# Yoshika Takezawa

# 4/7/19

# Computer Vision hw1

# I pledge my honor that I have abided by the stevens honor system

# how to run: "python hw1.py <image>"

import math

import random

import sys

import numpy as np

import cv2

# given an image, square filter matrix and the length,

# outputs the filtered image

def filter(img, fm, len):

height, width = img.shape

# create black image

res = np.zeros((height, width), np.uint8)

off = len//2

for i in range(off, height-off):

for j in range(off, width-off):

fval = block\_sum(img, fm, len, i, j)

res[i][j] = fval

return res

# using the filtermatrix's weight distribution, adds all the values

def block\_sum(img, fm, len, x, y):

total = 0.0

off = len//2

# iterating through each matrix element

for i in range(len):

for j in range(len):

curx = x + i - off

cury = y + j - off

total += img[curx][cury] \* fm[i][j]

if (total <= 0):

total = 0

if (total >= 255):

total = 255

return total

# applies sobel filter and first nonmax suppression

# for the hessian function to get a clearer input

def sobel\_nms(img, threshold = 0.225):

# applying vertical and horizontal sobel filters

hsob = [[1, 2, 1],

[0, 0, 0],

[-1, -2, -1]]

I\_x = filter(img, hsob, 3)

cv2.imwrite('xsobel.png', I\_x)

vsob = [[1, 0, -1],

[2, 0, -2],

[1, 0, -1]]

I\_y = filter(img, vsob, 3)

cv2.imwrite('ysobel.png', I\_y)

highthesh = 255 \* threshold

height, width = img.shape

res = np.power(np.power(I\_x, 2.0) + np.power(I\_y, 2.0), 0.5)

# Non-maximum suppression

for i in range(height):

for j in range(width):

# weak pixels become black

if (res[i][j] < highthesh):

I\_x[i][j] = 0

I\_y[i][j] = 0

res[i][j] = 0

# image edge pixels become 0

if (i == 0 or i == height-1 or j == 0 or j == width - 1):

I\_x[i][j] = 0

I\_y[i][j] = 0

res[i][j] = 0

continue

# left right

elif res[i][j] <= res[i][j-1] or res[i][j] <= res[i][j+1]:

I\_x[i][j] = 0

I\_y[i][j] = 0

res[i][j] = 0

# / diagonal

elif res[i][j] <= res[i-1][j+1] or res[i][j] <= res[i+1][j-1]:

I\_x[i][j] = 0

I\_y[i][j] = 0

res[i][j] = 0

# up down

elif res[i][j] <= res[i-1][j] or res[i][j] <= res[i+1][j]:

I\_x[i][j] = 0

I\_y[i][j] = 0

res[i][j] = 0

# \ diagonal

elif res[i][j] <= res[i-1][j-1] or res[i][j] <= res[i+1][j+1]:

I\_x[i][j] = 0

I\_y[i][j] = 0

res[i][j] = 0

return I\_x, I\_y, res

# gets the second partial derivatives and the determinants

# are thresholded. takes neighboring pixels into account

# to allow for a more well-rounded view of the overall picture

def hessian(img, x, y, threshold=35000):

hsob = [[1, 2, 1],

[0, 0, 0],

[-1, -2, -1]]

vsob = [[1, 0, -1],

[2, 0, -2],

[1, 0, -1]]

xx = filter(x, hsob, 3)

yy = filter(y, vsob, 3)

xy = filter(x, vsob, 3)

height, width = img.shape

res = np.zeros((height, width), np.uint8)

one = [[1, 1, 1],

[1, 1, 1],

[1, 1, 1]]

for i in range(1, height-1):

for j in range(1, width-1):

totalxx = block\_sum(xx, one, 3, i, j)

totalyy = block\_sum(yy, one, 3, i, j)

totalxy = block\_sum(xy, one, 3, i, j)

deter = (totalxx \* totalyy) - (totalxy)\*\*2

# deter = xx[i][j]\* yy[i][j] - xy[i][j]\*\*2

if (deter > threshold):

res[i][j] = deter

else:

res[i][j] = 0

return res

# non-maximum suppression: if the current pixel is darker than a

# neighboring pixel, the current pixel is turned to black.

def nms(img):

height, width = img.shape

res = img.copy()

# Non-maximum suppression

for i in range(height):

for j in range(width):

# image edge pixels become 0

if (i == 0 or i == height-1 or j == 0 or j == width - 1):

res[i][j] = 0

continue

# left right

elif res[i][j] <= res[i][j-1] or res[i][j] <= res[i][j+1]:

res[i][j] = 0

# / diagonal

elif res[i][j] <= res[i-1][j+1] or res[i][j] <= res[i+1][j-1]:

res[i][j] = 0

# up down

elif res[i][j] <= res[i-1][j] or res[i][j] <= res[i+1][j]:

res[i][j] = 0

# \ diagonal

elif res[i][j] <= res[i-1][j-1] or res[i][j] <= res[i+1][j+1]:

res[i][j] = 0

# brighten up points

points = []

for i2 in range(1, height-1):

for j2 in range(1, width-2):

if res[i2][j2] != 0:

res[i2][j2] = 255

points.append([i2, j2])

return res, points

# takes the shortest distance between a line formed by

# two points(x1,y1 and x2, y2) and a point x0,y0

def shortest\_dist(x1, y1, x2, y2, x0, y0):

numerator = abs((y2 - y1)\*x0 - (x2-x1)\*y0 + x2\*y1 - y2\*x1)

denom = math.sqrt((y2-y1)\*\*2 + (x2-x1)\*\*2)

return numerator/denom

# out of a list of coordinates, pulls the two furthest points

def get\_extreme(lst):

dist = 0

a = (lst[0][1], lst[0][0])

b = (lst[1][1], lst[1][0])

for i in range(0, len(lst)-1):

for j in range(i+1, len(lst)):

y1,x1= lst[i]

y2,x2= lst[j]

curdist = math.sqrt((x1-x2)\*\*2 + (y1-y2)\*\*2)

if (curdist>dist) :

dist = curdist

a = (x1,y1)

b = (x2, y2)

return a,b

# picks two random points and sees if there is enough support for that line

# dist\_thresh is the max distance allowed to be considered part of the line

# inlier\_thresh is the number of points needed to be considered a viable line

def ransac(img, pt\_lst, dist\_thresh = 7, inlier\_thresh = 20):

blue=[255, 0, 0]

green=[0, 255, 0]

red=[0,0, 255]

cyan=[255, 255, 0]

colors = [blue, green, red, cyan]

lines\_needed = 4

numlines= 0

pt\_used = []

while (numlines<lines\_needed):

rand1 = random.randint(0, len(pt\_lst)-1)

rand2 = random.randint(0, len(pt\_lst)-1)

x1,y1= pt\_lst[rand1]

x2,y2 = pt\_lst[rand2]

if (pt\_lst[rand1] in pt\_used or pt\_lst[rand2] in pt\_used or rand1 ==rand2):

continue

cur\_line = [pt\_lst[rand1], pt\_lst[rand2]]

for p in range(len(pt\_lst)):

if (len(cur\_line) >= inlier\_thresh):

# have enough points

break

if (pt\_lst[p]in pt\_used or pt\_lst[p] in cur\_line):

# point is used, next point

continue

dist = shortest\_dist(x1, y1, x2, y2, pt\_lst[p][0], pt\_lst[p][1])

if (dist <= dist\_thresh):

cur\_line.append(pt\_lst[p])

if (len(cur\_line) < inlier\_thresh):

# line does not have enough, next line

continue

# line has enough points

for q in range(len(cur\_line)):

i,j = cur\_line[q]

img[i-1][j-1] = img[i][j-1]= img[i+1][j-1] = colors[numlines]

img[i-1][j] = img[i][j] = img[i+1][j] = colors[numlines]

img[i-1][j+1] = img[i][j+1] = img[i+1][j+1] = colors[numlines]

# set point to used

pt\_used.append(cur\_line[q])

# get outliers

out1, out2 = get\_extreme(cur\_line)

# plot line

cv2.line(img,out1,out2,colors[numlines],1)

cur\_line =[]

numlines+= 1

return img

# for every point, there is exists a rho and theta s.t.

# rho =xcos(theta) + ysin(theta). for every theta (0->180),

# add a "vote" in H[theta][rho].

# extract the theta and rho combo with the most votes and

# that is the line with the most support

def hough(img, pts, numlines=4):

height = img.shape[0]

width = img.shape[1]

H = np.zeros(shape=(181, int(math.sqrt(height\*\*2 + width\*\*2))))

for p in pts:

for theta in range(181):

x,y = p

rho = x\*math.cos(np.deg2rad(theta)) + y\*math.sin(np.deg2rad(theta))

H[theta][int(rho)] += 1

largest =[(0,0)]

min = 0

for t in range(181):

for r in range(int(math.sqrt(height\*\*2 + width\*\*2))):

if len(largest) < 4:

largest.append([t, r])

else:

if (H[t][r] > min):

for lpt in range(len(largest)):

if min == H[largest[lpt][0]][largest[lpt][1]]:

largest[lpt] = (t,r)

break

# readjusting min of largests

min = H[largest[0][0]][largest[0][1]]

for f in range(len(largest)):

if H[largest[f][0]][largest[f][1]]<min:

min = H[largest[f][0]][largest[f][1]]

# has the theta and rho of lines, need to plot the lines

for i in largest:

j,k = i

j= math.radians(j)

y1 = 0

x1 = int((k/math.sin(j)) -(y1\*math.cos(j)/math.sin(j)))

if (x1>width):

x1 = width

y1 = int((k/math.cos(j)) - (x1\*math.sin(j)/math.cos(j)))

elif (x1<0):

x1 = 0

y1 = int((k/math.cos(j)) - (x1\*math.sin(j)/math.cos(j)))

x2 = 0

y2 = int((k/math.cos(j)) - (x2\*math.sin(j)/math.cos(j)))

if (y2>height):

y2= height

x2 = int((k/math.sin(j)) -(y2\*math.cos(j)/math.sin(j)))

elif (y2<0):

y2 = 0

x2 = int((k/math.sin(j)) -(y2\*math.cos(j)/math.sin(j)))

cv2.line(img, (x1, y1), (x2, y2), (255, 255, 0),1)

return img

if \_\_name\_\_ == '\_\_main\_\_':

if (len(sys.argv) != 2):

sys.exit("usage: python hw1.py <img>")

orig = cv2.imread(sys.argv[1], 0)

print("Working on it!")

# applying gaussian filter

gaussian = [[0.077847, 0.123317, 0.077847],

[0.123317, 0.195346, 0.123317],

[0.077847, 0.123317, 0.077847]]

g\_res = filter(orig, gaussian, 3)

cv2.imwrite('gauss.png', g\_res)

x, y, together = sobel\_nms(g\_res)

cv2.imwrite('xnms.png', x)

cv2.imwrite('ynms.png', y)

cv2.imwrite('tnms.png', together)

hes = hessian(together, x, y)

cv2.imwrite('hessian.png', hes)

partone, pts = nms(hes)

cv2.imwrite('partone.png', partone)

print("Part one finished.")

rsc = ransac(cv2.cvtColor(orig, cv2.COLOR\_GRAY2RGB), pts)

cv2.imwrite('ransac.png', rsc)

print("Part two finished.")

huff = hough(cv2.cvtColor(partone, cv2.COLOR\_GRAY2RGB), pts)

cv2.imwrite('hough.png', huff)

print("Part three finished.")

print("Done!")

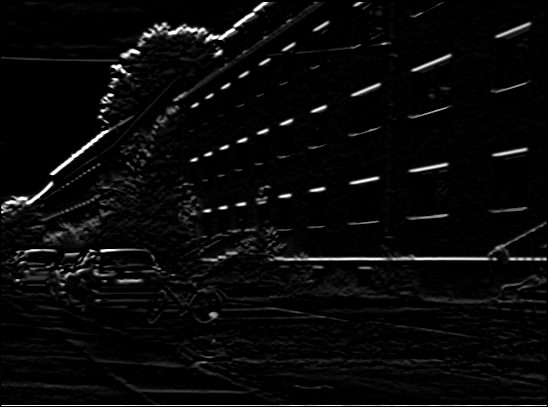
Original photo:



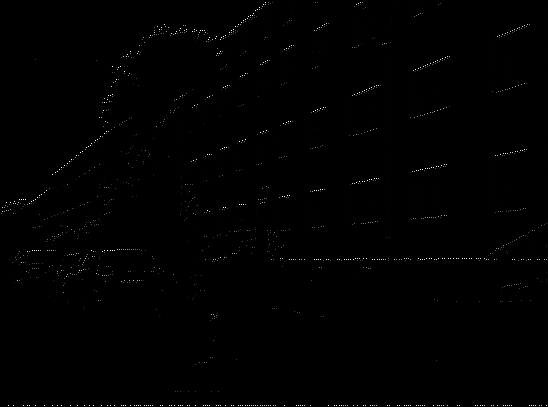
Gaussian filter:



Horizontal sobel filter:



Horizontal sobel + non max supp



Vertical sobel filter:



Vertical sobel + non max supp

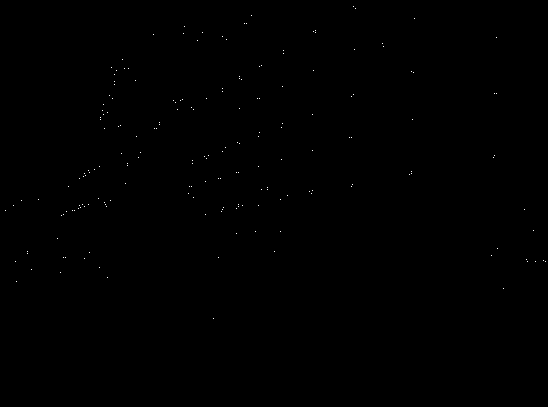


Vertical + horizontal sobel filter + non max suppression:  


Hessian:

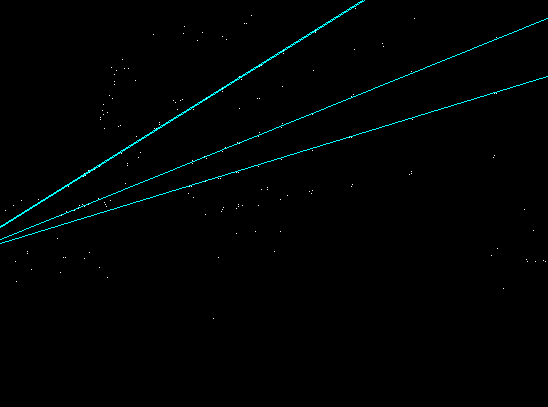


Non max suppression:



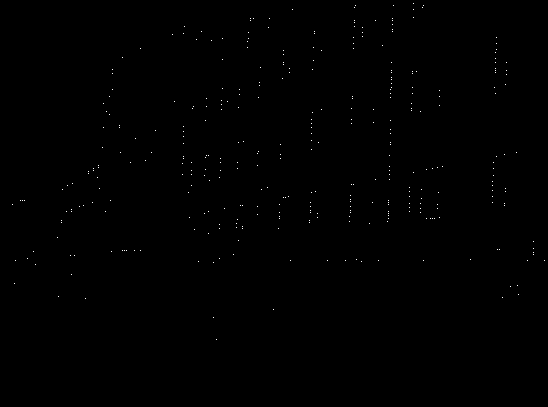
Ransac (over original photo):



Hough transform:  


As illustrated by the pictures, the general work flow was to apply the gaussian filter, apply the vertical and horizontal sobel filters, apply non-max suppression, threshold the hessian determinant, apply the non-max suppression again, run ransac and run hough.

I decided to apply the non-maximum suppression once after the sobel filters to give the hessian function a clearer input and to ensure that weak pixels would not show up later.

In the hessian function, I chose to add the sum of the neighboring pixels’ derivatives because without adding them, I found that the resulting image has a vertical edge bias. (The image to the right only takes the current pixel’s derivatives into account and has a threshold of 15. )

For RANSAC, I followed the algorithm outlined from the slides (Week 3, page 58).

For Hough, I followed the algorithm outlined from the slides (Week 4, page 14).