assignment4

January 18, 2021

1 Machine Learning and Computer Vision

1.1 Assigment 4

This assignment contains 1 programming exercises with 2 sections.

1.2 Problem 1: Hough Transform

This problem we will introduce Hough Transform. The Hough transform is a feature extraction technique used in image analysis, computer vision, and digital image processing. The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure.

(i) Implement the Hough Transform (HT) using the (phi, theta) parameterization as described in GW Third Edition p. 733-738 (please see 'HoughTransform.pdf' provided in the folder). Use accumulator cells with a resolution of 1 degree in theta and 1 pixel in phi.

(ii) Produce a simple 11 x 11 test image made up of zeros with 5 ones in it, arranged like the 5 points in GW Third Edition Figure 10.33(a).

Compute and display its Hough Transform; the result should look like GW Third Edition Figure 10.33(b). Threshold the HT by looking for any (phi, theta) cells that contains more than 2 votes then plot the corresponding lines in (x,y)-space on top of the original image.

(iii) Load in the image 'lane.png'.

Compute and display its edges using the edge detector, you can use canny edge detector, which you have implemented in Problem 1, or use OpenCV edge detection operator, such as Sobel, etc.

Now compute and display the Hough Transform of the binary edge image. As before, threshold the HT and plot the corresponding lines atop the original image; this time, use a threshold of 75% maximum accumulator count over the entire HT, i.e. 0.75*max(HT(:)).

(iv) We would like to only show line detections in the driver's lane and ignore any other line detections such as the lines resulting from the neighboring lane closest to the bus, light pole, and sidewalks. Using the thresholded HT from the 'lanes.png' image in the previous part, show only the lines corresponding to the line detections from the driver's lane by thresholding the HT again using a specified range of theta this time. What are the approximate theta values for the two lines in the driver's lane?

Things to turn in:

- Hough Transform plot should have colorbars next to them
- Line overlays should be clearly visible (adjust line width if needed)
- HT image axes should be properly labeled with name and values (see Figure 10.33(b) in the HoughTransform PDF for example)
- 3 images from 2(ii): original image, Hough Transform plot, original image with detected lines
- 4 images from 2(iii): original image, binary edge image, Hough Transform plot, original image with detected lines
- 1 image from 2(iv): original image with detected lines
- theta values from 2(iv)

[84]:

```
#Hough Transform Function

def Hough_transform(img):
    # Rho and Theta ranges
    thetas = np.deg2rad(np.arange(-90.0, 90.0, 1))
    width, height = img.shape
    diag_len = int(round(math.sqrt(width * width + height * height)))   #␣
 ↪max_dist
    rhos = np.linspace(-diag_len, diag_len, diag_len * 2)

    # Cache some resuable values
    cos_t = np.cos(thetas)
    sin_t = np.sin(thetas)
    num_thetas = len(thetas)

    # Hough accumulator array of theta vs rho
    accumulator = np.zeros((2 * diag_len, num_thetas), dtype=np.uint64)
    y_idxs, x_idxs = np.nonzero(img)  # (row, col) indexes to edges

    # Vote in the hough accumulator
    for i in range(len(x_idxs)):
        x = x_idxs[i]
        y = y_idxs[i]

        for t_idx in range(num_thetas):
            # Calculate rho. diag_len is added for a positive index
            rho = round(x * cos_t[t_idx] + y * sin_t[t_idx]) + diag_len
            accumulator[rho, t_idx] += 1

    return accumulator, rhos, thetas

def hough_line_peaks(accumulator, threshold):
    '''
    Edition 1
```

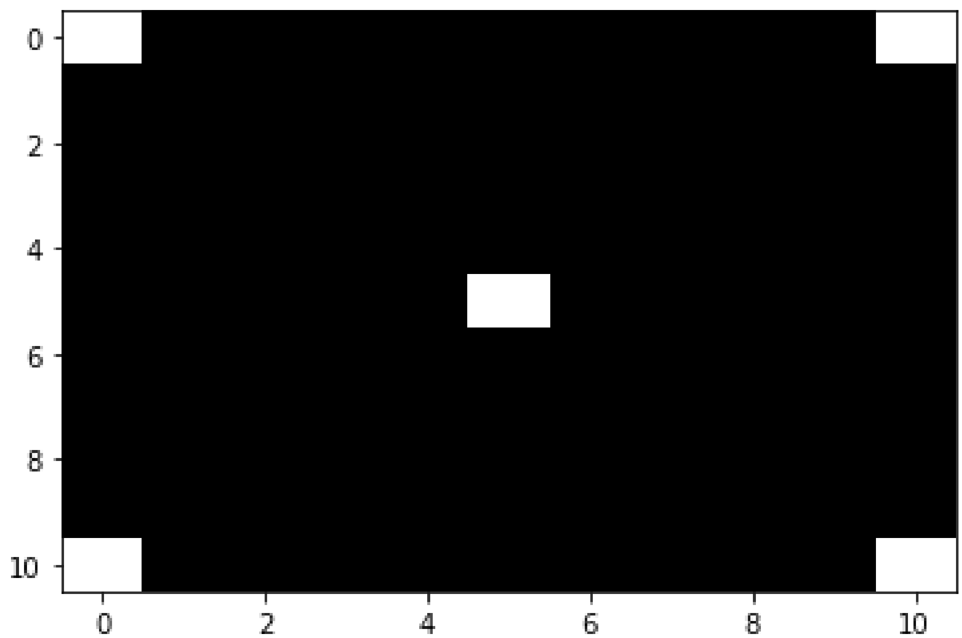
```
    #find the intersections
    accum = accumulator.ravel()
    peaks = signal.find_peaks(accum)
    print(peaks)

    for i in range(len(peaks)):
        rho = rhos[int(peaks[i] / accumulator.shape[1])]
        theta = thetas[int(peaks[i] % accumulator.shape[1])]

        y_idxs, x_idxs = np.nonzero(img)
        y_list = []
        x_list = []
        for j in range(len(x_idxs)):
            x = x_idxs[j]
            y = y_idxs[j]

            if (x*np.cos(theta)+y*np.sin(theta) == rho):a
                y_list = np.append(y_list,y)
                x_list = np.append(x_list,x)
        ax[2].plot(x_list, y_list, color='white', linewidth=10)
        '''
     #find the intersections
    cells = np.where(accumulator > threshold)
    print(cells[0])
    print(cells[1])
    rhos = np.unique(cells[0], return_counts=True,)
    thetas = np.zeros(rhos[0].shape)
    sum = 0

    for i in range(len(rhos[1])):
        thetas[i] = cells[1][round(rhos[1][i]/2)+sum]
        sum = sum + rhos[1][i]

    thetas = thetas.astype(int)
    return rhos[0],thetas

def plot_detected_line(img, accumulator, rhos, thetas, threshold):
    import matplotlib.pyplot as plt
    from scipy import signal

    fig, ax = plt.subplots(1, 3, figsize=(10, 10))

    ax[0].imshow(img, cmap=plt.cm.gray)
    ax[0].set_title('original image')
    ax[0].axis('image')
```

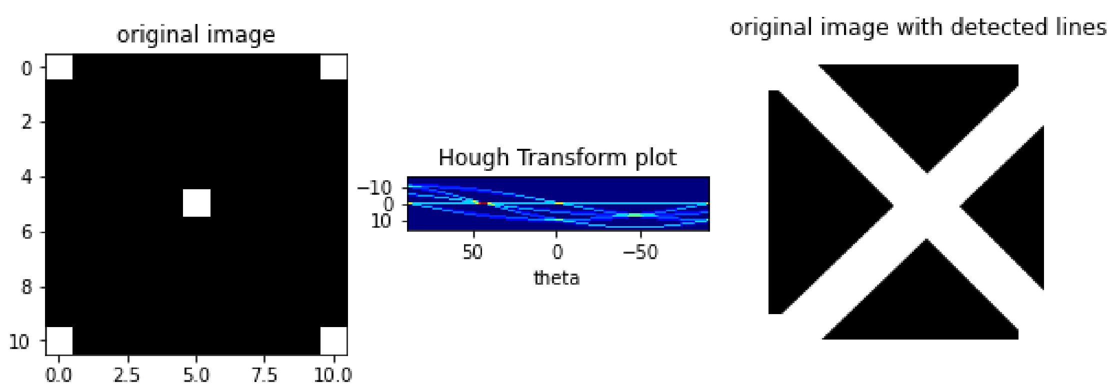
```
    ax[1].imshow(accumulator, cmap='jet', extent=[np.rad2deg(thetas[-1]), np.
↪rad2deg(thetas[0]), rhos[-1], rhos[0]])
    ax[1].set_aspect('equal', adjustable='box')
    ax[1].set_title('Hough Transform plot')
    ax[1].set_xlabel('theta')
    ax[1].set_ylabel('rou')
    ax[1].axis('image')

    ax[2].imshow(img, cmap=plt.cm.gray)
    origin_x = np.array((0, img.shape[1]))
    rho, theta = hough_line_peaks(accumulator, threshold)
    print(theta)
    print(rho)
    for i in range(len(rho)):
        dist = rhos[rho[i]]
        angle = thetas[theta[i]]
        y0, y1 = (dist - origin_x * np.cos(angle)) / np.sin(angle)
        ax[2].plot(origin_x, (y0, y1), color='white', linewidth=25)
    ax[2].set_xlim(origin_x)
    ax[2].set_ylim((img.shape[0], 0))
    ax[2].set_axis_off()
    ax[2].set_title('original image with detected lines')
    ax[2].axis('image')
```

[88]:

```
import numpy as np
from imageio import imread
import matplotlib.pyplot as plt
import math
from scipy import signal


np.set_printoptions(threshold=np.inf)

%matplotlib inline
# create your 11x11 test image and load "lane.png"
test = np.zeros([11, 11])
test[0, 0] = 1
test[0, 10] = 1
test[5, 5] = 1
test[10, 0] = 1
test[10, 10] = 1
plt.imshow(test,interpolation='nearest', aspect='auto',cmap='gray')

#Sample call and plot
accumulator, rhos, thetas = Hough_transform(test)
threshold = 2
```

```
plot_detected_line(test, accumulator, rhos, thetas, threshold)
```





[87]:
```
import numpy as np
from imageio import imread
import matplotlib.pyplot as plt
import math
from scipy import signal
%matplotlib inline

def Hough_transform(img):
    # Rho and Theta ranges
```

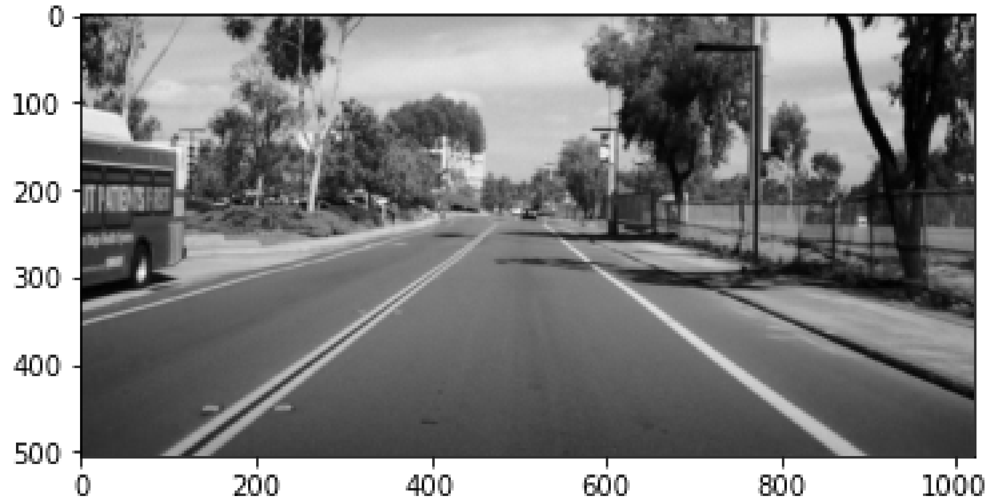
```
    thetas = np.deg2rad(np.arange(-90.0, 90.0, 1))
    width, height = img.shape
    diag_len = int(round(math.sqrt(width * width + height * height)))   #␣
↪max_dist
    rhos = np.linspace(-diag_len, diag_len, diag_len * 2)

    # Cache some resuable values
    cos_t = np.cos(thetas)
    sin_t = np.sin(thetas)
    num_thetas = len(thetas)

    # Hough accumulator array of theta vs rho
    accumulator = np.zeros((2 * diag_len, num_thetas), dtype=np.uint64)
    y_idxs, x_idxs = np.nonzero(img)  # (row, col) indexes to edges

    # Vote in the hough accumulator
    for i in range(len(x_idxs)):
        x = x_idxs[i]
        y = y_idxs[i]

        for t_idx in range(num_thetas):
            # Calculate rho. diag_len is added for a positive index
            rho = round(x * cos_t[t_idx] + y * sin_t[t_idx]) + diag_len
            accumulator[rho, t_idx] += 1

    return accumulator, rhos, thetas

def hough_line_peaks(accumulator, threshold):
     #find the intersections
    cells = np.where(accumulator > threshold)
    rhos = np.unique(cells[0], return_counts=True,)
    thetas = np.zeros(rhos[0].shape)
    sum = 0

    for i in range(len(rhos[1])):
        thetas[i] = cells[1][round(rhos[1][i]/2)+sum]
        sum = sum + rhos[1][i]

    thetas = thetas.astype(int)
    return rhos[0],thetas

def plot_detected_line(img, accumulator, rhos, thetas, threshold):
    import matplotlib.pyplot as plt
    from scipy import signal

    fig, ax = plt.subplots(1, 3, figsize=(10, 10))
```

```
    ax[0].imshow(img, cmap=plt.cm.gray)
    ax[0].set_title('original image')
    ax[0].axis('image')

    ax[1].imshow(accumulator, cmap='jet', extent=[np.rad2deg(thetas[-1]), np.
 ↪rad2deg(thetas[0]), rhos[-1], rhos[0]])
    ax[1].set_aspect('equal', adjustable='box')
    ax[1].set_title('Hough Transform plot')
    ax[1].set_xlabel('theta')
    ax[1].set_ylabel('rou')
    ax[1].axis('image')

    ax[2].imshow(img, cmap=plt.cm.gray)
    origin_x = np.array((0, img.shape[1]))
    rho, theta = hough_line_peaks(accumulator, threshold)
    for i in range(len(rho)):
        dist = rhos[rho[i]]
        angle = thetas[theta[i]]
        if (angle == 0):
            y0 = dist
            y1 = 0
        else:
            y0, y1 = (dist - origin_x * np.cos(angle)) / np.sin(angle)
        ax[2].plot(origin_x, (y0, y1), color='white', linewidth=25)
    ax[2].set_xlim(origin_x)
    ax[2].set_ylim((img.shape[0], 0))
    ax[2].set_axis_off()
    ax[2].set_title('original image with detected lines')
    ax[2].axis('image')

#define a function rgb to gray
def rgb2gray(rgb):
    rgb_weights = [0.2989, 0.5870, 0.1140]
    grayimg = np.dot(rgb[...,:3], rgb_weights)
    return grayimg


#define a function smoothing
def img_smoothing(img):
    #generate a guassian kernel
    t = 1 - np.abs(np.linspace(-1, 1, 7))
    kernel = t.reshape(7, 1) * t.reshape(1, 7)
    kernel /= kernel.sum()

    img_smoothed = signal.convolve2d(img, kernel, mode='same')
    return img_smoothed
```

```
#define a function finding gradients
def img_finding_gradients(img):
    #initialize
    img_gradients_norm = np.zeros((img.shape[0], img.shape[1]))
    img_gradients_angel = np.zeros((img.shape[0], img.shape[1]))

    #generate sobel operaters
    kx = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    ky = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])


    img_delta_x = signal.convolve2d(img, kx, mode='same')
    img_delta_y = signal.convolve2d(img, kx, mode='same')

    for i in range(0, img.shape[0]):
        for j in range(0, img.shape[1]):
            img_gradients_norm[i][j] =
 ↪(img_delta_x[i][j]**2+img_delta_y[i][j]**2)**(1/2)
            img_gradients_angel[i][j] = np.arctan(img_delta_y[i][j]/
 ↪img_delta_x[i][j])
    return img_gradients_norm, img_gradients_angel

#define a function for Non-maximum Suppression
def non_maximum_suppression(norm, phase):
    img_with_nms = np.zeros(norm.shape)

    for i in range(img_with_nms.shape[0]):
        for j in range(img_with_nms.shape[1]):
            #initialize the degrees
            if phase[i][j] < 0:
                phase[i][j] += 360

            if ((j+1) < img_with_nms.shape[1]) and ((j-1) >= 0) and ((i+1) <
 ↪img_with_nms.shape[0]) and ((i-1) >= 0):
                # 0 degrees
                if (phase[i][j] >= 337.5 or phase[i][j] < 22.5) or (phase[i][j]
 ↪>= 157.5 and phase[i][j] < 202.5):
                    if norm[i][j] >= norm[i][j + 1] and norm[i][j] >= norm[i][j
 ↪- 1]:
                        img_with_nms[i][j] = norm[i][j]
                # 45 degrees
                if (phase[i][j] >= 22.5 and phase[i][j] < 67.5) or (phase[i][j]
 ↪>= 202.5 and phase[i][j] < 247.5):
                    if det[i][j] >= det[i - 1][j + 1] and det[i][j] >= norm[i +
 ↪1][j - 1]:
                        img_with_nms[i][j] = norm[i][j]
```

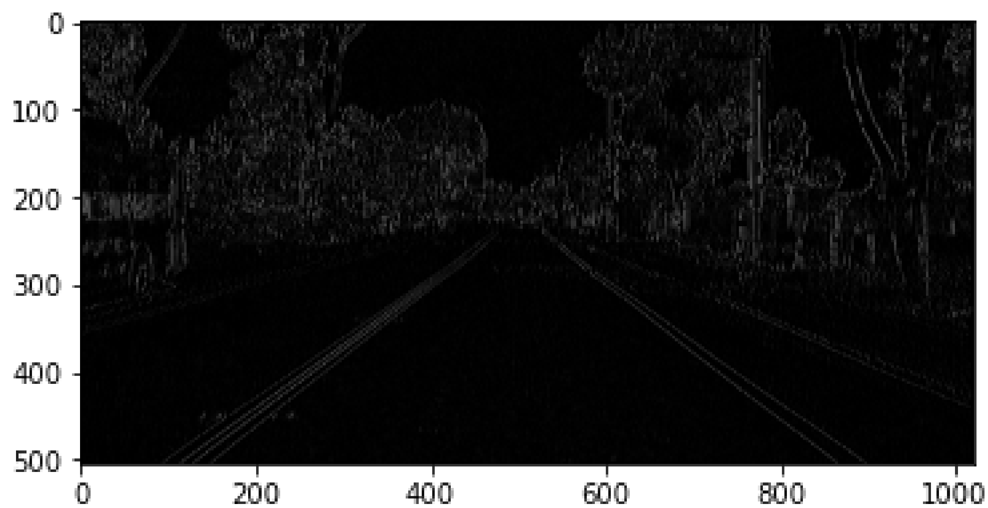
```
                # 90 degrees
                if (phase[i][j] >= 67.5 and phase[i][j] < 112.5) or␣
 ↪(phase[i][j] >= 247.5 and phase[i][j] < 292.5):
                    if det[i][j] >= det[i - 1][j] and det[i][j] >= det[i +␣
 ↪1][j]:
                        img_with_nms[i][j] = norm[i][j]
                # 135 degrees
                if (phase[i][j] >= 112.5 and phase[i][j] < 157.5) or␣
 ↪(phase[i][j] >= 292.5 and phase[i][j] < 337.5):
                    if det[i][j] >= det[i - 1][j - 1] and det[i][j] >= det[i +␣
 ↪1][j + 1]:
                        img_with_nms[i][j] = norm[i][j]

    return img_with_nms

#define a function for thresholding
def img_thresholding(img,te):
    img_with_thresholding = img

    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            if (img[i][j] <= te):
                img_with_thresholding[i][j] = 0
    return img_with_thresholding

#Canny Edge Detection Function
def canny_edge(img, te):
    img_smoothed = img_smoothing(img)
    img_strength, img_phase = img_finding_gradients(img_smoothed)
    img_after_NMS = non_maximum_suppression(img_strength, img_phase)
    detected_img = img_thresholding(img_after_NMS,te)
    return detected_img


lane = imread('lane.png')
plt.imshow(lane.astype(np.uint8), cmap="gray") # Add 'cmap="gray"' in imshow to␣
 ↪enforce grayscale
plt.show()
print(lane.shape)
#Sample call and plot
img_gray = rgb2gray(lane)
print(img_gray.shape)
te = 2
threshold = 2
img_edge = canny_edge(img_gray, te)
plt.imshow(img_edge.astype(np.uint8), cmap="gray")
plt.show()
```

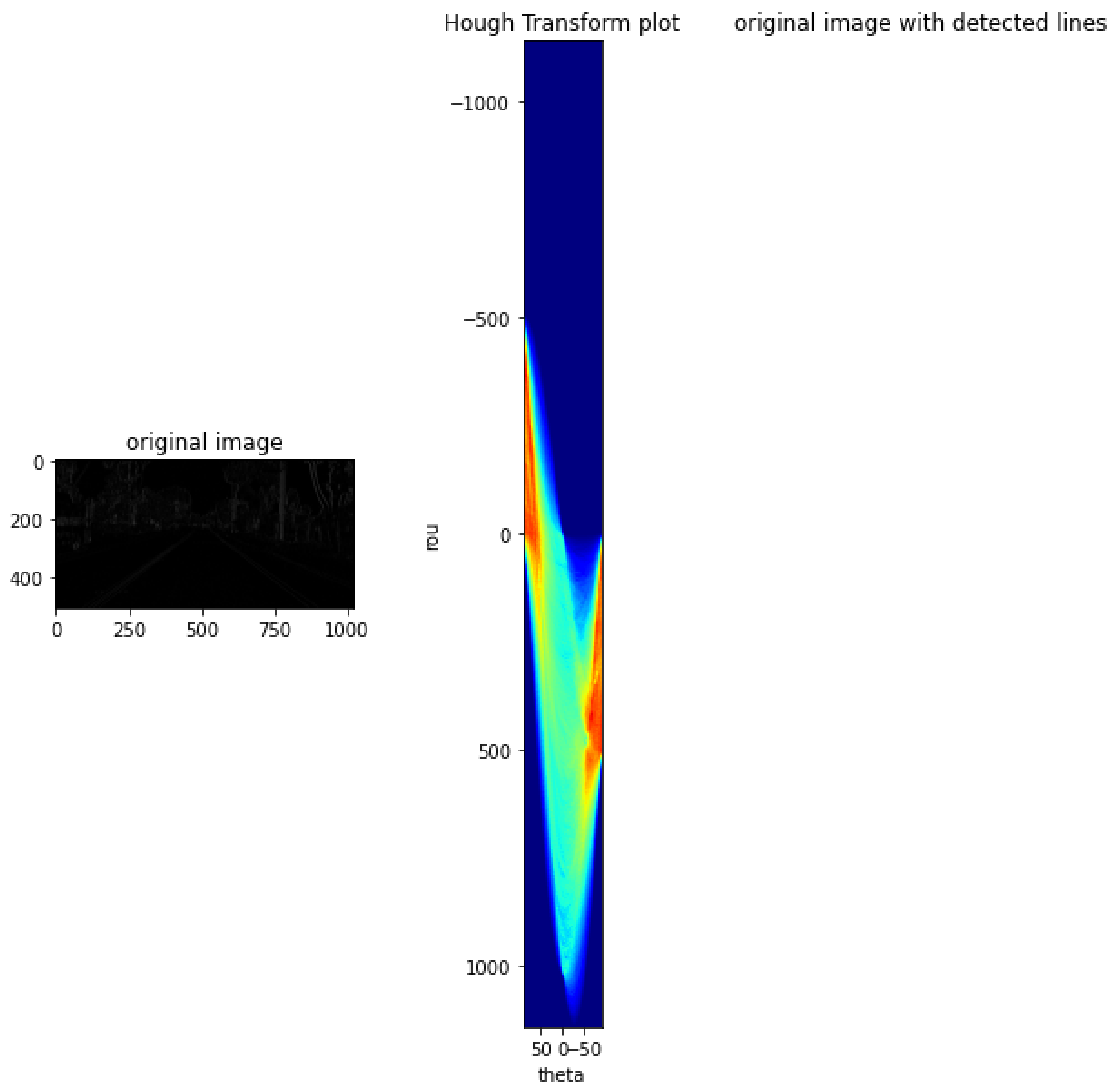
```
accumulator, rhos, thetas = Hough_transform(img_edge)
plot_detected_line(img_edge, accumulator, rhos, thetas, threshold)
```




```
(505, 1022, 4)
(505, 1022)
```

```
<ipython-input-87-ca36385a8759>:119: RuntimeWarning: invalid value encountered
in double_scalars
  img_gradients_angel[i][j] = np.arctan(img_delta_y[i][j]/img_delta_x[i][j])
```





1.3 Conclusion

Have you accomplished all parts of your assignment? What concepts did you used or learned in this assignment? What difficulties have you encountered? Explain your result for each section. Please wirte one or two short paragraph in the below Markdown window (double click to edit).

**** Your Conclusion: ****

–

** Submission Instructions**
Remember to submit you pdf version of this notebook to Gradescope. You can find the export option at File → Download as → PDF via LaTeX