# EE 569 3D Vision: Post-Lab Report 3

Furkan Gül, Student ID: 28795

November 19, 2020

## 1 Experiments in Lab

This first part of the report is an introductory step to understand scripts, and functions implemented in Lab deeply.

Before diving into the first filtering method discussed in Lab, it will be better to mention the main script that calls all the required functions benefited in Lab. It is named "lab3main", which can be found in the Appendix. Because there exist fundamentally three parts that will be addressed in Lab, the main script is divided into three sections by starting the code line with two comment characters (%%). Each section refers correspondingly to three distinct functions made for every three problems in Lab. At the beginning of all those three functions, there are a couple of lines of codes in common to make them generic. They all start with checking the input images if it is RGB or gray scale. If they are RGB, they will convert it into a grayscale format. To avoid arithmetic errors due to using image matrices of uint8 data type, which is the default option for image matrices in Matlab, those three functions will also convert data types of image matrices to double format at the beginning. Finally, right before those three functions show returned images in the figure, the data type of the returned image's matrix form must be converted back into uint8. So that the returned image can be displayed without any issues.

In the first section of the main script, Sobel filtering will be addressed. "lab3sobel" named function will be called with an input image and a threshold value for choosing strong edge elements in the magnitude of image gradient. It will return four images consist of the first derivatives of the input image in x and y directions, magnitude of the Sobel gradient of an input image, and Sobel edge elements in an input image. This function can be found in the Appendix. 3 by 3 Sobel kernels for x and y directions are discussed in the class. Then, they are used in Lab for obtaining the first derivatives for both x and y directions. In this function, the same sliding operation as it is in the "lab2gaussfilt" will be processed. But this time, instead of convolving the input image with the gaussian filter, the input image will be convolved separately with two Sobel derivative filters in each x and y directions to obtain x and y derivatives of an input image. After getting x and y derivatives of an image, L2 norm of those derivatives will be calculated to get the magnitude of the gradient of the input image. Then, it will be thresholded with the given threshold value to capture strong edge elements in a given input image. Lastly, the images of two first derivative arrays in x and

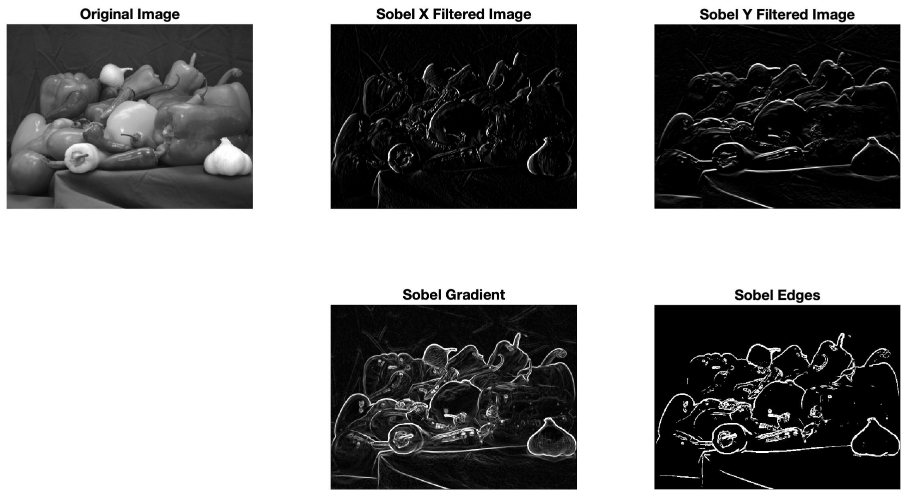


Figure 1: Sobel Filtered Images, Gradient and Image with Edge Elements

y direction, the magnitude of the Sobel gradient image, and the thresholded Sobel gradient image will be displayed in the Matlab figure. This can be seen in Figure 1.

In the second section of the main script, there is Prewitt derivative operation to get the edge elements in a given input image. The content and flow of this function are almost the same as "lab3sobel" named function. The only difference is the derivative masks for x and y directions. "lab3prewitt" named function will be called with an input image and a threshold value for choosing strong edge elements in the magnitude of image gradient. It will return four images consist of the first derivatives of the input image in x and y directions, magnitude of the Prewitt gradient of an input image, and Prewitt edge elements in an input image. This function can be found in the Appendix. 3 by 3 Prewitt kernels for x and y directions are discussed in the class. Then, they are used in Lab for obtaining the first derivatives for both x and y directions. In this function, the input image will be convolved separately with two Prewitt derivative filters in each x and y directions to obtain x and y derivatives of an input image. After getting x and y derivatives of an image, L2 norm of those derivatives will be calculated to get the magnitude of the Prewitt gradient of the input image. Then, it will be thresholded with the given threshold value to obtain strong edge elements in a given input image. Lastly, the images of two first derivative arrays in x and y direction, the magnitude of the Prewitt gradient image, and the thresholded Prewitt gradient image will be displayed in the Matlab figure. This can be seen in Figure 2.

Note that in Lab 3 document, it shows that the positive direction for Prewitt x derivative filter is the right. When it is chosen like this, the result is the same as in Figure 2. When the positive direction for x is reversed, the result for Prewitt X filtered image is the same as in Figure 3. Thus, Prewitt X filtered image in Figure 2 is different than the one in the Lab 3 document which can also be seen in Figure 3. It can be observed by examining the garlic in the bottom-right of Prewitt X filtered image in Figures 2&3 . Here in Figure 2, the

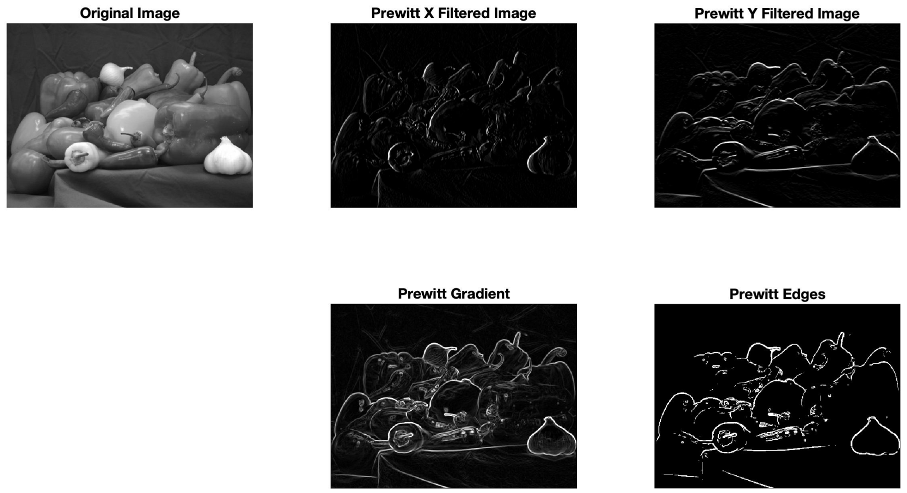


Figure 2: Prewitt Filtered Images, Gradient and Image with Edge Element. Note: The positive direction for Prewitt x derivative filter is to the right.

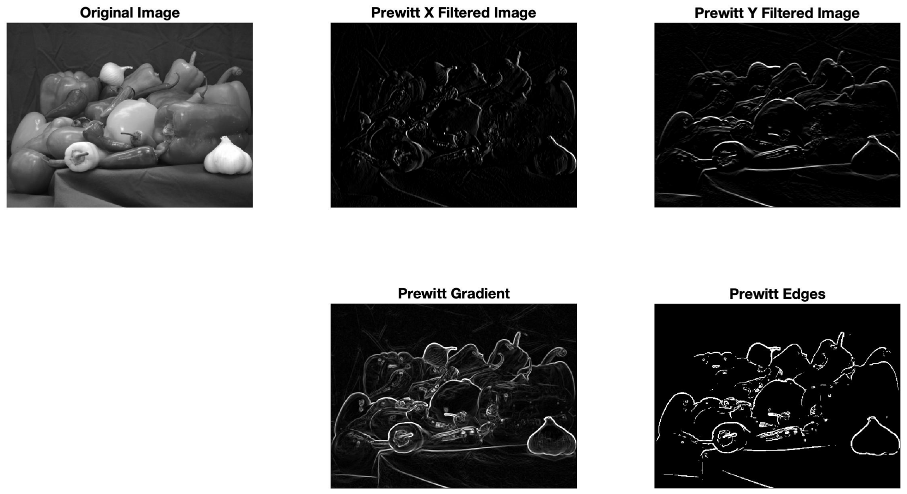


Figure 3: Prewitt Filtered Images, Gradient and Image with Edge Element. Note: The positive direction for Prewitt x derivative filter is to the left.

left edge of the garlic can be observed not the right edge of it. However, in Figure 3 or Lab 3 document, the right edge of the garlic is visible not the left. The reason for that comes from choosing a different positive direction for x derivative Prewitt filter. Thus, it can be concluded that there is an inconsistency between Prewitt X filtered image and positive x direction of Prewitt x derivative filter in Lab 3 document.

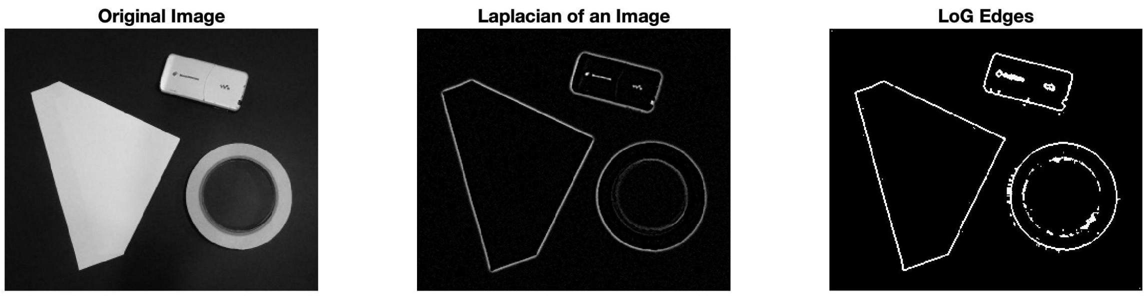


Figure 4: Laplacian of Gaussian (LoG) Filtered Images and Image with Edge Element

In the third part of the main script, LoG filtering will be addressed to find out the LoG edge elements in a given input image. "lab3log" named function will be called with an input image and two threshold values for the center pixel value in the sliding window and slopes to choose strong edge elements. This function can be found in the Appendix. Firstly, the input image will be smoothed with a Gaussian kernel to remove noises before finding edge elements. Since derivative operations for finding edge elements are very sensitive to noise, it is very crucial to remove noise in first place. 3 by 3 LoG kernel as given in the problem will be used in the function. The input image is convolved with this LoG filter with the same padding to obtain Laplacian of the input image. Then, every pixel position in this Laplacian of the input image is checked with the corresponding threshold value. After this, there are two different paths to follow. First path: if it exceeds the threshold, then there is another checkpoint for zero-crossings. Zero-crossings can be observed by looking if there is a sign change between the center pixel and its left-right or up-down neighbors. If there exist at least one zero crossings, then the slopes will be thresholded with the second threshold value in the inputs of this function. If any slope exceeds the corresponding threshold value, then this pixel position will be white and considered a strong edge element. Second path: if it does not exceed the threshold for the center pixel value in the sliding window, there is a different checkpoint for zero-crossings. Here, Zero-crossings can be observed by looking if there is a sign change between right-left or up-down neighbors of the center pixel. The remaining is the same as in the first path. Thus, strong edge elements are found and put into a new image matrix, LoG edge matrix. Lastly, original input image, LoG image, and LoG edge image will be displayed in the Matlab figure. This can be seen in Figure 4.

## 2 Results & Discussions

To get a better understanding of how accurately Sobel, Prewitt, and Laplacian of Gaussian(LoG) derivative filters will extract edge pixels from a given input image, four different

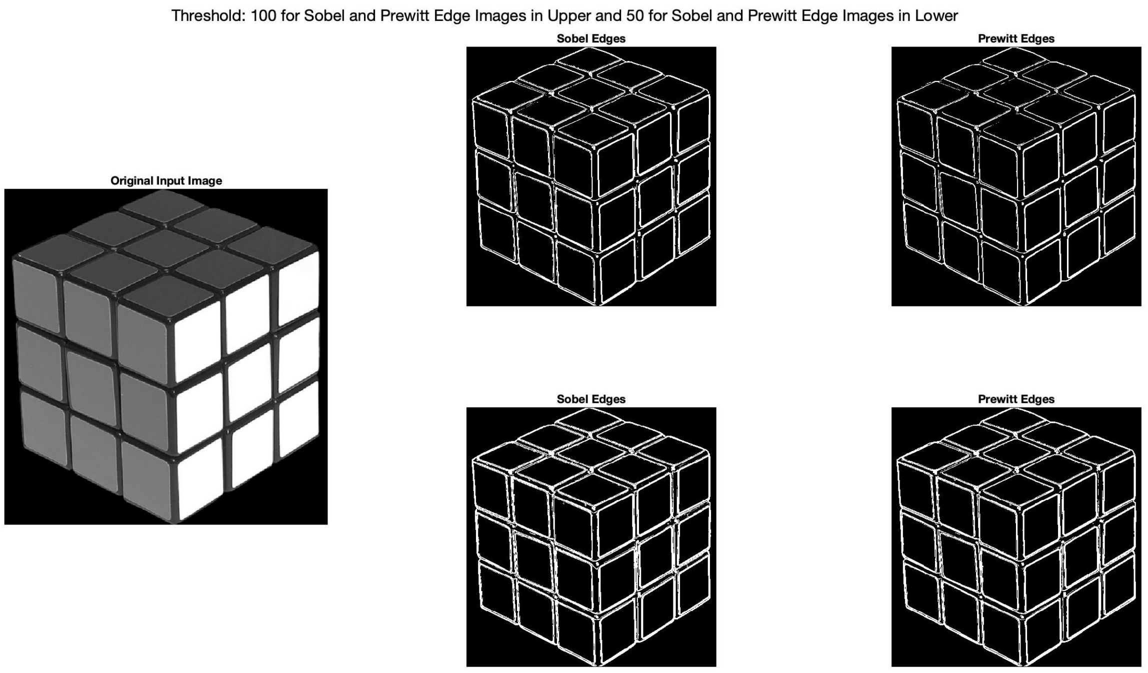

Threshold: 100 for Sobel and Prewitt Edge Images in Upper and 50 for Sobel and Prewitt Edge Images in Lower
Original Input Image
Sobel Edges
Prewitt Edges
Sobel Edges
Prewitt Edges

Figure 5: Sobel and Prewitt Edges in Rubik's Cube Image

input images found online will be processed with those three derivative filters with different hyperparameters like thresholds. These input images are Rubik's cube, a ball, daisy, and Vinfast brand car (It is a Vietnam car brand).

Rubik's cube is given as an input image to "lab3sobel" and "lab3prewitt" functions together with a threshold of 100 and 50 for detecting strong edge pixels. The resulting edge images for each of those four cases can be seen in Figure 5. In Figure 5, the top two edge images are obtained with the threshold of 100 and the bottom two edge images are outputted with the threshold of 50. When those Sobel and Prewitt Edge images in the top or the bottom are examined, it is clear that the edges in Sobel Edge image is thicker than the edges in Prewitt Edge image. It is also obvious that the discontinuities in edges in Prewitt Edge image occur many times compared to Sobel Edge image. Those two deductions arise from the fact that Sobel derivative filter is giving more weights to central element pixels. By doing so, in contrast to Prewitt derivative filters, Sobel filters can capture more edge elements with fewer discontinuities in edges, circles, or any feature you want to extract from images. On the other hand, when two Sobel Edge images or two Prewitt Edge images with different thresholds of 100 and 50 are examined, it can be easily noticed that in both cases (Sobel and Prewitt derivative operations) when the threshold is decreased, edges become thicker and discontinuities in edges reduce. The reason for that deduction comes from that when threshold value decreases, more edge elements are allowed to become what it is called "strong edge elements". What is seen in Sobel Edge and Prewitt Edge images are those strong edge elements which surpass the corresponding threshold value. Let's try the same comparison with another input image, which is a kind of ball image. This time, the threshold values are 100 for images at the top of Figure 6 and 200 for images at the bottom of Figure 6. As the effects of the threshold value and using Sobel filter are stated earlier, the effects of

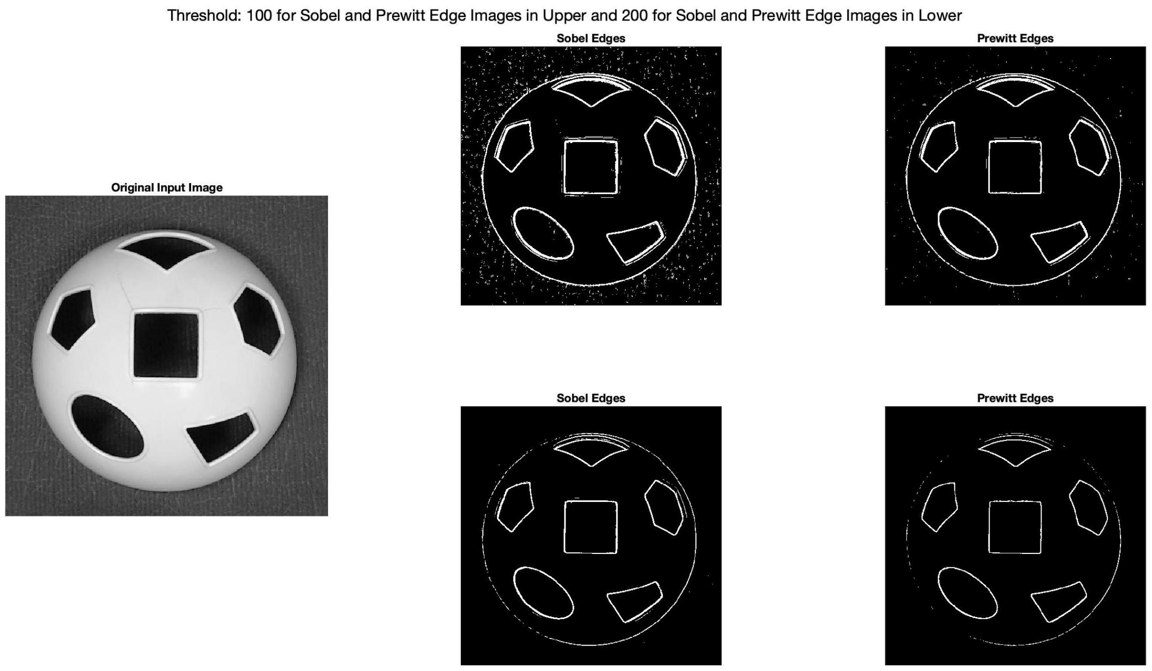

Threshold: 100 for Sobel and Prewitt Edge Images in Upper and 200 for Sobel and Prewitt Edge Images in Lower
Original Input Image
Sobel Edges
Prewitt Edges
Sobel Edges
Prewitt Edges

Figure 6: Sobel and Prewitt Edges in Ball Image

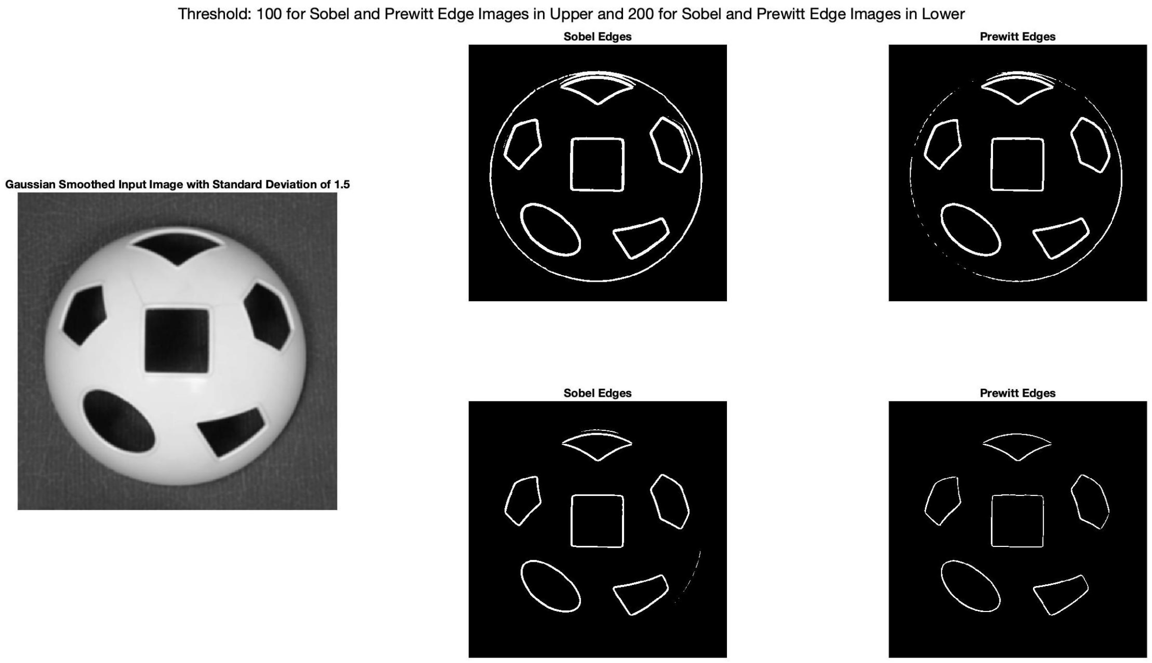

Threshold: 100 for Sobel and Prewitt Edge Images in Upper and 200 for Sobel and Prewitt Edge Images in Lower
Gaussian Smoothed Input Image with Standard Deviation of 1.5
Sobel Edges
Prewitt Edges
Sobel Edges
Prewitt Edges

Figure 7: Sobel and Prewitt Edges in Rubik's Cube Image

using lower threshold and Sobel derivatives over Prewitt can be also easily figured out like in Figure 5. Here, there is another important thing to notice. The input image has some noise outside of the ball object. Since derivative operations are very sensitive to noise in images, it affects the filtering process. In Figure 6, Sobel and Prewitt Edge images with a threshold of 100 have many outlier edge elements outside of this ball object due to that noise. There are two tricks to overcome this issue. The first one is that one can increase the threshold value to get rid of those outlier edge elements. Even though optimization for the threshold value is done manually, it causes to generate some discontinuities and thinning in the shapes (rectangle, polygons, circle) in the images. This is a bad side effect for the first method since the edges without noise can not be visualized better for the input image. The second method for overcoming this problem is to apply Gaussian smoothing to the input image in the first place. Then, this smoothed input image can be processed to obtain Sobel and Prewitt Edge images. The effect of the second method can be seen in Figure 7. As it is seen from Figure 7, by using the threshold of 100 and Sobel derivative filter, shapes in the input image can be directly observed without any discontinuities and noisy edge elements. When the threshold is increased to 200, some of the shapes in the input image start to be invisible noticeably. When Edge images with the threshold of 200 in Figures 6 & 7 are examined, it can be seen that even though the circle can be seen in Figure 6, it becomes almost invisible in Figure 7. The reason for this is that Gaussian smoothing operation lowers the contrast of this circle to a greater extent.

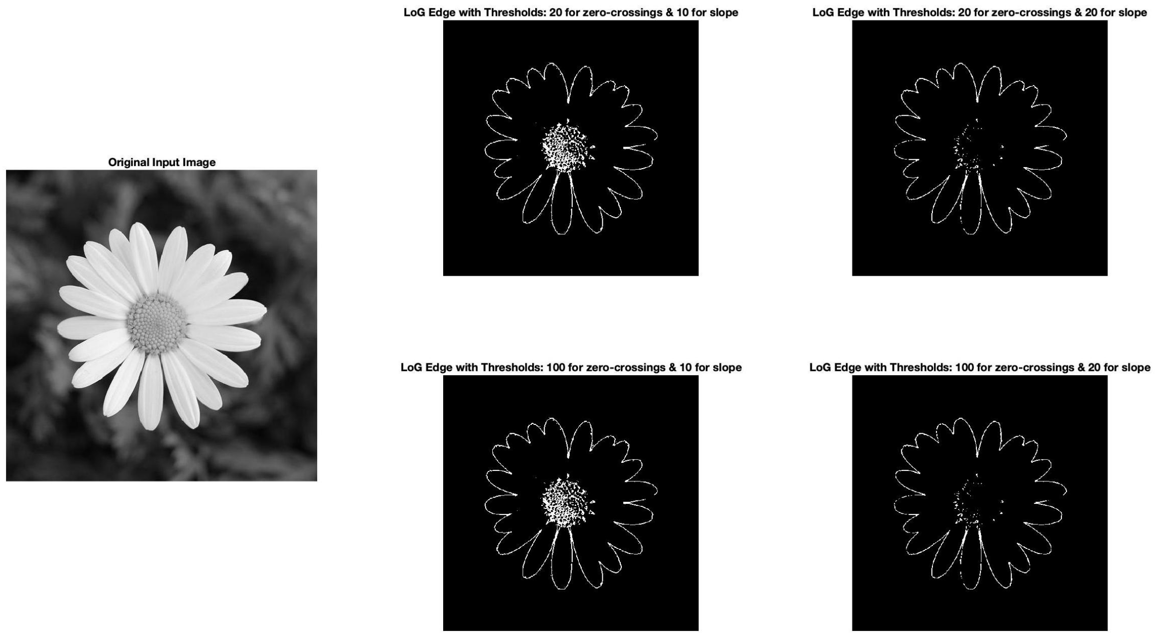

Original Input Image
LoG Edge with Thresholds: 20 for zero-crossings & 10 for slope
LoG Edge with Thresholds: 20 for zero-crossings & 20 for slope
LoG Edge with Thresholds: 100 for zero-crossings & 10 for slope
LoG Edge with Thresholds: 100 for zero-crossings & 20 for slope

Figure 8: LoG Edge Images of Daisy with Different Thresholds

To observe the effect of two threshold values in LoG filtering, Daisy image is processed with different thresholds. There are two threshold values for LoG filtering. One is for the center value in the sliding window, the other is for the slope. By looking at LoG Edge images with the same threshold for the slope in Figure 8, it can be deduced that even though there exists a significant change in the threshold for the center value in the sliding window, there is almost no difference in the output LoG Edge images. However, when the threshold for the slope is increased, there will be significant changes like discontinuities in the shapes (Note that there is no thinning effect in shapes) due to letting fewer edge elements to become strong edge elements.

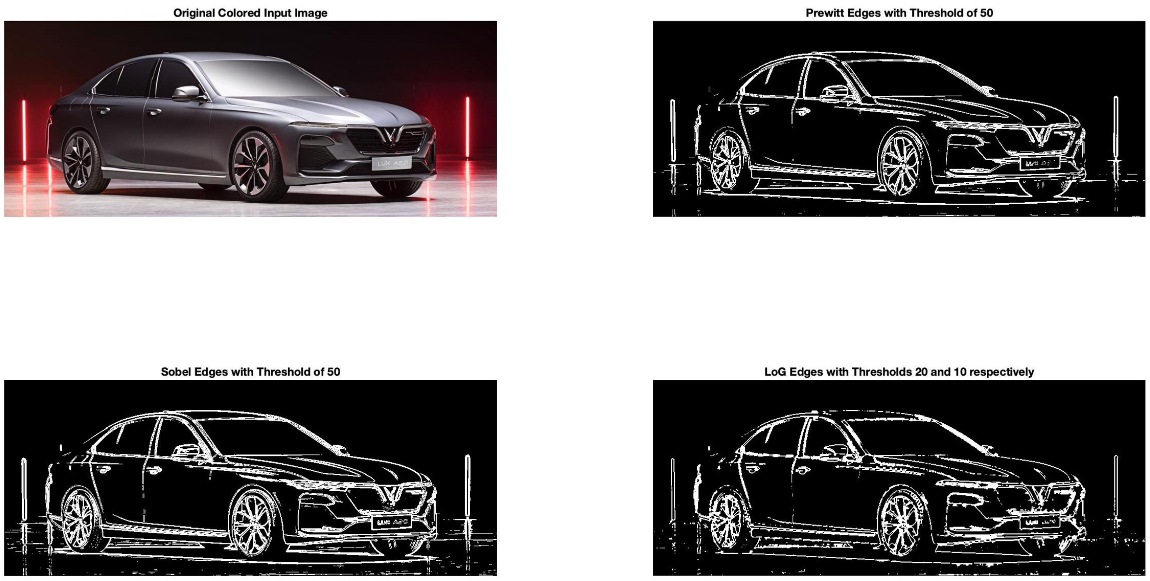

Original Colored Input Image
Prewitt Edges with Threshold of 50
Sobel Edges with Threshold of 50
LoG Edges with Thresholds 20 and 10 respectively

Figure 9: Edges in Vinfast Car

Lastly, Vinfast car model will be processed with those three different edge filters to extract edge elements in that car. I choose to analyze this image since it is more complex than the other three input images. I want to see how those three edge operators will react to that complex image. Threshold values are optimized by hand so that it will give the best edge image for each of those three cases in Figure 9. In Figure 9, it seems that both Sobel and Prewitt give better results compared to LoG operator. LoG Edge image has some noise and many discontinuities in edges. It is probably due to the fact that LoG is the second-order derivative operation and derivatives are strongly sensitive to noise.

## A Appendix

All the below codes and their corresponding outputs were obtained as latex output format without any manual intervention by using the "Publish" Method in Matlab.

## lab3main Script

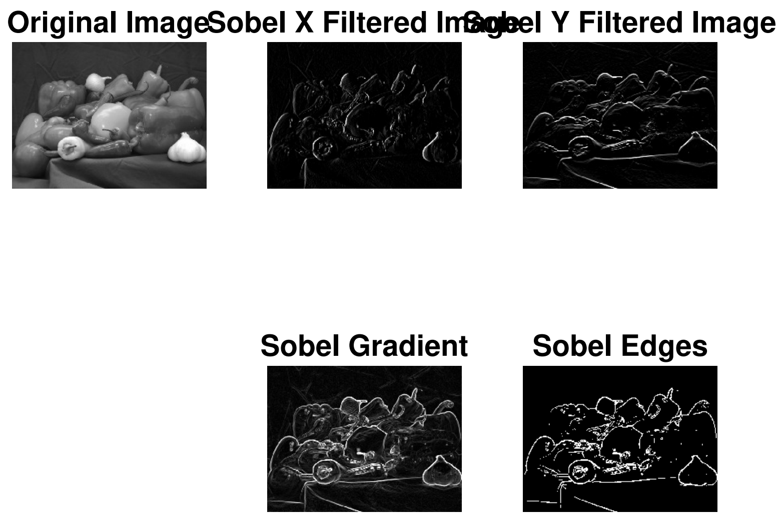
### Contents of lab3main Script

- Part 1: Sobel

- Part 2: Prewitt
- Part 3: Laplacian of Gaussian (LoG)

## Part 1: Sobel

```
clc;clear;close;
% Let's get peppers image
img = imread("peppers.png");
% In order to remove orange warning, replace function output with ~ since
% we'll not use it.
% Threshold for strong edge pixels
Thresh1 = 100;
[~] = lab3sobel(img, Thresh1);
```



## lab3sobel Function

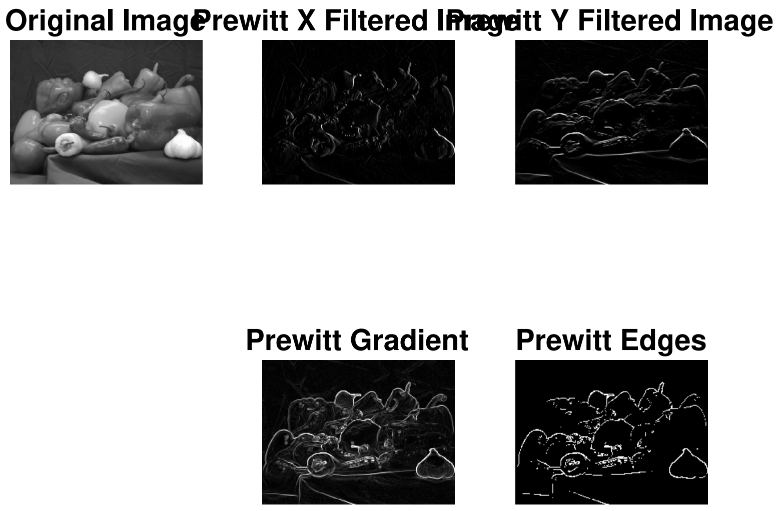
```
function [threshMag] = lab3sobel(img, T)
% Chech the original image if it is RGB or monocolor
[h, w, c] = size(img);

% If the image is RGB (colored), then convert it to grayscale image
if c == 3
    img = rgb2gray(img); % It is overwritting. Not good in general.
end

img = double(img); % In order to have proper matrix operations, convert
% datatype from uint8 to double

% Define sobel filters for x and y directions
```

```
x_Filter = [-1 0 1;-2 0 2; -1 0 1];
y_Filter = [-1 -2 -1; 0 0 0 ; 1 2 1];

k = 1; % Since we have 3 by 3 sobel filters

% Initialize X and Y sobel derivatives as 0s with the same dimension of
% the original matrix
x_filtered =  zeros(size(img));
y_filtered =  zeros(size(img));

% Go over center pixels. We have total 2k+1 parameters in every row & column of
% filter. Since we have a valid boundary, at the start, the center of the image
% coincides the filter at (k+1, k+1) position. The filter (window) will slide
% one by one until (h-k,w-k) position in the image.
for i = k+1:h-k
    for j = k+1:w-k
        Window = img(i-k:i+k, j-k:j+k);
        % Calculate the first x and y derivative pixel values of the substracted
        % part of image by multiplying each filter element-wise with window
        Xvalue = sum(sum(Window.*x_Filter));
        Yvalue = sum(sum(Window.*y_Filter));
        x_filtered(i,j) = Xvalue;
        y_filtered(i,j) = Yvalue;
    end
end

% Calculate gradient magnitude
mag = sqrt(x_filtered.^2 + y_filtered.^2);

% Threshold mag with T
threshMag = zeros(size(mag));
indices = mag>T;
threshMag(indices) = 255;

% Change the datatype of the images from double to uint8
% so that it can be displayed without any problem.
img = uint8(img);
x_filtered = uint8(x_filtered);
y_filtered = uint8(y_filtered);
mag = uint8(mag);
threshMag = uint8(threshMag);

% Plot the original, sovel derivatives, magnitude, and edge images via
% subplot function
figure()
```

```
subplot(2,3,1);
imshow(img);
title("Original Image", 'FontSize',18);

subplot(2,3,2);
imshow(x_filtered);
title("Sobel X Filtered Image", 'FontSize',18);

subplot(2,3,3);
imshow(y_filtered);
title("Sobel Y Filtered Image", 'FontSize',18);

subplot(2,3,5);
imshow(mag);
title("Sobel Gradient", 'FontSize',18);

subplot(2,3,6);
imshow(threshMag);
title("Sobel Edges", 'FontSize',18);

end
```

## Part 2: Prewitt

```
clc;clear;close;
% Let's get peppers image
img = imread("peppers.png");
% In order to remove orange warning, replace function output with ~ since
% we'll not use it.
Thresh1 = 100;
[~] = lab3prewitt(img, Thresh1);
```



## lab3prewitt Function

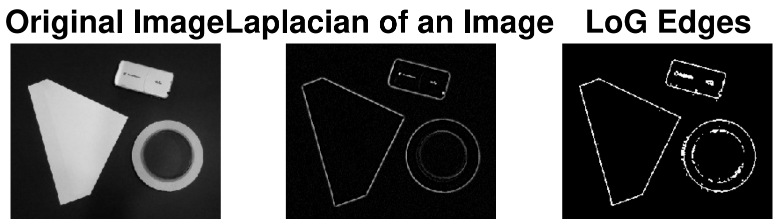
```
function [threshMag] = lab3prewitt(img, T)
% Chech the original image if it is RGB or monocolor
[h, w, c] = size(img);

% If the image is RGB (colored), then convert it to grayscale image
if c == 3
    img = rgb2gray(img); % It is overwritting. Not good in general.
end

img = double(img); % In order to have proper matrix operations, convert
% datatype from uint8 to double

% Define prewitt filters for x and y
x_Filter = -1*[-1 0 1;-1 0 1; -1 0 1];
y_Filter = [-1 -1 -1; 0 0 0 ; 1 1 1];

k = 1; % Since we have 3 by 3 prewitt filters

% Initialize X and Y prewitt derivatives as 0s with the same dimension of
% the original matrix
x_filtered =  zeros(size(img));
y_filtered =  zeros(size(img));

% Go over center pixels. We have total 2k+1 parameters in every row & column of
% filter. Since we have a valid boundary, at the start, the center of the image
% coincides the filter at (k+1, k+1) position. The filter (window) will slide
% one by one until (h-k,w-k) position in the image.
for i = k+1:h-k
```

```
    for j = k+1:w-k
        Window = img(i-k:i+k, j-k:j+k);
        % Calculate the first x and y derivative pixel values of the substracted
        % part of image by multiplying each filter element-wise with window
        Xvalue = sum(sum(Window.*x_Filter));
        Yvalue = sum(sum(Window.*y_Filter));
        x_filtered(i,j) = Xvalue;
        y_filtered(i,j) = Yvalue;
    end
end

% Calculate gradient magnitude
mag = sqrt(x_filtered.^2 + y_filtered.^2);

% Threshold mag with T
threshMag = zeros(size(mag));
indices = mag>T;
threshMag(indices) = 255;

% Change the datatype of the images from double to uint8
% so that it can be displayed without any problem.
img = uint8(img);
x_filtered = uint8(x_filtered);
y_filtered = uint8(y_filtered);
mag = uint8(mag);
threshMag = uint8(threshMag);

% Plot the original, prewitt derivatives, magnitude, and edge images via
% subplot function
figure()

subplot(2,3,1);
imshow(img);
title("Original Image", 'FontSize',18);

subplot(2,3,2);
imshow(x_filtered);
title("Prewitt X Filtered Image", 'FontSize',18);

subplot(2,3,3);
imshow(y_filtered);
title("Prewitt Y Filtered Image", 'FontSize',18);

subplot(2,3,5);
imshow(mag);
```

```
title("Prewitt Gradient", 'FontSize',18);

subplot(2,3,6);
imshow(threshMag);
title("Prewitt Edges", 'FontSize',18);

end
```

## Part 3: Laplacian of Gaussian (LoG)

```
clc;clear;close;
% Let's get object contour image
img = imread("Object_contours.jpg");
% In order to remove orange warning, replace function output with ~ since
% we'll not use it.
Thresh1 = 1; % Threshold for determining zero crossings
Thresh2 = 7; % Threshold for the slope to get strong edge pixels
[~] = lab3log(img, Thresh1, Thresh2);
```



## lab3logFunction

```
function [E] = lab3log(img, T1, T2)
% Chech the original image if it is RGB or monocolor
[h, w, c] = size(img);

% If the image is RGB (colored), then convert it to grayscale image
if c == 3
    img = rgb2gray(img); % It is overwritting. Not good in general.
end

% Blur the image, using Gaussian builtin function
J = imgaussfilt(img, 1.1);

% datatype from uint8 to double


% Define laplace filter for x and y
```

```
laplac_Filter = [0 1 0;1 -4 1; 0 1 0];
G = conv2(J, laplac_Filter, "same");

% Find the zero crossing of the laplacian and compare the local change at
% this point to a threshold
E = zeros(h,w);

k = 1; % Since we have 3 by 3 LoG filter

% Go over center pixels. We have total 2k+1 parameters in every row & column of
% filter. Since we have a valid boundary, at the start, the center of the image
% coincides the filter at (k+1, k+1) position. The filter (window) will slide
% one by one until (h-k,w-k) position in the image.
for i = k+1:h-k
    for j = k+1:w-k
        % Apply thresholding at each pixel
        if abs(G(i,j)) >= T1
            % Check zero crossings occur at (i,j) pixel or not
            if (G(i,j)*G(i-1,j)<0 )|| (G(i,j)*G(i+1, j)<0)|| (G(i,j)*G(i, j-1)<0)...
                || (G(i,j)*G(i, j+1)<0)
                % Apply thresholding to slopes for finding strong edge
                % pixels
                if abs(G(i,j)-G(i-1,j))>= T2 || abs(G(i,j)-G(i+1,j))>= T2...
                    || abs(G(i,j)-G(i, j-1))>= T2 || abs(G(i,j)-G(i, j+1))>= T2
                    % Make strong edge pixels white
                    E(i,j) = 255;
                end
            end
        else
            % Check zero crossings occur at (i,j) pixel or not
            if (G(i+1,j)*G(i-1,j)<0 )|| (G(i,j+1)*G(i, j-1)<0)
                % Apply thresholding to slopes for finding strong edge
                % pixels
                if abs(G(i, j+1) - G(i, j-1))>= T2 || abs(G(i+1, j) -G(i-1,j))>= T2
                    % Make strong edge pixels white
                    E(i,j) = 255;
                end
            end
        end
    end
end

% Change the datatype of the images from double to uint8
% so that it can be displayed without any problem.
img = uint8(img);
```

```
G = uint8(G);
E = uint8(E);

% Plot the original image, laplacian of image and log edge images
% via subplot function
figure()

subplot(1,3,1);
imshow(img);
title("Original Image", 'FontSize',18);

subplot(1,3,2);
imshow(G,[]);
title("Laplacian of an Image", 'FontSize',18);

subplot(1,3,3);
E= medfilt2(E);
imshow(E);
title("LoG Edges", 'FontSize',18);
end
```

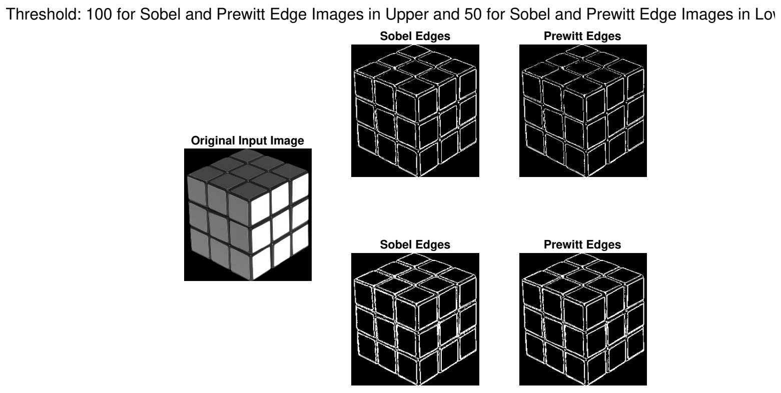
## Examples4Lab3 Script

### Contents of Examples4Lab3 Script

### Part 1: Sobel vs Prewitt

```
clc;clear;close;
% Let's get ball and rubicks cube image
%img = imread("ball.jpg");
img = imread("rubicsCube.png");
%img = imgaussfilt(img, 1.5); % smooth for noise before
%applying derivative since it is sensitive to noise
% Threshold for strong edge pixels
Thresh1 = 100;
% Do not plot figures in those two functions
set(0,'DefaultFigureVisible','off')
[thSobelGrad] = lab3sobel(img, Thresh1);
[thPrwGrad] = lab3prewitt(img, Thresh1);
```

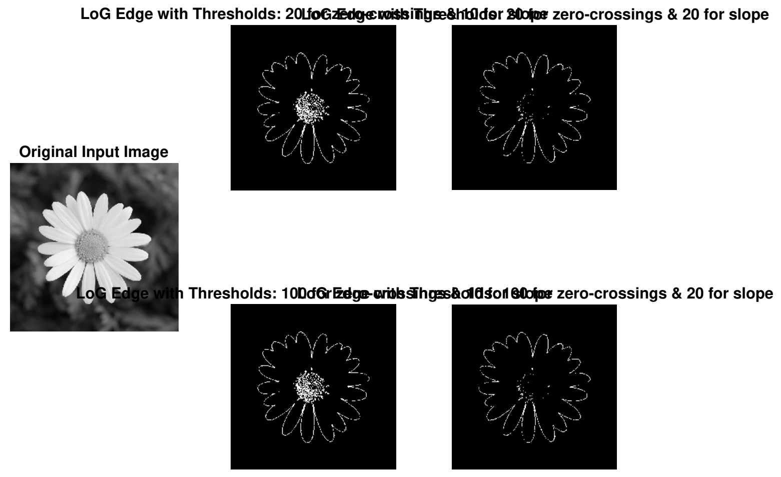
```
Thresh1 = 50;
% Do not plot figures in those two functions
[thSobelGrad2] = lab3sobel(img, Thresh1);
[thPrwGrad2] = lab3prewitt(img, Thresh1);
set(0,'DefaultFigureVisible','on')

figure
subplot(2,3,2)
imshow(thSobelGrad)
title("Sobel Edges")
subplot(2,3,[1,4])
imshow(rgb2gray(img))
title("Original Input Image")
subplot(2,3,3)
imshow(thPrwGrad)
title("Prewitt Edges")

subplot(2,3,5)
imshow(thSobelGrad2)
title("Sobel Edges")
subplot(2,3,6)
imshow(thPrwGrad2)
title("Prewitt Edges")

sgt = sgtitle('Threshold: 100 for Sobel and Prewitt Edge Images in Upper and 50 for Sobe
```


Threshold: 100 for Sobel and Prewitt Edge Images in Upper and 50 for Sobel and Prewitt Edge Images in Lo
Sobel Edges
Prewitt Edges
Original Input Image
Sobel Edges
Prewitt Edges

## Part 2: Laplacian of Gaussian (LoG) with Different Thresholds

```
clc;clear;close;
% Let's get object contour image
img = imread("papatya.jpg");
% Do not plot figures in those two functions
```

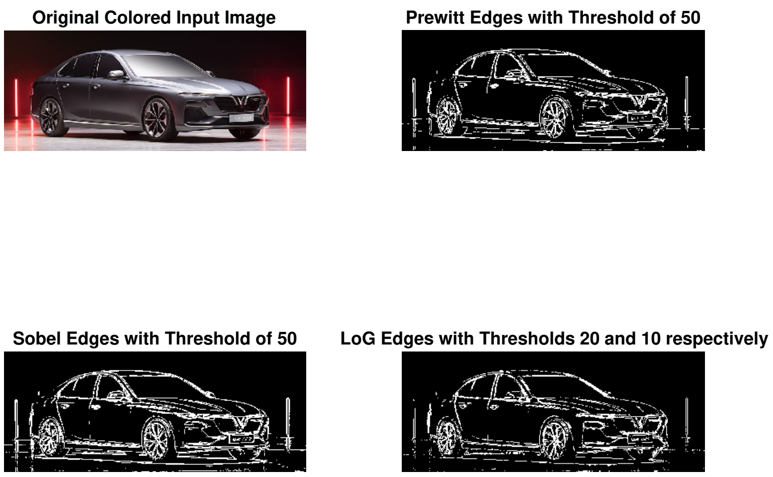
```
set(0,'DefaultFigureVisible','off')
Thresh1 = 20; % Threshold for determining zero crossings
Thresh2 = 10; % Threshold for the slope to get strong edge pixels
[logEdg1] = lab3log(img, Thresh1, Thresh2);
Thresh2 = 20;
[logEdg2] = lab3log(img, Thresh1, Thresh2);

Thresh1 = 100; % Threshold for determining zero crossings
Thresh2 = 10;
[logEdg3] = lab3log(img, Thresh1, Thresh2);
Thresh1 = 100;
Thresh2 = 20;
[logEdg4] = lab3log(img, Thresh1, Thresh2);
set(0,'DefaultFigureVisible','on')

figure

subplot(2,3,[1,4])
imshow(rgb2gray(img))
title("Original Input Image")

subplot(2,3,2)
imshow(logEdg1)
title("LoG Edge with Thresholds: 20 for zero-crossings & 10 for slope")

subplot(2,3,3)
imshow(logEdg2)
title("LoG Edge with Thresholds: 20 for zero-crossings & 20 for slope")

subplot(2,3,5)
imshow(logEdg3)
title("LoG Edge with Thresholds: 100 for zero-crossings & 10 for slope")

subplot(2,3,6)
imshow(logEdg4)
title("LoG Edge with Thresholds: 100 for zero-crossings & 20 for slope")
```


LoG Edge with Thresholds: 20
zero-crossings & 20 for slope
Original Input Image
LoG Edge with Thresholds: 100
zero-crossings & 20 for slope

## Part 2: Sobel, Prewitt, LoG Edges Comparison

```
clc;clear;close;
% Let's get vietnam car image and resize it since it takes too long to
% process
img = imread("vinfast.jpg");
img = imresize(img, 0.25);
% Do not plot figures in those two functions
set(0,'DefaultFigureVisible','off')
Thresh1 = 20; % Threshold for determining zero crossings
Thresh2 = 10; % Threshold for the slope to get strong edge pixels
[logEdg1] = lab3log(img, Thresh1, Thresh2);

Thresh1 = 100;
% Do not plot figures in those two functions
[thSobelGrad2] = lab3sobel(img, Thresh1);
[thPrwGrad2] = lab3prewitt(img, Thresh1);
set(0,'DefaultFigureVisible','on')

figure
subplot(2,2,1)
imshow(img)
title("Original Colored Input Image")

subplot(2,2,2)
imshow(thPrwGrad2)
title("Prewitt Edges with Threshold of 50")

subplot(2,2,3)
imshow(thSobelGrad2)
title("Sobel Edges with Threshold of 50")
```

```
subplot(2,2,4)
imshow(logEdg1)
title("LoG Edges with Thresholds 20 and 10 respectively")
```


Original Colored Input Image
Prewitt Edges with Threshold of 50
Sobel Edges with Threshold of 50
LoG Edges with Thresholds 20 and 10 respectively