Computer Vision Laboratories

Project 2

EE/CPE 428 - 03

Computer Vision

Winter 2023

Group 4

Students: Addison Sandvik, Matthew Palma, Nathan Jaggers

Project Due: 02/02/23

Instructor: Dr. Zhang

Part A – Image Filtering

Assignment

In the following you will find two imperfect images (electronic versions can be found on Canvas). First analyze these images and discuss ways to find out problems in them. Then use image filtering techniques covered in class to reduce noise in these images. Display your filtered images and submit it along with the report. It should include the technique(s) you applied, comparisons of different techniques you used (include linear and nonlinear filtering, and masks of different sizes), if applied, and a discussion on the results. Please write your own code (i.e. no MATLAB built-in function) to implement image filtering.

Procedure

Use words/sample_code to describe the procedures (How you did it). Explain the steps taken if needed.

First, we wrote a function, myfilter(), that takes a grayscale image, the name of a filter type, and the size of the filter and returns the filtered image. For a Gaussian filter, a sigma value must also be supplied.

```
function [filtered_image]=myfilter(image, f_mode, f_size, f_sigma)
% image must be a 2-dimensional matrix
% f_mode is the type of filter:
%   -'average'
%   -'gaussian'
%   -'median'
% f_size is the x and y dimensions of the square filter kernel
% f_sigma is a parameter used in the gaussian filter
%
% Note: Dimensions of image must be larger than f_size
```

For the averaging and Gaussian filters, the filter kernel is generated using the filter parameters. This is straightforward for the averaging filter, but the Gaussian kernel is generated with fspecial(). To convolve the image with the filter, each pixels' neighborhood is isolated. For the averaging and gaussian filters, the neighborhood is multiplied by the kernel values and summed. For the median filter, the pixel is reassigned to the median value in the neighborhood. Pixels near the edge where the kernel does not have a complete neighborhood are set to zero.

For both the 'Boat2.tif' and 'building.gif' images, each filter with different kernel sizes is applied and the results for each size are combined into one image.

Results

The following results show the effects of different filters and sizes on an image with the original in the top left, 3x3 in the top right, 5x5 in the bottom left, and 7x7 in the bottom right.

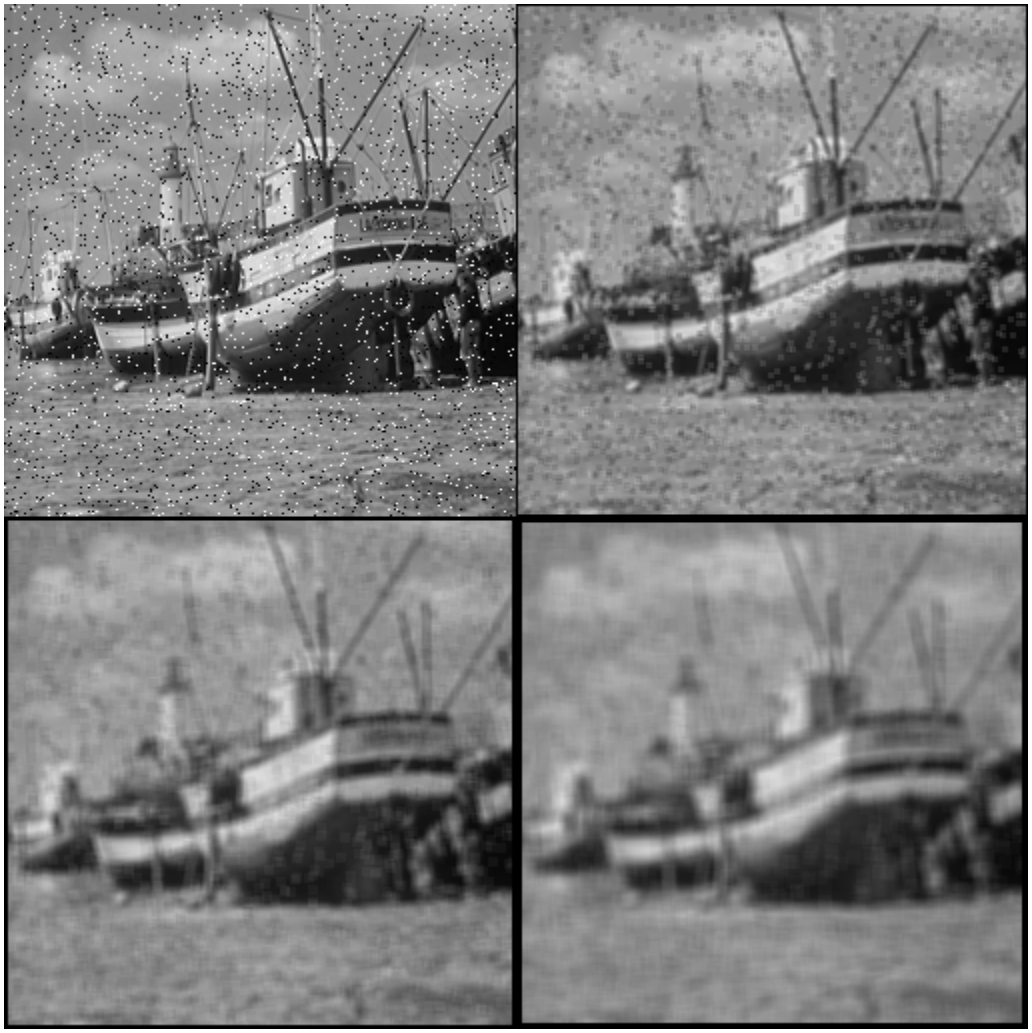

Figure 1: Boat with averaging filters of different sizes

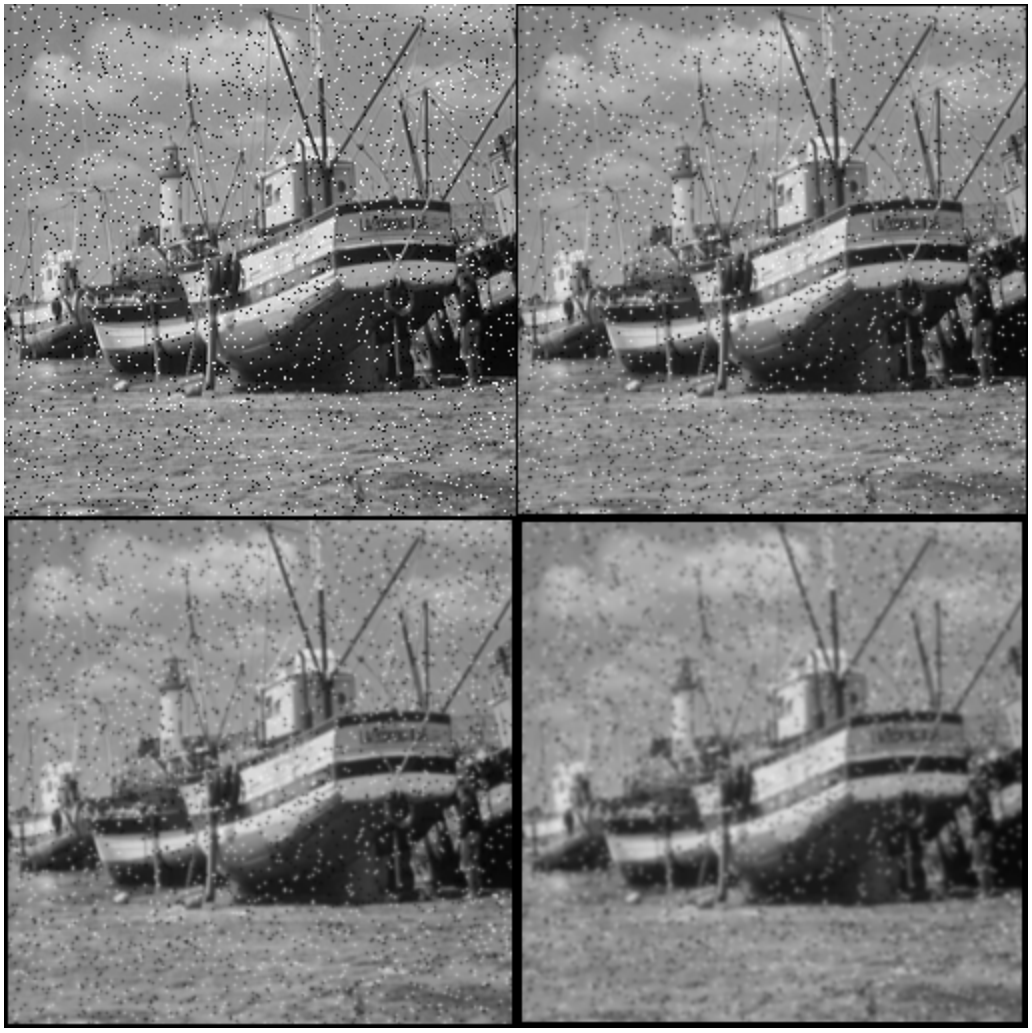

Figure 2: Boat with Gaussian filter of different sizes

Figure 3: Boat with median filter of different sizes

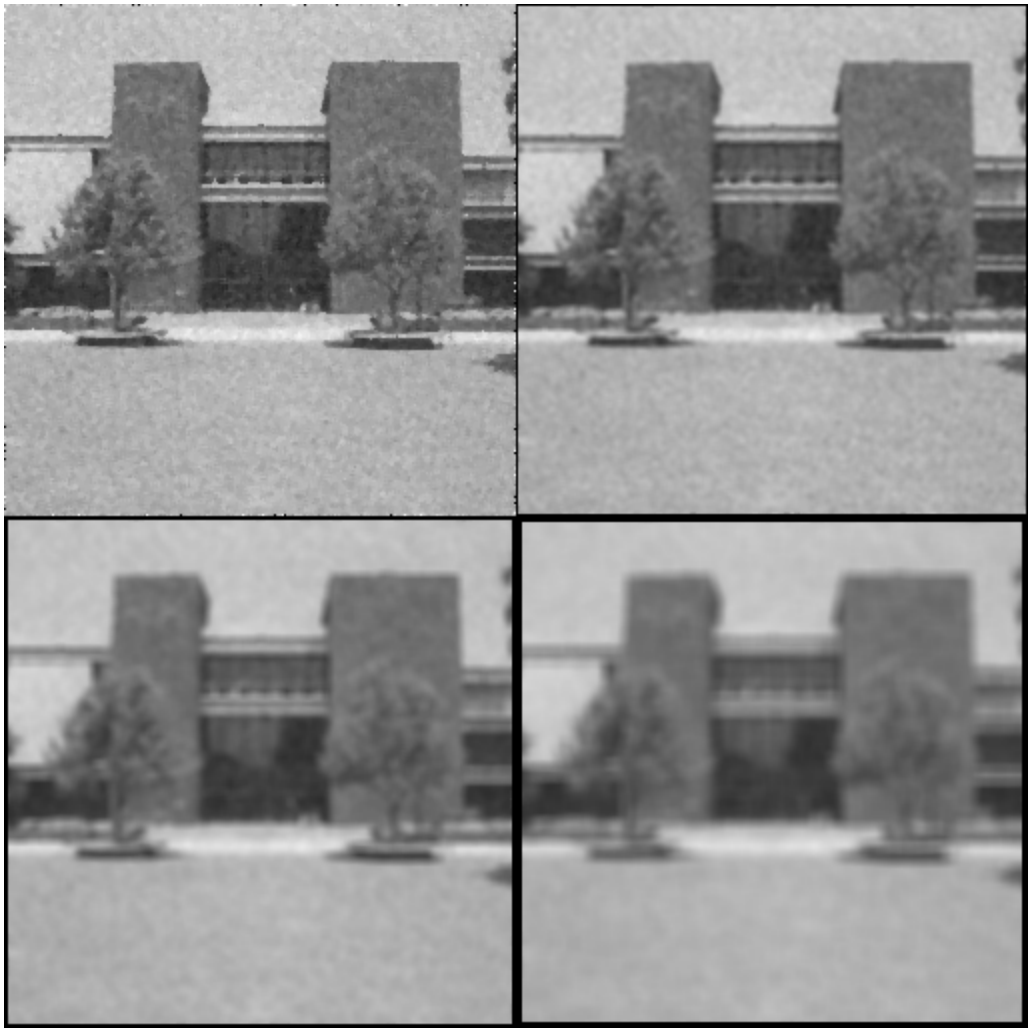

Figure 4: Building with averaging filter of different sizes

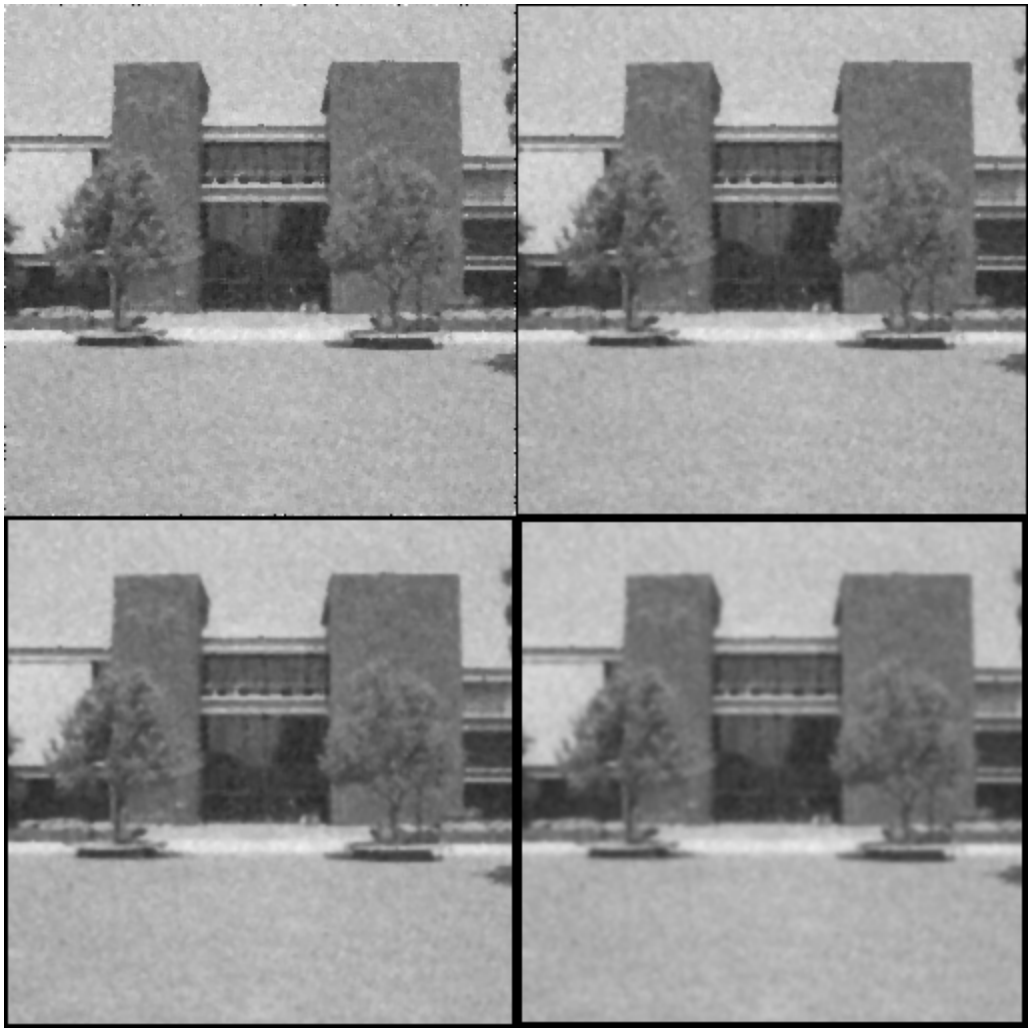

Figure 5: Building with gaussian filter of different sizes

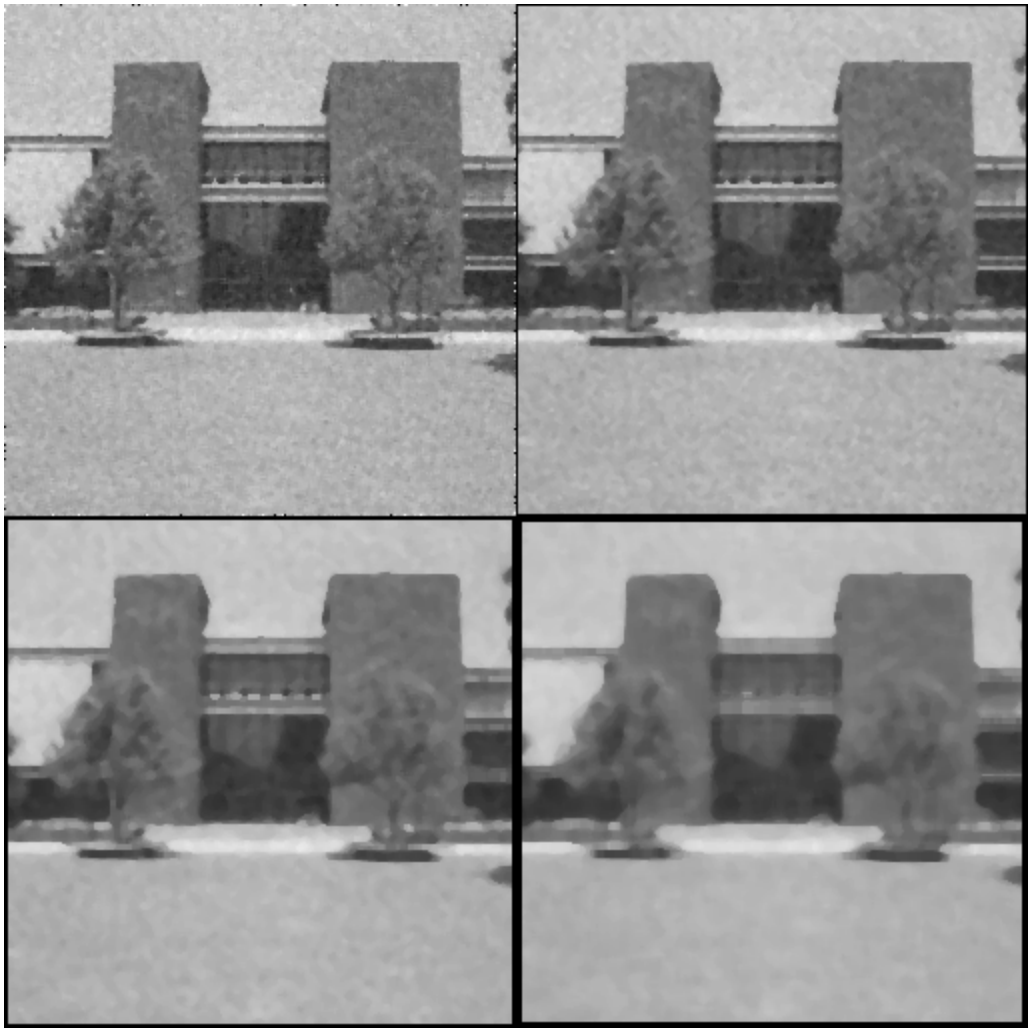

Figure 6: Building with median filter of different sizes

Analysis

The image of the boat has much salt and pepper noise, which is best resolved with the 3x3 median filter. Increasing the size of the filter does not increase its effectiveness in this case, as details begin to be lost with the 5x5 and 7x7 filters. The linear filters do not handle this kind of noise well when compared to the median filter.

The building image has Gaussian noise, which is best resolved with the Gaussian filter. In this case, the averaging and median filters must be large to be effective, causing the details to be lost. For the building, the best size for the Gaussian filter is 5x5, because it balances noise removal while preserving the details.

Part B – Edge Detection

Assignment

Test your edge detectors on the following images (electronic versions on Canvas). For color images, convert them to gray-scale images before edge detection.

1. *Implement the Prewitt Operator and the Sobel Operators. Discuss which operator gives better results. In your report, include the following for one test image.*
    a. *The original image.*
    b. *Compute and display the sum of squared gradient magnitude.*
    c. *Select a threshold that produces the best edge map and display it.*
2. *Implement the LoG Operator. Consider the masks approximating the LoG with σ = 0.5, 1, and 1.2. Discuss the effects of mask size on the output. In your report, include the following for one test image.*
    a. *The original image*
    b. *Output image from LoG with various values of σ c. Determine the σ value that produces the best edge map.*
3. *Apply the Canny Edge Detector to the images. You can use MATLAB built-in function for this step. Discuss the effects of threshold and σ on the output.*
    a. *The original image*
    b. *Output image with several threshold and σ values for one test image, then show the best results for each of the four test images.*
4. *Using outputs from the optimal operator from the above edge detectors, apply the Hough Transform to extract lines from the test images. You can use MATLAB built-in function for this step. Discuss your method for selecting likely lines from the Hough Space. In your report, include results for all four test images.*
    a. *the original image*
    b. *image of detected lines overlaid onto original image*
5. *Note: You can earn extra credit if you implement your own Canny Edge Detector and Hough Transform.*

Procedure

Use words/sample_code to describe the procedures (How you did it). Explain the steps taken if needed.

1. Prewitt and Sobel
    a. The Prewitt and Sobel Operators for the x and y directions are added to the myfilter() function with their kernels hard coded, allowing myfilter() to produce the gradients Gx and Gy.
    b. The gradients are combined to produce a single edge map, which is then thresholded to isolate the strong edges.

2. Laplacian of Gaussian
    a. The LoG Operator is added to the myfilter() function where fspecial() is used to generate the kernel.
    b. After applying the operator, the zero crossings are isolated by investigating the 3x3 neighborhood around each pixel. If there is a sign change between opposite neighbors (e.g. top left corner and bottom right corner) and the magnitude of the center pixel is less than both magnitudes of the opposite neighbors, that pixel is considered and edge and set to one. This results in a binary edge map where the edges are one pixel thick.
    c. The sigmas of 0.5, 1, and 1.2 are compared.
    d. To investigate the effects of the LoG parameters, the sigma values of 1, 2, 3, and 4 are shown side by side in two images with different mask sizes. The first has a large mask size of 31x31, and the second has a small mask size of 9x9.
3. Canny Edge Detector
    a. Canny edge maps were created using the built-in Matlab function edge() and specifying ‘canny’ as one of the arguments. With no other arguments, the function will pick high and low threshold values automatically and the default sigma value used is sqrt(2).
    b. Threshold and sigma values were varied and the best results acquired for each image are shown under “Canny and Hough” in the result section.
    c. Additionally, a to compare to the built-in Matlab canny edge detector was created. First, it takes an image converts it to grayscale, and smooths it with a Gaussian filter. Then using a Sobel filter the gradient magnitude and direction are extracted so the non-maximum suppression can be applied. Finally, hysteresis thresholding is applied to connect stronger edges to related weaker edges.
4. Hough Transform
    a. Using edge maps generated previously and the built-in function for Hough Transforms, lines from images were extracted and mapped to a Hough matrix.
    b. To display the various images along with the lines detected from the Hough Transform, the built-in functions houghpeaks() and houghlines() were used.

Results

Prewwit and Sobel

Figure 7: Original Bike Image

Figure 8: Prewitt edge map

Figure 9: Sobel edge map

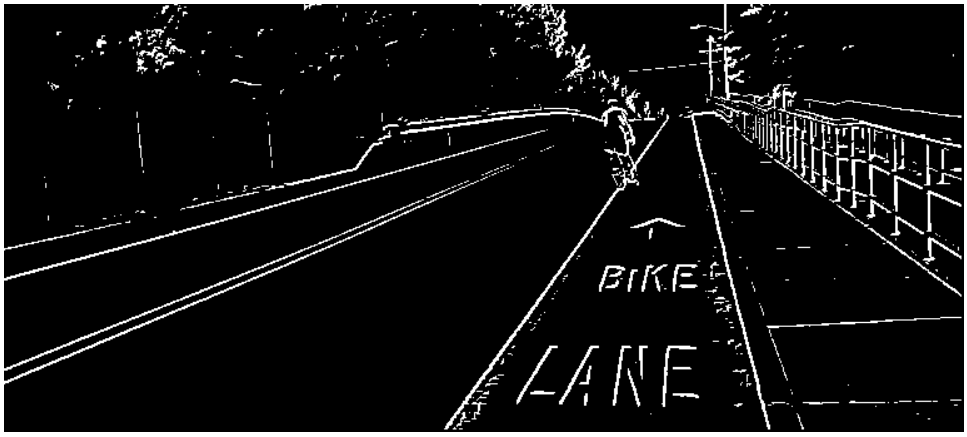

Figure 10: Prewitt thresholded edge map. Threshold = 90

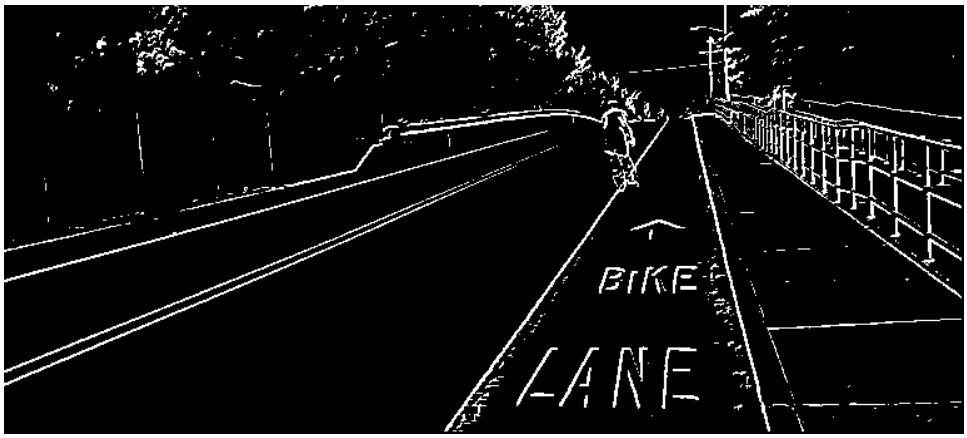

Figure 11: Sobel thresholded edge map. Threshold = 120

LoG

Figure 12: Original Corridor Image

Figure 13: LoG edge maps with filter size of 9x9 and sigmas of 0.5, 1, and 1.2

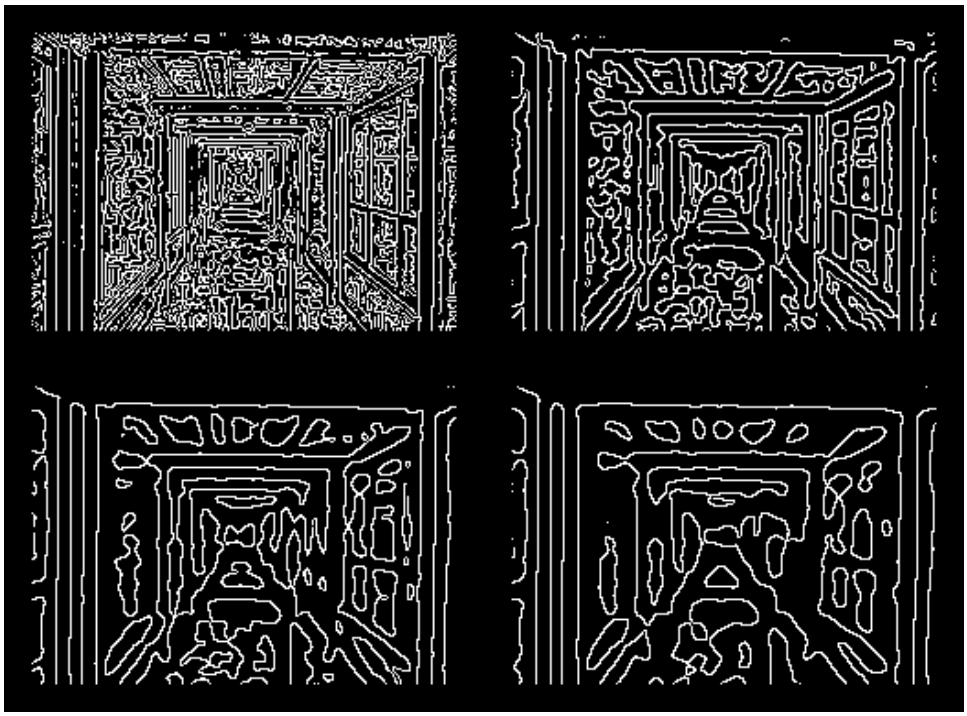

Figure 14: LoG edge maps with filter size of 31x31 and sigmas of 1, 2, 3, and 4.

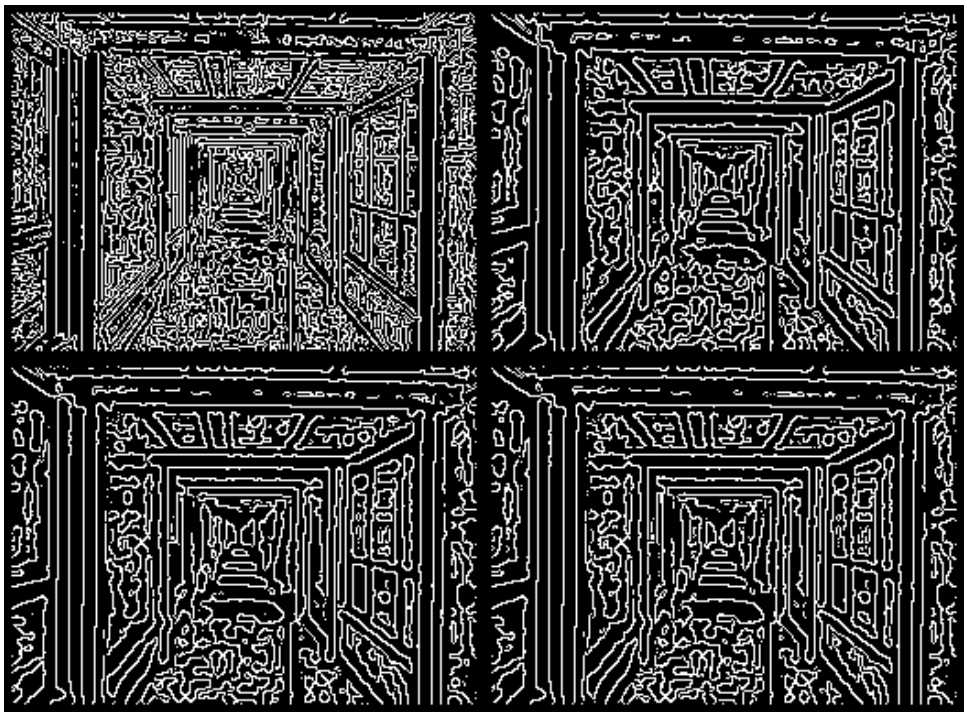

Figure 15: LoG edge maps with filter size of 9x9 and sigmas of 1, 2, 3, and 4.

Canny and Hough

Figure 16: Original NYC Image

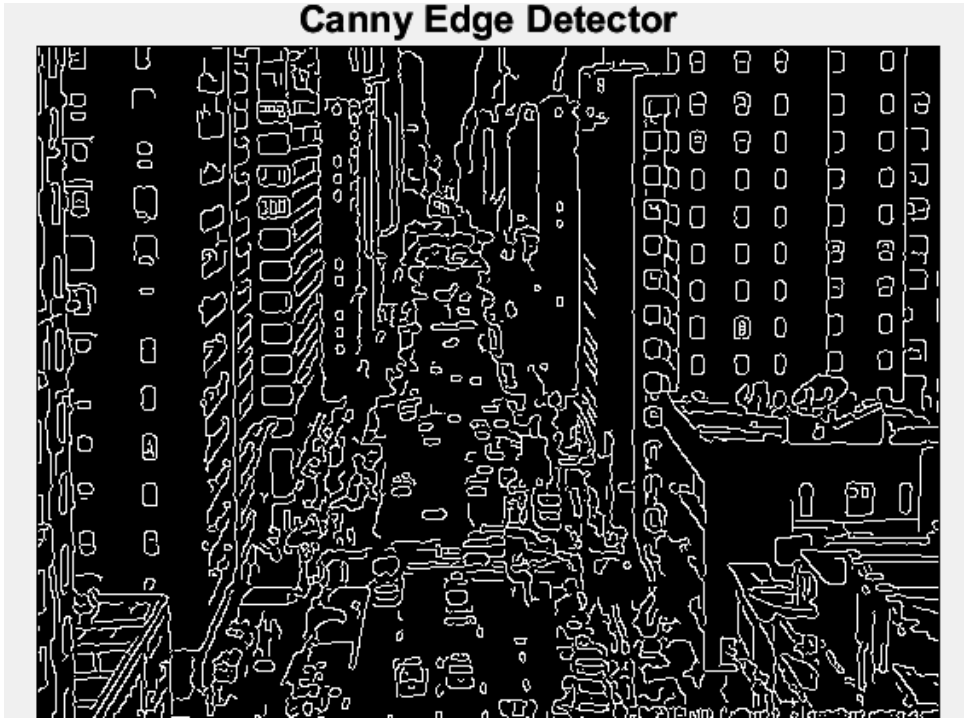

Figure 17: Canny Edge Detector Applied to NYC Image

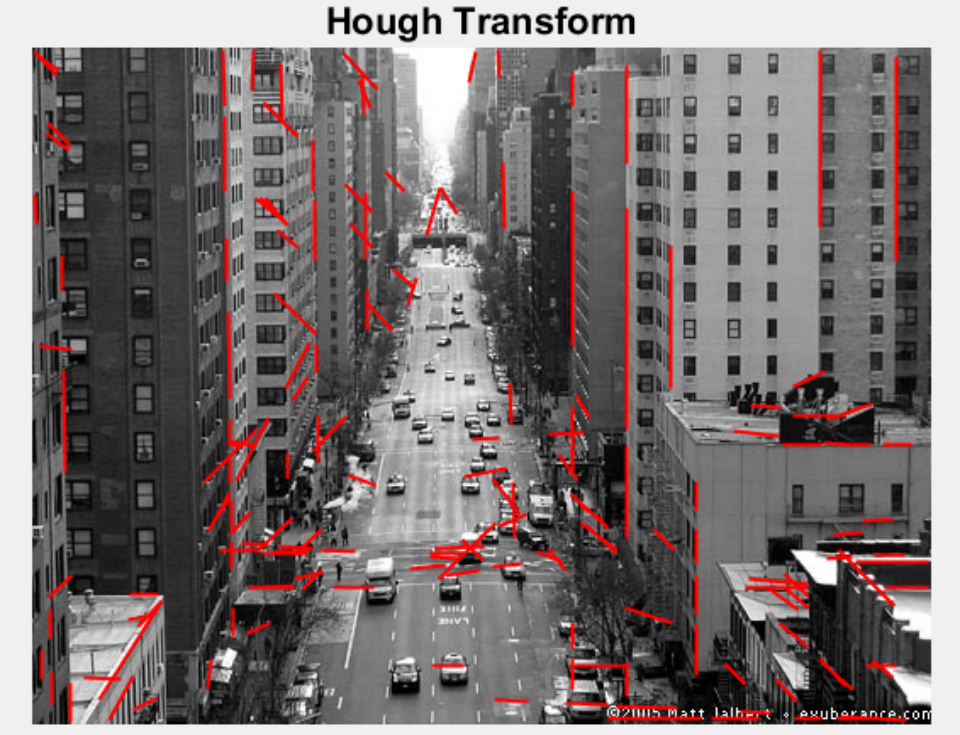

Figure 18: Hough Transform Applied to NYC Image

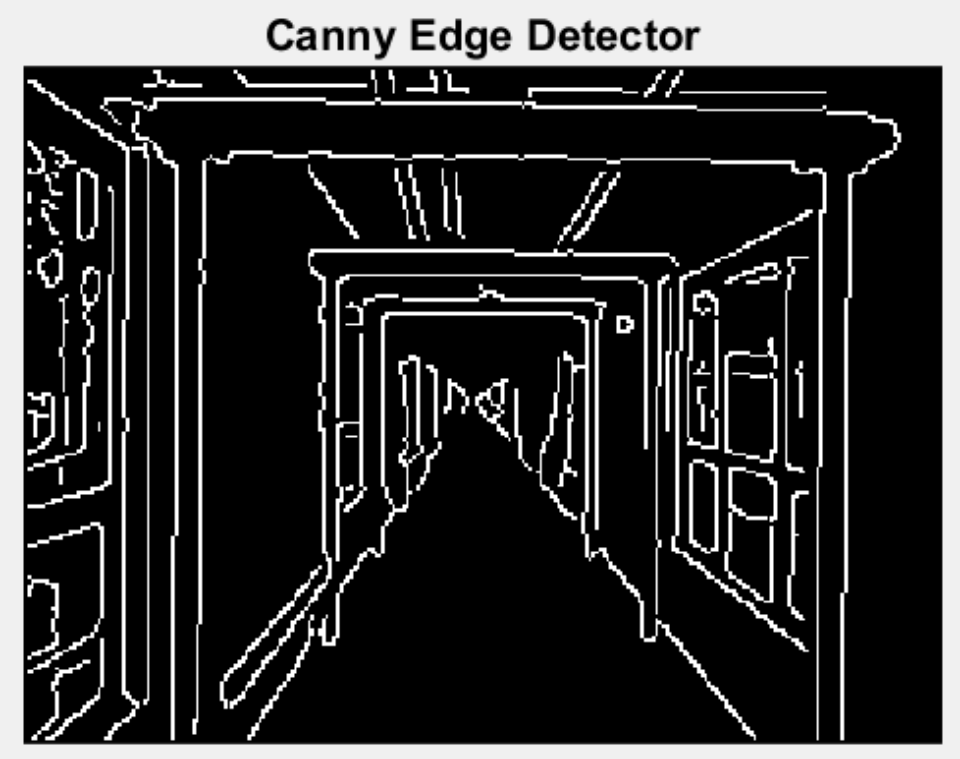

Figure 19: Canny Edge of the Hall image

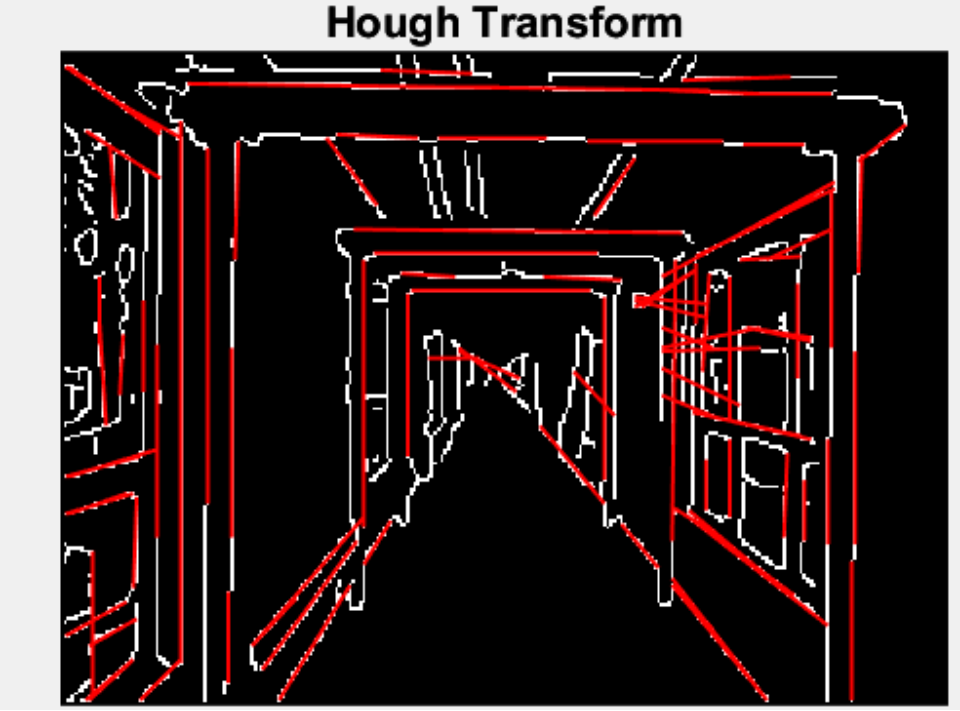

Figure 20: Hough Transform Applied to Corridor

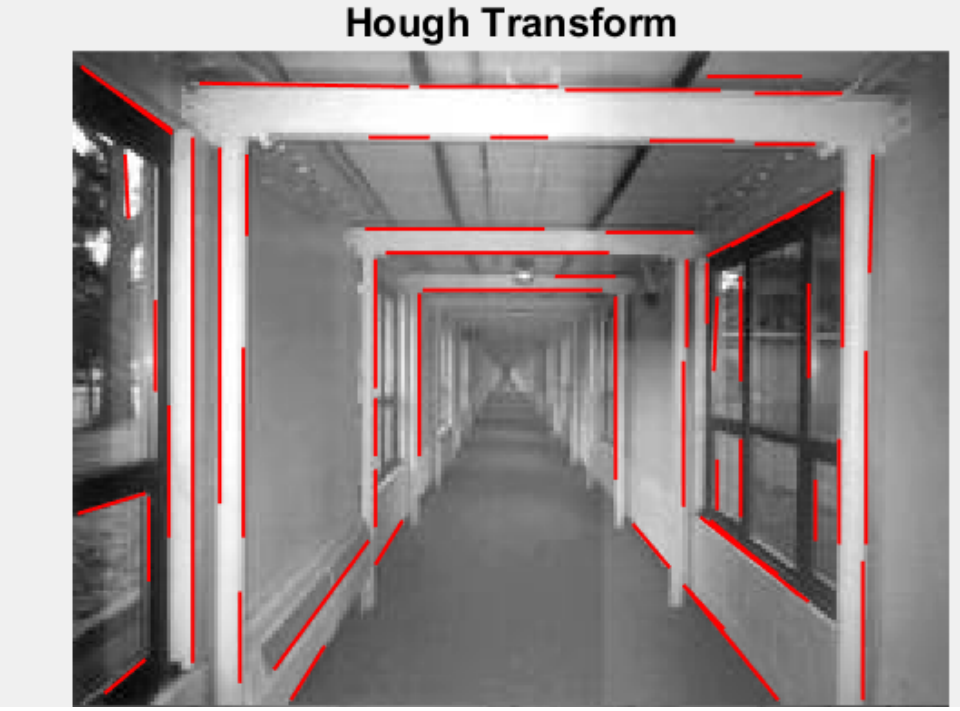

Figure 21: Hall Image with FIllgap of 2

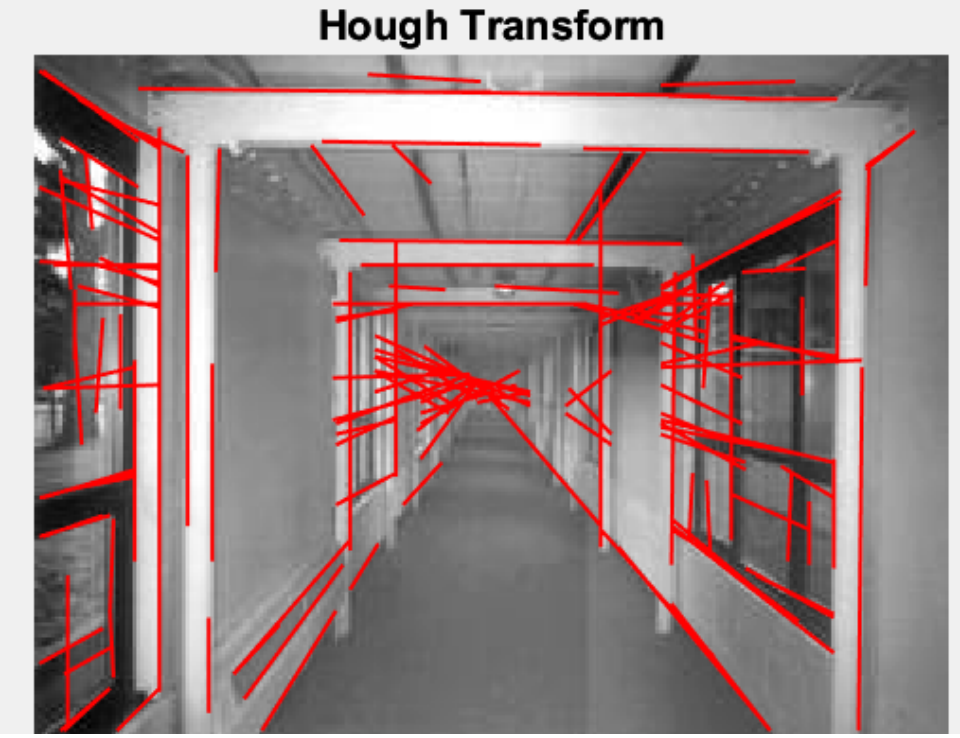

Figure 22: Hall Image with FIllgap of 7

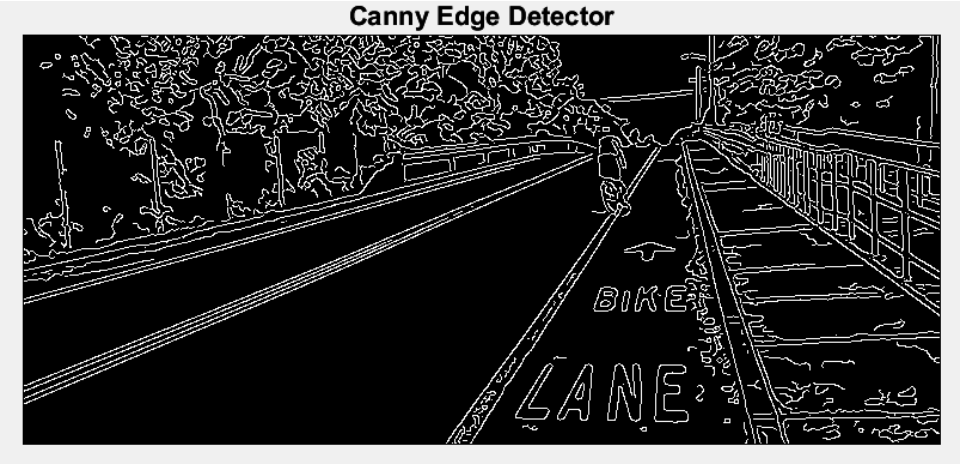

Figure 23: Canny Image of Bike lane

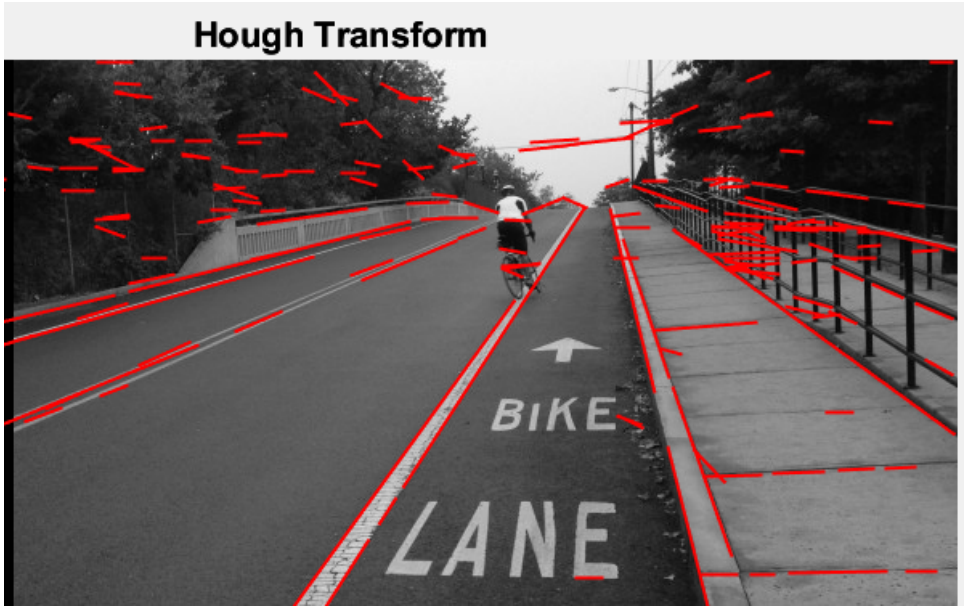

Figure 24: Hough Transform of Bike Lane Image FillGap of 5

Figure 25: Original Window Image

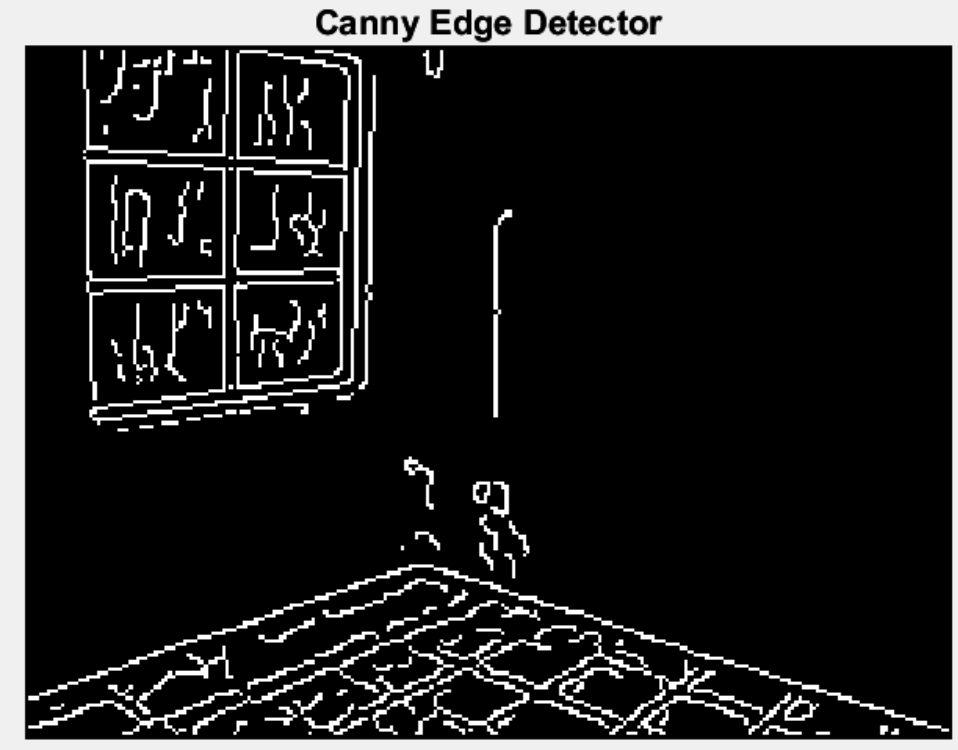

Figure 26: Canny Edge of Corner Image

Figure 27: Hough Transform of Corner Image FillGap of 5

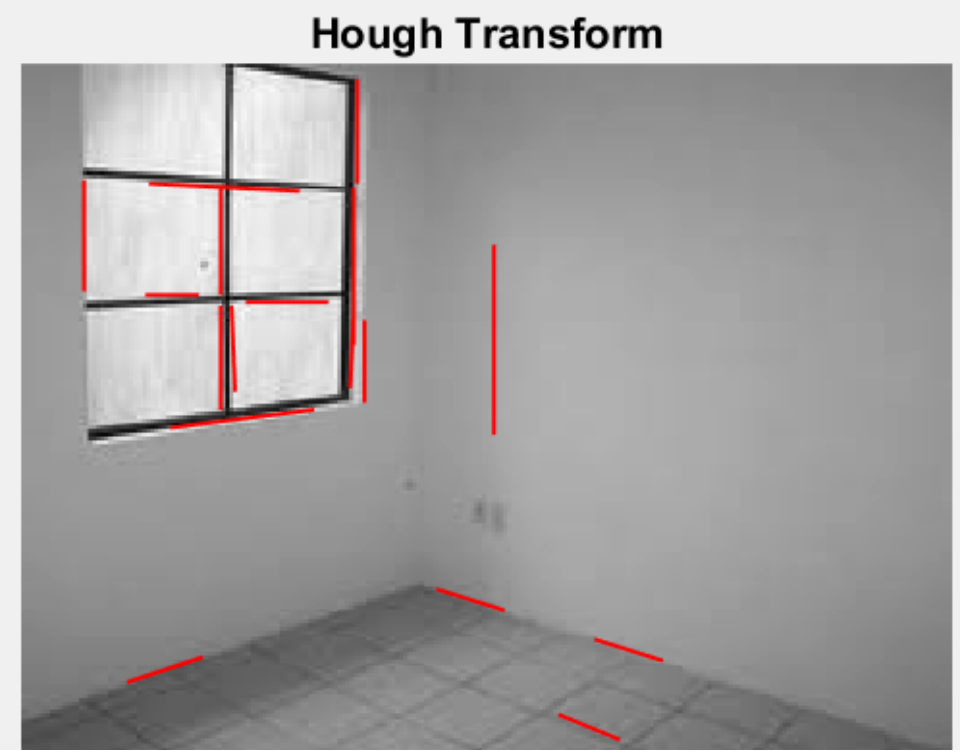

Figure 28: Hough Transform of Corner Image FillGap of 2

Analysis

Evaluate the results. Use words to describe the results and provide explanations if needed.

1. The differences between the Prewitt and Sobel edge maps are minimal. It is not clear whether one has an advantage over the other. The main difference is the threshold required to create similar edge maps.
2. The LoG edge maps are very susceptible to noise, and increasing the sigma value tends to improve the result. With the corridor photo, the suggested sigma values of 0.5, 1, and 1.2 are too small, so Figures 14 and 15 show larger sigma values at different mask sizes. Larger masks result in more lost pixels around the edge, but smaller masks can restrict the LoG operator's sigma parameter.
3. The Canny Edge is a fairly straightforward operation. They can be set to be very sensitive, where the reflection in a window or shadows are called lines.
4. The Hough Transform relies on the canny edge detector to generate lines. Changing the FillGAP and minLen parameters changes the resulting lines(in red) that are generated. If the Canny Edge detector is implemented properly, the Hough Detector can be fairly accurate, but it also has to be adjusted for each image that it processes.

Teamwork

Addison

- Part A
    - procedure, results, and analysis
- Part B
    - Prewitt and Sobel: procedure, results, and analysis
    - LoG: procedure, results, and analysis
- Appendix
    - Part A Script File
    - myfilter()
    - Part B Script File
    - edges_ps()
    - edges_log()

Matthew

- Part B
    - Canny Transform
        - Built in function
        - Extra credit attempt
    - Hough Transform
- Document Formatting

Nathan

- Set up Document
- Did Part B exercise separately and compared results
- Part B
    - Canny: procedure
    - Hough: procedure
- Document formatting and review
    - Small grammar/spelling corrections
    - Adding missing pictures/elements

Appendix

Part A Script File

```
% Boat with 3x3 filters
boat_orig = imread('Boat2.tif');
boat_avg = myfilter(boat_orig, 'average', 3);
boat_gau = myfilter(boat_orig, 'gaussian', 3, 0.5);
boat_med = myfilter(boat_orig, 'median', 3);
boat_array = [boat_orig boat_avg; boat_gau boat_med];
imtool(boat_array);

% Boat with 5x5 filters
boat_avg = myfilter(boat_orig, 'average', 5);
boat_gau = myfilter(boat_orig, 'gaussian', 5, 0.7);
boat_med = myfilter(boat_orig, 'median', 5);
boat_array = [boat_orig boat_avg; boat_gau boat_med];
imtool(boat_array);

% Boat with 7x7 filters
boat_avg = myfilter(boat_orig, 'average', 7);
boat_gau = myfilter(boat_orig, 'gaussian', 7, 1);
boat_med = myfilter(boat_orig, 'median', 7);
boat_array = [boat_orig boat_avg; boat_gau boat_med];
imtool(boat_array);

% Building with 3x3 filters
build_orig = imread('building.gif');
build_avg = myfilter(build_orig, 'average', 3);
build_gau = myfilter(build_orig, 'gaussian', 3, 0.5);
build_med = myfilter(build_orig, 'median', 3);
build_array = [build_orig build_avg; build_gau build_med];
imtool(build_array);

% Building with 5x5 filters
build_avg = myfilter(build_orig, 'average', 5);
build_gau = myfilter(build_orig, 'gaussian', 5, 0.7);
build_med = myfilter(build_orig, 'median', 5);
build_array = [build_orig build_avg; build_gau build_med];
imtool(build_array);

% Building with 7x7 filters
build_avg = myfilter(build_orig, 'average', 7);
build_gau = myfilter(build_orig, 'gaussian', 7, 1);
build_med = myfilter(build_orig, 'median', 7);
build_array = [build_orig build_avg; build_gau build_med];
imtool(build_array);
```

Function: myfilter()

```
function [filtered_image]=myfilter(image, f_mode, f_size, f_sigma)
% image must be a 2-dimensional matrix
% f_mode is the type of filter:
%   -'average'
%   -'gaussian'
%   -'median'
%   -'prewittx'
%   -'prewitty'
%   -'sobelx'
%   -'sobely'
%   -'log'
% f_size is the x and y dimensions of the square filter kernel
% sigma is a parameter used in the gaussian and log filters
%
% Note: Dimensions of image must be larger than f_size

[i_rows, i_cols] = size(image);

% Edge detecting filters are all 3x3
if (strcmp(f_mode, 'prewittx') || strcmp(f_mode, 'prewitty') || ...
    strcmp(f_mode, 'sobelx')  || strcmp(f_mode, 'sobely'))
    f_size = 3;
end

% Edge pixels of image will be omitted from new image.
% trim holds number of pixels from edge that are ignored.
trim = (f_size-1)/2;

% Start and end indicies of new image
col_start = trim+1;
col_end = i_cols-trim;
row_start = trim+1;
row_end = i_rows-trim;

kernel = zeros(f_size, f_size);

switch f_mode
    case 'average'
        kernel = fspecial('average', f_size);
    case 'gaussian'
        kernel = fspecial('gaussian', f_size, f_sigma);
    case 'median'
        % Do nothing here
    case 'prewittx'
        kernel = [-1 -1 -1; 0 0 0; 1 1 1];
    case 'prewitty'
        kernel = [-1 0 1; -1 0 1; -1 0 1];
    case 'sobelx'
        kernel = [-1 -2 -1; 0 0 0; 1 2 1];
    case 'sobely'
        kernel = [-1 0 1; -2 0 2; -1 0 1];
    case 'log'
        kernel = fspecial('log', f_size, f_sigma);
    otherwise
```

```
        error('Unsupported filter mode');
end

image = double(image);
%filtered_image = uint8(zeros(i_rows-2*tb_trim, i_cols-2*lr_trim));

if (strcmp(f_mode, 'log'))
    filtered_image = double(zeros(i_rows, i_cols));
else
    filtered_image = uint8(zeros(i_rows, i_cols));
end

f_col = col_start;
for C=col_start:1:col_end
    f_row = row_start;
    for R=row_start:1:row_end

        % Extract neighborhood from image
        neighbors = image(R-trim:R+trim,C-trim:C+trim);

        switch f_mode
            case 'average'
                pixel = sum(neighbors.*kernel, 'all');
            case 'gaussian'
                pixel = sum(neighbors.*kernel, 'all');
            case 'median'
                pixel = median(neighbors, 'all');
            case 'prewittx'
                pixel = sum(neighbors.*kernel, 'all');
            case 'prewitty'
                pixel = sum(neighbors.*kernel, 'all');
            case 'sobelx'
                pixel = sum(neighbors.*kernel, 'all');
            case 'sobely'
                pixel = sum(neighbors.*kernel, 'all');
            case 'log'
                pixel = double(sum(neighbors.*kernel, 'all'));
            otherwise
                error('Unsupported filter mode');
        end

        if (strcmp(f_mode, 'log'))
            filtered_image(f_row, f_col) = double(pixel);
        else
            filtered_image(f_row, f_col) = uint8(pixel);
        end

        f_row = f_row+1;
    end
    f_col = f_col+1;
end
end
```

Part B Script File

```
% Part 1: Prewitt and Sobel
edges_ps('bike-lane.jpg', 'prewitt', 90);
edges_ps('bike-lane.jpg', 'sobel', 120);

% Part 2: Laplacian of Gaussian
log_1 = edges_log('corridor.jpg', 9, 0.5);
log_2 = edges_log('corridor.jpg', 9, 1);
log_3 = edges_log('corridor.jpg', 9, 1.2);
log_4 = zeros(size(log_3));

imtool([log_1 log_2; log_3 log_4], []);

log_1 = edges_log('corridor.jpg', 31, 1);
log_2 = edges_log('corridor.jpg', 31, 2);
log_3 = edges_log('corridor.jpg', 31, 3);
log_4 = edges_log('corridor.jpg', 31, 4);

imtool([log_1 log_2; log_3 log_4], []);

log_1 = edges_log('corridor.jpg', 9, 1);
log_2 = edges_log('corridor.jpg', 9, 2);
log_3 = edges_log('corridor.jpg', 9, 3);
log_4 = edges_log('corridor.jpg', 9, 4);

imtool([log_1 log_2; log_3 log_4], []);
```

Function: edges_ps() (generates and shows Prewitt and Sobel edge maps)

```
function edges_ps(image, mode, threshold)
% image must be a 2-dimensional matrix
% mode is the type of operator:
%   -'prewittx'
%   -'sobelx'
% threshold is an 8 bit value

img = imread(image);
img = rgb2gray(img);

switch mode
    case 'prewitt'
        Gx = myfilter(img, 'prewittx');
        Gy = myfilter(img, 'prewitty');
    case 'sobel'
        Gx = myfilter(img, 'sobelx');
        Gy = myfilter(img, 'sobely');
    otherwise
        error('mode must be "prewitt" or "sobel"');
end

% Calculate magnitude
edges = ((double(Gx).^2) + (double(Gy).^2)).^0.5;

% Isolate strong edges
threshold = (edges > threshold).*255;

% All images together
% blank = ones(size(img)).*255;
% imArray = [img edges; threshold blank];
% imshow(imArray);

% Seperate images
imshow(img);
imshow(edges, []);
imshow(threshold);

end
```

Function: edges_log()

```
function [edges]=edges_log(image, f_size, f_sigma)
% image must be a 2-dimensional matrix
% f_size is the x and y dimensions of the square filter kernel
% f_sigma is a parameter used in the gaussian and log filters
%
% edges is a binary edge map


img = imread(image);
img = rgb2gray(img);
[i_rows, i_cols] = size(img);

% initialize output
edges = zeros(size(img));

% Start and end indicies of edge map
col_start = 2;
col_end = i_cols-1;
row_start = 2;
row_end = i_rows-1;

% Filter with log
log = myfilter(img, 'log', f_size, f_sigma);

% Find zero grossings
f_col = col_start;
for C=col_start:1:col_end
    f_row = row_start;
    for R=row_start:1:row_end

        % Extract neighborhood from image
        % nw |  n  | ne
        %  w | cen | e
        % sw |  s  | se
        neighbors = log(R-1:R+1,C-1:C+1);
        nw = neighbors(1,1);
        n = neighbors(1,2);
        ne = neighbors(1,3);
        w = neighbors(2,1);
        cen = neighbors(2,2);
        e = neighbors(2,3);
        sw = neighbors(3,1);
        s = neighbors(3,2);
        se = neighbors(3,3);

        % Check for zero crossings in each direction
        if ((nw * se < 0 && abs(cen) < min(abs(nw), abs(se))) || ...
            (n * s < 0 && abs(cen) < min(abs(n), abs(s))) || ...
            (ne * sw < 0 && abs(cen) < min(abs(ne), abs(sw))) || ...
            (e * w < 0 && abs(cen) < min(abs(e), abs(w))))
            edges(f_row, f_col) = 1;
        end

        f_row = f_row+1;
```

```
    end
    f_col = f_col+1;
end
end
```

LoG Operator function(matt)

```
function edges = LoG(image, sigma)
% Create a Gaussian filter
filterSize = ceil(3 * sigma) * 2 + 1;
gaussianFilter = fspecial('gaussian', filterSize, sigma);
% Compute the Laplacian of the Gaussian
laplacianFilter = fspecial('log', filterSize, sigma);
% Convolve the image with the Laplacian of Gaussian filter
edges = imfilter(image, laplacianFilter, 'replicate');
end
```

Canny Edge Detector(extra credit part)

```
function edges = cannyEdgeDetector(image)
% Convert image to grayscale
image = rgb2gray(image);
% Apply Gaussian filter to smooth image
image = imgaussfilt(image, 2);
% Compute gradient magnitude and direction using Sobel filter
[Gx, Gy] = imgradientxy(image, 'sobel');
G = sqrt(Gx.^2 + Gy.^2);
Theta = atan2(Gy, Gx);
% Non-maximum suppression
edges = zeros(size(G));
for i = 2:size(G, 1) - 1
   for j = 2:size(G, 2) - 1
       angle = Theta(i, j);
       if angle < 0
           angle = angle + pi;
       end
       angle = round(angle / (pi / 4)) * (pi / 4);
       switch angle
           case 0
               if G(i, j) > G(i, j - 1) && G(i, j) > G(i, j + 1)
                   edges(i, j) = G(i, j);
               end
           case pi / 4
               if G(i, j) > G(i - 1, j + 1) && G(i, j) > G(i + 1, j - 1)
                   edges(i, j) = G(i, j);
               end
           case pi / 2
               if G(i, j) > G(i - 1, j) && G(i, j) > G(i + 1, j)
                   edges(i, j) = G(i, j);
               end
           case 3 * pi / 4
               if G(i, j) > G(i - 1, j - 1) && G(i, j) > G(i + 1, j + 1)
                   edges(i, j) = G(i, j);
               end
       end
   end
end
% Hysteresis thresholding
highThreshold = 0.15 * max(edges(:));
lowThreshold = 0.05 * highThreshold;
for i = 1:size(edges, 1)
   for j = 1:size(edges, 2)
       if edges(i, j) < lowThreshold
           edges(i, j) = 0;
       elseif edges(i, j) > highThreshold
           edges(i, j) = 1;
       else
           if any(any(edges(i - 1:i + 1, j - 1:j + 1)))
               edges(i, j) = 1;
           else
               edges(i, j) = 0;
           end
       end
   end
```

```
end
End
```

Canny and Hough Threshold

```
% NYC =imread('New York City.jpg');
% NYC = im2double(rgb2gray(NYC));
% hall=imread('corridor.jpg');
% hall=im2double(rgb2gray(hall));
% figure, imshow(hall)
corner = imread('corner_window.jpg');
corner = im2double(rgb2gray(corner));
% bike = imread("bike-lane.jpg");
%bike = im2double(rgb2gray(bike));
% bikes = edge(bike, 'canny');
% figure, imshow(bikes)
% title('Canny Edge Detector')
cornerS = edge(corner, 'canny');
figure, imshow(cornerS)
title('Canny Edge Detector')
figure(1), imshow(cornerS, []), title("Hough Transform"), hold on
[H, T, R] = hough(cornerS);
imshow(H);
npeax = 80;
peax = houghpeaks(H, npeax, 'Threshold', 25);
lines = houghlines(cornerS, T, R, peax, 'FillGap', 5, 'minLen', 3);
figure(1), hold on
for a=1:length(lines)
   m = [lines(a).point1; lines(a).point2];
   plot(m(:,1), my(:,2), 'LineWidth', 1, 'Color', 'r');
end
```