Kunal Dhaimade - CS 558 Homework Assignment 2

Source Code:

```
# -*- coding: utf-8 -*-
"""
Created on Sun Mar 19 21:36:40 2017

@author: Kunal
"""

from PIL import Image, ImageDraw
import random as rn
import numpy as np
import math as mt
import itertools as it

'''To resize the image to replicate border pixels before convolution as per the filter_size'''
def resize_image(ip_im, filter_size):
    r, c = ip_im.shape
    filter_n = int((filter_size-1)/2)
    op_r = r+2*(filter_n)
    op_c = c+2*(filter_n)
    op_im = np.zeros((op_r,op_c))
    for i in range(r):
        for j in range(c):
            op_im[i+filter_n][j+filter_n] = ip_im[i][j]
    for i in range(filter_n):
        for j in range(filter_n):
            op_im[i][j] = op_im[filter_n][filter_n]
    for i in range(filter_n):
        for j in range(op_c-filter_n, op_c):
            op_im[i][j] = op_im[filter_n][op_c-filter_n-1]
    for i in range(op_r-filter_n, op_r):
        for j in range(filter_n):
            op_im[i][j] = op_im[op_r-filter_n-1][filter_n]
    for i in range(op_r-filter_n, op_r):
        for j in range(op_c-filter_n, op_c):
            op_im[i][j] = op_im[op_r-filter_n-1][op_c-filter_n-1]
    for i in range(filter_n):
        for j in range(filter_n, op_c-filter_n):
            op_im[i][j] = op_im[filter_n][j]
    for i in range(op_r-filter_n, op_r):
        for j in range(filter_n, op_c-filter_n):
            op_im[i][j] = op_im[op_r-filter_n-1][j]
    for i in range(filter_n, op_r-filter_n):
        for j in range(filter_n):
            op_im[i][j] = op_im[i][filter_n]
    for i in range(filter_n, op_r-filter_n):
        for j in range(op_c-filter_n, op_c):
            op_im[i][j] = op_im[i][op_c-filter_n-1]
```

```
    return op_im

'''To perform convolution of ip with filter'''
def convolution(ip,filter):
    filter_size = int(mt.sqrt(filter.size))
    filter_n = int((filter_size-1)/2)
    ip_r, ip_c = ip.shape
    r = ip_r - 2*filter_n
    c = ip_c - 2*filter_n
    op_im = np.zeros((r, c))
    for i in range(r):
        for j in range(c):
            for k in range(filter_size):
                for l in range(filter_size):
                    op_im[i][j] = op_im[i][j] + (filter[k][l] * ip[i+k][j+l])
    return op_im

'''To create the gaussian filter'''
def gauss_filter(im, size, sigma):
    size = int(size)
    sigma = float(sigma)
    og_im = np.array(im)
    filter = np.zeros((size,size))
    filter_n = int((size-1)/2)
    y, x = np.ogrid[float(-filter_n):float(filter_n+1),float(-filter_n):float(filter_n+1)]
    sum = 0
    for i in range(size):
        for j in range(size):
            e = mt.exp((-((x[0][j]**2)+(y[i][0]**2))/(2*(sigma**2))))
            filter[i][j] = e*(1/(2*mt.pi*(sigma**2)))
            sum = sum + filter[i][j]
    for i in range(size):
        for j in range(size):
            filter[i][j] = filter[i][j]/sum
    r, c = og_im.shape
    m_im = resize_image(og_im, size)
    m_r, m_c = m_im.shape
    op_im = convolution(m_im, filter)
    op_im = Image.fromarray(op_im)
    return op_im

'''To obtain the X-derivative'''
def grad_x(ip_im):
    filter_x = [[-1,0,+1], [-2,0,+2], [-1,0,+1]]
    filter_x = np.array(filter_x)
    m_im = resize_image(ip_im, 3)
    op_im = convolution(m_im, filter_x)
    return op_im
```

```
'''To obtain the Y-derivative'''
def grad_y(ip_im):
    filter_y = [[+1,+2,+1], [0,0,0], [-1,-2,-1]]
    filter_y = np.array(filter_y)
    m_im = resize_image(ip_im, 3)
    op_im = convolution(m_im, filter_y)
    return op_im

'''To obtain the output of the Hessian Detector'''
def hessian_detector(im):
    ip_im = np.array(im)
    r, c = ip_im.shape
    op_im = np.zeros((r, c))
    f_im = op_im
    im_xx = grad_x(grad_x(ip_im))
    im_yy = grad_y(grad_y(ip_im))
    im_xy = grad_x(grad_y(ip_im))
    for i in range(r):
        for j in range(c):
            op_im[i][j] = (im_xx[i][j] * im_yy[i][j]) - (im_xy[i][j] ** 2)
    op_im_max = np.amax(op_im)
    op_im_min = np.amin(op_im)
    old_range = op_im_max - op_im_min
    for i in range(r):
        for j in range(c):
            op_im[i][j] = (op_im[i][j] - op_im_min) * (255/old_range)
    for i in range(r):
        for j in range(c):
            if(op_im[i][j] < 125):
                op_im[i][j] = 0
            else:
                op_im[i][j] = 255
    op_im = resize_image(op_im, 3)
    r, c = op_im.shape
    for i in range(1, r-1):
        for j in range(1, c-1):
            if(op_im[i][j] != max(op_im[i-1][j-1], op_im[i-1][j], op_im[i-1][j+1], op_im[i][j-1], op_im[i][j],
op_im[i][j+1], op_im[i+1][j-1], op_im[i+1][j], op_im[i+1][j+1])):
                f_im[i-1][j-1] = 0
            else:
                f_im[i-1][j-1] = op_im[i][j]
    f_im = f_im.astype(np.uint8)
    f_im = Image.fromarray(f_im)
    return f_im

'''To generate the sample space of potential point pairs for RANSAC operation'''
def gen_sample_space(im):
```

```
    ip_im = np.asarray(im)
    r, c = ip_im.shape
    n_points = 0
    sample_space = []
    for i in range(r):
        for j in range(c):
            if(ip_im[i][j] != 0):
                n_points = n_points + 1
    for i in range(r):
        for j in range(int(c/3),c):
            if(ip_im[i][j] != 0):
                sample_space.append([j, i])
    sample_space = list(it.combinations(sample_space, 2))
    return sample_space, n_points

'''To perform the RANSAC operation and obtain potential line models'''
def ransac(im, sample_space, n_points):
    ip_im = np.asarray(im)
    r, c = ip_im.shape
    r_iter = 500
    r_dist = 3
    r_no_inlier = 70
    r_ratio = 0.06
    fit_pt_mod = []
    while(r_iter != 0):
        sample_space, pt_mod = select_one(sample_space)
        l_mod = line_model(pt_mod)
        inliers = []
        no_inlier = 0
        put_flag = 1
        for i in range(r):
            for j in range(c):
                if(ip_im[i][j] != 0):
                    x, y = find_inter_pt(l_mod, j, i)
                    dist = mt.sqrt(((x - j)**2) + ((y - i)**2))
                    if(dist < r_dist):
                        inliers.append([j, i])
                        no_inlier = no_inlier + 1
        if(no_inlier/n_points > r_ratio and no_inlier > r_no_inlier):
            ratio = no_inlier/n_points
            inliers.sort()
            in_len = len(inliers)
            if([ratio, [inliers[0], inliers[in_len - 1]]] not in fit_pt_mod):
                if(fit_pt_mod == []):
                    fit_pt_mod.append([ratio, [inliers[0], inliers[in_len - 1]]])
                for k in fit_pt_mod:
                    if(inliers[0] in k[1] or inliers[in_len - 1] in k[1] or ratio == k[0]):
                        put_flag = 0
```

```
                    break
                if(put_flag == 1):
                    fit_pt_mod.append([ratio, [inliers[0], inliers[in_len - 1]]])
        r_iter = r_iter - 1
    fit_pt_mod.sort()
    return fit_pt_mod

'''To plot the 4 line models with the strongest support obtained from RANSAC'''
def plot_ransac(ip_im, ip_im2, fit_pt_mod):
    no_plot = 4
    i = len(fit_pt_mod) - 1
    while(no_plot != 0):
        pt_mod = fit_pt_mod[i][1]
        x1 = pt_mod[0][0]
        y1 = pt_mod[0][1]
        x2 = pt_mod[1][0]
        y2 = pt_mod[1][1]
        ip_im_draw = ImageDraw.Draw(ip_im)
        ip_im_draw = endpt_draw(ip_im_draw, x1, y1, x2, y2)
        ip_im_draw.line((x1, y1, x2, y2), fill = 255)
        ip_im2_draw = ImageDraw.Draw(ip_im2)
        ip_im2_draw = endpt_draw(ip_im2_draw, x1, y1, x2, y2)
        ip_im2_draw.line((x1, y1, x2, y2), fill = 255)
        i = i - 1
        no_plot = no_plot- 1
    return ip_im, ip_im2

'''To create a line model(m and c) from two points'''
def line_model(pt_list):
    x1 = pt_list[0][0]
    y1 = pt_list[0][1]
    x2 = pt_list[1][0]
    y2 = pt_list[1][1]
    if((x2 - x1) == 0):
        m = mt.inf
    else:
        m = (y2 - y1)/(x2 - x1)
    c = y2 - (m*x2)
    return [m, c]

'''To find the intersection point of the normal from (x, y) to the line represented by line_model'''
def find_inter_pt(line_model, x, y):
    m = line_model[0]
    c = line_model[1]
    xi = (x + (m*y) - (m*c))/(1 + m**2)
    yi = ((m*x) + ((m**2)*y) - ((m**2)*c))/(1 + m**2) + c
    return xi, yi
```

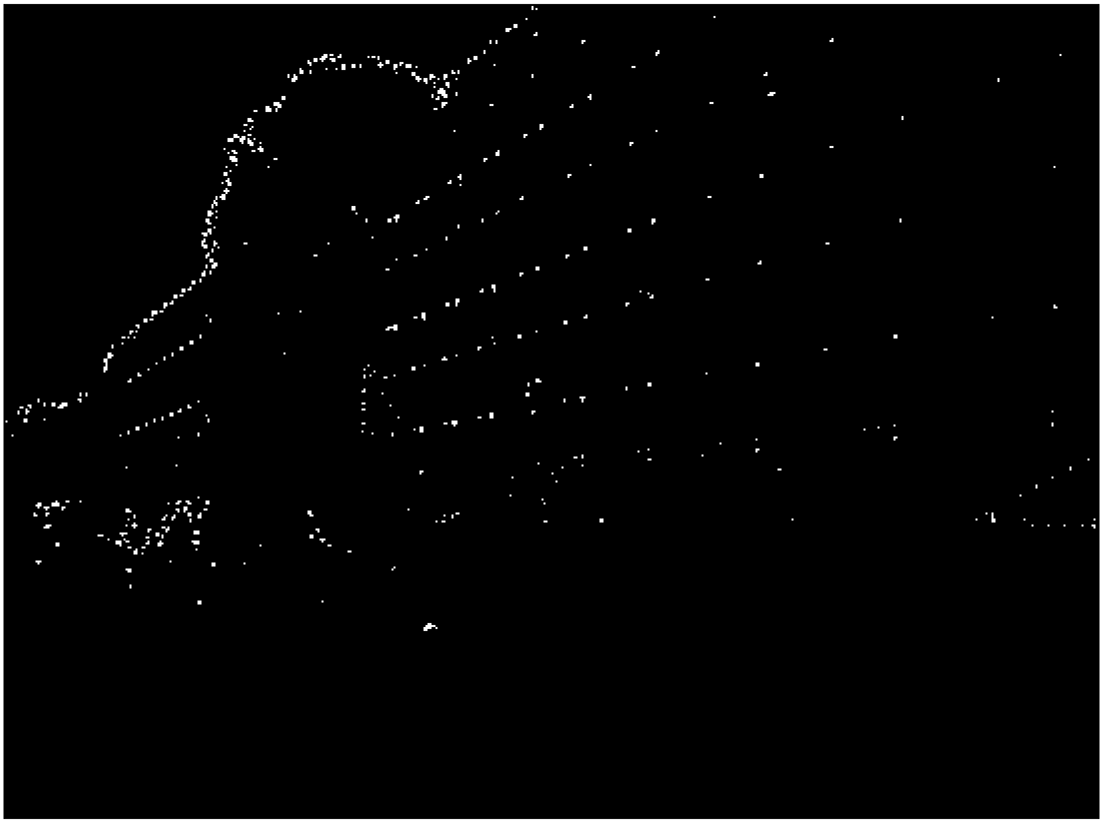
Kunal Dhaimade - CS 558 Homework Assignment 2

```
'''To select a single pair of points from the sample space'''
def select_one(sample_space):
    space_len = len(sample_space)
    pti = rn.randint(0, space_len-1)
    l_mod = sample_space[pti]
    sample_space.remove(l_mod)
    return sample_space, l_mod

'''To obtain the Hough transform, also with local maxima calculated'''
def hough(ip_im):
    ip_im = np.asarray(ip_im)
    r, c = ip_im.shape
    bin_r = int(((r + (c/2)) * 2) + 1)
    offset = int((bin_r - 1) / 2)
    bin_c = 181
    bin_space = np.zeros((bin_r, bin_c))
    for i in range(r):
        for j in range(c):
            if(ip_im[i][j] != 0):
                for angle in range(bin_c):
                    r = int(((j * mt.cos(mt.radians(angle))) + (i * mt.sin(mt.radians(angle)))) + offset)
                    bin_space[r][angle] = bin_space[r][angle] + 5
    bin_space = bin_space.astype(np.uint8)
    bin_space_im = Image.fromarray(bin_space)
    bin_space = resize_image(bin_space, 3)
    op_im = np.zeros((bin_r, bin_c))
    for i in range(1, bin_r-1):
        for j in range(1, bin_c-1):
            if(bin_space[i][j] != max(bin_space[i-1][j-1], bin_space[i-1][j], bin_space[i-1][j+1], bin_space[i][j-1],
bin_space[i][j], bin_space[i][j+1], bin_space[i+1][j-1], bin_space[i+1][j], bin_space[i+1][j+1])):
                op_im[i-1][j-1] = 0
            else:
                op_im[i-1][j-1] = bin_space[i][j]
    op_im = Image.fromarray(op_im)
    return op_im, bin_space_im

'''To plot the 4 lines obtained from the Hough transform'''
def hough_map(ip_im, ip_im2, hough_im):
    hough_im_c = np.asarray(hough_im)
    bin_r, bin_c = hough_im_c.shape
    r, col = np.asarray(ip_im).shape
    offset = int((bin_r - 1) / 2)
    max_pt_list = []
    max_pt = 0
    no_of_pts = 10
    ip_im_draw = ImageDraw.Draw(ip_im)
    ip_im2_draw = ImageDraw.Draw(ip_im2)
    hough_im_c.setflags(write = 1)
```

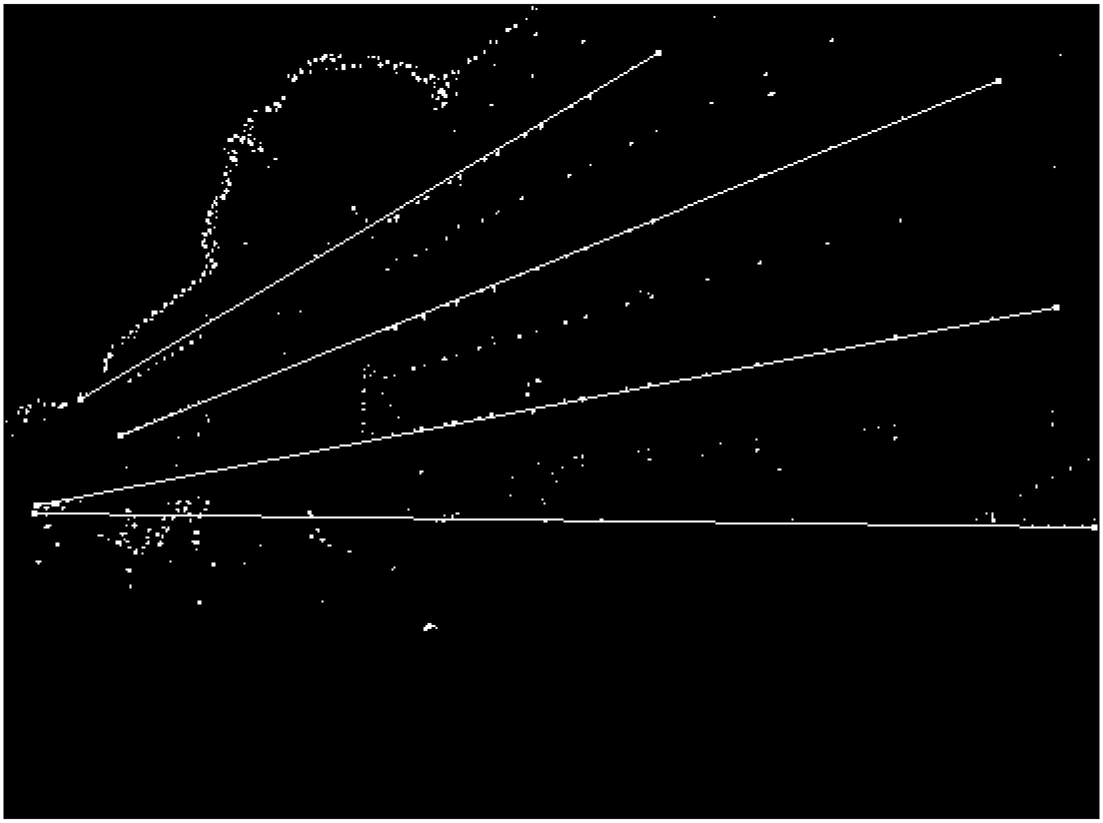
```
    while(no_of_pts != 0):
        for i in range(bin_r):
            for j in range(bin_c):
                if(hough_im_c[i][j] > max_pt):
                    max_pt = hough_im_c[i][j]
                    r = i - offset
                    angle = j
                    acc_i = i
                    acc_j = j
        hough_im_c[acc_i][acc_j] = 0
        a = mt.cos(mt.radians(angle))
        b = mt.sin(mt.radians(angle))
        x0 = int(r * a)
        y0 = int(r * b)
        m = (-mt.cos(mt.radians(angle)))/(mt.sin(mt.radians(angle)))
        c = r/(mt.sin(mt.radians(angle)))
        x1 = x0 + 400
        y1 = (x1 * m) + c
        x2 = x0 - 50
        y2 = (x2 * m) + c
        #ip_im = ip_im.convert('RGB')
        if(no_of_pts in [10, 9, 8, 2]):
            ip_im_draw = endpt_draw(ip_im_draw, x1, y1, x2, y2)
            ip_im_draw.line((x0, y0, x1, y1), fill = 255)
            ip_im_draw.line((x0, y0, x2, y2), fill = 255)
            ip_im2_draw = endpt_draw(ip_im2_draw, x1, y1, x2, y2)
            ip_im2_draw.line((x0, y0, x1, y1), fill = 255)
            ip_im2_draw.line((x0, y0, x2, y2), fill = 255)
        hough_im_c[r+offset][angle] = 0
        max_pt_list.append([r+offset, angle])
        no_of_pts = no_of_pts - 1
        max_pt = 0
    return ip_im, ip_im2

'''To draw the 3x3 square at the end-points'''
def endpt_draw(ip_im_draw, x0, y0, x1, y1):
    ip_im_draw.line((x0-1, y0+1, x0+1, y0+1), fill = 255)
    ip_im_draw.line((x0-1, y0+1, x0-1, y0-1), fill = 255)
    ip_im_draw.line((x0+1, y0+1, x0+1, y0-1), fill = 255)
    ip_im_draw.line((x0-1, y0-1, x0+1, y0-1), fill = 255)
    ip_im_draw.line((x1-1, y1+1, x1+1, y1+1), fill = 255)
    ip_im_draw.line((x1-1, y1+1, x1-1, y1-1), fill = 255)
    ip_im_draw.line((x1+1, y1+1, x1+1, y1-1), fill = 255)
    ip_im_draw.line((x1-1, y1-1, x1+1, y1-1), fill = 255)
    return ip_im_draw

'''Main function'''
def main():
```

```
    og_im = Image.open("road.png")
    og_im.show()
    sigma = 0.5
    size = 5
    g_im = gauss_filter(og_im, size, sigma)
    h_im = hessian_detector(g_im)
    h_im.show()
    r_im = h_im.copy()
    r_og_im = og_im.copy()
    sample_space, n_pts = gen_sample_space(r_im)
    draw_model = ransac(r_im, sample_space, n_pts)
    r_op_im, r_op_im_raw = plot_ransac(r_og_im, r_im, draw_model)
    r_op_im_raw.show()
    r_op_im.show()
    ho_im = h_im.copy()
    ho_og_im = og_im.copy()
    ho_int_im, ho_tr_im = hough(ho_im)
    ho_tr_im.show()
    ho_op_im, ho_op_im_raw = hough_map(ho_og_im, ho_im, ho_int_im)
    ho_op_im_raw.show()
    ho_op_im.show()

main()
```

Output Images:

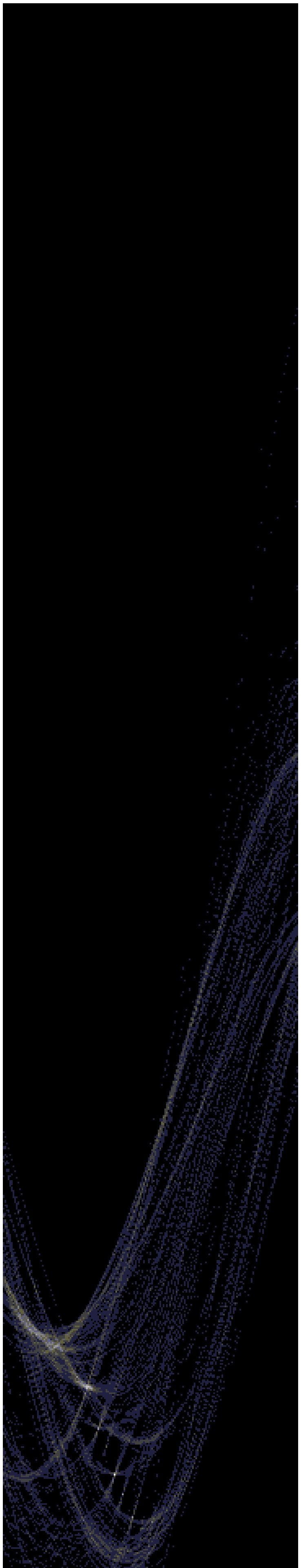


Output of the Hessian Detector

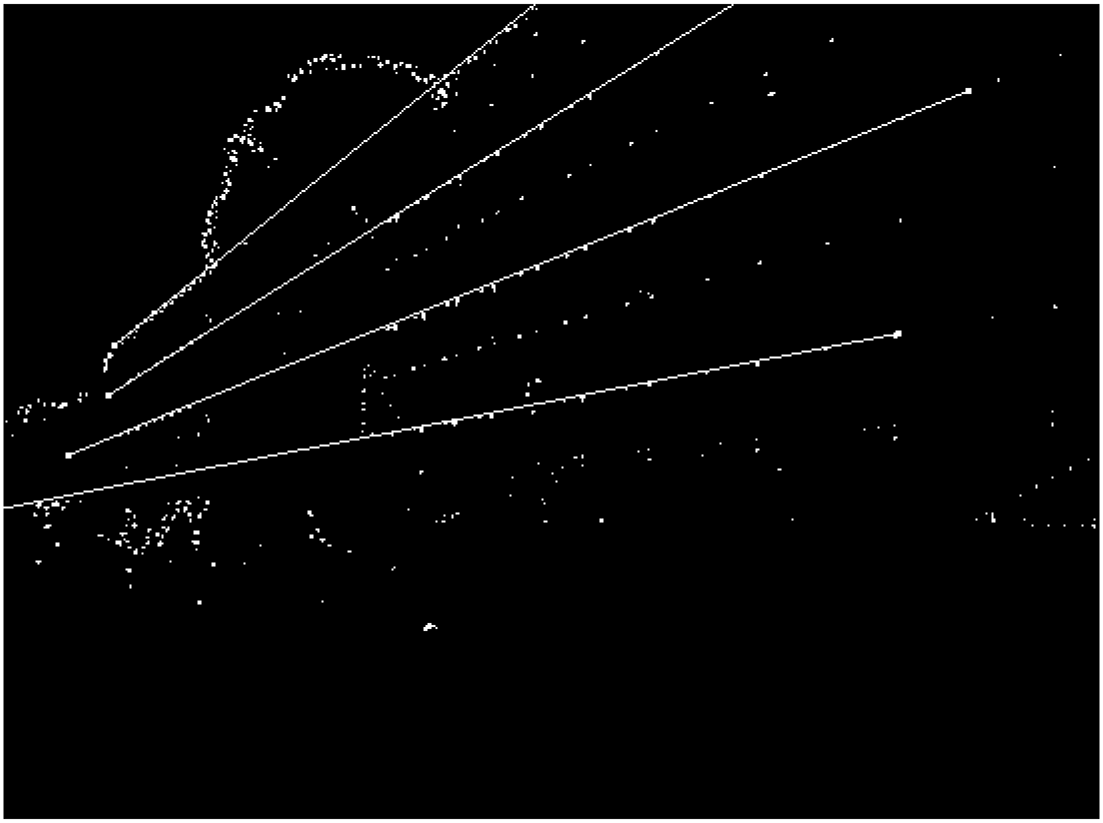


Output of RANSAC on the result of Hessian Detector. Repetitions are detected sometimes.

Lines found using RANSAC on the original image.



The Hough Transform obtained from the result of Hessian Detector



Lines found using Hough Transform drawn on the result of Hessian Detector

Lines found using Hough Transform drawn on the original image

Notes:

For the Hessian Detector, the threshold is the primary parameter. A good threshold results in well-defined points. The corners of the vegetation are also detected, which can errors in the process of line detection. It is crucial to normalize the intensities of the pixels after the process of calculating the Determinant of Hessian. If this step is not performed, then a chipped image form is obtained. Non-maximum suppression is performed to obtain the local maxima. The initial step is a Gaussian blur, performed with a sigma value of 0.5 and a filter size of 5.

For RANSAC operation, a sample space is necessary, which is generated using the output of the Hessian Detector. The ransac function has initial parameters, such as the number of iterations (set to 500), threshold distance (set to 3), the required number of inliers for a potential fit model (set to 70) and the inlier to total points ratio (set to 0.06). There may be repetitions in detection sometimes. Lowering the number of iterations will result in less detections for the same ratio and the same number of required inliers. Hence, the tweaking of parameters has to be done carefully. Lowering the number of iterations may require lowering the required number of inliers as well. Line models that seem like a potential fit are stored and the strongest 4 lines are then plotted.

For Hough Transform, the parameters of the bin space are important. The angle parameter goes from 0 to 180, while the parameter r may vary, and depends upon the angle, as bin space is generated using the parametric form. The local maxima from the Hough transform are obtained, and then converted back to lines. 4 of such lines that look like the best fit are drawn.