BBM418 Computer Vision Lab

Harun Alperen Toktaş-21727795
harunalperentoktas@gmai1.com

Abstract

As part of the first assignment of the computer vision lesson, we are expected to determine the license plate from the given car picture with the help of Hough Transform.

1. Keywords

Hough Transformation, Computer Vision, Cars, Detection

2. Edge Detection Algorithms

2.1. Sobel Edge Detector

Since the Sobel Edge Detection kernel examines the gradient changes in the horizontal and vertical axis for a given image, it finds the horizontal and vertical edges separately. However, we need a more complicated approach than that, so the edge determinations we made on the given image should give more robust results. Let's examine how it works, with examples from the data set containing our car images.

X – Direction Kernel

-1	0	1
-2	0	2
-1	0	1

Y – Direction Kernel

-1	-2	-1
0	0	0
1	2	1

Figure 1. Sobel Filter Weights

When we examine the Sobel filter, we see that, unlike an ordinary vertical or horizontal edge detection filter, more weight is given to the pixels in the middle. Gaussian distribution is applied. In this way, it is provided to give better results and smoother transitions are achieved. Now let's see what result we got with the help of a simple code.

The code reduces the noise with Gaussian blur. Then it turns the image into gray scale and applies the Sobel filter.

```
import cv2 as cv
from google.colab.patches import cv2_imshow as cv_imshow
from matplotlib import pyplot as plt

img_path = '/content/drive/MyDrive/Assignments/images/Cars0.png'

example_image = cv.imread(img_path,1) #1 means read as rgb
example_image = cv.GaussianBlur(example_image, (3, 3), 0) # I applied gaussian blur for reduce to noise
example_image_gray = cv.cvtColor(example_image, cv.COLOR_BGR2GRAY) #Converted to grayscale image


example_image_vertical = cv.Sobel(example_image_gray, cv.CV_16S, 1, 0, ksize=3, scale=1, delta=0, borderType=cv.BORDER_DEFAULT) # (1,0) GRADYAN X-VERTICAL
example_image_horizontal = cv.Sobel(example_image_gray, cv.CV_16S, 0, 1, ksize=3, scale=1, delta=0, borderType=cv.BORDER_DEFAULT)#(0,1) GRADYAN Y-HORIZONTAL

plt.subplot(221),plt.imshow(example_image,cmap = 'BrBG')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])

plt.subplot(222),plt.imshow(example_image_gray,cmap = 'gray')
plt.title('Original Image in Grayscale'), plt.xticks([]), plt.yticks([])

plt.subplot(223),plt.imshow(example_image_vertical, 'gray')
plt.title('Vertical Edge - Gradyan X'), plt.xticks([]), plt.yticks([])

plt.subplot(224),plt.imshow(example_image_horizontal,cmap = 'gray')
plt.title('Horizontal Edge - Gradyan Y'), plt.xticks([]), plt.yticks([])

plt.show()
```

Figure 2. Sobel Edge Detection Code

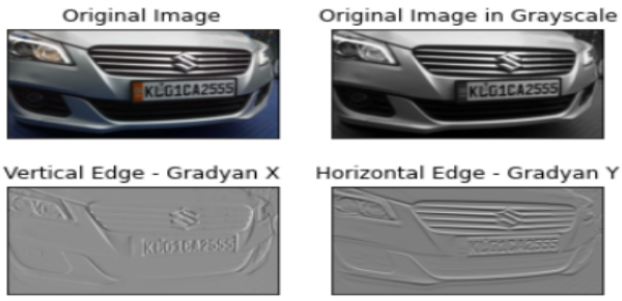


Figure 3. Sobel Edge Detection Results

As you can see in Figure 3, the Sobel filter can detect the vertical and horizontal edges for the given picture by finding the changes in the x-axis and the changes in the y-axis . Let's try the other algorithms to find out which edge detection algorithm to use.

2.2. Laplacian Edge Detector

Laplacian Edge Detector : Laplacian edge detection algorithm also obtains results through gradient change. I haven't mentioned it before, when the Sobel filter looks at change in the x and y axis, it actually takes a first order derivative. During the lesson, our teacher explained it in detail. There are examples in the lecture slides. The Laplacian filter detects both vertical and horizontal edges at once with the help of a filter by taking its second order derivation with a single filter.

Laplacian Filter Weights

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Basic Laplacian Filter **Diagonal Included Laplacian Filter**

Figure 4. Laplacian Filters

These are two examples of Laplacian filters. According to the information I have obtained from my research on these filters, these filters applied with second order derivatives are more sensitive to noise in the images. Therefore, it is beneficial to apply the Gaussian filter definitely before filtering. Let's try this filtering method on our previous image and observe the result.

```
import cv2 as cv
from google.colab.patches import cv2_imshow as cv_imshow
from matplotlib import pyplot as plt

img_path = '/content/drive/MyDrive/Assignments/Images/Cars0.png'

example_image = cv.imread(img_path,1) #1 means read as rgb
example_image = cv.GaussianBlur(example_image, (3, 3), 0) # I applied gaussian blur for reduce to noise
example_image_gray = cv.cvtColor(example_image, cv.COLOR_BGR2GRAY) #Converted to grayscale image

#sobel filter
example_image_vertical = cv.Sobel(example_image_gray, cv.CV_16S, 1, 0, ksize=3, scale=1, delta=0, borderType=cv.BORDER_DEFAULT)  #(1,0) GRADYAN X-VERTICAL
example_image_horizontal = cv.Sobel(example_image_gray, cv.CV_16S, 0, 1, ksize=3, scale=1, delta=0, borderType=cv.BORDER_DEFAULT) #(0,1) GRADYAN Y-HORIZONTAL

#laplacian filter
example_image_laplacian = cv.Laplacian(example_image_gray,cv.CV_64F)

#plot part
plt.subplot(221),plt.imshow(example_image,cmap = 'BrBG')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])

plt.subplot(222),plt.imshow(example_image_laplacian,cmap = 'gray')
plt.title('Original Image in Grayscale'), plt.xticks([]), plt.yticks([])

plt.subplot(223),plt.imshow(example_image_vertical, 'gray')
plt.title('Vertical Edge - Gradyan X'), plt.xticks([]), plt.yticks([])

plt.subplot(224),plt.imshow(example_image_horizontal,cmap = 'gray')
plt.title('Horizontal Edge - Gradyan Y'), plt.xticks([]), plt.yticks([])

plt.show()
```

Figure 5. Laplacian Filter Code

By adding the codes I showed with the red rectangle to the code I used before, I have obtained the Laplacian filter result.

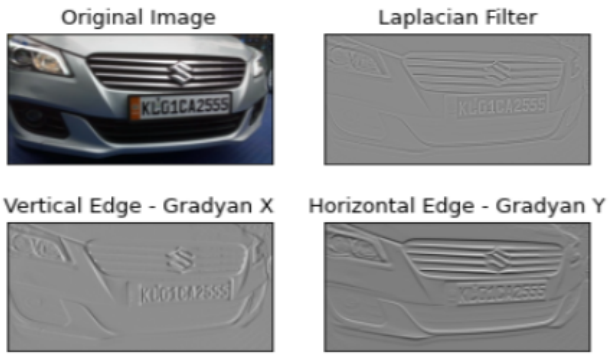


Figure 6. Laplacian Filter Result with Sobel Filter Results

The Laplacian filter worked fine. It seems that the result of the Sobel filter with 2 filters is obtained with a single filter. In my opinion, the Sobel filter has obtained sharper results because it detects edges separately. Edges in the Laplacian filter seem to have smoother gradations. Finally, let's look at the Canny Edge Detection algorithm and make our decision.

2.3. Canny Edge Detector

Canny edge detector is actually trying to improve its results by taking advantage of the Sobel filter. The algorithm consists of 4 stages. The first step is the Gaussian Blur process we always do, which we use to reduce the noise in the images. With this process, after the noise is reduced, gradient changes are calculated along both the x and y axis with the filtering method we apply in the Sobel filter. Then the intensity gradient value is calculated for each pixel. After this stage, we move on to the Non-maximum Suppression stage(third stage). At this stage, we check whether it creates a local maxima in the gradient direction for each pixel. If not, we suppress it by pulling it to zero. As a result of this method, we get thin edges before proceeding to the final stage. What we want is a single thick edge. That's why we use the Hysteresis Thresholding (last stage) method. What we're doing here is to see if the edge information we have actually creates an edge. We set two values for this. One is the minimum and the other is the maximum. If the intensity gradient of the edge we have is more than this maximum value, we consider this edge as the exact edge. If the intensity gradient of the edge we have is lower than the minimum value, we directly judge that it is not an edge. If the intensity gradient of the edge we have is between minimum and maximum values, then we examine whether this edge has a connection with an edge that we consider as a definite edge. If there is a link, we hold this edge, if not, we decide that this edge does not actually represent an edge. In this way, this algorithm aims to obtain better edge maps, now let's examine this algorithm by trying it with other filters

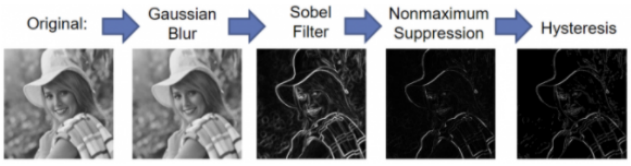


Figure 7. Canny Edge Detection Steps

I was expecting a better result as the Canny algorithm uses the Sobel filter and develops new methods on this process. As can be seen from the figure 8, it looks very successful in finding a single and solid edge. For example, in the Sobel filter, the transitions were softer, I think that if we tried to extract the edge directly from that filter, there would not be a sharp and single edge map as the Canny algorithm did. I prefer the Canny algorithm because it produces solutions for thin edges and also contains the Sobel filter.

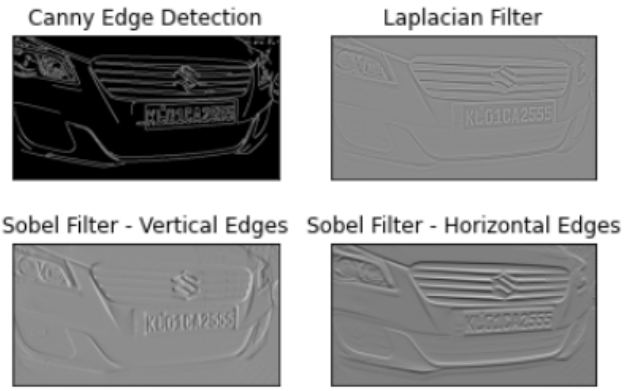


Figure 8. Canny Edge Detection Steps

2.4. Canny Edge Detection Parameter Selection

```
def canny_edge_parameter_selection(read_img_path,  write_image_path):
    """
    read_img_path: images/*.png
    write_img_path: current directory path 'os.getcwd()'
    """
    if 'Parameter_Images' not in os.listdir(write_image_path):
        os.mkdir(os.path.join(write_image_path,'Parameter_Images'))

    write_image_path = os.path.join(write_image_path,'Parameter_Images')

    example_image = cv.imread(read_img_path,1) #1 means read as rgb
    example_image = cv.GaussianBlur(example_image, (3, 3), 0) # I applied gaussian blur for reduce to noise
    example_image_gray = cv.cvtColor(example_image, cv.COLOR_BGR2GRAY) #Converted to grayscale image

    #canny edge detection
    #default parameters for min value = 100 max value = 200
    #let's try to find better parameters
    parameter_list = [50,75,100,125,150,175,200]

    for i in (parameter_list):
        for j in (parameter_list):
            if (i != j and i < j):
                example_image_canny = cv.Canny(example_image_gray,i,j)
                cv.imwrite(os.path.join(write_image_path,'example_image_min{}_max{}.png'.format(i,j)),example_image_canny)
```

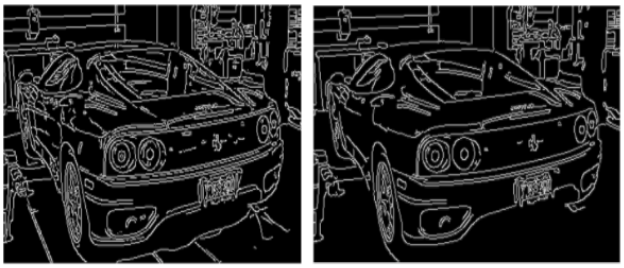
Figure 9. Canny Edge Detection Parameter Selection Code

I implemented a simple code for selecting parameters. The code I have implemented creates edge map images obtained by canny edge method at certain interval values for the desired image. Since there are too many images here, instead of showing them, I created a folder in the directory where you run the program and brought the pictures there. Instead of showing all the pictures, I'll show you the pictures that help me make up my mind.

The code creates images between a minimum of 50 and a maximum of 200 values as the intensity gradient value for a pixel. I kept the minimum limit lower here(default min 100, max 200 in opencv doc) and I thought that the information we will add as the edge may give better results.

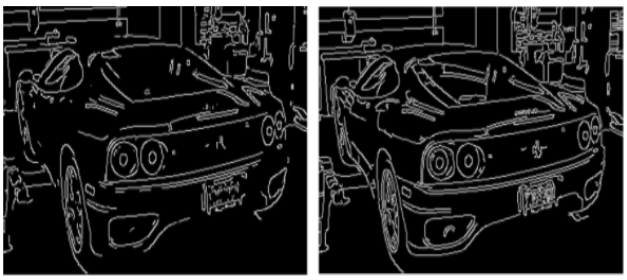
As you can see in figure 10, reducing the maximum value increases the number of edges in the image, so unnecessary information increases. I think this is not something we want. Let's try others. As you can see in figure 11, Increasing the minimum threshold value resulted in the loss of information that could be an edge. A balance is needed. I don't find it very logical to handle the upper limit. I think increasing the lower limit a little will reduce unnecessary information and provide the best protection of the necessary knowledge.

I choose minimum value as 125 and maximum value as



Min 50 - Max 75 Min 50 - Max 200

Figure 10. Canny Edge Detection Results



Min 175 - Max 200 Min 50 - Max 200

Figure 11. Canny Edge Detection Results

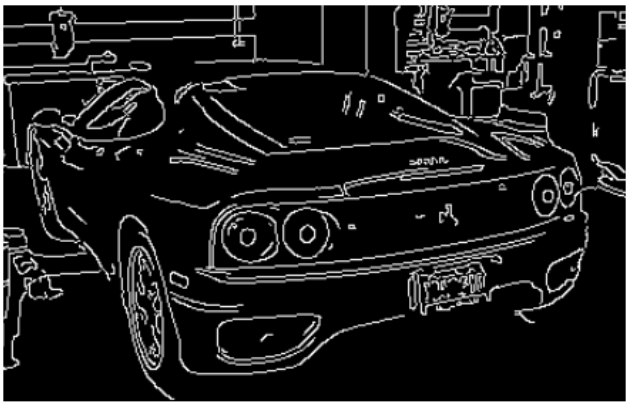


Figure 12. Canny Edge Detection Chosen Interval

200 for pixel intensity gradients. In this way, I tried not to lose the license plate information while trying to reduce the noise.

2.5. Edge Maps Results on 10 Different Images

Now that the method and parameters I will use are certain, I made the Canny Edge Detection method into a function and got 10 outputs with a simple code. With the help of matplotlib, I have obtained the desired images as a table from 2 rows and 5 columns.

```
def canny_edge_detection(img_path):
    example_image = cv.imread(img_path,1) #1 means read as rgb
    example_image = cv.GaussianBlur(example_image, (3, 3), 0) # I applied gaussian blur for reduce to noise
    example_image_gray = cv.cvtColor(example_image, cv.COLOR_BGR2GRAY) #Converted to grayscale image

    #canny edge detection
    example_image_canny = cv.Canny(example_image_gray,125,200) #choosen parameters

    return example_image_canny

def canny_edge_detection_on_10_dif_images(images_folder):
    fig = plt.figure(figsize=(20, 7)) #Create template for all images

    rows = 2 #I showed them 2 row 5 column,may be I change it depends report page
    columns = 5

    image_count = 0
    for img_path in os.listdir(images_folder):
        if (image_count < 10): #total 10 images
            img = canny_edge_detection(os.path.join(images_folder,img_path))
            fig.add_subplot(rows, columns, image_count + 1)


            plt.imshow(img,cmap = 'gray') #represent in grayscale
            plt.axis('off')
            plt.title("Edge Image - {}".format(image_count + 1))

        image_count += 1
```

Figure 13. Code snippet: Canny Edge Detection for 10 Images

Figure 14. Canny Edge Detection Result for 10 Images

3. All Steps of Hough Transform with Visual Examples and Code Snippets

3.1. What is Hough Transform

First of all, let's talk about why we need to use Hough transform. Normally, it is possible to find an edge in a picture using the edge detection algorithms I mentioned in section 2. However, these edge maps we found contain a lot of noise. That's why we need an algorithm that reduces this noise and allows us to get robust straight lines. This algorithm is Hough Transform. The Hough Transform is based on converting lines in cartesian coordinates to polar coordinates.

3.2. Cartesian Coordinates to Polar Coordinates

I can explain the logic here as follows. Points form a line in Cartesian coordinates. Since these points are on the same line, their slopes are the same. So these points point to the same line. If I move these points to polar coordinates, Hough space using the necessary equations, the points that make up the same line will intersect at a common point. This common point will correspond to the values rho and theta that make up that line. rho value corresponds to the perpendicular distance of the detected line to its origin, and theta value corresponds to the angle between the line and the origin.

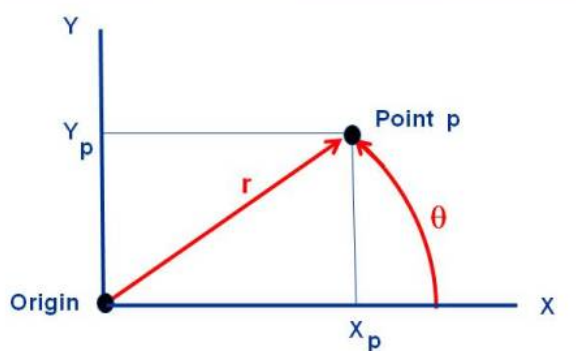


Figure 15. Cartesian and Polar representation of Point p

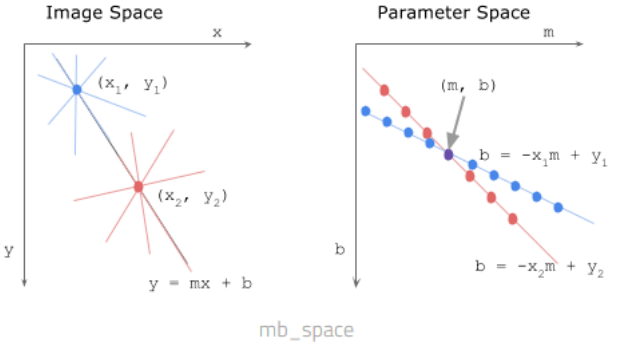


Figure 16. Points on the same line intersect at a common point.

Here, too, we are voting using the edge map we have, using this transformation. If a point in Hough space has received more votes than others, the points that make up that line point to straight lines with less noise than others.

3.3. Step by Step Visual and Code Explanations

```
def Hough_Transform(image_path,min_theta,max_theta,threshold):
    '''
    In my case min theta is -90, max theta is +90
    It can be try as 0 to 180,it is up to you
    If you change 0 to 180 you should uptade accumulator[ro][theta+90] with proper indexes

    '''
    original_image = cv.imread(image_path,1)
    edge_image = canny_edge_detection(image_path) #it return canny edge maps

    thetas_range = [x for x in range(min_theta,max_theta,1)] #-90 to +90

    max_ro = int (math.hypot(edge_image.shape[0], edge_image.shape[1])) #  find maximum range of ro value, it is diagonal lenght from origin to polar edge
    ro_range = [x for x in range(-1 * max_ro, max_ro +1 , 1)] #ro range is [-max_ro to +max_ro]

    accumulator = [[0 for i in range(len(thetas_range)+1)] for j in range((len(ro_range)) + 1)] #accumulator (row = ro range,column = theta range)


    for x in range(edge_image.shape[0]):
        for y in range(edge_image.shape[1]):
            if (edge_image[x,y] == 255): #255 means edge pixel
                for theta in range(len(thetas_range)): #np.cos function need radyan so i convert them to radyan

                    theta -= 90
                    ro = int((y * np.cos((np.pi/180) * theta) ) + (x * np.sin((np.pi/180) * theta))) + max_ro

                    #actually y means x, x means y , image coordinat and cartesian coordinat are different
                    #we need use radyan, for calculations 1 radyan * theta,e.g. 1 radyan * 30 is equal cos(30) or sin(30)
                    #tested and find negative value for d so I added diagonal length(max_ro) to formula

                    try:
                        accumulator[ro][theta+90] += 1
                    except IndexError:
                        print(' Boundary issue ro:{} theta:{} '.format(str(ro),str(theta)))
```

Figure 17. Hough Transformation Code.

Let me start by explaining the code from beginning to end. First of all, I made the Hough transform process into a function. It takes four parameters as inputs. The path of the desired image ,the minimum theta, the maximum theta and the threshold value to determine the local maxima.Then I extracted the edge map of the desired picture using the Canny Edge detection method.

After this stage, it is necessary to determine the size for the accumulator matrix (Hough Space) to be created. The dimensions of the matrix we will use to vote will be 2 * max rho as the number of rows, and the number of columns will be theta number in the range where we will get theta. I took it from -90 degrees to +90 degrees as theta range. Here, of course, a range from 0 to 180 can be preferred. Optionally, it may be necessary to pay attention to both matrix indexing and rho formula. We do not specify the range of the rho's capability. The maximum value rho can take for an image is the end-to-end diagonal length of that image. For example, if the image is 3x4, the maximum value of rho it can take is 5 in the + direction or 5 in the - direction. So I took the range of the value rho as the range [-max rho, + max rho].

Then I created a matrix of 0's in the dimensions I specified. Then I created a matrix of 0's in the dimensions I specified. I do this calculation as follows. The x and y coordinates in an image and the x and y coordinates I use in the formula do not correspond exactly. I use the y coordinate as x and the x coordinate as y to calculate the value of rho, in this way the calculation will be correct. I calculate the value of rho with the formula $ro = xcos(theta) + ysin(theta)$ and increase the number of votes by 1 for each acc[rho] [theta] on the accumulator matrix. It may be useful to mention 2 things here lastly. When you examine the code, you can see that I have removed from -90 and added +90, this is all about how I use the matrix. For example, column 0 represents -90 degrees.In addition, I added max rho to the value of rho, I cannot index the - valued result in the matrix.

```
#local maxima ,most most most most important part
#Firstly i did not use np matrix but I need now so, I converted to np aray
accumulator = np.array(accumulator)

kernel = 13
lines = list()

for row_index in range(accumulator.shape[0] -(kernel-1)):
    for column_index in range(accumulator.shape[1] -(kernel-1)):
        kernel_slice = accumulator[row_index:row_index+kernel,column_index:column_index+kernel]


        local_maxima = np.where(kernel_slice == np.amax(kernel_slice))
        #local_maxima = (int(sum(local_maximas[0])/len(local_maximas[0])), int(sum(local_maximas[1])/len(local_maximas[1])))
        if(local_maxima[1][0]+column_index == 90):
            if(accumulator[local_maxima[0][0]+row_index][local_maxima[1][0]+column_index] >= 40):
                #print(accumulator[local_maxima[0][0]+row_index][local_maxima[1][0]+column_index])
                line = (local_maxima[0][0]+row_index,local_maxima[1][0]+column_index)
                lines.append(line)

        if (accumulator[local_maxima[0][0]+row_index][local_maxima[1][0]+column_index]>= threshold):

            line = (local_maxima[0][0]+row_index,local_maxima[1][0]+column_index)
            lines.append(line)


return accumulator, list(set(lines)), thetas_range, ro_range,max_ro
```

Figure 18. Local Maxima

I think the most important part of this assignment is to find the values that peak locally. When I did various experiments here, the most difficult thing was to find vertical lines. Because the vertical lines in the picture are short, the number of points forming those lines is less, this directly prevents them from being maximum in the accumulator matrix we have. So I did the job of finding the local maxima in 2 parts. I determined the horizontal and vertical lines separately.

In fact, it would be better if I could talk about the method I use to determine the lines here. First I found the maximum index in each row in the accumulator matrix. In my first experiments using this method, I generally obtained more horizontal and cross lines. I needed vertical lines to detect plates, so I decided to change the method. Inspired by the convolution process used in Convolutional Neural Networks, I got the maximum values at the point where that filter corresponds by moving the filter over the accumulator, so that I can get better results. Using a small filter (3x3) does not give very good results, it detects too many unnecessary lines. I changed the filter size and examined the results. The higher the number of filters, the better the results. Of course, the point we need to be careful about here is not to get a good result for a single image. Finding a generalizing parameter. After several experiments I chose 13x13 as the filter size. A second parameter is of course the threshold value. In general, 125 for the threshold value works well for horizontal lines on average. Although the method I developed works well for horizontal lines, the vertical problem has not been solved. For vertical lines, I determined a value equal to one fourth of the threshold value and discussed their determination separately. First of all, let's look at the printouts on the noise-free pictures I created to show that the code I wrote works correctly.

```
import cv2 as cv
from matplotlib import pyplot as plt
original_image_path ='/content/drive/MyDrive/Assignments/Assignment1/code/hough_experiment.png'
org_img = cv.imread(original_image_path,1)
#detected_image = !python main.py original_image_path 100
detected_image = cv.imread('/content/drive/MyDrive/Assignments/Assignment1/code/Detected.png',1)


#plot part
fig = plt.figure(figsize=(25, 7))
fig.add_subplot(121),plt.imshow(org_img)
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
fig.add_subplot(122),plt.imshow(detected_image)
plt.title('Hough Image'), plt.xticks([]), plt.yticks([])
plt.show()
```

Figure 19. Hough Transform Code on Custom Image

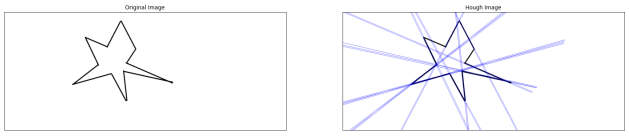


Figure 20. Hough Transform on Custom Image

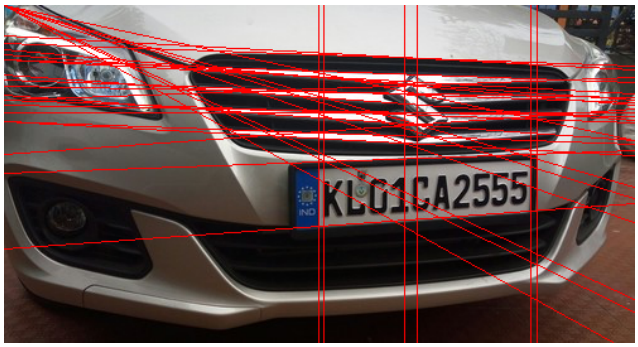


Figure 21. Hough Transform on Car Image via our dataset

If you look closely at figure 21, you can see that it has found all the lines. This is proof that the code is working correctly. Now let's take a look at examples from our car data set. In Figure 22, we see the Hough transform being tested on an example from the car data set. There are lines on the picture where I can find the plate. Now let's measure how I found the plate from these lines and measure the performance metric, then let's look at the Hough transform example from beginning to end and how the algorithm responds to other car images.

3.4. Plate Detection

```python
def find_x_y_coordinat(line, ro_range, thetas_range):
    x = line[0]
    y = line[1]
    #y = thetas, 90 vertical,0-180 horizontal
    #x,y = 567,19
    ro = ro_range[x]
    theta = thetas_range[y]

    #print(x,y,ro,theta)
    a = np.cos(theta * np.pi/180)
    b = np.sin(theta * np.pi/180)

    x0 = a * ro
    y0 = b * ro
    x1 = int(round(x0 + 1000 * (-b)))
    y1 = int(round(y0 + 1000 * (a)))
    x2 = int(round(x0 - 1000 * (-b)))
    y2 = int(round(y0 - 1000 * (a)))
    return x1,y1,x2,y2
```

Figure 22. Default polar to Cartesian Transformation Code

```python
def draw_and_detect(original_image,lines,ro_range,thetas_range):

    horizontal_lines = list()
    vertical_lines = list()

    for line in lines:

        x,y = line[0],line[1]
        x1,y1,x2,y2 = find_x_y_coordinat(line,ro_range,thetas_range)

        p1,p2 = Point(x1,y1),Point(x2,y2)
        line = Line(p1, p2)
        if (y == 90):
            vertical_lines.append(line)
        elif (y<=15 or y>=165):
            horizontal_lines.append(line)

    print('Number of Vertical Lines: {}'.format(len(vertical_lines)))
    print('Number of Horizontal Lines: {}'.format(len(horizontal_lines)))
```

Figure 23. Draw And Detect Code Part1

In this section, I first shared the codes and the final picture so that we can talk about them. Here we found the line parts for the given picture using the Hough transform, but that alone is not enough. We need to detect the plate using these lines correctly. We can call this stage post processing. At this stage, I first separated the vertical and horizontal lines. Because it is very difficult to find vertical lines in a picture to recognize license plates. While I was finding horizontal lines, I took those between a certain angle. Because I think, in general, a plate doesn't stand at a 60-degree angle. Then I calculated the intersection points for each vertical and horizontal line I found over combinations of each other. Finding all combinations increases the running time of the algorithm when the number of lines is high. The code, which normally runs for 30 seconds, can extend up to 6-7 minutes when the number of lines is too high. At this stage, it is also necessary to set more restrictive parameters so that we can find what we want. My first limiting parameter was the area of the rectangle I found, using the ground truth information of a few pictures, I roughly determined a plate area. average 9500 pixels. The second limiting parameter was to set the aspect ratio to make the shape I found rectangular. Here I tried to have the long edge to short edge ratio between 3.5 and 6. Although these parameters are restrictive, they are not sufficient. Therefore, of all the rectangles that provide these parameters, I chose the one whose midpoint is closer to the origin. The aim here is usually the plates are at the lower part of the picture. I thought it would be useful. As you can see in Figure 25, I have roughly found the plate.

```python
rectangles = list()
detected_part = [-1000000000,None]
for line1, line2 in itertools.combinations(vertical_lines, 2):
    for line3,line4 in itertools.combinations(horizontal_lines, 2):

        inters_1 = line1.intersection(line3)
        inters_2 = line1.intersection(line4)

        left_top = inters_2 if inters_2[0][1]>inters_1[0][1] else inters_1

        inters_3 = line2.intersection(line3)
        inters_4 = line2.intersection(line4)

        right_bottom = inters_3 if inters_4[0][1]>inters_3[0][1] else inters_4

        tall_edge = abs(left_top[0][0]-right_bottom[0][0])
        short_edge = abs(left_top[0][1]-right_bottom[0][1])

        area =int(tall_edge * short_edge)
        is_rect = (tall_edge) / (short_edge)

        if(area<=13000 and area >=6500):#9500 average
            if(is_rect >= 3.5 and is_rect <=6):

                diag_length = math.hypot(((tall_edge/2)+left_top[0][0]),(short_edge/2)+right_bottom[0][1])


                if(diag_length>detected_part[0]):
                    detected_part[0] = diag_length
                    detected_part[1] = (left_top,right_bottom)
                    #(left_top[0][0],left_top[0][1]),(right_bottom[0][0],right_bottom[0][1])
print(detected_part)
if(detected_part[1] == None):
    "It can find plate"
    return -1
result_image = cv.rectangle(original_image, (detected_part[1][0][0][0],detected_part[1][0][0][1]), (detected_part[1][1][0][0],detected_part[1][1][0][1]), (255,0,0), 2)
cv.imwrite('detected_result.png',result_image)
        #constraint
```

Figure 24. Draw And Detect Code Part2

Figure 25. Detection Result

3.5. 10 samples of my detected license plates and their IOU scores:Figure26-35

First of all it requires to explain something here. The IOU results of the results I generated are probably not numerically correct. I believe I have spelled the IOU function correctly. Despite my efforts for hours, I could not get a healthier result. This is probably due to the Line library I use. When I give the same points in the IOU function, I can get 100 percent accuracy. But for some reason the numerical result of the IOU does not make much sense. However, the license plate detection areas I made are correct. And I can easily comment on them through the pictures. he rectangles drawn in green are ground truth, the ones drawn in blue are the ones I predicted. I will make all my comments in the next section.

Figure 26. Sample1 - IOU Score : 0.96

Figure 27. Sample2 - IOU Score : 0.50

3.6. Average IOU score for all images and comment for results

There is a problem with the algorithm's running time. Since I look at all combinations through all the lines I have detected, the more lines I detect on the picture, the longer the algorithm runs. This makes it difficult to obtain more detailed results. Now my algorithm is running for 60 minutes. And it was able to finish 18 images in total. In addition, there are cases such as: When I reduce the threshold value I use for the accumulator, the number of lines it finds increases, but the working time increases. This is why I cannot reduce the threshold value very much. This causes the algorithm to not work well when there are plates

Figure 28. Sample3 - IOU Score : 0.20

Figure 29. Sample4 - IOU Score : 0.91

Figure 30. Sample5 - IOU Score : 0.22

Figure 31. Sample6 - IOU Score : 0.28

Figure 32. Sample7 - IOU Score : 0.97

Figure 33. Sample8 - IOU Score :0.26

Figure 34. Sample9 - IOU Score : 0.69

Figure 35. Sample10 - IOU Score : 1.31

to find. However, considering the overall average score, I can say that it is quite low. It is obvious from the results produced by the algorithm. There are many reasons why the results are low. The first of these is that besides the information required when obtaining the edge map, we also obtain unnecessary noise. This causes us to find too many unnecessary lines as it increases the peaks in the accumulator. When you increase the threshold value of the accumulator to eliminate these useless lines, you also eliminate the useful lines. At the same time, since we are looking for a plate in the image given, we need at least 2 lines close to the vertical. Since the vertical lines are short, the number of votes in the accumulator is low. That's why they stay below the threshold value and it becomes very difficult to find the vertical line.So somehow you have to do something to find vertical lines. For example, I tried to solve the vertical lines by considering them separately. Another reason for the low overall result may be related to the shape, color and cleanliness of the plate. For example, when a white car has a white plate, the algorithm does not produce good results while finding an edge from the very beginning. Because, as I explained earlier, the edge detection process is based on the difference between the pixel value. Switching from white to white makes it very difficult for the algorithm to see it as the edge. Also, there is no general template for the plates. Some take a very small place in the visual, some take a very large place. Or there is more than one car and license plate in the image. Since these are all separate parameters, it is very difficult to generate a generalizing result with Hough transform. I am not saying it is impossible, but it is neces-

sary to do pre-post processing very well. For example, I can find the license plate better in vehicles with medium-sized plates, which are mostly drawn in front of the vehicle. (E.g. Figure25-Figure36) Also, I would like to state that if the threshold value is taken low and the time is waited, it can better detect many pictures that it cannot detect well.

3.7. 5 images my method fails to detect license plates

Here again, there are 5 examples that I could not detect at all. Actually, I think I should have made some determinations for figure 37. Here, I think that the parameters (field, aspect ratio) that I mentioned earlier in my algorithm got stuck because of the low number of lines and could not produce a good result. As we can see in Figure36 and Figure 40, there are more than one plate. And their sizes are quite small. It is very normal that he was not detected. The reason why it could not be found in Figure 39 may be because I searched on 2 vertical and 2 horizontal lines. As you can see, the plate is not straight, it has a certain slope. It is not very easy to find the plate at this angle since I take the 90 degree ones directly on the vertical lines. In Figure 38, the situation is most likely again due to the threshold value. I think it might find something, but still it wouldn't work well because the plate is small.

Figure 36. Sample1 - IOU Score : No Detection

4. Resources

An Implementation of Sobel Edge Detection - Rhea. (2021). Retrieved 21 March 2021, from

Spatial Filters - Laplacian/Laplacian of Gaussian. (2021). Retrieved 21 March 2021, from

OpenCV 3 Image Edge Detection : Sobel and Laplacian - 2020. (2021). Retrieved 21 March 2021, from

OpenCV: Canny Edge Detection. (2021). Retrieved 21 March 2021, from

Canny Edge Detector Using LegUp – LegUp Computing Blog. (2017). Retrieved 21 March 2021, from

Figure 37. Sample2 - IOU Score :No Detection

Figure 38. Sample3 - IOU Score : No Detection

Figure 39. Sample4 - IOU Score : No Detection

Figure 40. Sample5 - IOU Score : No Detection