Project 2

Lab Partners: Ryan Myers, Mihir Deshmukh

Sections: 428-01 (Deshmukh), 428-01 (Myers)

Winter Quarter

1/25/2020

Professor: Jane Zhang

Part A

The ‘Boat2.tif’ image shows randomly distributed white and black pixels indicating that it has salt and pepper noise. In order to reduce the noise in this image, it is most optimal to use a median filter where each pixel is replaced with the median value of its neighbors. This filter is not the best choice when the number of noise pixels is greater than half the neighboring pixels, however, in the boat image the noise is fairly spread out making it a good choice. Below is the original image before any filtering has been applied to it

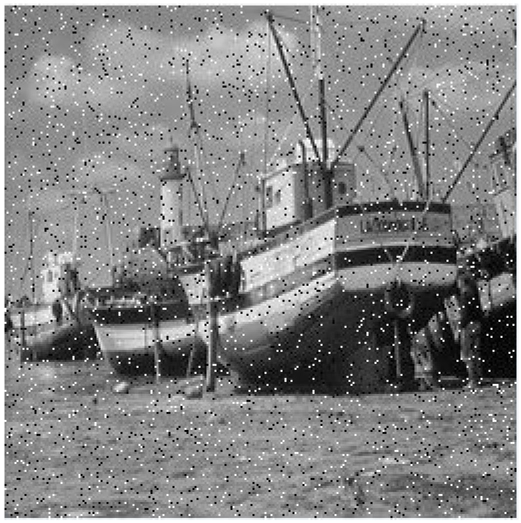

Figure 1: Unfiltered ‘Boat2.tif’ image displaying salt and pepper noise

And after we have applied our filter to the image, the result is the following

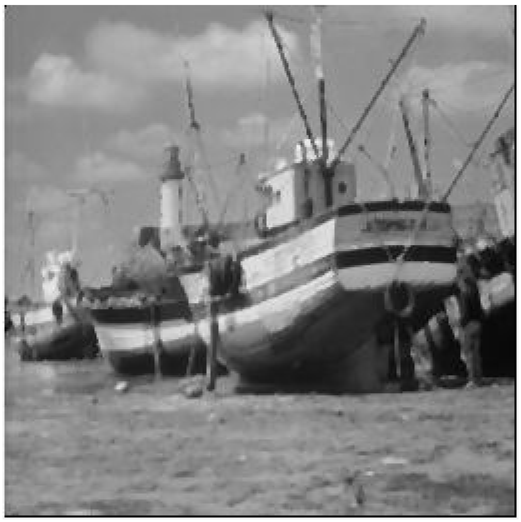

Figure 2: Median filtered ‘Boat2.tif’ image

As seen in the image, the colors are slightly more dull due to the blending effect of the filter, but all of the black and white pixel noise has been removed. This technique utilized non-linear filtering with a mask of 3x3 as a larger mask was not needed to remove the noise.

The ‘building.gif’ file shows spots of noise and blurring due to Gaussian distributed noise. In order to smooth the image and reduce the randomly distributed noise, it is best to apply a Gaussian filter to it. We decided the most optimal choice for smoothing the image without introducing too much of a blurring effect was a Gaussian filter with sigma = 2 and a convolutional matrix size of 5x5. The original image before filtering was applied can be seen below

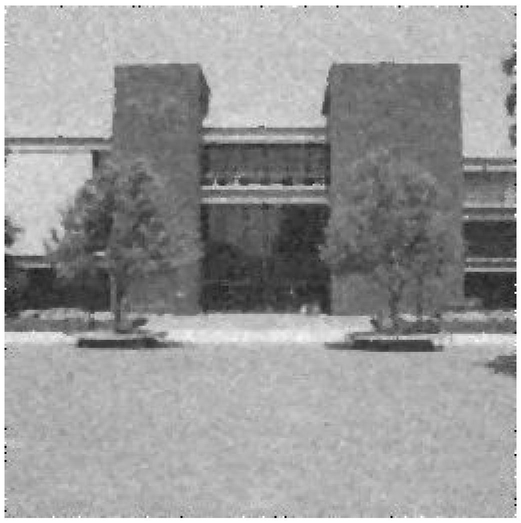

Figure 3: Original ‘building.gif’ image showing Gaussian noise

After filtering was applied, we were left with the following image

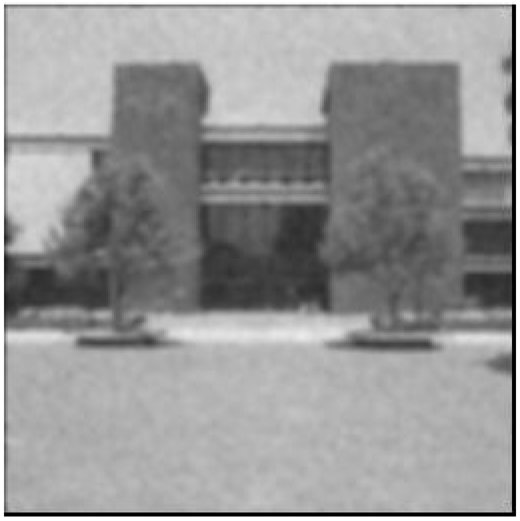

Figure 4: The ‘building.gif’ image after a Gaussian filter is applied

As can be seen from the filtered image, much of the white noise was able to be reduced and the overall image is smoother than before. Consequently, however, the image is less sharp than the original and it is harder to make out details. Overall this would be a useful technique for providing better edge definitions in the image.

Part B

1- Prewitt and Sobel Operators

We applied the Prewitt and Sobel operators to the corridor image. Below we can see a side by side comparison of the original image, and the original image with the operators applied to them.

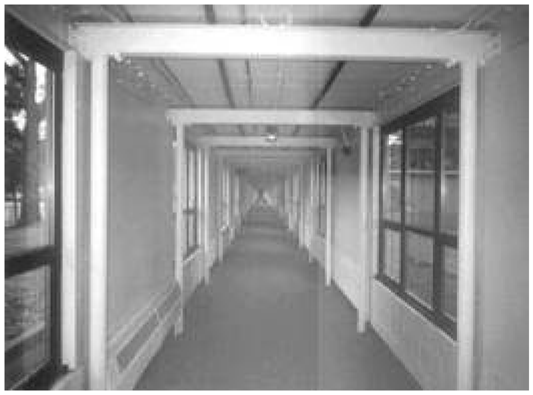

Figure 5: Original Image of corridor

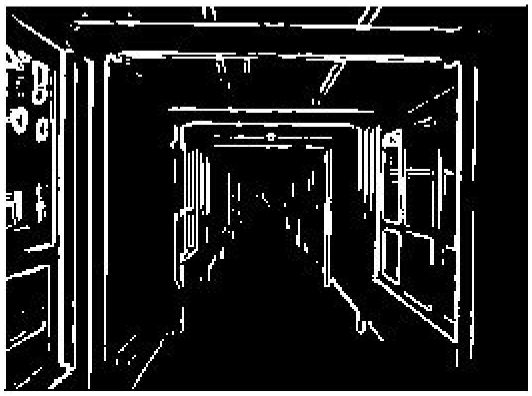

Figure 6: Prewitt operator applied

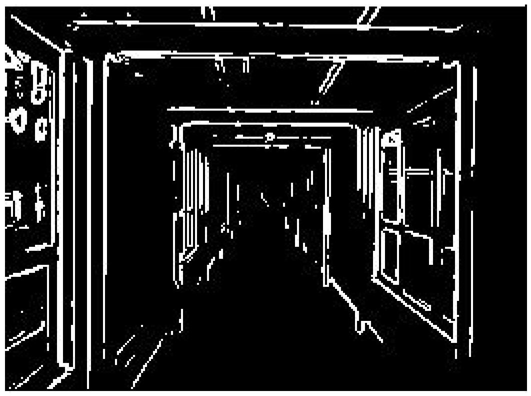

Figure 7: Sobel operator applied

The threshold value applied in both filters is the same at 0.17. While the differences are marginal we found that the Sobel operator performed better at detecting diagonal edges and resulted in edges appearing more connected. The Prewitt filter on the other hand was better at identifying strictly horizontal and diagonal lines. These differences can be mitigated by choosing slightly different threshold values for each. However, for this image the Sobel operator was better.

2 - Laplacian of Gaussian filter and Zero-Crossing

We applied the LoG filter to the corridor image above (figure 5). We found that an increased mask size reduced noise so the lines were more distinct. However, we end up losing some of the edges and the edges that remain are not as straight and crisp and more wavy. The below images show the effect of increasing the mask size by 4 (from 7 to 15) with the same $\boldsymbol{\sigma}^2 = 25$.

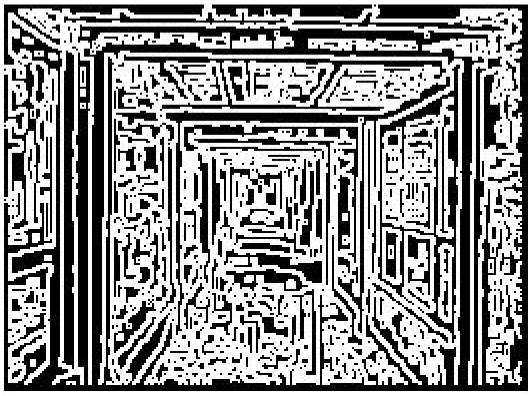
Figure 8: Mask = 7, $\boldsymbol{\sigma}^2 = 25$
LoG filter to corridor image

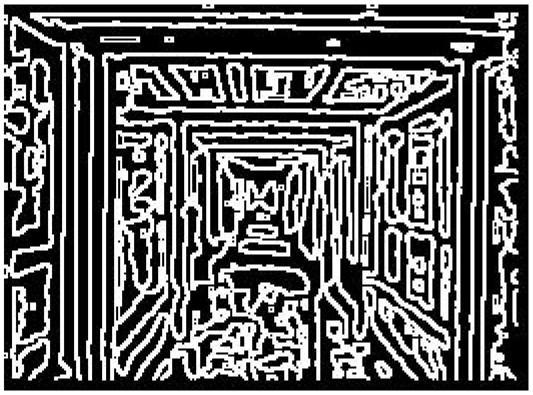
Figure 9: Mask = 7, $\boldsymbol{\sigma}^2 = 25$
LoG filter to corridor image

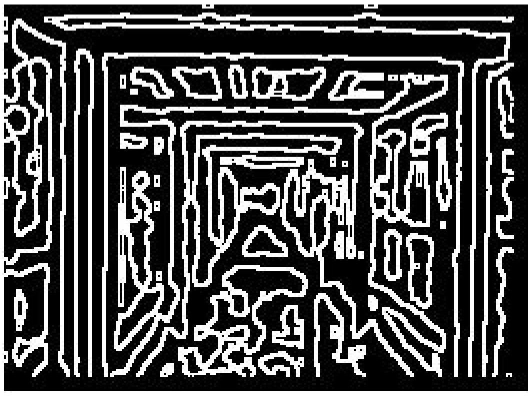
Figure 10: Mask = 7, $\boldsymbol{\sigma}^2 = 25$
LoG filter to corridor image

Below, we can see the effect of changing the $\boldsymbol{\sigma}$ value with a fixed mask size. We found through experimentation that the best mask size was 9 for this image. Increasing the $\boldsymbol{\sigma}$ value resulted in a clearer distinction between lines. However, the effect of the lines becoming clearer with an increase in $\boldsymbol{\sigma}$ was not linear. We can see that the difference between figure 11 and figure 12 is more pronounced than the difference between figure 12 and figure 13. The best $\boldsymbol{\sigma}$ value is 5 for a mask size of 9 as it preserved the main edges while removing minor edges that the filter detected. However, for larger mask sizes an increase in $\boldsymbol{\sigma}$ results in some edges being eliminated or two neighboring boxes to be combined into one big box. Our conclusion is that mask sizes and $\boldsymbol{\sigma}$ have an inverse relationship on the clarity of the edges and it's up to the user to find a balance.

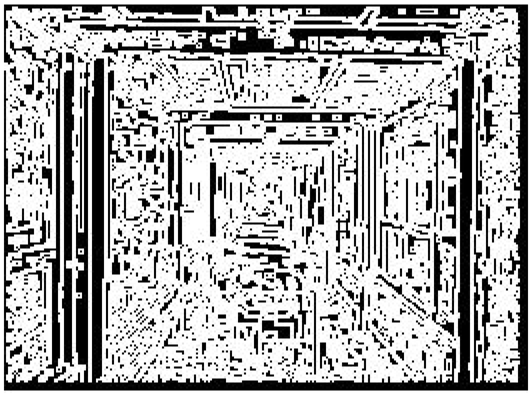
Figure 11: Mask = 7, $\boldsymbol{\sigma} = 1$

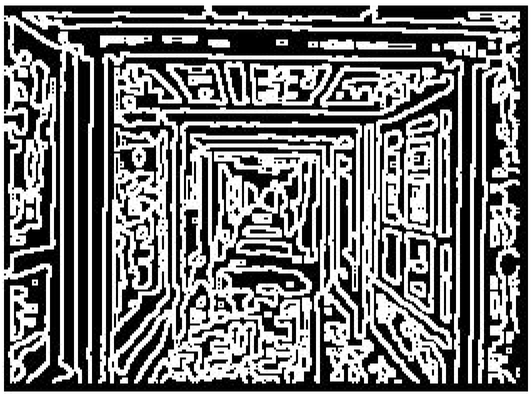
Figure 12: Mask = 7, $\boldsymbol{\sigma}^2 = 3$

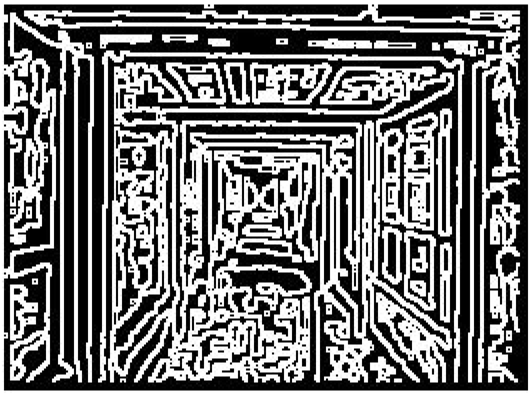
Figure 13: Mask = 7, $\boldsymbol{\sigma}^2 = 5$

3 - Canny Edge Detection

With each of the images we can see dramatic effects as we raise the values of sigma and the threshold. Increasing the value of sigma used on in the initial Gaussian filtering allows for more large scale features to be highlighted even if they do not have as dramatic a gradient as the finer features. This works along with the threshold value which suppresses smaller gradient values as it grows larger to eliminate the less useful edges in the image. The following shows the results for each of the test images

'corner_window.jpg'

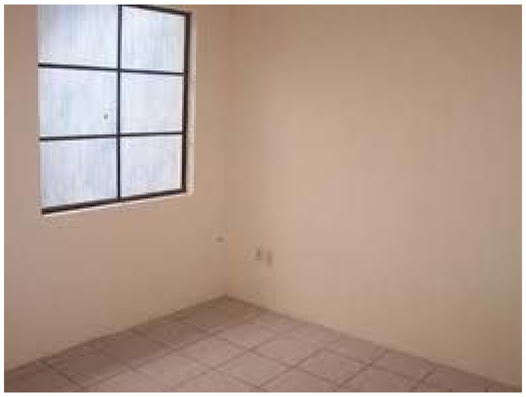

Figure 14: Original corner window image before any edge detection is applied

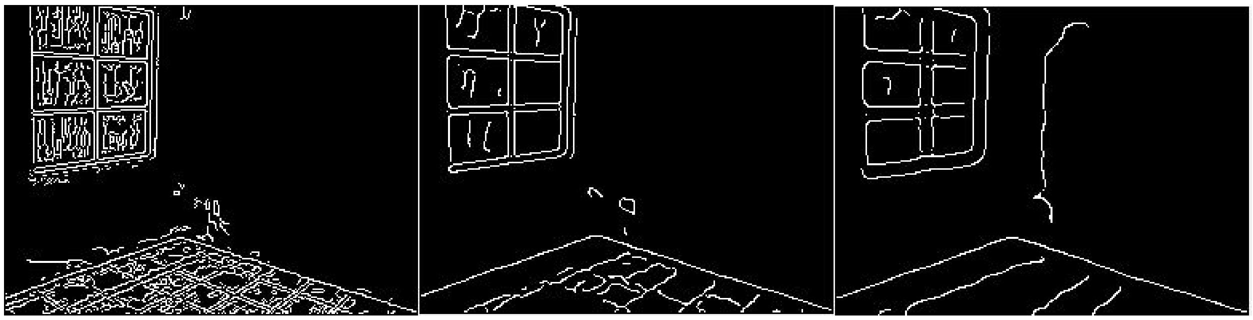

Figure 15: Corner window images with σ = 0.5, 2.0, and 4.0 respectively

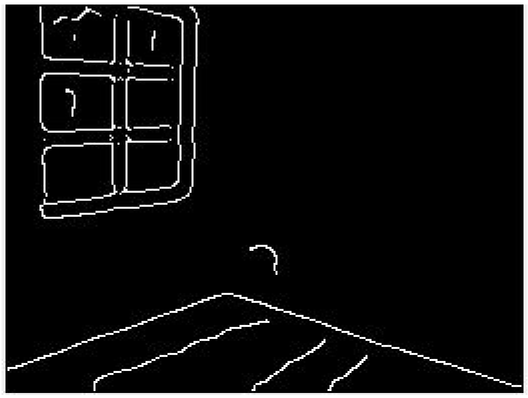

Figure 16: Best edge detection for the corner window image with σ = 3.5, negating noise through the window and the shadow on the wall

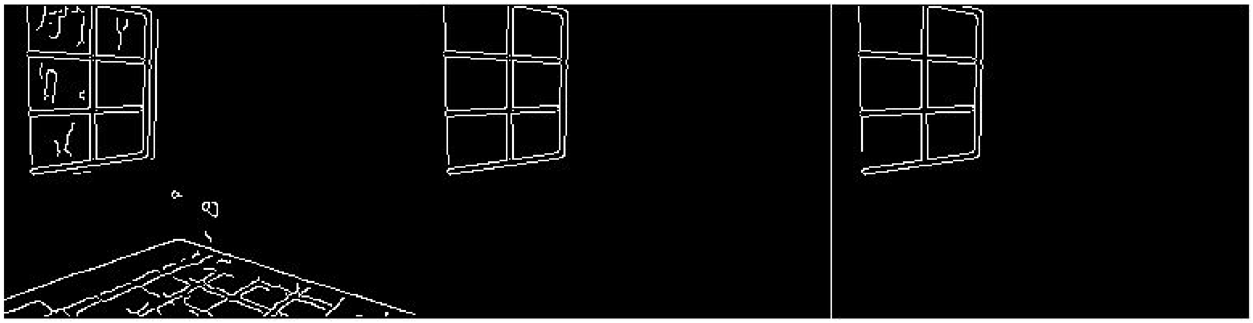

Figure 17: Corner window image with T = 0.5, 2.0, and 4.0 respectively

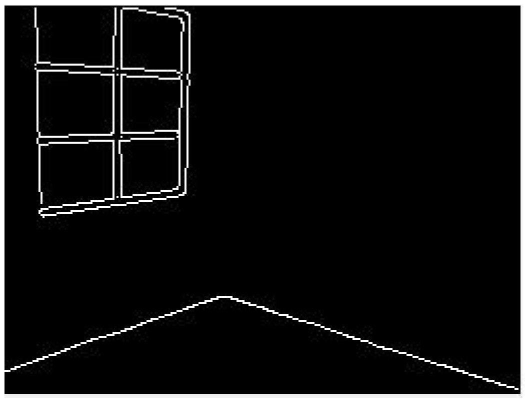

Figure 18: Corner window with optimal threshold T = 1.5, keeping all important edges in the final image

‘corridor.jpg’

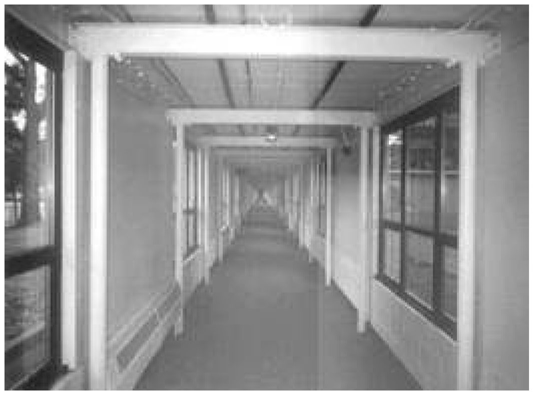
Figure 19: Original corridor image with no edge detection applied to it

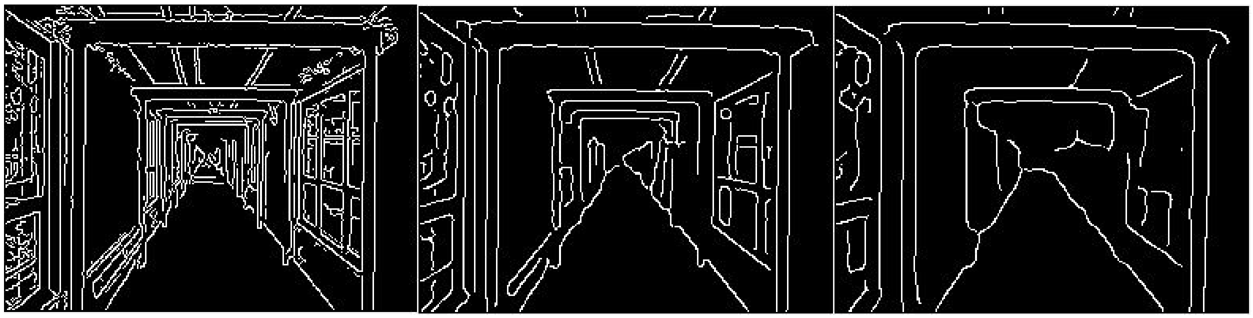
Figure 20: Corridor image with σ = 0.5, 2.0, 4.0 respectively

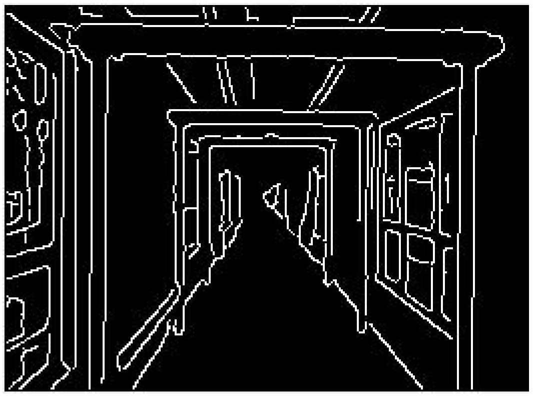
Figure 21: Corridor image with optimal σ value 1.5 to retain window and wall details

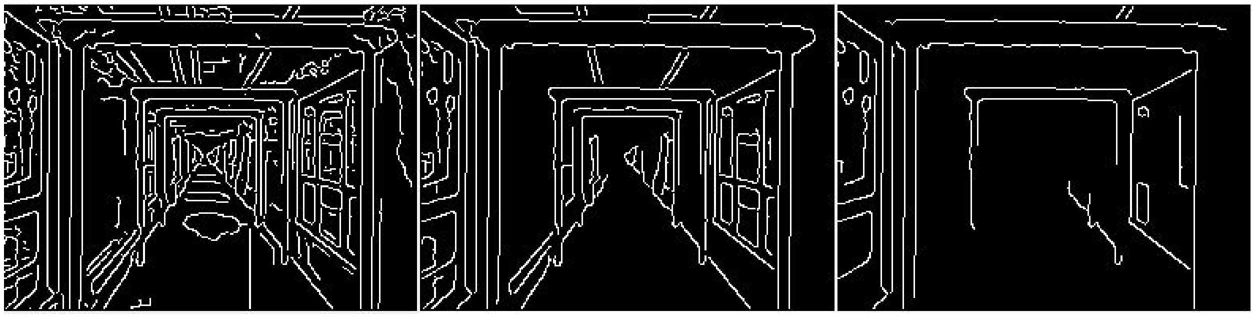
Figure 22: Corridor image with T = 0.5, 2.0, 4.0 respectively

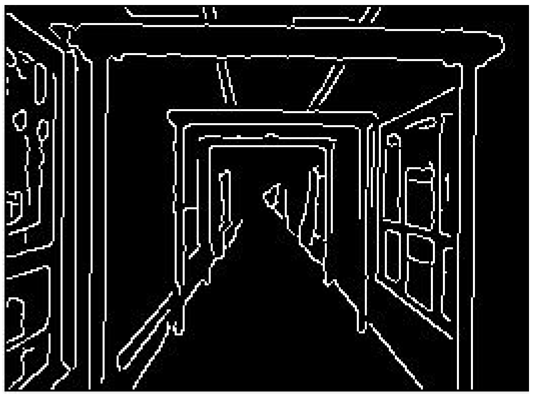
Figure 23: Corridor image with optimal threshold value T = 2.0, showing distinct details along the walls

‘New York City.jpg’

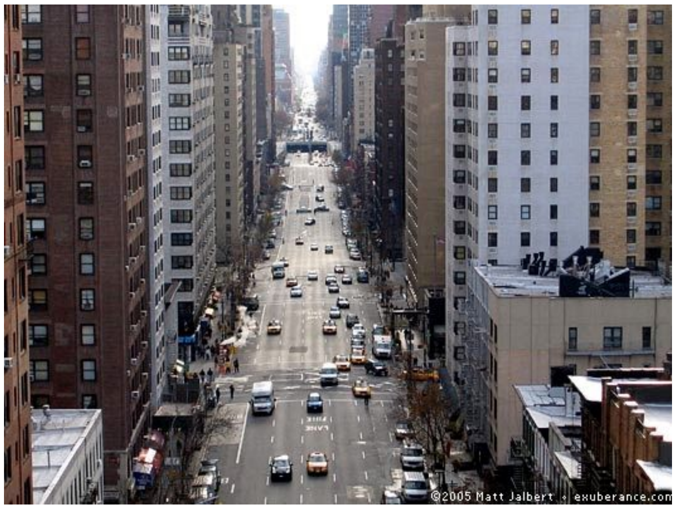

Figure 24: Original New York City image before any edge detection is applied

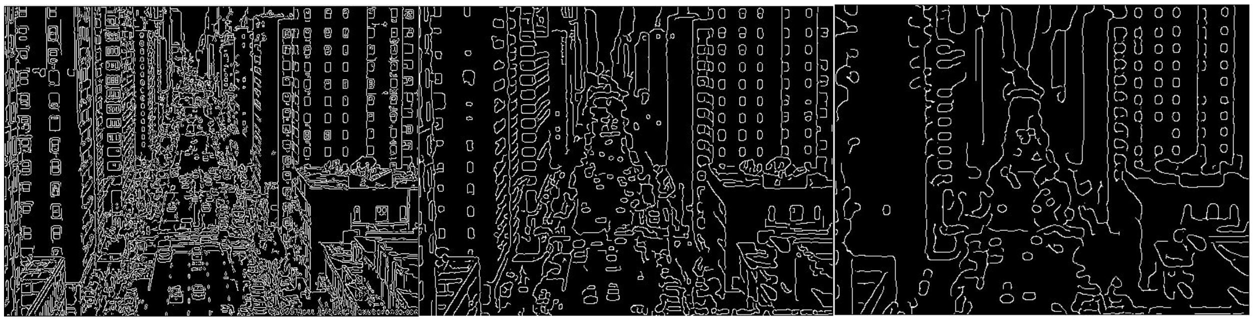
Figure 25: New York City image with σ = 0.5, 2.0, 4.0 respectively

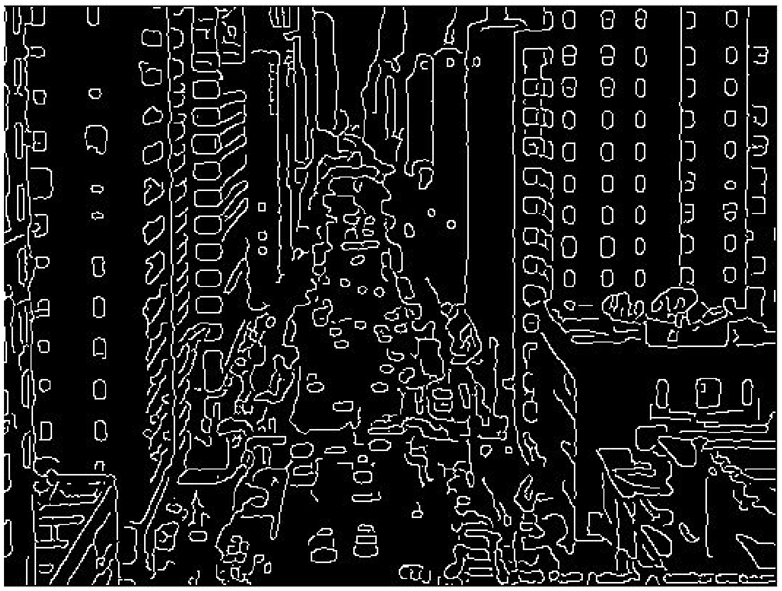
Figure 26: New York City image with optimal σ = 2.0 so that building lines and cars are still distinguishable

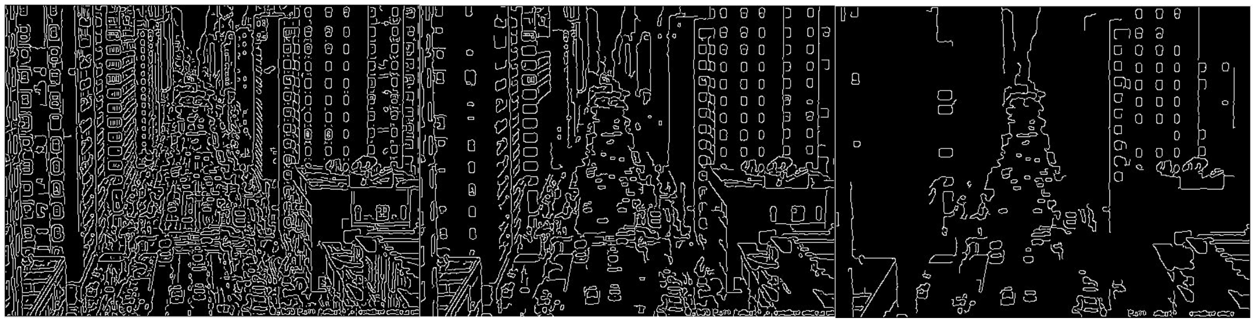
Figure 27: New York City image with threshold T = 0.5, 2.0, 4.0 respectively

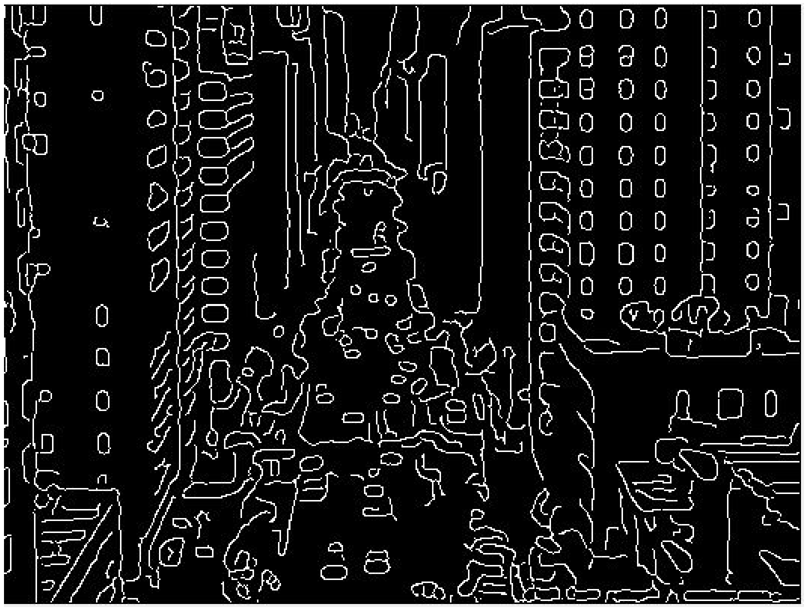
Figure 28: New York City image with optimal threshold value applied, highlighting the most important details of buildings and cars

‘bike-lane.jpg’

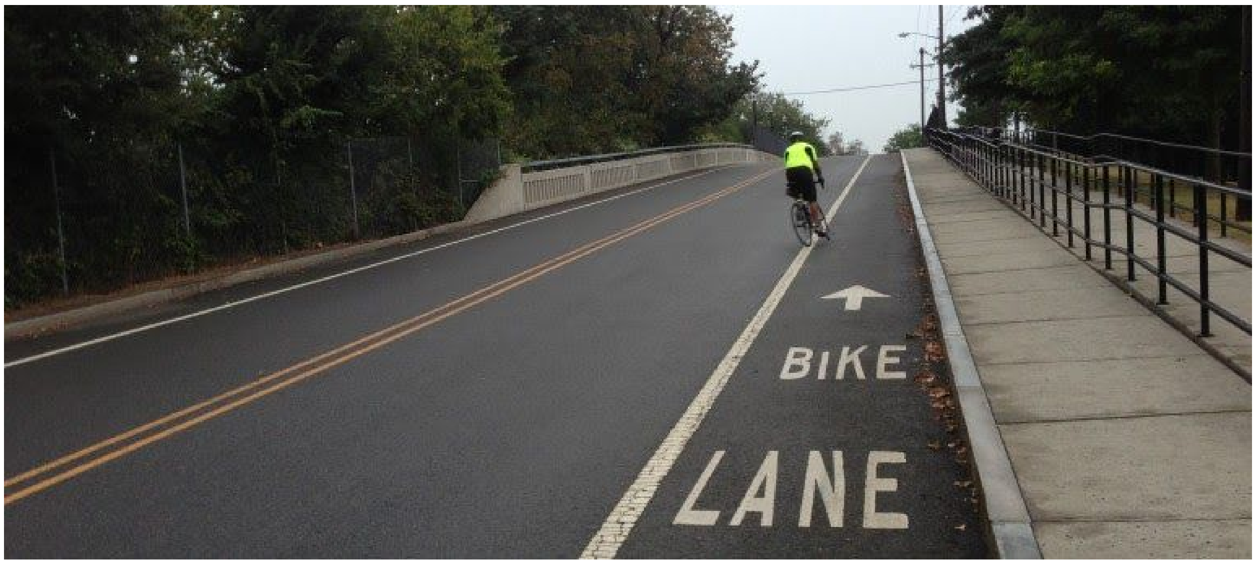

Figure 29: Original bike-lane image with no edge detection applied

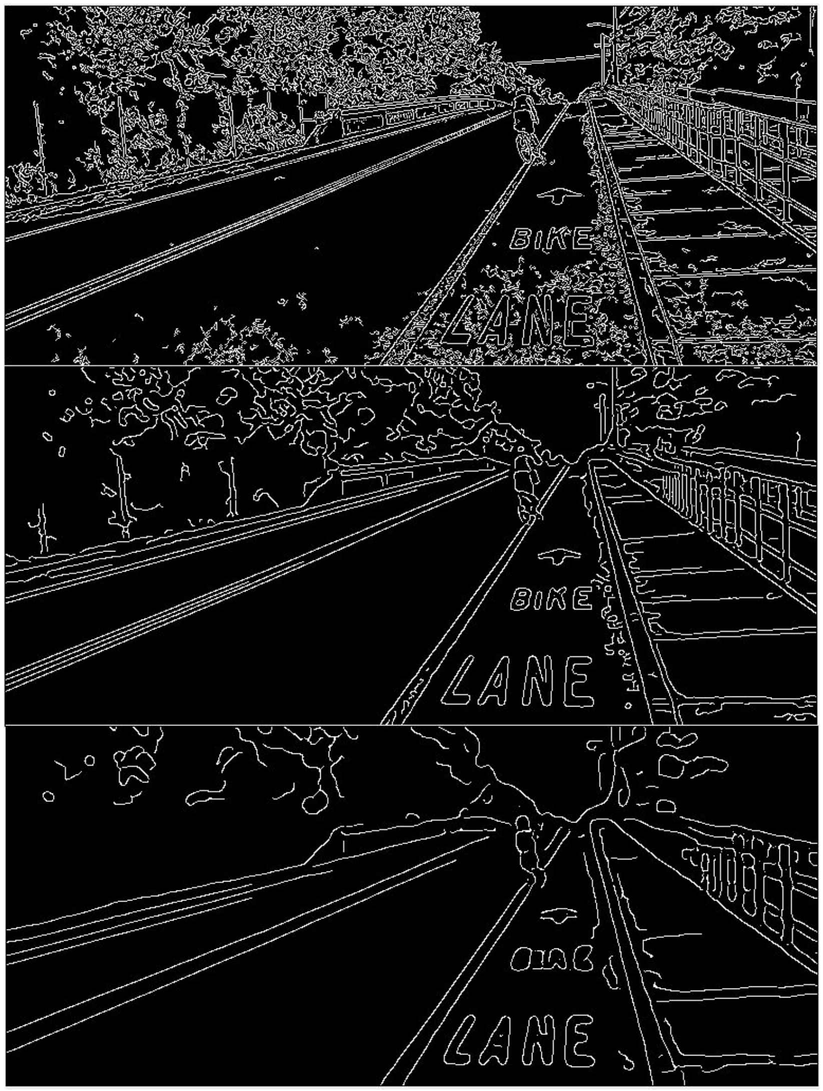

Figure 30: Bike lane image with σ = 0.5, 2.0, 4.0 respectively

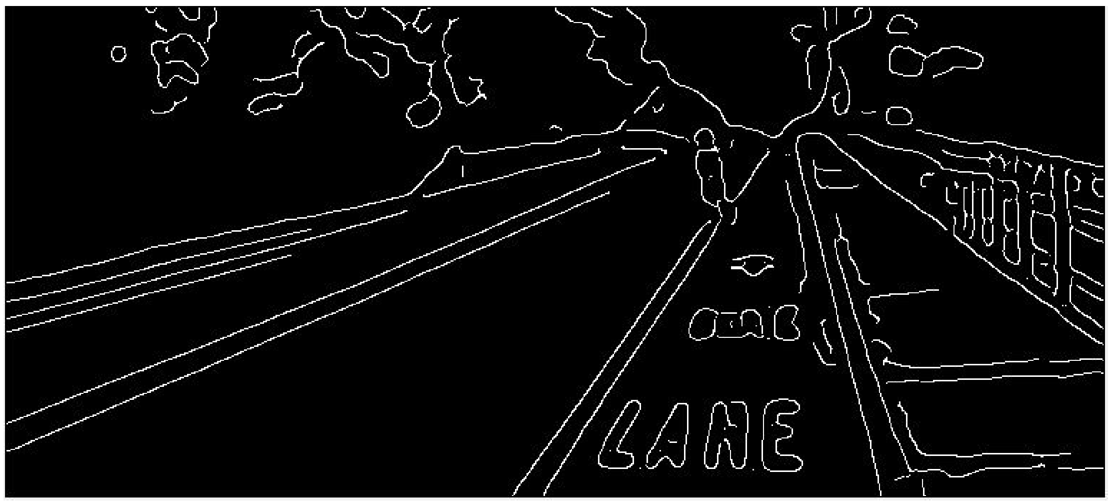

Figure 31: Bike lane image with optimal σ value 4.5 so that most background activity is negated

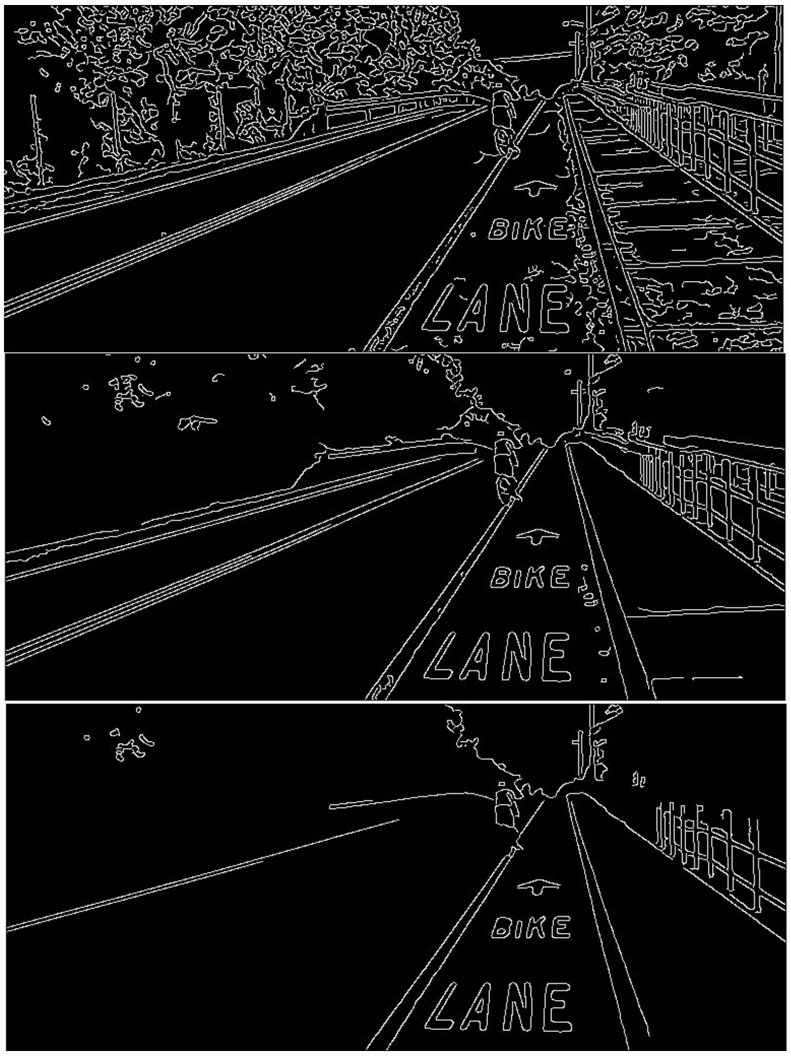

Figure 32: Bike lane image with threshold T = 0.5, 2.0, 4.0 respectively

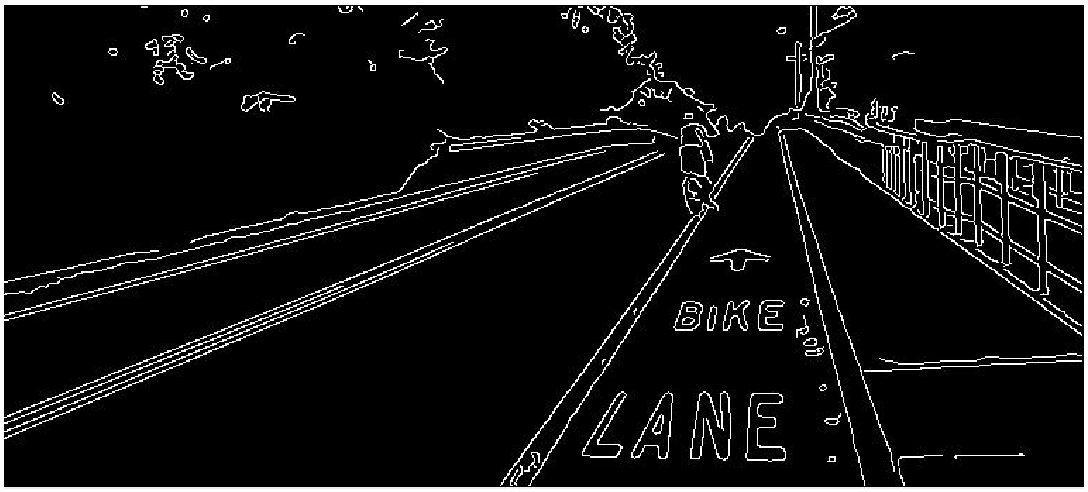

Figure 33: Bike lane image with optimal threshold T = 2.0, keeping most features

It can be seen that the sigma value has a greater effect on fine tuning the image features where as the threshold value can have a more significant effect in removing background activity and creating sharper lines. There is a balance between threshold and sigma values and maximizing either does not produce the most useful image.

Appendix - Code

Part A code

```
clc

im = imread('Boat2.tif');
figure(1);
imshow('Boat2.tif');

filtered = median_filter(im, 3);
figure(2);
imshow(filtered);

im = imread('building.gif');

figure(4);imshow(im);
filtered = gaussian_filter(im, 2, 5);
figure(5);imshow(filtered);
```

Part B code

```
clc, clear all;
close all;

im = imread('corridor.jpg');

figure(1);
imshow(im);

figure(2);
imshow(prewitt_filter(im));

figure(3);
imshow(sobel_filter(im));

%Varying sigma of corridor image

laplacian = laplacian_filter(im, 9, 1);
figure(3);
imshow(laplacian);

laplacian = laplacian_filter(im, 9, 3);
figure(4);
imshow(laplacian);

laplacian = laplacian_filter(im, 9, 5);
figure(5);
imshow(laplacian);
```

Gaussian Filter code

```
function [filtered] = gaussian_filter(im, sigma, mask_size)
%GAUSSIAN_FILTER
%Calculate the mask size
x=[-floor(mask_size / 2):1:floor(mask_size / 2)];

%Create the filters for px and py
px=1/(sqrt(2*pi)*sigma)*exp(-x.^2/(2*sigma^2));
py=px';
g = py * px;
normalizing_factor=sum(sum(g));
g = g/normalizing_factor;

%Pad the original image
im_size = size(im);
filtered = uint8(zeros(im_size));
pad_size = floor(mask_size / 2);
im = padarray(im, [pad_size pad_size], 0, 'both');

%Apply the filter to every pixel
for i = pad_size + 1:im_size(1)
    for j = pad_size + 1:im_size(2)
        neighbors = im(i-pad_size:i+pad_size, j-pad_size:j+pad_size);
        neighbors = double(neighbors);
        cmat = neighbors.*g;
        cval = sum(sum(cmat));
        filtered(i - pad_size, j - pad_size) = uint8(cval);
    end
end

end
```

Median Filter code

```
function [filtered] = median_filter(im, mask_size)
%   Detailed explanation goes here
if mod(mask_size, 2) == 0 || mask_size < 3
    return
End

% Pad the original image
im_size = size(im);
filtered = uint8(zeros(im_size));
pad_size = floor(mask_size / 2);
im = padarray(im, [pad_size pad_size], 0, 'both');

% Apply the median filter to every pixel by taking the median of
neighboring pixels
for i = pad_size + 1:im_size(1)
    for j = pad_size + 1:im_size(2)
        neighbors = im(i-pad_size:i+pad_size, j-pad_size:j+pad_size);
        neighbors = neighbors(:)';
        filtered(i - pad_size, j - pad_size) =
uint8(floor(median(neighbors)));
    end
end
```

Prewitt Operator code

```
function [filtered] = prewitt_filter(img)
im = rgb2gray(img);

%The sobel filter
px= [-1,-1,-1;0,0,0;1,1,1];
py=px';

%Create the filters for px and py and the output arrays
im_size = size(im);
gx = double(zeros(im_size));
gy = double(zeros(im_size));
filtered = double(zeros(im_size));
pad_size = floor(3 / 2);
im = padarray(im, [pad_size pad_size], 0, 'both');

%Apply the px filter to get Gx
for i = pad_size + 1:im_size(1)
    for j = pad_size + 1:im_size(2)
        neighbors = im(i-pad_size:i+pad_size, j-pad_size:j+pad_size);
        neighbors = double(neighbors);
        cmat = neighbors.*px;
        cval = sum(sum(cmat));
        gx(i - pad_size, j - pad_size) = double(cval);
    end
end

%Apply the py filter to get Gy
for i = pad_size + 1:im_size(1)
    for j = pad_size + 1:im_size(2)
        neighbors = im(i-pad_size:i+pad_size, j-pad_size:j+pad_size);
        neighbors = double(neighbors);
        cmat = neighbors.*py;
        cval = sum(sum(cmat));
        gy(i - pad_size, j - pad_size) = double(cval);
    end
end

%Compute the magnitude of the gradient
filtered = (gx.^2 + gy.^2).^0.5;
```

```
temp = filtered;
filtered = double(rescale(temp));

%Apply the threshold and output the image
filtered = filtered>0.17*max(filtered(:));

end
```

Sobel Operator code

```
function [filtered] = sobel_filter(img)

im = rgb2gray(img);

%The sobel filter
px= [-1,-2,-1;0,0,0;1,2,1];
py=px';

%Create the filters for px and py and the output arrays
im_size = size(im);
gx = double(zeros(im_size));
gy = double(zeros(im_size));
filtered = double(zeros(im_size));
pad_size = floor(3 / 2);
im = padarray(im, [pad_size pad_size], 0, 'both');

%Apply the px filter to get Gx
for i = pad_size + 1:im_size(1)
    for j = pad_size + 1:im_size(2)
        neighbors = im(i-pad_size:i+pad_size, j-pad_size:j+pad_size);
        neighbors = double(neighbors);
        cmat = neighbors.*px;
        cval = sum(sum(cmat));
        gx(i - pad_size, j - pad_size) = double(cval);
    end
end

%Apply the py filter to get Gy
for i = pad_size + 1:im_size(1)
    for j = pad_size + 1:im_size(2)
        neighbors = im(i-pad_size:i+pad_size, j-pad_size:j+pad_size);
        neighbors = double(neighbors);
        cmat = neighbors.*py;
        cval = sum(sum(cmat));
        gy(i - pad_size, j - pad_size) = double(cval);
    end
end

%Compute the magnitude of the gradient
```

```
filtered = (gx.^2 + gy.^2).^0.5;
temp = filtered;
filtered = double(rescale(temp));

%Apply the threshold and output the image
filtered = filtered>0.17*max(filtered(:));

end
```

Laplacian of Gaussian code

```
function [filtered] = laplacian_filter(img, mask_size, sig)

im = rgb2gray(img);

%Create the Laplacian of gaussian filter
lp = fspecial('log', mask_size, sig);

%Create the output array and pad the original image to add the filter
im_size = size(im);
filtered_log = double(zeros(im_size));
pad_size = floor(mask_size / 2);
im = padarray(im, [pad_size pad_size], 0, 'both');

%Apply the filter to each pixel
for i = pad_size + 1:im_size(1)
    for j = pad_size + 1:im_size(2)
        neighbors = im(i-pad_size:i+pad_size, j-pad_size:j+pad_size);
        neighbors = double(neighbors);
        cmat = neighbors.*lp;
        cval = sum(sum(cmat));
        filtered_log(i - pad_size, j - pad_size) = double(cval);
    end
end

%Create the output array
im_size = size(filtered_log);
filtered = uint8(zeros(im_size));
pad_size = floor(3 / 2);
filtered_temp = padarray(filtered_log, [pad_size pad_size], 0,
'both');

%Apply the zero crossing to the filtered image
for i = pad_size + 1:im_size(1)
    for j = pad_size + 1:im_size(2)
        neighbors = filtered_temp(i-pad_size:i+pad_size,
j-pad_size:j+pad_size);
        neighbors = double(neighbors);
        if (neighbors(1,2)*neighbors(3,2)) < 0 || (neighbors(2,1)*
neighbors(2,3)) < 0 || (neighbors(1,1) * neighbors(3,3)) < 0 ||
(neighbors(1,3) * neighbors(3,1) < 0)
```

```
            filtered(i - pad_size, j - pad_size) = 255;
        else
            filtered(i - pad_size, j - pad_size) = 0;
        end
    end
end

end
```