CS 558: Homework Assignment 1

Student Name: Yiqun Peng

CWID: 10450566

Problem 1: Pre-processing

1.Load the image and convert it to a single channel gray image. (By using OpenCV)

Code:

```
pre_image = cv2.imread("road.png")
gray_image = cv2.cvtColor(pre_image, cv2.COLOR_BGR2GRAY)
```

Result:

2. The 5 X 5 gaussian filter operator is made by using the gaussian function formula.
Formula:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

Code:

```
sigma = 1
gaussian_sum = 0
gaussian = np.zeros([5, 5])
for i in range(5):
    for j in range(5):
        gaussian[i, j] = math.exp((-1 / (2 * sigma * sigma)) * (np.square(i-3)
                 + np.square(j-3))) / (2 * math.pi * sigma * sigma)
        gaussian_sum = gaussian_sum + gaussian[i, j]

gaussian = gaussian / gaussian_sum
```

3. Gaussian filtering is accomplished by convolution of gaussian operator with original image matrix. flatten the graph by increasing the standard deviation of sigma.
Code:

```
row, col = img.shape
result = np.zeros([row - 5, col - 5])
for i in range(row - 5):
    for j in range(col - 5):
        result[i, j] = np.sum(img[i:i + 5, j:j + 5] * gaussian)
```

Result:

Sigma = 0.5

Sigma = 5

4. By using the gaussian filtering results and the convolution of Soebl operator with the original image, the vertical and horizontal gradient values of each point are calculated, and the gradient value of a certain point is calculated by combining the two results.
Sobel operator:

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} \quad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix}$$

Code:

```
sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
sobel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
result = np.zeros(img.shape)
sx = np.zeros(img.shape)
sy = np.zeros(img.shape)
row, col = img.shape
```

```
for i in range(1,row-2):
    for j in range(1,col - 2):
        sx[i + 1, j + 1] = np.sum(img[i:i + 3, j:j + 3] * sobel_x)
        sy[i + 1, j + 1] = np.sum(img[i:i + 3, j:j + 3] * sobel_y)
        ''' gradient values '''
        result[i + 1, j + 1] = np.sqrt(np.square(sx[i + 1, j + 1])
                                       + np.square(sx[i + 1, j + 1]))

return result,sx,sy
```

Result:

5. The value of the second derivative is calculated by the gradient value. By calculating the average value of 3X3 neighbors, four values in the hessian matrix are obtained, and two eigenvalues are calculated. According to the eigenvalue, the formula is used to find the value of the corner point. Where lambda =0.05 is used to suppress visible lines. Using non-maximum suppression: the relationship between the threshold parameter and the corner value is compared, and the corner value satisfying the requirement is reserved.

Fomula:

$$\begin{bmatrix} I_x^2(\mathbf{x},\sigma_D) & I_x(\mathbf{x},\sigma_D)I_y(\mathbf{x},\sigma_D) \\ I_x(\mathbf{x},\sigma_D)I_y(\mathbf{x},\sigma_D) & I_y^2(\mathbf{x},\sigma_D) \end{bmatrix}$$

$$\text{cornerness} = \det(M) - \lambda \operatorname{trace}(M)$$

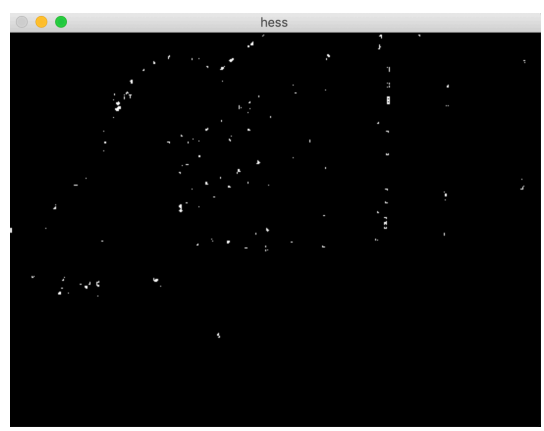
Code:

```
dy,dx = np.gradient(img)
Ixx = dx ** 2
Iyy = dy ** 2
Ixy = dx * dy
row, col = img.shape
hess_img = np.zeros(img.shape)
cornermax = 0
```

```
for i in range(1,row - 1):
    for j in range(1, col - 1):
        DIxx = Ixx[i - 1:i + 2, j - 1:j + 2]
        DIyy = Iyy[i - 1:i + 2, j - 1:j + 2]
        DIxy = Ixy[i - 1:i + 2, j - 1:j + 2]
        axx = DIxx.sum()/9
        ayy = DIyy.sum()/9
        axy = DIxy.sum()/9
        det = (axx * ayy) - (axy ** 2)
        trace = axx + ayy
        corner = det - 0.05 * (trace ** 2)
        if corner > cornermax:
            cornermax = corner
        if corner > thre:
            hess_img[i][j] = 255
```

Result:

Threshold = 2000000



Threshold = 10000000

Problem 2: RANSAC

1.Set the distance threshold and the maximum number of inlier. Generate x and y coordinates at random and draw a line according to the coordinates.

Code:

```
def ransac(img, inlier_number, thre):
    row, col = img.shape
    x1 = x2 = y1 = y2 = 1
    best_lines = []
    best_points = []
    best_b = 0
    best_k = 0
    current = 0
    times = 4
    inliers = np.zeros(img.shape)
    while(times):
        pre = 0
        while(pre<inlier_number):
            while(img[y1,x1] != 255 or img[y2,x2] != 255
                  or x1 == x2 or inliers[y1,x1]==225 or inliers[y2,x2]==225):
                x1 = random.randint(1, col-1)
                x2 = random.randint(1, col-1)
                y1 = random.randint(1, row-1)
                y2 = random.randint(1, row-1)
            k = (y2 - y1 ) / (x2 - x1)
            b = y1 - k * x1
```

2.Traverse the feature point matrix and calculate the distance from each point to the line. If the distance is less than the distance threshold, the feature points are recorded as inlier. Similarly, the number n line is generated. If the number of inlier is greater than that of the previous line, the current number is set as the maximum and the equation of the line is recorded. Continue to look for the other three lines and remove the previously recorded inlier point coordinates through the inlier matrix.

Code:

```
    inlier = []
    for i in range(1, row - 1):
        for j in range(1, col - 1):
            if img[i,j] == 255:
                distance = abs((i - k * j - b)/math.sqrt(1 + k ** 2))
                #print("d=",distance)
                if distance < thre:
                    current = current + 1
                    inlier.append((i,j))
    if current > pre:
        pre = current
        if(pre >= inlier_number):
            best_lines.append([b,k,current])
            best_points.append([x1,y1,x2,y2])
            for inl in inlier:
                ii,ij = inl
                inliers[ii,ij]=225
        print("dayule")
    x1 = x2 = y1 = y2 = 1
    print(pre)
times = times-1
```

3.Change the distance threshold and the maximum number of inlier, find the most suitable line segment, and draw the image.

Code:

```
best_points,best_lines = ransac(hess_image,30,6)
ransac_image = copy.deepcopy(hess_image)
for best_point in best_points:
    px1,py1,px2,py2 = best_point
    cv2.line(ransac_image,(px1,py1),(px2,py2),255,2)
    cv2.line(pre_image,(px1,py1),(px2,py2),(0,255,0),2)
```

Result:

Threshold=5, Inlier=20

Threshold=6, Inlier=30

Problem 3: Hough Transform.

1. Using canny edge detection. Accept the gradient value and direction of x and y in sobel_img, divide different angles and keep them in the direction matrix.

Code:

```
def canny(img, sx,sy,tmin,tmax):
    row,col = img.shape
    direction = np.zeros(sx.shape)
    dir = np.zeros(sx.shape)
    for i in range(1, row - 2):
        for j in range(1, col - 2):
            if(sx[i + 1, j + 1]!=0):
                direction[i + 1, j + 1] = np.arctan(sy[i + 1, j + 1]/sx[i + 1, j + 1])
            if direction[i + 1, j + 1] >= 0.375*math.pi and direction[i + 1, j + 1] <0.5*math.pi:
                dir[i + 1, j + 1] = 2
            elif direction[i + 1, j + 1] >= -0.375*math.pi and direction[i + 1, j + 1] < -0.125*math.pi:
                dir[i + 1, j + 1] = 3
            elif direction[i + 1, j + 1] >= -0.125*math.pi and direction[i + 1, j + 1] < 0.125*math.pi:
                dir[i + 1, j + 1] = 0
            elif direction[i + 1, j + 1] >= 0.125*math.pi and direction[i + 1, j + 1] < 0.375*math.pi:
                dir[i + 1, j + 1] = 135
            else:
                dir[i + 1, j + 1] = 2
```

2. Non - maximum suppression of gradient in the same direction. The edge detection is completed by the hysteresis threshold filtering.

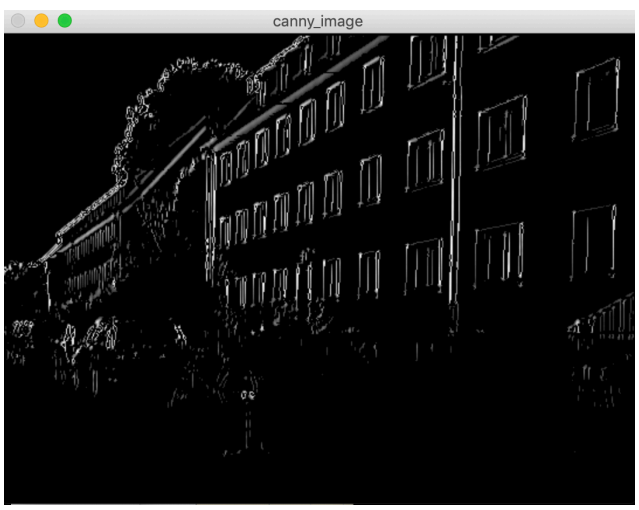
Code:

```
result = np.zeros(img.shape)
for k in range(1, row - 2):
    for l in range(1, col - 2):
        t = img[k - 1, l]
        b = img[k + 1, l]
        r = img[k, l + 1]
        lt = img[k, l - 1]
        tr = img[k - 1, l + 1]
        tl = img[k - 1, l - 1]
        lb = img[k + 1, l - 1]
        rb = img[k + 1, l + 1]

        if dir[k, l] == 0:
            g1 = r
            g2 = lt
        elif dir[k, l] == 1:
            g1 = tr
            g2 = lb
```

```
        elif dir[k, l] == 2:
            g1 = b
            g2 = t
        elif dir[k, l] == 3:
            g1 = tl
            g2 = rb
        if img[k,l]>=g1 and img[k,l]>=g2:
            result[k,l] = img[k,l]
        else:
            result[k, l] = 0
        if img[k, l] >tmax:
            result[k, l] = img[k, l]
        if img[k, l] < tmin:
            result[k, l] = 0
```

Result:



Min = 50, Max = 100

3.Hough transform was used to detect the four most favorable supported lines. Go through the picture, and I'm going to represent x and y in polar coordinates. Define the theta range (-90 to 90), and calculate the value by x,y, and theta. Use bins of the accumulator to store the theta and R, and if the image is 255 pixels, add up the value of bins of the accumulator.

Code：

```
acc = copy.deepcopy(accu)
amax = np.max(acc)
print(amax)
acrow,accol = acc.shape
times = 10
pre_theta = 180
pre_r = acrow
while times:
    amax = np.max(acc)
    print("max,",amax)
    for i in range(acrow-1):
        for j in range(accol-1):
            if acc[i,j] == amax:
                theta = j-90
                r = i-acrow/2
```

```
r_len = int(np.sqrt(row**2+col**2))
print(r_len)
for i in range(1,row-2):
    for j in range(1, col-2):
        if img[i][j] == 255:
            points.append((j, i))
new_points = []
for point in points:
    tmp = []
    for theta in range(-90, 90):
        x, y = point
        r2 = x * np.cos(theta / 180 * np.pi) + y * np.sin(theta / 180 * np.pi)
        tmp.append((theta, int(r2)))
    new_points.append(tmp)

accu = np.zeros([r_len*2, 180]).astype(np.uint8)
accu_img = np.zeros([r_len*2, 180]).astype(np.uint8)
for new_point in new_points:
    for p in new_point:
        theta, r2 = p
        accu[r2+r_len][theta+90]= accu[r2+r_len][theta+90]+1
        if accu_img[r2+r_len][theta+90]<=225:
            accu_img[r2+r_len][theta+90] = accu[r2+r_len][theta+90]+80
        if accu_img[r2+r_len][theta+90] > 225:
            accu_img[r2+r_len][theta+90] = 225
return accu,accu_img,new_points
```

4.The index of the four positions with the largest calculator value is reverted to theta and r. Find the equation of the line b and k. Draw the equations of lines and polar coordinates.

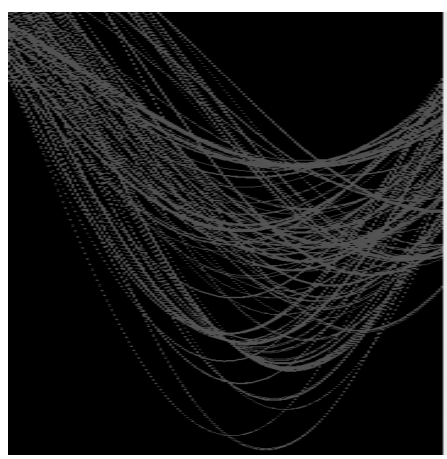
Code：

```
maxlist = []
acc = copy.deepcopy(accu)
amax = np.max(acc)
print(amax)
acrow,accol = acc.shape
times = 10
pre_theta = 180
pre_r = acrow
while times:
    amax = np.max(acc)
    print("max,",amax)
    for i in range(acrow-1):
        for j in range(accol-1):
            if acc[i,j] == amax:
                theta = j-90
                r = i-acrow/2
```

```
if abs(theta-pre_theta)>10 or abs(r-pre_r)>10:
    acc[i,j]=0
    pre_theta = theta
    pre_r = r
    i=acrow-1
    j=accol-1
    print("i:",i,"j",j)
else:
    acc[i,j]=0
```

```
if np.sin(theta/180*np.pi)!=0:
    k = -np.cos(theta/180*np.pi) / np.sin(theta/180*np.pi)
    b = r/np.sin(theta/180*np.pi)
    cv2.line(hough_image, (int(-b / k), 0), (0, int(b)), (255, 0, 0), 2)
times = times-1
```

Result：



Complete code:

```
from PIL import Image
import cv2
import math
import numpy as np
import matplotlib.pyplot as plt
import copy
import random

def gaussianfiller(img):
    ''' Make 5x5 gaussian filter operator '''
    sigma = 1
    gaussian_sum = 0
    gaussian = np.zeros([5, 5])
    for i in range(5):
        for j in range(5):
            gaussian[i, j] = math.exp((-1 / (2 * sigma * sigma)) * (np.square(i-
3)
                    + np.square(j-3))) / (2 * math.pi * sigma * sigma)
            gaussian_sum = gaussian_sum + gaussian[i, j]

    gaussian = gaussian / gaussian_sum
    print("Gau:", gaussian)

    '''
    Gaussian filtering is accomplished by
    convolution of gaussian operator with
    original image matrix
    '''
    row, col = img.shape
    result = np.zeros([row - 5, col - 5])
    for i in range(row - 5):
        for j in range(col - 5):
            result[i, j] = np.sum(img[i:i + 5, j:j + 5] * gaussian)

    return result


def sobel(img):
```

```python
    """Soebl operator"""
    sobel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    sobel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])
    result = np.zeros(img.shape)
    sx = np.zeros(img.shape)
    sy = np.zeros(img.shape)
    row, col = img.shape
    ''' convolution of soebl operator with the original image '''
    for i in range(1,row-2):
        for j in range(1,col - 2):
            sx[i + 1, j + 1] = np.sum(img[i:i + 3, j:j + 3] * sobel_x)
            sy[i + 1, j + 1] = np.sum(img[i:i + 3, j:j + 3] * sobel_y)
            ''' gradient values '''
            result[i + 1, j + 1] = np.sqrt(np.square(sx[i + 1, j + 1])
                                + np.square(sx[i + 1, j + 1]))

    return result,sx,sy

def Hessian(img, thre):
    """ second derivative is calculated """
    dy,dx = np.gradient(img)
    Ixx = dx ** 2
    Iyy = dy ** 2
    Ixy = dx * dy
    row, col = img.shape
    hess_img = np.zeros(img.shape)
    cornermax = 0
    for i in range(1,row - 1):
        for j in range(1, col - 1):
            DIxx = Ixx[i - 1:i + 2, j - 1:j + 2]
            DIyy = Iyy[i - 1:i + 2, j - 1:j + 2]
            DIxy = Ixy[i - 1:i + 2, j - 1:j + 2]
            ''' calculating the average value of 3X3 neighbors '''
            axx = DIxx.sum()/9
            ayy = DIyy.sum()/9
            axy = DIxy.sum()/9
            ''' two eigenvalues are calculated '''
            det = (axx * ayy) - (axy ** 2)
            trace = axx + ayy
            corner = det - 0.05 * (trace ** 2)
            ''' non-maximum suppression '''
```

```
            if corner > cornermax:
                cornermax = corner
            if corner > thre:
                hess_img[i][j] = 255
            if corner > 20000000:
                hess_img[i][j] = 0
    print("cornermax: ",cornermax)
    return hess_img

def ransac(img, inlier_number, thre):
    row, col = img.shape
    x1 = x2 = y1 = y2 = 1
    best_lines = []
    best_points = []
    best_b = 0
    best_k = 0
    current = 0
    times = 4
    inliers = np.zeros(img.shape)
    while(times):
        pre = 0
        while(pre<inlier_number):
            while(img[y1,x1] != 255 or img[y2,x2] != 255
                 or x1 == x2 or inliers[y1,x1]==225 or inliers[y2,x2]==225):
                x1 = random.randint(1, col-1)
                x2 = random.randint(1, col-1)
                y1 = random.randint(1, row-1)
                y2 = random.randint(1, row-1)
            k = (y2 - y1 ) / (x2 - x1)
            b = y1 - k * x1
            current = 0
            print(x1,y1,x2,y2)
            inlier = []
            '''Traverse the feature point matrix and calculate the distance'''
            for i in range(1, row - 1):
                for j in range(1, col - 1):
                    if img[i,j] == 255:
                        distance = abs((i - k * j - b)/math.sqrt(1 + k ** 2))
                        #print("d=",distance)
                        '''
                        If the distance is less than the distance threshold,
```

```
                the feature points are recorded as inlier.
                '''
                if distance < thre:
                    current = current + 1
                    inlier.append((i,j))
        if current > pre:
            '''
            current number is set as the maximum
            and the equation of the line is recorded.
            '''
            pre = current
            if(pre >= inlier_number):
                best_lines.append([b,k,current])
                best_points.append([x1,y1,x2,y2])
                for inl in inlier:
                    ii,ij = inl
                    inliers[ii,ij]=225
            print("dayule")
        x1 = x2 = y1 = y2 = 1
        print(pre)
    times = times-1

# print("best",best_lines)
# x = []
# y = []
# for i in range(1, row - 1):
#     for j in range(1, col - 1):
#         if img[i,j]==255:
#             x.append(j)
#             y.append(i)
# X = np.array(x)
# fig = plt.figure()
# ax1 = fig.add_subplot(1, 1, 1)
# ax1.scatter(x, y, s=1.5)
# ax1.set_xlabel("x")
# ax1.set_ylabel("y")
# for pl in range (0,4):
#     Y = best_lines[pl][1] * X + best_lines[pl][0]
#     ax1.plot(X, Y)
# plt.show()
return best_points,best_lines
```

```
def canny(img, sx,sy,tmin,tmax):
    row,col = img.shape
    direction = np.zeros(sx.shape)
    dir = np.zeros(sx.shape)
    for i in range(1, row - 2):
        for j in range(1, col - 2):
            if(sx[i + 1, j + 1]!=0):
                direction[i + 1, j + 1] = np.arctan(sy[i + 1, j + 1]/sx[i + 1, j +
1])
            if direction[i + 1, j + 1] >= 0.375*math.pi and direction[i + 1, j +
1] <0.5*math.pi:
                dir[i + 1, j + 1] = 2
            elif direction[i + 1, j + 1] >= -0.375*math.pi and direction[i + 1, j
+ 1] < -0.125*math.pi:
                dir[i + 1, j + 1] = 3
            elif direction[i + 1, j + 1] >= -0.125*math.pi and direction[i + 1, j
+ 1] < 0.125*math.pi:
                dir[i + 1, j + 1] = 0
            elif direction[i + 1, j + 1] >= 0.125*math.pi and direction[i + 1, j
+ 1] < 0.375*math.pi:
                dir[i + 1, j + 1] = 135
            else:
                dir[i + 1, j + 1] = 2

    result = np.zeros(img.shape)
    for k in range(1, row - 2):
        for l in range(1, col - 2):
            t = img[k - 1, l]
            b = img[k + 1, l]
            r = img[k, l + 1]
            lt = img[k, l - 1]
            tr = img[k - 1, l + 1]
            tl = img[k - 1, l - 1]
            lb = img[k + 1, l - 1]
            rb = img[k + 1, l + 1]

            if dir[k, l] == 0:
                g1 = r
                g2 = lt
            elif dir[k, l] == 1:
```

```
            g1 = tr
            g2 = lb
        elif dir[k, l] == 2:
            g1 = b
            g2 = t
        elif dir[k, l] == 3:
            g1 = tl
            g2 = rb
        if img[k,l]>=g1 and img[k,l]>=g2:
            result[k,l] = img[k,l]
        else:
            result[k, l] = 0
        if img[k, l] >tmax:
            result[k, l] = img[k, l]
        if img[k, l] < tmin:
            result[k, l] = 0

    return result

def hough(img):
    points = []
    row,col = img.shape
    r_len = int(np.sqrt(row**2+col**2))
    print(r_len)
    for i in range(1,row-2):
        for j in range(1, col-2):
            if img[i][j] == 255:
                points.append((j, i))
    new_points = []
    for point in points:
        tmp = []
        for theta in range(-90, 90):
            x, y = point
            r2 = x * np.cos(theta / 180 * np.pi) + y * np.sin(theta / 180 *
np.pi)
            tmp.append((theta, int(r2)))
        new_points.append(tmp)

    accu = np.zeros([r_len*2, 180]).astype(np.uint8)
    accu_img = np.zeros([r_len*2, 180]).astype(np.uint8)
    for new_point in new_points:
```

```
        for p in new_point:
            theta, r2 = p
            accu[r2+r_len][theta+90]= accu[r2+r_len][theta+90]+1
            if accu_img[r2+r_len][theta+90]<=225:
                accu_img[r2+r_len][theta+90] = accu[r2+r_len][theta+90]+80
            if accu_img[r2+r_len][theta+90] > 225:
                accu_img[r2+r_len][theta+90] = 225
    return accu,accu_img,new_points


''' Load the image and convert it to a single channel gray image '''
pre_image = cv2.imread("road.png")
gray_image = cv2.cvtColor(pre_image, cv2.COLOR_BGR2GRAY)
print("gray:",gray_image)

hough_image = copy.deepcopy(pre_image)

'''gaussian'''
gaussian_image = gaussianfiller(gray_image).astype('uint8')
print("gau:",gaussian_image)

s_image, sx, sy = sobel(gaussian_image)
sobel_image = s_image.astype('uint8')
print(sobel_image)

'''hessian'''
hess_image = Hessian(sobel_image,11000000).astype('uint8')
row,col = hess_image.shape
for i in range(0,row-1):
    for j in range(0, col-1):
        if hess_image[i,j]==255:
            cv2.circle(pre_image, (j, i),2, (0, 0, 255), 0)

'''ransac'''
best_points,best_lines = ransac(hess_image,30,5)
ransac_image = copy.deepcopy(hess_image)
for best_point in best_points:
    px1,py1,px2,py2 = best_point
    cv2.line(ransac_image,(px1,py1),(px2,py2),255,2)
    cv2.line(pre_image,(px1,py1),(px2,py2),(0,255,0),2)
```

```
'''hough'''
canny_image = canny(sobel_image,sx,sy,80,200).astype('uint8')

accu,hough_space,points = hough(hess_image)
maxlist = []
acc = copy.deepcopy(accu)
amax = np.max(acc)
print(amax)
acrow,accol = acc.shape
times = 10
pre_theta = 180
pre_r = acrow
while times:
    amax = np.max(acc)
    print("max,",amax)
    for i in range(acrow-1):
        for j in range(accol-1):
            if acc[i,j] == amax:
                theta = j-90
                r = i-acrow/2
                print("r",r,"theta,",theta,"r_pre",pre_r,"theta_pre,",pre_theta)
                if abs(theta-pre_theta)>10 or abs(r-pre_r)>10:
                    acc[i,j]=0
                    pre_theta = theta
                    pre_r = r
                    i=acrow-1
                    j=accol-1
                    print("i:",i,"j",j)
                else:
                    acc[i,j]=0

    if np.sin(theta/180*np.pi)!=0:
        k = -np.cos(theta/180*np.pi) / np.sin(theta/180*np.pi)
        b = r/np.sin(theta/180*np.pi)
        cv2.line(hough_image, (int(-b / k), 0), (0, int(b)), (255, 0, 0), 2)
    times = times-1

cv2.imshow("gau", gaussian_image)
cv2.imshow("sobel", sobel_image)
cv2.imshow("hess", hess_image)
cv2.imshow("ransac_image", ransac_image)
```

```
cv2.imshow("canny_image", canny_image)
cv2.imshow('hough_space', hough_space)
cv2.imshow("hough_image", hough_image)
cv2.imshow("pre", pre_image)

cv2.waitKey()
cv2.destroyAllWindows()
```