatr, direction)
    % --------------------------------
    % This function perform the edge detection with sobel filtering
    % ref: https://www.geeksforgeeks.org/matlab-image-edge-detection-using-sobel-operator-from-scratch/
    % return transformed image matrix
    % --------------------------------

    imgMatr = double(imgMatr);

    hg = size(imgMatr,1); % get the height of image
    wd = size(imgMatr,2); % get the width of image

    % zero padding on image matrix
    filtSize = 3;
    shift = double(floor(filtSize / 2));
    paddMatr = padarray(imgMatr,[shift shift],0);

    % sobel filter
    sobelx = [-1,0,1; -2,0,2; -1,0,1];
    sobely = [-1,-2,-1; 0,0,0; 1,2,1];

    % create the matrix with width and height
    % exacly same as the input image to store the transformed pixel values
    gx = zeros(hg,wd);
    gy = zeros(hg,wd);
    for row = 1 : hg
        for col = 1 : wd
            % x direction
            mul = paddMatr(row : row + (filtSize - 1),col : col + (filtSize - 1)).* sobelx;
            gx(row,col) = sum(mul(:));

            % y direction
            mul = paddMatr(row : row + (filtSize - 1),col : col + (filtSize - 1)).* sobely;
            gy(row,col) = sum(mul(:));

            switch direction
                case 'vertical'
                    gx(row,col) = 0; % remove gx
                case 'horizontal'
                    gy(row,col) = 0; % remove gy
            end

        end
    end

    grad = sqrt((gx .^2) + (gy .^2));
    transImgMatr = uint8(grad);

end
```

→ LoG filtering

```
function transImgMatr = laplacian_gaussian_filter(imgMatr,sigma)
    % --------------------------------
    % This function perform the edge detection with log filtering
    % ref: https://www.mathworks.com/matlabcentral/fileexchange/59816-log_edgedetection-image-sigma
    % return transformed image matrix
    % --------------------------------

    imgMatr = double(imgMatr);

    % filter size is estimated by: sigma * constant
    filtSz = ceil(sigma) * 5;
    filtSz = (filtSz - 1) / 2;

    shift = double(floor(filtSz / 2));
    [x, y] = meshgrid(-shift : shift, -shift : shift);
    % The two parts of the LoG equation
    part_1 = (x .^ 2 + y .^ 2 - 2 * sigma ^ 2) / sigma ^ 4;
    part_2 = exp( - (x .^ 2 + y .^ 2) / (2 * sigma ^ 2) );
    part_2 = part_2 / sum(part_2(:));
    % The LoG filter
    LoG = - part_1 .* part_2;
    % The normalized LoG filter
    nLoG = LoG - mean2(LoG);

    transImgMatr = imfilter(imgMatr, nLoG, 'replicate');

    transImgMatr = transImgMatr * 255.0;
    transImgMatr = uint8(transImgMatr);

end
```

2. Canny Edge Detection.
   In this implementation, I divided steps into several function.

→ Canny Edge Detection Main Function

```matlab
function transImgMatr = canny_edge_detection(imgMatr, sigma, low_thres, high_thres)
    % -------------------------------
    % This function perform the canny edge detection
    % ref:
    % 1. https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123
    % 2. https://www.mathworks.com/matlabcentral/fileexchange/41221-canny-edge-detector
    % return transformed image matrix
    % -------------------------------

    imgMatr = double(imgMatr);
    % 1. Smoothing
    transImgMatr = smoothing_with_gaussian(imgMatr, sigma);
    % 2. Gradient Calculation
    [transImgMatr, theta] = gradient_calculation(transImgMatr);
    % 3. Non-Maxima Suppression
    transImgMatr = non_maxima_suppression(transImgMatr, theta);
    % 4. Double Thresholding
    transImgMatr = double_thresholding(transImgMatr, low_thres, high_thres);
    % 5. Hysteresis Edge Tracking
    transImgMatr = hysteresis_edge_tracking(transImgMatr);

    transImgMatr = uint8(transImgMatr);
end
```

→ Smoothing with Gaussian

```matlab
function transImgMatr = smoothing_with_gaussian(imgMatr, sigma, filtSz)
    hg = size(imgMatr,1); % get the height of image
    wd = size(imgMatr,2); % get the width of image

    % filter size is estimated by: sigma * constant
    filtSz = ceil(sigma) * 5;
    filtSz = (filtSz - 1) / 2;

    % replicate padding on image matrix
    shift = double(floor(filtSz / 2));
    paddMatr = padarray(imgMatr,[shift shift],'replicate');

    % Make 2D Gaussian kernel
    x=-ceil(shift):ceil(shift);
    Gauss1D = exp(-(x.^2/(2*sigma^2)));
    Gauss1D = Gauss1D/sum(Gauss1D(:));

    GaussX=reshape(Gauss1D,[length(Gauss1D) 1]);
    GaussY=reshape(Gauss1D,[1 length(Gauss1D)]);

    GaussMatr = zeros(length(GaussX),length(GaussY));
    for x = 1 : length(GaussX)
        for y = 1 : length(GaussY)
            GaussMatr(x,y) = GaussX(x) * GaussY(y);
        end
    end

    clearvars GaussX GaussY Gauss1D
```

```matlab
    % create the matrix with width and height
    % exacly same as the input image to store the transformed pixel values
    transImgMatr = zeros(hg,wd);

    for row = 1 : hg
        for col = 1 : wd

            tmp = 0;
            for x = 1 : filtSz
                for y = 1 : filtSz
                    tmp = tmp + GaussMatr(x,y) * paddMatr(x + (row - 1), y + (col - 1));
                end
            end

            transImgMatr(row,col) = tmp;

        end
    end

end
```

→ Gradient Calculation Function

```
function [transImgMatr, theta] = gradient_calculation(imgMatr)
    hg = size(imgMatr,1); % get the height of image
    wd = size(imgMatr,2); % get the width of image

    % =========================================
    % 1. gradients computing
    % =========================================
    % zero padding on image matrix
    filtSize = 3;
    shift = double(floor(filtSize / 2));
    paddMatr = padarray(imgMatr,[shift shift],0);

    % sobel filter
    sobelx = [-1,0,1; -2,0,2; -1,0,1];
    sobely = [-1,-2,-1; 0,0,0; 1,2,1];

    % create the matrix with width and height
    % exacly same as the input image to store the transformed pixel values
    gx = zeros(hg,wd);
    gy = zeros(hg,wd);
    for row = 1 : hg
        for col = 1 : wd
            % x direction
            mul = paddMatr(row : row + (filtSize - 1),col : col + (filtSize - 1)).* sobelx;
            gx(row,col) = sum(mul(:));

            % y direction
            mul = paddMatr(row : row + (filtSize - 1),col : col + (filtSize - 1)).* sobely;
            gy(row,col) = sum(mul(:));


        end
    end
    grad = sqrt((gx .^2) + (gy .^2));
    %grad = grad / max(grad(:)) * 255;
    transImgMatr = grad;

    % =========================================
    % 2. angle of gradients computing
    % =========================================
    [gradHg, gradWd] = size(gx);
    epsilon = 0.00000000001; % prevent infinity results (in case there could be 0 in gx)
    theta = atan(gy./(gx + epsilon)) * (180/3.1412);

    % note that the range of arctan is (-pi/2, pi/2)
    % make all elements in theta positive
    for row = 1 : gradHg
        for col = 1 : gradWd
            if (theta(row,col) < 0)
                theta(row,col) = theta(row,col) + 180;
            end
        end
    end
    % fix all element of theta into 4 degrees
    for row = 1 : gradHg
        for col = 1 : gradWd
            if ( (0 <= theta(row,col)) && (theta(row,col) < 22.5)) || ((157.5 <= theta(row,col)) && (theta(row,col) <= 180) )
                theta(row,col) = 0;
            elseif (22.5 <= theta(row,col)) && (theta(row,col) < 67.5)
                theta(row,col) = 45;
            elseif (67.5 <= theta(row,col)) && (theta(row,col) < 112.5)
                theta(row,col) = 90;
            elseif (112.5 <= theta(row,col)) && (theta(row,col) < 157.5)
                theta(row,col) = 135;
            end
        end
    end


end
```

→ Non Maxima Suppression Function

```
function transImgMatr = non_maxima_suppression(Grad, theta)
    GradHg = size(Grad, 1); % get the height of gradient
    GradWd = size(Grad, 2); % get the width of gradient

    paddMatr = padarray(Grad,[1 1],0); % zero padding

    % create the matrix with width and height
    % exacly same as the input image to store the transformed pixel values
    transImgMatr = zeros(GradHg,GradWd);
```

```
for row = 1 : GradHg
    for col = 1 : GradWd
        lt = 0;
        rt = 0;
        act_row = row + 1;
        act_col = col + 1;
        switch theta(row,col)
            case 0
                lt = paddMatr(act_row, act_col - 1);
                rt = paddMatr(act_row, act_col + 1);
            case 45
                lt = paddMatr(act_row + 1, act_col - 1);
                rt = paddMatr(act_row - 1, act_col + 1);
            case 90
                lt = paddMatr(act_row + 1, act_col);
                rt = paddMatr(act_row - 1, act_col);
            case 135
                lt = paddMatr(act_row - 1, act_col - 1);
                rt = paddMatr(act_row + 1, act_col + 1);
        end

        % check if current pixel value is the maximum in surrounding
        if (Grad(row,col) >= lt && Grad(row,col) >= rt)
            transImgMatr(row,col) = Grad(row,col);
        else
            transImgMatr(row,col) = 0;
        end

    end
end

end
```

→ Double Thresholding Function

```
function Grad = double_thresholding(Grad, low_thres, high_thres)
    GradHg = size(Grad, 1); % get the height of gradient
    GradWd = size(Grad, 2); % get the width of gradient

    weak = 25;
    strong = 255;
    zero = 0;

    for row = 1 : GradHg
        for col = 1 : GradWd
            if (Grad(row,col) >= high_thres)
                Grad(row,col) = strong;
            elseif (Grad(row,col) > low_thres)
                Grad(row,col) = weak;
            else
                Grad(row,col) = zero;
            end
        end
    end

end
```

→ Hysteresis Edge Tracking Function

```
function Grad = hysteresis_edge_tracking(Grad)
    GradHg = size(Grad, 1); % get the height of gradient
    GradWd = size(Grad, 2); % get the width of gradient

    paddMatr = padarray(Grad,[1 1],0); % zero padding

    weak = 25;
    strong = 255;
    zero = 0;

    for row  = 1 : GradHg
        for col = 1 : GradWd
            act_row = row + 1;
            act_col = col + 1;

            if (Grad(row,col) == weak)
                if (paddMatr(act_row - 1, act_col - 1) == strong || ... % top-left
                    paddMatr(act_row - 1, act_col) == strong ||     ... % upper
                    paddMatr(act_row - 1, act_col + 1) == strong || ... % top-right
                    paddMatr(act_row, act_col - 1) == strong ||     ... % left
                    paddMatr(act_row, act_col + 1) == strong ||     ... % right
                    paddMatr(act_row + 1, act_col - 1) == strong || ... % bottom-left
                    paddMatr(act_row + 1, act_col) == strong ||     ... % bottom
                    paddMatr(act_row + 1, act_col + 1) == strong)       % bottom-right

                    Grad(row,col) = strong;
                else
                    Grad(row,col) = zero;
                end
            end


        end
    end

end
```

3. Thresholding.

→ Thresholding and convert image to binary image

```
function imgMatr = convert_to_binary_image(imgMatr, threshold)
    % --------------------------------
    % This function perform the conversion from image to binary image
    % note that threshold value is between 0 to 1
    % ref: it's same to builtin function --> im2bw()
    % return transformed image matrix
    % --------------------------------

    imgMatr = uint8(imgMatr);
    th = 255 * threshold;

    imgMatr(imgMatr < th) = 0;
    imgMatr(imgMatr >= th) = 1;

    imgMatr = double(imgMatr);

end
```

4. Preprocessing Functions

This actually already described on previous assignment, here will only show the preprocessing functions that are used in this assignment.

→ Gaussian Smoothing Filter

```matlab
function transImgMatr = gaussian_filtering(imgMatr,sigma,filtSize)
    % ---------------------------------
    % This function perform the noise removal with gaussian filtering
    % return transformed image matrix
    % ---------------------------------

    hg = size(imgMatr,1); % get the height of image
    wd = size(imgMatr,2); % get the width of image

    % replicate padding on image matrix
    shift = double(floor(filtSize / 2));
    paddMatr = padarray(imgMatr,[shift shift],'replicate');

    % Make 2D Gaussian kernel
    x=-ceil(shift):ceil(shift);
    Gauss1D = exp(-(x.^2/(2*sigma^2)));
    Gauss1D = Gauss1D/sum(Gauss1D(:));

    GaussX=reshape(Gauss1D,[length(Gauss1D) 1]);
    GaussY=reshape(Gauss1D,[1 length(Gauss1D)]);

    GaussMatr = zeros(length(GaussX),length(GaussY));
    for x = 1 : length(GaussX)
        for y = 1 : length(GaussY)
            GaussMatr(x,y) = GaussX(x) * GaussY(y);
        end
    end

    clearvars GaussX GaussY Gauss1D

    % create the matrix with width and height
    % exacly same as the input image to store the transformed pixel values
    transImgMatr = zeros(hg,wd);
    % Do spatial filtering
    for row = 1 : hg
        for col = 1 : wd

            tmp = 0;
            for x = 1 : filtSize
                for y = 1 : filtSize
                    tmp = tmp + GaussMatr(x,y) * paddMatr(x + (row - 1), y + (col - 1));
                end
            end

            transImgMatr(row,col) = tmp;

        end
    end

    transImgMatr = uint8(transImgMatr);

end
```

→ Adaptive Filter

```
function transImgMatr = adaptive_filtering(imgMatr,filtWd,filtHg)
    %---------------------------------
    % This function perform the noise removal with adaptive filtering
    % return transformed image matrix
    % refer to https://www.imageeprocessing.com
    % /2011/12/adaptive-filtering-local-noise-filter.html
    %---------------------------------

    imgMatr = double(imgMatr);

    hg = size(imgMatr,1); % get the height of image
    wd = size(imgMatr,2); % get the width of image

    % zero padding on image matrix
    shiftHg = double(floor(filtHg / 2));
    shiftWd = double(floor(filtWd / 2));
    paddMatr = padarray(imgMatr,[shiftHg shiftWd],0);

    % create the matrix with width and height
    % exacly same as the input image to store the transformed pixel values
    transImgMatr = zeros(hg,wd);

    % create matrices of local variance,
    % local means,
    local_var = zeros(hg,wd);
    local_mean = zeros(hg,wd);

    % Do the spatial filtering computation
    for row = 1 : hg
        for col = 1 : wd
            % assign filter pixel values
            filter = paddMatr(row : row + (filtHg - 1),col : col + (filtWd - 1));

            % get local mean for the local region
            local_mean(row,col) = mean(filter(:));
            % get local variance for the local region
            local_var(row,col) = mean(filter(:).^2) - mean(filter(:)).^2;

        end
    end

    % calculate noise variance
    noise_var = mean(local_var(:));

    % replace local variances which smaller than noice variance
    local_var = max(local_var,noise_var);

    % apply adaptive expression formula to image matrix
    transImgMatr = imgMatr - (noise_var./local_var).*(imgMatr - local_mean);

    transImgMatr = uint8(transImgMatr);

end
```

→ Bilateral Filter

```matlab
function transImgMatr = bilateral_filtering(imgMatr,filtSize,sigma_d,sigma_g)
    % --------------------------------
    % This function perform the noise removal with bilateral filtering
    % return transformed image matrix
    % Code refer to https://www.mathworks.com/matlabcentral/fileexchange/12191-bilateral-filtering
    % --------------------------------
    imgMatr = double(imgMatr);

    hg = size(imgMatr,1); % get the height of image
    wd = size(imgMatr,2); % get the width of image

    % replicate padding on image matrix
    shift = double(floor(filtSize / 2));
    paddMatr = padarray(imgMatr,[shift shift],0);

    % Make 2D Gaussian kernel
    x=-ceil(shift) : ceil(shift);
    Gauss1D = exp(-(x.^2 / (2 * sigma_d^2)));
    Gauss1D = Gauss1D / sum(Gauss1D(:));

    GaussX=reshape(Gauss1D,[length(Gauss1D) 1]);
    GaussY=reshape(Gauss1D,[1 length(Gauss1D)]);

    GaussMatr = zeros(length(GaussX),length(GaussY));
    for x = 1 : length(GaussX)
        for y = 1 : length(GaussY)
            GaussMatr(x,y) = GaussX(x) * GaussY(y);
        end
    end

    clearvars GaussX GaussY Gauss1D

    % create the matrix with width and height
    % exacly same as the input image to store the transformed pixel values
    transImgMatr = zeros(hg,wd);
```

```matlab
    % Do spatial filtering
    for row = 1 : hg
        for col = 1 : wd

            % Extract local region
            L = paddMatr(row : row + (filtSize - 1),col : col + (filtSize - 1));
            % Compute Gaussian Intensity Weights
            H = exp(-(L - imgMatr(row,col).^2) / (2 * sigma_g^2));
            % Compute Bilateral filter response
            Bilt = H.* GaussMatr;

            transImgMatr(row,col) = sum(Bilt(:).* L(:)) / sum(Bilt(:));

        end
    end

    transImgMatr = uint8(transImgMatr);
end
```

5. Sharpening Code.

   → Laplacian Filter

```
function transImgMatr = laplacian_filtering(imgMatr,Lapl_filt)
    % --------------------------------
    % This function perform sharpening with laplacian filtering
    % return transformed image matrix
    % refer to https://bohr.wlu.ca/hfan/cp467/12/notes/cp467_12_lecture6_sharpening.pdf
    % pg13
    % --------------------------------

    imgMatr = double(imgMatr);

    hg = size(imgMatr,1); % get the height of image
    wd = size(imgMatr,2); % get the width of image

    % zero padding on image matrix
    filtSize = 3;
    shift = double(floor(filtSize / 2));
    paddMatr = padarray(imgMatr,[shift shift],0);

    % create the matrix with width and height
    % exacly same as the input image to store the transformed pixel values
    transImgMatr = zeros(hg,wd);

    %Lapl_filt = [1,1,1 ; 1,-8,1 ; 1,1,1];
    %if (filtType == 1)
        %Lapl_filt = [0,1,0 ; 1,-4,1 ; 0,1,0];
    %end


    filt = zeros(hg,wd);
    for row = 1 : hg
        for col = 1 : wd
            mul = paddMatr(row : row + (filtSize - 1),col : col + (filtSize - 1)).* Lapl_filt;
            filt(row,col) = sum(mul(:));
        end
    end

    transImgMatr = imgMatr - filt;
    transImgMatr = uint8(transImgMatr);
end
```

6. Color Space Conversion Code.

→ Conversion of RGB to HSI

```
function HSI = RGB_to_HSI(imgMatr)
    %-----------------------------------
    % This function perform the tranformation from RGB to HSI
    % return RGB
    % refer to https://www.imageeprocessing.com/2013/05/converting-rgb-image-to-hsi.html
    %-----------------------------------

    imgMatr = double(imgMatr);

    R = imgMatr(:,:,1) / 255;
    G = imgMatr(:,:,2) / 255;
    B = imgMatr(:,:,3) / 255;

    % Hue
    numerator = 1/2 * ((R - G) + (R - B));
    denominator = ((R - G).^2+((R - B).*(G - B))).^0.5;

    % To avoid divide by zero exception add a small number in the denominator
    H = acosd(numerator./(denominator + 0.000001));

    % If B>G then H= 360-Theta
    H(B > G) = 360 - H(B > G);

    % Normalize to the range [0 1]
    H = H / 360;

    % Saturation
    S = 1 - (3./ (sum(imgMatr,3) + 0.000001)).* min(imgMatr,[],3);

    % Intensity
    I = sum(imgMatr,3)./ 3;

    % HSI
    HSI = cat(3,H,S,I);

end
```

→ Conversion of HSI to RGB

```
function RGB = HSI_to_RGB(imgMatr)
    %-----------------------------------
    % This function perform the tranformation from HSI to RGB
    % return RGB
    % refer to https://www.imageeprocessing.com/2013/06/convert-hsi-image-to-rgb-image.html
    %-----------------------------------

    imgMatr = double(imgMatr);

    hg = size(imgMatr,1); % get the height of image
    wd = size(imgMatr,2); % get the width of image

    H = imgMatr(:,:,1) * 360;
    S = imgMatr(:,:,2);
    I = imgMatr(:,:,3);

    % Preallocate the R,G and B components
    R = zeros(hg,wd);
    G = zeros(hg,wd);
    B = zeros(hg,wd);

    % RG Sector(0<=H<120)
    % When H is in the above sector, the RGB components equations are
    B(H < 120) = I(H < 120).* (1 - S(H < 120));
    R(H < 120) = I(H < 120).* (1 + ((S(H < 120).* cosd(H(H < 120)))./ cosd(60 - H(H < 120))));
    G(H < 120) = 3.* I(H < 120) - (R(H < 120) + B(H < 120));

    % GB Sector(120<=H<240)
    % When H is in the above sector, the RGB components equations are
    % Subtract 120 from Hue
    H_adj = H - 120;
    R(H >= 120 & H < 240) = I(H >= 120 & H < 240).*(1 - S(H >= 120 & H < 240));
    G(H >= 120 & H < 240) = I(H >= 120 & H < 240).*(1 + ((S(H >= 120 & H < 240).*cosd(H_adj(H >= 120 & H <240)))./cosd(60 - H_adj(H >=120 & H < 240))));
    B(H >= 120 & H < 240) = 3.*I(H >= 120 & H < 240) - (R(H >= 120 & H < 240) + G(H >= 120 & H < 240));

    % BR Sector(240<=H<=360)
    % When H is in the above sector, the RGB components equations are
    % Subtract 240 from Hue
    H_adj = H - 240;
    G(H >= 240 & H <= 360) = I(H >= 240 & H <= 360).*(1 - S(H >= 240 & H <= 360));
    B(H >= 240 & H <= 360) = I(H >= 240 & H <= 360).*(1 + ((S(H >= 240 & H <= 360).*cosd(H_adj(H >= 240 & H <= 360)))./cosd(60 - H_adj(H >= 240 & H <= 360))));
    R(H >= 240 & H <= 360) = 3.*I(H >= 240 & H <= 360) - (G(H >= 240 & H <= 360) + B(H >= 240 & H <= 360));

    %Form RGB Image
    RGB = uint8(cat(3,R,G,B));

end
```

→ Conversion of RGB to HSV

```
function HSV = RGB_to_HSV(imgMatr)
    %-----------------------------------
    % This function perform the tranformation from RGB to HSV
    % return HSV
    % refer to https://www.mathworks.com/matlabcentral/fileexchange/48864-rgb_to_hsv-m
    %-----------------------------------

    imgMatr = double(imgMatr);

    hg = size(imgMatr,1); % get the height of image
    wd = size(imgMatr,2); % get the width of image

    imgMatr_R = imgMatr(:,:,1);
    imgMatr_G = imgMatr(:,:,2);
    imgMatr_B = imgMatr(:,:,3);

    H_matr = zeros(hg,wd);
    S_matr = zeros(hg,wd);
    V_matr = zeros(hg,wd);
```

```
for row = 1 : hg
    for col = 1 : wd
        R = imgMatr_R(row,col);
        G = imgMatr_G(row,col);
        B = imgMatr_B(row,col);

        Max = max(R,max(G,B));
        Min = min(R,min(G,B));

        % Compute H
        H = 0;
        delta = Max - Min;
        if (delta == 0)
            H = 0; % undefined
        elseif (Max == R)
            H = 60 * (G - B) / delta;
            if (G < B) H = H + 360; end
        elseif (Max == G)
            H = 60 * (B - R) / delta + 120;
        elseif (Max == B)
            H = 60 * (R - G) / delta + 240;
        end

        % check if the value of H is negative
        if (H < 0)
            H = H + 360;
        end
        H = H / 360.0;


       % Compute S
       if (Max == 0)
           S = 0;
       else
        S = 1 - Min / Max;
       end
```

```
        % Compute V
        V = Max;

        H_matr(row,col) = H;
        S_matr(row,col) = S;
        V_matr(row,col) = V;

    end
end

HSV = cat(3,H_matr,S_matr,V_matr);

end
```

→ Conversion of HSV to RGB

```
function RGB = HSV_to_RGB(imgMatr)
    %-----------------------------------
    % This function perform the tranformation from HSV to RGB
    % return RGB
    % refer to https://www.rapidtables.com/convert/color/hsv-to-rgb.html
    %-----------------------------------

    imgMatr = double(imgMatr);

    hg = size(imgMatr,1); % get the height of image
    wd = size(imgMatr,2); % get the width of image

    imgMatr_H = imgMatr(:,:,1) * 360;
    imgMatr_S = imgMatr(:,:,2);
    imgMatr_V = imgMatr(:,:,3) / 255;

    R_matr = zeros(hg,wd);
    G_matr = zeros(hg,wd);
    B_matr = zeros(hg,wd);
```

```
    for row = 1 : hg
        for col = 1 : wd
            H = imgMatr_H(row,col);
            S = imgMatr_S(row,col);
            V = imgMatr_V(row,col);

            % transform to RGB
            C = V * S;
            X = C * (1 - abs( mod(H/60,2) - 1 ));
            m = V - C;

            R = 0; G = 0; B = 0;
            if (H >= 0 && H < 60)
                R = C; G = X; B = 0;
            elseif (H >= 60 && H < 120)
                R = X; G = C; B = 0;
            elseif (H >= 120 && H < 180)
                R = 0; G = C; B = X;
            elseif (H >= 180 && H < 240)
                R = 0; G = X; B = C;
            elseif (H >= 240 && H < 300)
                R = X; G = 0; B = C;
            elseif (H >= 300 && H < 306)
                R = C; G = 0; B = X;
            end


            R_matr(row,col) = (R + m) * 255;
            G_matr(row,col) = (G + m) * 255;
            B_matr(row,col) = (B + m) * 255;

        end
    end

    RGB = uint8(cat(3,R_matr,G_matr,B_matr));

end
```

→ Conversion of RGB to Grayscale

```
function gray = rgb_to_gray(imgMatr)
    % --------------------------------
    % This function perform the edge detection with sobel filtering
    % ref: https://www.mathworks.com/matlabcentral/fileexchange/29392-rgb2gray
    % return grayscale image matrix
    % --------------------------------

    imgMatr = double(imgMatr);
    [hg wd ch] = size(imgMatr); % get height, width and channel of the image

    R = imgMatr(:,:,1);
    G = imgMatr(:,:,2);
    B = imgMatr(:,:,3);

    gray = 0.29900 * R + 0.58700 * G + 0.11400 * B;
    gray = uint8(gray);

end
```

→ Conversion of RGB to YCbCr

```
function YCbCr = RGB_to_YCbCr(imgMatr)
    %-----------------------------------
    % This function perform the tranformation from RGB to YCbCr
    % return YCbCr
    % refer to https://stackoverflow.com/questions/6311460/rgb-to-ycbcr-conversion-in-matlab
    %-----------------------------------

    imgMatr = double(imgMatr);

    coeffs = [65.481, -37.797, 112; ...        % R coef
              128.553, -74.203, -93.786; ...   % G coef
              24.966, 112, -18.214];           % B coef

    YCbCr = reshape(imgMatr ./ 255, [], 3) * coeffs;
    YCbCr(:,1) = YCbCr(:,1) + 16;
    YCbCr(:,2) = YCbCr(:,2) + 128;
    YCbCr(:,3) = YCbCr(:,3) + 128;

    YCbCr = reshape(uint8(YCbCr),size(imgMatr));

end
```

→ Conversion of YCbCr to RGB

```
function RGB = YCbCr_to_RGB(imgMatr)
    %-----------------------------------
    % This function perform the tranformation from YCbCr to RGB
    % return RGB
    % refer to https://github.com/arrayfire/arrayfire/issues/532
    %-----------------------------------

    imgMatr = double(imgMatr);

    coeffs = [0.0045662, 0.0045662, 0.0045662; ...        % Y coef
              0.0, -0.0015363, 0.0079107; ...             % Cb coef
              0.0062589, -0.0031881, 0.0];                % Cr coef

    RGB = reshape(imgMatr, [], 3) * coeffs;
    RGB(:,1) = RGB(:,1) + -0.8742024;
    RGB(:,2) = RGB(:,2) + 0.5316682;
    RGB(:,3) = RGB(:,3) + -1.0856326;
    RGB = RGB .* 255;
    RGB = reshape(uint8(RGB),size(imgMatr));

end
```

## Section 4: References


1. https://medium.com/@pomelyu5199/canny-edge-detector-%E5%AF%A6%E4%BD%9C-opencv-f7d1a0a57d19
2. https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123
3. https://dsp.stackexchange.com/questions/15458/edge-detection-on-a-color-image
4. https://www.mathworks.com/matlabcentral/answers/247003-how-to-detect-edges-in-color-image
5. https://www.mathworks.com/matlabcentral/answers/435919-how-to-detect-the-edges-in-the-picture-using-robert-sobel-and-prewitt-s-operator
6. https://www.mathworks.com/discovery/edge-detection.html
7. https://stackoverflow.com/questions/59582056/how-to-apply-edge-detection-filters-on-colored-images-on-sobel
8. https://github.com/winniesolves/Sobel-Filter-Implementation/blob/master/Code/2Dconvolution.py
9. https://www.imageeprocessing.com/2013/06/convert-hsi-image-to-rgb-image.html