BBM418 - Computer Vision Lab.

Burak Özüesen 21827761

March 2021

1 Introduction

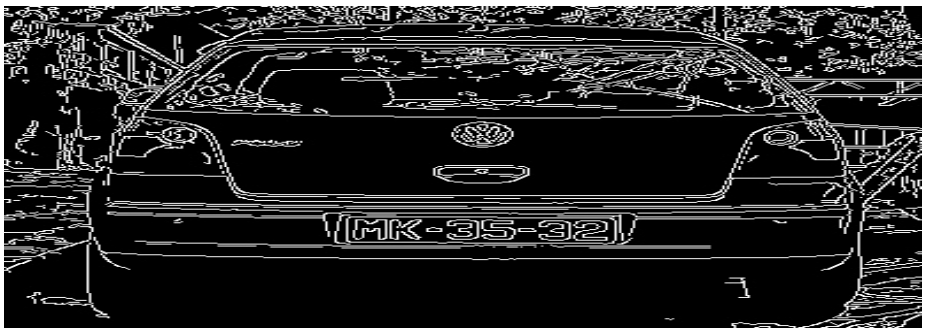
In this assignment, we get familiar with edge detection methods and Hough Transform. First of all, we extracted and obtain the edge points from an image simple by using an edge detection method. Then we used these edge points to detect license plates by giving the map including edge points as an input for the Hough Transform. As a dataset, we used use the dataset provided from you.

2 Edge Detection

In Edge Detection part, I used Canny Edge detection method to apply my Hough Transform method. After loading image I apply gray filter and take white pixels for my Hough Transform method.

```
img = cv2.imread("images/" + sys.argv[1] + ".png")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray, 75, 150)
```

Here sample detected edge image.



3 Hough Transform

First of all we need to initialize width, shape, diagonal length of image, theta range, rho range and create 2D zero matrix which named accumulator.

```
height, width = edges.shape
diagonal_length = int(np.ceil(np.sqrt(width * width + height * height)))
thetas = np.deg2rad(np.arange(-90.0, 90.0))
rhos = np.arange(-diagonal_length, diagonal_length)
accumulator = np.zeros((2 * diagonal_length, len(thetas)))
```

After the initializing step, we need to traverse all edge points and for each edge point we calculate all possible rho values according to every possible theta values. For every calculated rho value we go to our accumulator and increment corresponding index. Also I create two new arrays for horizontal and vertical lines.

```
for i in range(width):
    for j in range(height):
        if edges[j][i] == 255:
            for k in range(len(thetas)):
                rho = round(i * np.cos(thetas[k]) + j * np.sin(thetas[k]))
                accumulator[rho + diagonal_length][k] += 1

horizontal_lines = []
vertical_lines = []
```

At step 3, when our traverse is done I took at least 30 votes for evaluates as potential line. Also I took only vertical and horizantal lines when I filter angles. Because most of the plates are rectangle not quad. At last but not least, I want to avoid from overlapping potantial lines. So I compared last two rho values and if they are too close to each other I do not take last coming potential line to horizantal/vertical line array.

```
for i in range(len(accumulator)):
    for k in range(len(accumulator[0])):
        if 50 > accumulator[i][k] > 30 and 92 > k > 88:
            rho = rhos[i]
            theta = thetas[k]
            if len(vertical_lines) == 0 or rho - 50 > vertical_lines[-1][0]:
                vertical_lines.append([rho, theta])

        if 50 > accumulator[i][k] > 30 and (2 > k or k > 178):
            rho = rhos[i]
            theta = thetas[k]
            if len(horizontal_lines) == 0 or rho - 50 > horizontal_lines[-1][0]:
                horizontal_lines.append([rho, theta])
```

At step 4, I reduct my potential lines to just 4 lines. I took 2 vertical lines at

the middle from image and took 2 horizontal lines at lower quertile from my image. My lines array holds last 4 ordered pair which they are my rectangle lines.

```
rho1 = vertical_lines[len(vertical_lines) // 2 - 1][0]
rho2 = vertical_lines[len(vertical_lines) // 2 + 1][0]
rho3 = horizontal_lines[int(np.floor(len(horizontal_lines) // 4 * 1))][0]
rho4 = horizontal_lines[int(np.floor(len(horizontal_lines) // 4 * 1 - 1))][0]
theta1 = vertical_lines[len(vertical_lines) // 2 - 1][1]
theta2 = vertical_lines[len(vertical_lines) // 2 + 1][1]
theta3 = horizontal_lines[int(np.floor(len(horizontal_lines) // 4 * 1))][1]
theta4 = horizontal_lines[int(np.floor(len(horizontal_lines) // 4 * 1 - 1))][1]

lines = [rho1, theta1, rho2, theta2, rho3, theta3, rho4, theta4]
rectangleLines = []
```

At step 5, I do some math from "open-cv" documentation to draw lines. I found two intersection points from last four line and thanks to two points I draw my predicted rectangle.

```
for i in range(0, len(lines), 2):
    rho = lines[i]
    theta = lines[i + 1]
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a * rho
    y0 = b * rho
    x1 = int(x0 + 1000 * (-b))
    y1 = int(y0 + 1000 * a)
    x2 = int(x0 - 1000 * (-b))
    y2 = int(y0 - 1000 * a)
    rectangleLines.append([x1, y1, x2, y2])

L1 = line([rectangleLines[0][0], rectangleLines[0][1]], [rectangleLines[0][2], rectangleLines[0][3]])
L2 = line([rectangleLines[2][0], rectangleLines[2][1]], [rectangleLines[2][2], rectangleLines[2][3]])
L3 = line([rectangleLines[1][0], rectangleLines[1][1]], [rectangleLines[1][2], rectangleLines[1][3]])
L4 = line([rectangleLines[3][0], rectangleLines[3][1]], [rectangleLines[3][2], rectangleLines[3][3]])
p1 = intersection(L1, L2)
p2 = intersection(L3, L4)
cv2.rectangle(img, (round(p1[0]), round(p1[1])), (round(p2[0]), round(p2[1])), (0, 0, 255), 3)
readXML_and_calculateIOU(p1, p2)
```

4 Sample Correct Plates

Image	Car Number	IOU Score
	Cars116	0.56
	Cars111	0.52
	Cars113	0.49

	Cars11	0.48
	Cars150	0.48
	Cars80	0.47

	Cars12	0.44
	Cars144	0.44
	Cars72	0.44

	Cars10	0.43

5 IOU Score Analysis

My average IOU score for 146 car images is 0.17. For 5 of them I faced with some bugs but I could not find and solve that bugs.

6 Bad Results

I got zero points from these images.

For this image I suppose text at right-bottom mislead my prediction model.

For this image I suppose model could not decide which plate to pick.

For this image I suppose model could not decide which plate to pick.

For this image I suppose model could not find correct lines Because my main focus was on finding horizantal and vertical lines.

For this image I could not find any problem but model could not find plate.