**Project 2 Report**

Image Filtering and Edge Detection

CPE428-01,02 - Computer Vision

Prepared for: Professor Xiaozheng Zhang

Prepared By: The “A” Team; Nikhil Patolia, Alec Hardy

Date Submitted: Saturday, January 25, 2020

Table of Contents

[**Part A - Image Filtering**](#_w38306v5l8dn) **3**

[Analysis of Images](#_ktc528fq0kv0) 3

[Methods Used](#_568ygpl05dx) 3

[MATLAB Implementation](#_89ml0dd92hcj) 4

[Conclusion](#_49oiyxkeqmsf) 4

[**Part B - Edge Detection**](#_yc6ewwpk8nw3) **5**

[Prewitt and Sobel Operator](#_45pwe342ghdu) 5

[Prewitt-Filtered Images](#_x0n57yrgauem) 6

[Best thresholded images using Prewitt:](#_241jrsk1g06o) 6

[Sobel-Filtered Images](#_dhdrekp0nz4t) 7

[Best thresholded images using Sobel:](#_xy6eb3k1mfw1) 7

[LoG Operator](#_m29wm1iormr8) 8

[LoG-Filtered Images](#_whbnxgsqsjnj) 8

[Best σ2 value images using LoG Operator:](#_igahaxjhd4d) 9

[Canny Edge Detector](#_2c6h0ph5bkm9) 10

[Best of Canny Edge Detection:](#_bvn8h33778hf) 11

[Hough Transform](#_4qkynblom6a7) 12

[**Appendix A - Part A Images (Large Scale)**](#_ucsymbrr4vqb) **13**

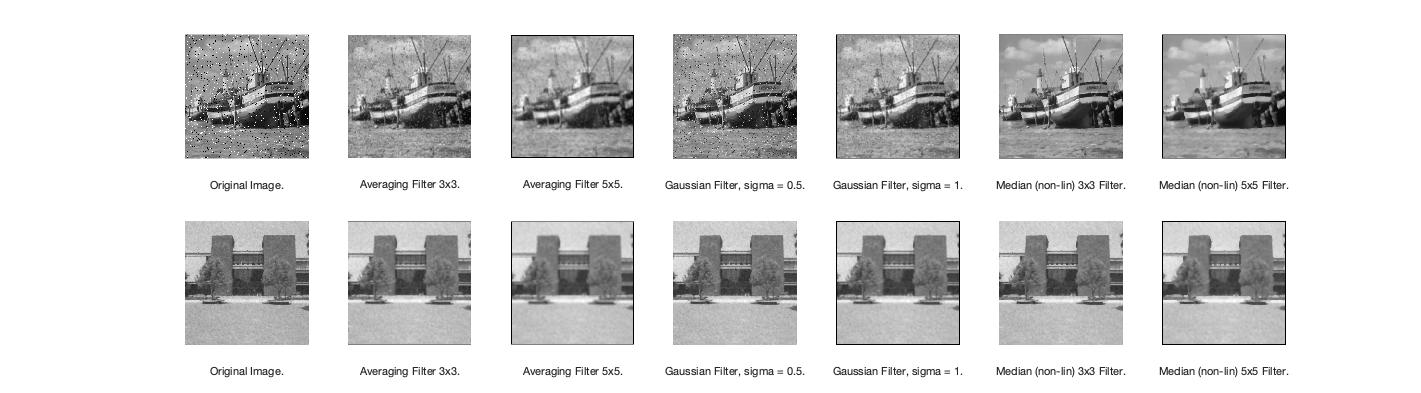
[**Appendix B - Part A Code.**](#_h3d4rvkx3wsm) **14**

[**Appendix C - Part B Code.**](#_eosvbnuectdd) **17**

# **Part A - Image Filtering**

### **Analysis of Images**

Two noisy images were provided - one with salt-and-pepper noise and the other with gaussian noise. Each image was noise-reduced using many various methods. It is expected that the image suffering from salt-and-pepper noise to best be repaired using median filtering and that the gaussian-noise image to best be repaired with either the averaging filter or the Gaussian filter. The original and filtered images are shown below, and a larger copy is included in Appendix A.

****

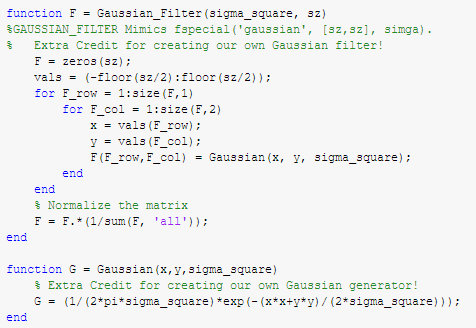
### Methods Used

The following methods were used to reduce the noise in the images of the boat and the building

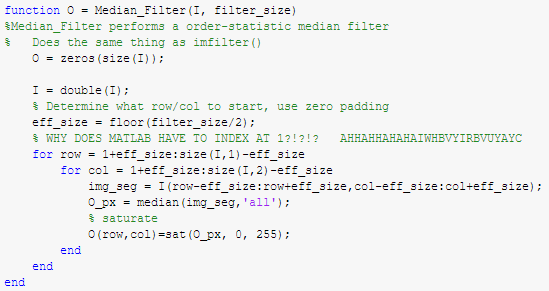
* 3x3 Averaging filter:
  + This filter uses a 3x3 pixel square and averages the values of all the pixels in that square and assigns that average value to the pixel in the middle of the square. This was accomplished by using a linear filter with all the filter values equal to 1/9, as shown in the code snippet below where I is the image matrix and *lin\_img\_conv()* mimics the built-in function *imfilter()*.



* 5x5 Averaging filter:
  + This filter uses the same method as the 3x3 Averaging filter but uses a larger 5x5 filter instead with filter values equal to 1/25.
* Gaussian filter:
  + The Gaussian filter uses the gaussian function to determine the filter values. Values from the gaussian function are quantized and normalized such that the sum of all filter values equals 1. This was implemented with a custom MATLAB function shown below. The gaussian filter is then used with either *imfilter()* or *lin\_img\_conv()*.



* Median Filter:
  + The Median filter uses a 3x3 square of pixels and gets the median of those 9 pixels and assigns the median value to the middle pixel of the square of the output image. This is accomplished as shown below.



### MATLAB Implementation

The code used to generate the output filtered images is shown in Appendix B.

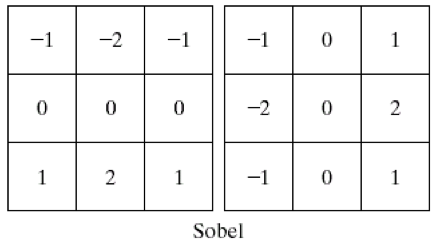
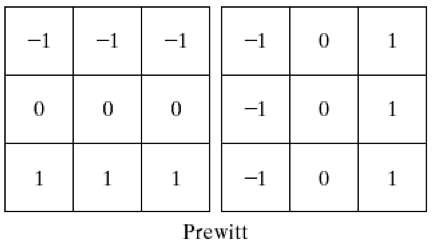
### Conclusion

For the image of the boat that had a large amount of ‘salt and pepper noise’, the median 3x3 filter worked best by keeping the image sharp while also getting rid of all the high-intensity and low-intensity pixels. As for the image of the building, the gaussian filter with a sigma of 0.5 performed the best by making the sky and grass look more uniform while keeping the details in the building.

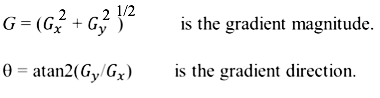
# Part B - Edge Detection

### Prewitt and Sobel Operator

The Prewitt and Sobel operators work by using the following for Gx and Gy:



Then the magnitude and direction of the gradient are calculated using the following formulas.



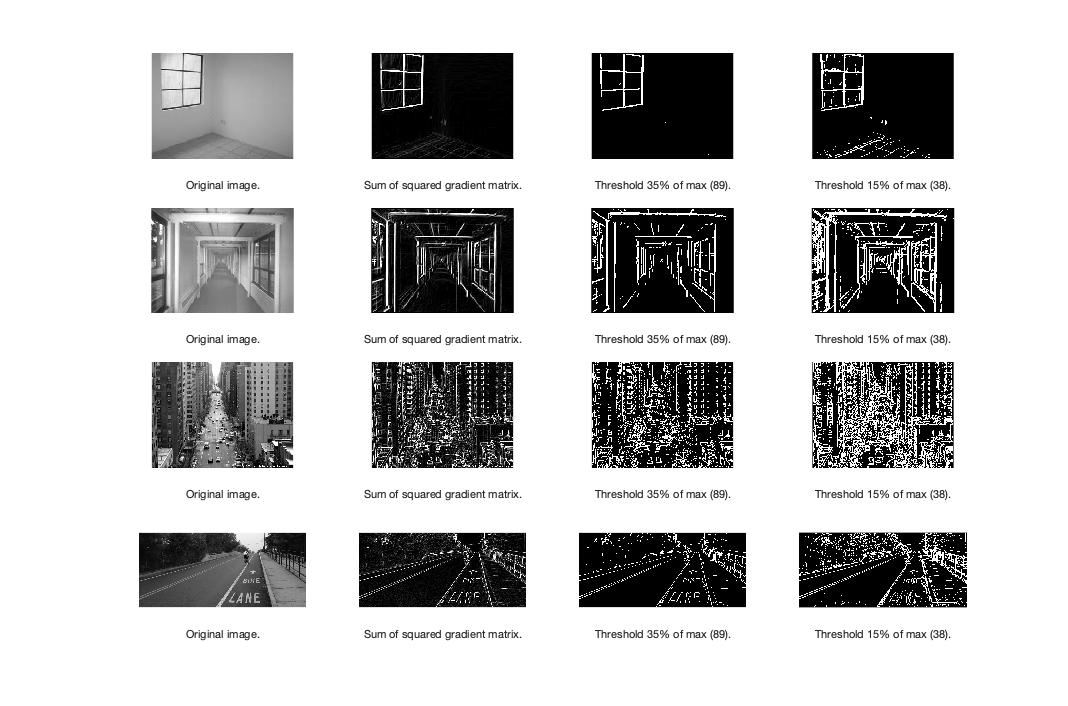
Using these two methods, these are the results we received on the 4 images provided below.

### **Prewitt-Filtered Images**

### Best thresholded images using Prewitt:

|  |  |
| --- | --- |
|  |  |
|  |  |

### Sobel-Filtered Images



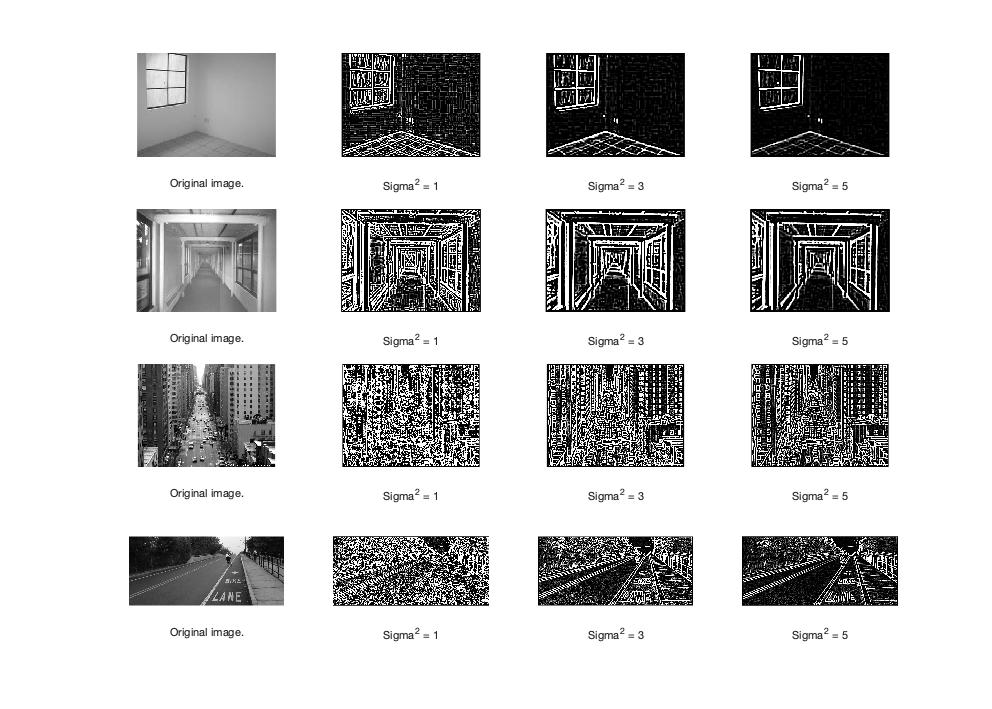
### Best thresholded images using Sobel:

|  |  |
| --- | --- |
|  |  |
|  |  |

### LoG Operator

The LoG operator uses a Gaussian blur to smooth the image before convolving the image with a Laplacian mask. For this project, we used sigma^2 values of 1,3, and 5. The results can be seen below. The mask size with the LoG changes the blurring of the image. For these particular images that were selected for this project, there was a lot of noise in the images which requires a larger Gaussian blur mask to be applied. That is why the best images were produced using a sigma^2 value of 5 for all of the images.

### LoG-Filtered Images



### Best σ2 value images using LoG Operator:

|  |  |
| --- | --- |
|  |  |
|  |  |

### Canny Edge Detector

The Canny Edge Detector works by first applying a Gaussian filter to reduce the noise in the image. Then, we get the magnitude and the direction of the gradient to determine the edges of the image. We used combinations of sigma values 3 and 5 and thresholds of 0.1 and 0.3 to get different results for the prominent edges. A higher sigma value decreases the detail in the image and a higher threshold only keeps the prominent edges in an image.



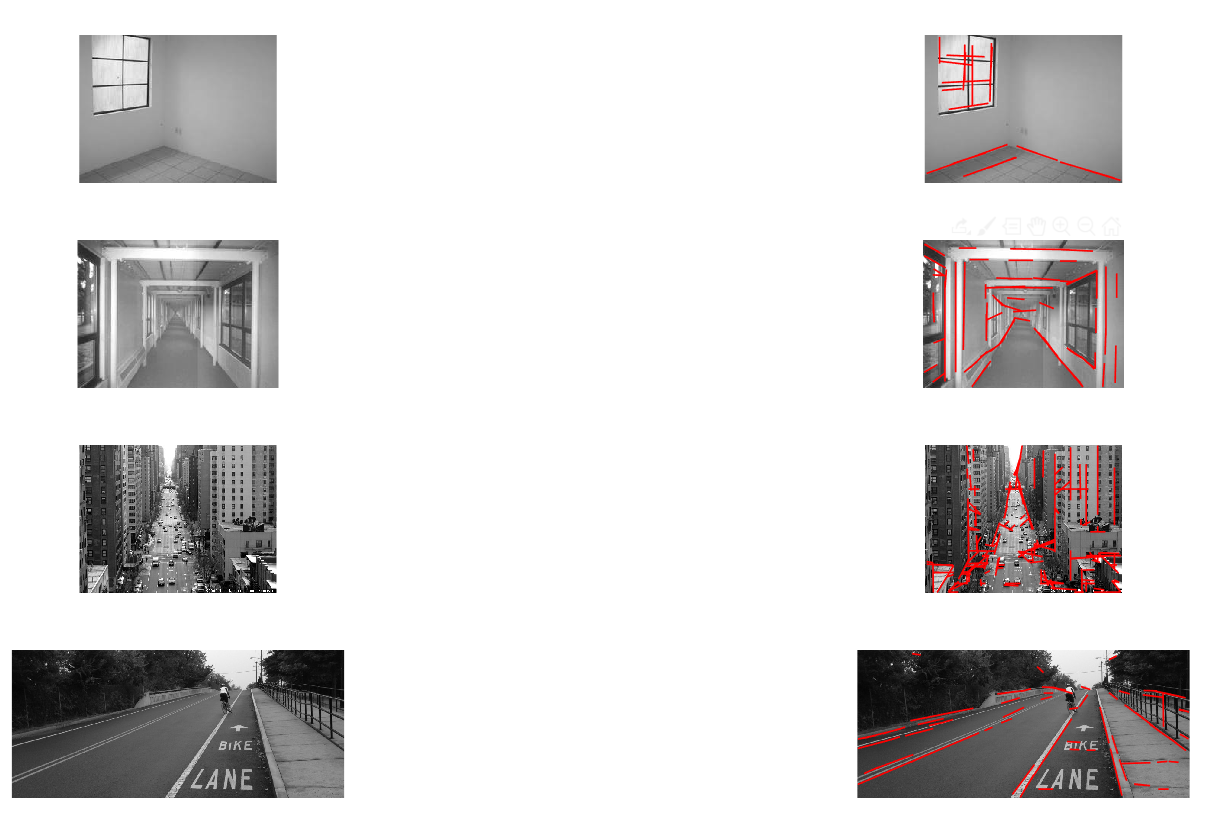
### Best of Canny Edge Detection:

|  |  |
| --- | --- |
|  |  |
|  |  |

### 

### Hough Transform

A Hoguh transform was implemented as a line-detector for each of the sample images. The best edge-map from the Canny Edge Detector was used as the input to the Hough Transformation. Those images are shown above. The detected lines from the transformation are superimposed over the original images below (original images in the left column, superimposed detected lines in the right column). Different parameters were changed for each image in order to produce the best line detection. Parameters such as threshold, numPeaks, fill-gap, and minimum line length were tweaked for each image in order to generate the below results. The code used to generate the image is found as the *houghDisplay()* function in Appendix C.



# Appendix A - Part A Images (Large Scale)



# Appendix B - Part A Code.

clc;

I1 = imread('Boat2.tif');

I2 = imread('building.gif');

figure();

set(gcf,'color','w');

for col = (0:1)

if col==0

I = I1;

elseif col==1

I = I2;

end

numCols = 7;

subplot(2,numCols,(numCols\*col)+1);

imshow(I);

xlabel("Original Image.")

subplot(2,numCols,(numCols\*col)+2);

imshow(lin\_img\_conv(I, 1/9.\*[1 1 1;1 1 1;1 1 1]), []);

xlabel("Averaging Filter 3x3.")

subplot(2,numCols,(numCols\*col)+3);

imshow(lin\_img\_conv(I, 1/25.\*[1 1 1 1 1;1 1 1 1 1;1 1 1 1 1;1 1 1 1 1;1 1 1 1 1]), []);

xlabel("Averaging Filter 5x5.")

subplot(2,numCols,(numCols\*col)+4);

imshow(lin\_img\_conv(I, Gaussian\_Filter(.25, 3)),[]);

xlabel("Gaussian Filter, sigma = 0.5.")

subplot(2,numCols,(numCols\*col)+5);

imshow(lin\_img\_conv(I, Gaussian\_Filter(1, 5)),[]);

xlabel("Gaussian Filter, sigma = 1.")

subplot(2,numCols,(numCols\*col)+6);

imshow(Median\_Filter(I, 3),[]);

xlabel("Median (non-lin) 3x3 Filter.")

subplot(2,numCols,(numCols\*col)+7);

imshow(Median\_Filter(I, 5),[]);

xlabel("Median (non-lin) 5x5 Filter.")

end

function O = lin\_img\_conv(I,F)

%LIN\_IMG\_CONV Does the same thing as imfilter()

% Does the same thing as imfilter()

O = zeros(size(I));

F = double(F);

I = double(I);

% Determine what row/col to start, use zero padding

eff\_size = floor(size(F,1)/2);

% WHY DOES MATLAB HAVE TO INDEX AT 1?!?!? AHHAHHAHAHAIWHBVYIRBVUYAYC

for row = 1+eff\_size:size(I,1)-eff\_size

for col = 1+eff\_size:size(I,2)-eff\_size

img\_seg = I(row-eff\_size:row+eff\_size,col-eff\_size:col+eff\_size);

O\_px = sum(img\_seg.\*F,'all');

% saturate

O(row,col)=sat(O\_px, 0, 255);

end

end

end

function F = Gaussian\_Filter(sigma\_square, sz)

%GAUSSIAN\_FILTER Mimics fspecial('gaussian', [sz,sz], simga).

% Extra Credit for creating our own Gaussian filter!

F = zeros(sz);

vals = (-floor(sz/2):floor(sz/2));

for F\_row = 1:size(F,1)

for F\_col = 1:size(F,2)

x = vals(F\_row);

y = vals(F\_col);

F(F\_row,F\_col) = Gaussian(x, y, sigma\_square);

end

end

% Normalize the matrix

F = F.\*(1/sum(F, 'all'));

end

function G = Gaussian(x,y,sigma\_square)

% Extra Credit for creating our own Gaussian generator!

G = (1/(2\*pi\*sigma\_square)\*exp(-(x\*x+y\*y)/(2\*sigma\_square)));

end

function O = Median\_Filter(I, filter\_size)

%Median\_Filter performs a order-statistic median filter

% Does the same thing as imfilter()

O = zeros(size(I));

I = double(I);

% Determine what row/col to start, use zero padding

eff\_size = floor(filter\_size/2);

% WHY DOES MATLAB HAVE TO INDEX AT 1?!?!? AHHAHHAHAHAIWHBVYIRBVUYAYC

for row = 1+eff\_size:size(I,1)-eff\_size

for col = 1+eff\_size:size(I,2)-eff\_size

img\_seg = I(row-eff\_size:row+eff\_size,col-eff\_size:col+eff\_size);

O\_px = median(img\_seg,'all');

% saturate

O(row,col)=sat(O\_px, 0, 255);

end

end

End

function sat\_x = sat(x, min, max)

%sat\_x Saturates input value with upper and lower limits

% Detailed explanation goes here

if x > max

sat\_x = max;

elseif x < min

sat\_x = min;

else

sat\_x = x;

end

End

function O = threshold(I, threshold\_val)

% Detailed explanation goes here

O = I;

O(O > threshold\_val) = 255;

O(O <= threshold\_val) = 0;

end

# Appendix C - Part B Code.

I1 = rgb2gray(imread('corner\_window.jpg'));

I2 = rgb2gray(imread('corridor.jpg'));

I3 = rgb2gray(imread('New York City.jpg'));

I4 = rgb2gray(imread('bike-lane.jpg'));

edgeDisplay(I1, I2, I3, I4, 'sobel');

edgeDisplay(I1, I2, I3, I4, 'prewitt');

LoGDisplay(I1, I2, I3, I4);

cannyDisplay(I1, I2, I3, I4);

houghDisplay(I1, I2, I3, I4);

function edgeDisplay(I1, I2, I3, I4, mode)

figure('Name', mode);

set(gcf,'color','w');

for col = (0:3)

% I'm still salty that MATLAB indexes at 1...but not this time!

if col==0

I = I1;

elseif col==1

I = I2;

elseif col==2

I = I3;

elseif col==3

I=I4;

end

E1 = edge\_detect(I, mode);

subplot(4,4,(4\*col)+1);

imshow(I);

xlabel("Original image.")

subplot(4,4,(4\*col)+2);

imshow(E1);

xlabel("Sum of squared gradient matrix.")

subplot(4,4,(4\*col)+3);

t = max(E1(:))\*.35;

imshow(threshold(E1, t));

xlabel("Threshold 35% of max ("+t+").")

subplot(4,4,(4\*col)+4);

t = max(E1(:))\*.15;

imshow(threshold(E1, t));

xlabel("Threshold 15% of max ("+t+").")

end

end

function LoGDisplay(I1, I2, I3, I4)

mode = 'LoG';

figure('Name', mode);

set(gcf,'color','w');

for col = (0:3)

if col==0

I = I1;

elseif col==1

I = I2;

elseif col==2

I = I3;

elseif col==3

I=I4;

end

s1 = edge\_detect(I, mode, 1);

s2 = edge\_detect(I, mode, 3);

s3 = edge\_detect(I, mode, 5);

subplot(4,4,(4\*col)+1);

imshow(I);

xlabel("Original image.")

subplot(4,4,(4\*col)+2);

imshow(s1);

xlabel("Sigma^2 = 1")

subplot(4,4,(4\*col)+3);

imshow(s2);

xlabel("Sigma^2 = 3")

subplot(4,4,(4\*col)+4);

imshow(s3);

xlabel("Sigma^2 = 5")

end

end

function cannyDisplay(I1, I2, I3, I4)

figure('Name', 'Canny');

set(gcf,'color','w');

for col = (0:3)

if col==0

I = I1;

elseif col==1

I = I2;

elseif col==2

I = I3;

elseif col==3

I=I4;

end

E1 = canny\_edge\_detect(I,3, 0.1);

E2 = canny\_edge\_detect(I,3, 0.3);

E3 = canny\_edge\_detect(I,5, 0.1);

E4 = canny\_edge\_detect(I,5, 0.3);

subplot(5,5,(5\*col)+1);

imshow(I);

xlabel("Original image.")

subplot(5,5,(5\*col)+2);

imshow(E1);

xlabel("\sigma^2 = 3. Threshold of 0.1")

subplot(5,5,(5\*col)+3);

imshow(E2);

xlabel("\sigma^2 = 3. Threshold of 0.3")

subplot(5,5,(5\*col)+4);

imshow(E3);

xlabel("\sigma^2 = 5. Threshold of 0.1")

subplot(5,5,(5\*col)+5);

imshow(E4);

xlabel("\sigma^2 = 5. Threshold of 0.3")

end

end

function houghDisplay(I1, I2, I3, I4)

%% Generate edge maps using Canny Edge Detector

% These canny parameters were selected from the "Best of Canny Edge

% Detection" part of the write-up. Definitely not arbitrary.

E1 = canny\_edge\_detect(I1, 5, 0.1);

E2 = canny\_edge\_detect(I2, 5, 0.1);

E3 = canny\_edge\_detect(I3, 3, 0.3);

E4 = canny\_edge\_detect(I4, 5, 0.1);

%% Display Stuff...

figure();

set(gcf,'color','w');

% Display all of the original images

subplot(4,2,1);

imshow(I1);

subplot(4,2,3);

imshow(I2);

subplot(4,2,5);

imshow(I3);

subplot(4,2,7);

imshow(I4);

%% Let's do all the crazy Hough stuff here, and display it as we go.

for crazy\_hough\_stuff = (0:3)

if crazy\_hough\_stuff == 0

E = E1;

I = I1;

threshold = 30;

numPeaks = 50;

fillgap = 8;

minlen = 15;

subplot(4,2,2);

elseif crazy\_hough\_stuff == 1

E = E2;

I = I2;

threshold = 30;

numPeaks = 50;

fillgap = 10;

minlen = 15;

subplot(4,2,4);

elseif crazy\_hough\_stuff == 2

E = E3;

I = I3;

threshold = 30;

numPeaks = 50;

fillgap = 20;

minlen = 15;

subplot(4,2,6);

else

E = E4;

I = I4;

threshold = 30;

numPeaks = 50;

fillgap = 10;

minlen = 20;

subplot(4,2,8);

end

% I wish I had a bit more time to make my own hough() function for

% extra credit :(

[H, theta, rho] = hough(E);

peaks = houghpeaks(H,numPeaks,'Threshold', threshold);

lines = houghlines(E, theta, rho, peaks, 'FillGap', fillgap, 'minLen', minlen);

imshow(I), hold on;

for k=1:length(lines)

xy = [lines(k).point1; lines(k).point2];

plot(xy(:,1), xy(:,2), 'LineWidth', 1, 'Color', 'r'), hold on;

end

end

end

function O = edge\_detect(I, mode, options)

% Detailed explanation goes here

if strcmp(mode,'sobel')

Fx = [-1 0 1;-2 0 2;-1 0 1];

Fy = [-1 -2 -1; 0 0 0; 1 2 1];

n\_conv = 2;

elseif strcmp(mode, 'prewitt')

Fx = [1 0 -1;1 0 -1;1 0 -1];

Fy = [1 1 1;0 0 0;-1 -1 -1];

n\_conv = 2;

elseif strcmp(mode, 'LoG')

sigma\_square = options(1);

if (sigma\_square >= 3); gs = 7; else; gs = 5; end

F = fspecial('log', gs, sqrt(sigma\_square));

n\_conv = 1;

end

if n\_conv==1

O = lin\_img\_conv(I,F);

elseif n\_conv==2

Gx = lin\_img\_conv(I, Fx);

Gy = lin\_img\_conv(I, Fy);

O = uint8(sqrt(Gx.^2 + Gy.^2));

end

end

function O = canny\_edge\_detect(I, sigma, threshold)

% gaus = lin\_img\_conv(I, Gaussian\_Filter(sigma\*sigma, 5));

gaus = imgaussfilt(I, sigma);

O = edge(gaus, 'Canny', threshold);

end