Hough Transformation

The Hough Transform is an image analysis technique for object detection, identifying shapes within images. A common use case is the recognition of lines in an image, for example to recognize a road lane in a picture taken by a camera mounted on a car.

In this first task are 4 test images given and we have to perform the Hough Transformation on them. The idea of the Hough Transformation is to transform the image from the Cartesian coordinate system to the Hough space. In the Hough space, the lines are represented by a point. The point is the intersection of the lines in the Cartesian coordinate system. The Hough space is a 2D space with the d and α axis. The d axis is the distance from the origin to the line and the α axis is the angle between the x-axis and the line. The Hough Transformation is a mapping from the Cartesian coordinate system to the Hough space. The mapping is done by the following formula:

$$d = x \cdot \cos(\alpha) + y \cdot \sin(\alpha)$$

Each point in the image votes for all the lines that could pass through it in the parameter space (Hough space). The accumulator space (a 2D array for lines) tallies votes for each potential line. The results are shown in Figure 1.

We can observe the detection of the points in the hough space is represented as a local maximum, which the parameter set with the most votes. We can see this clearly in the 4th example with 2 peaks in the parameter space, which represented each of the line. The Hough Transformation is a very useful tool for detecting lines in an image. However, it is not very efficient. The Hough Transformation is a very expensive operation. The complexity of the Hough Transformation is $O(n^2)$, where n is the number of points in the image. The Hough Transformation is also very sensitive to noise. The noise in the image can cause the Hough Transformation to detect lines that are not in the image. We can see this in the 3rd example, where are several maximum in the parameter space.

Hsin-Feng Ho
03770686

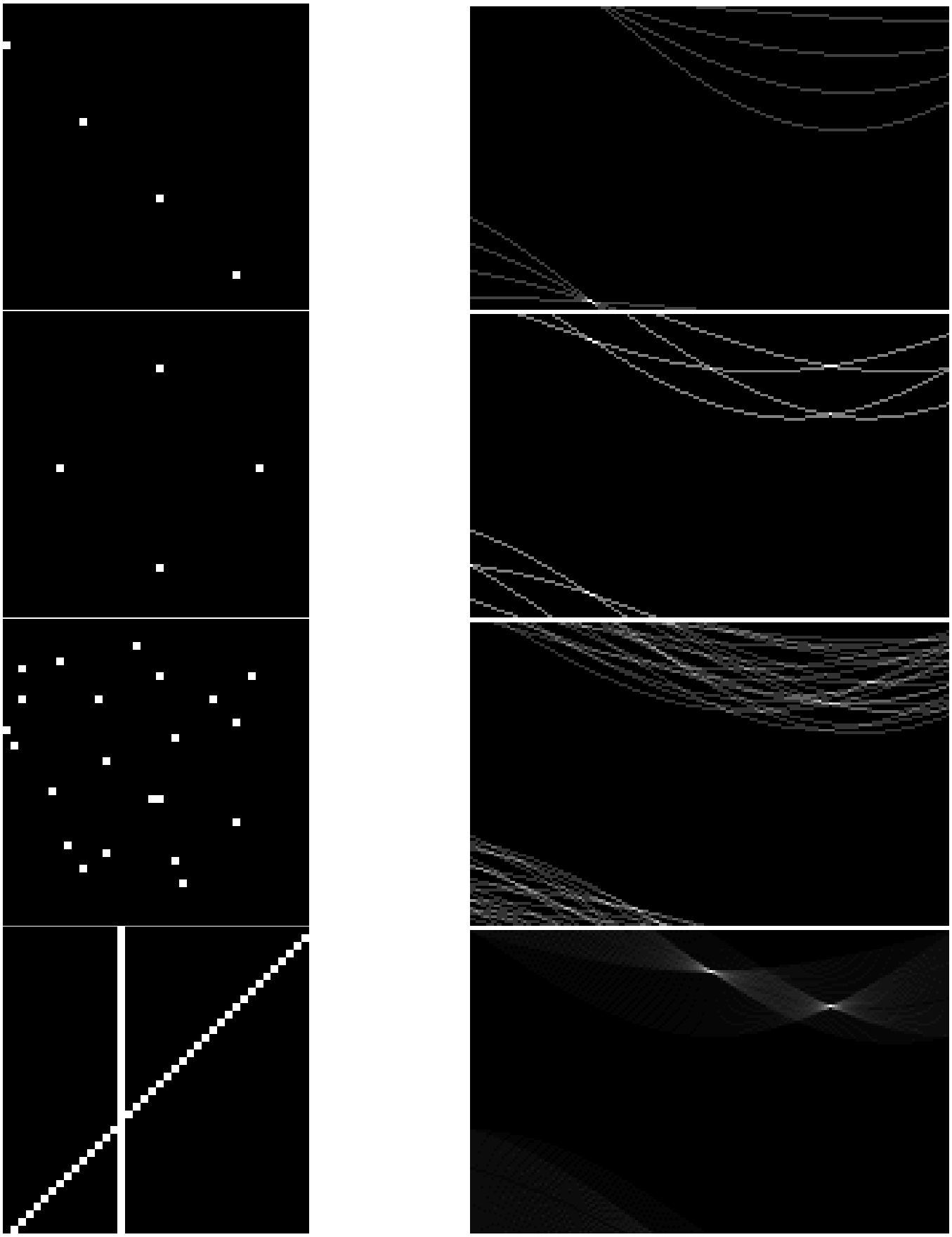


Figure 1: The results of the Hough Transformation on the 4 test images.

Validation of the Hough Transformation results is achieved by reprojecting the detected lines onto the image. The reprojected lines are shown in Figure 2. We can see that the lines are detected correctly.

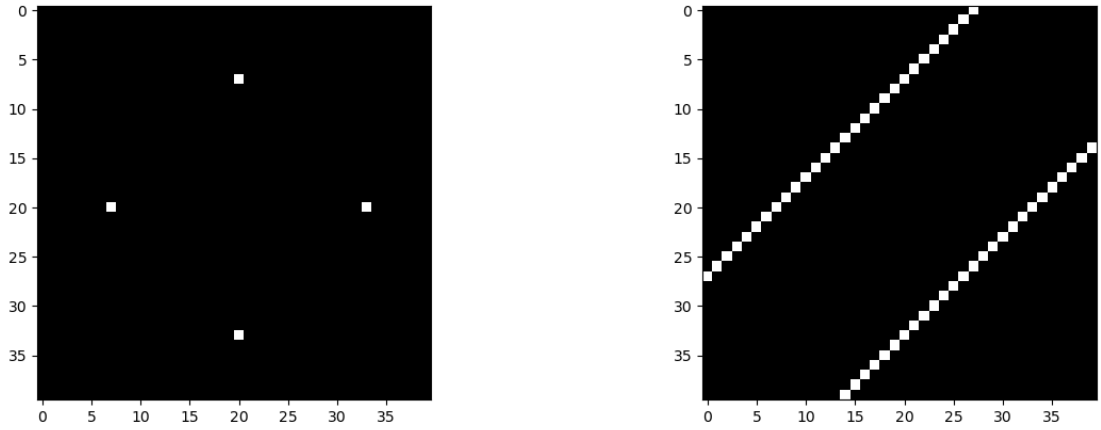


Figure 2: The reprojected lines in the image.

The calculated values for d and α are shown in below:

```
Line 1: d= 19 alpha= 45.0 accumulated votes= 2
Line 2: d= 37 alpha= 45.0 accumulated votes= 2
```

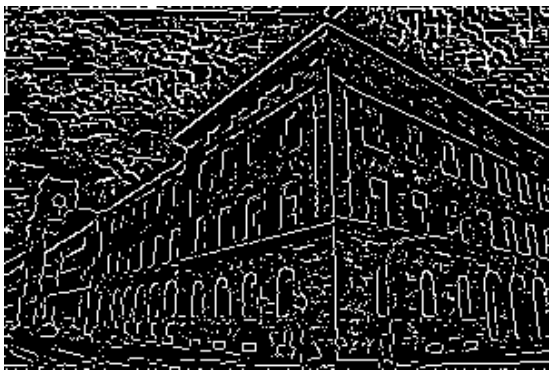
Line detection using Sobel

In this task we have to detect the lines in the image using the Sobel operator. The Sobel operator is a discrete differentiation operator. It is used to compute an approximation of the gradient of the image intensity function. The Sobel operator is based on convolving the image with a small, separable, and integer valued filter in horizontal and vertical direction and is therefore relatively inexpensive in terms of computations. The results are shown in Figure 3.

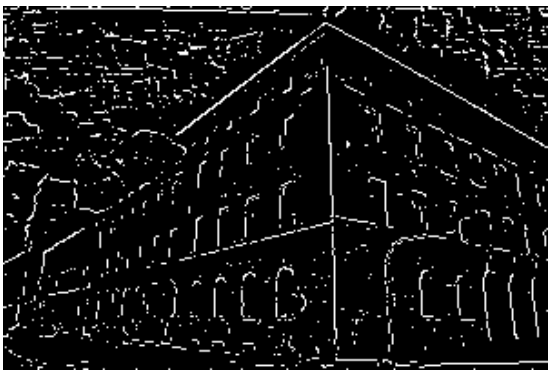
Sobel direction Sobel magnitude

Figure 3: Results of Sobel operator

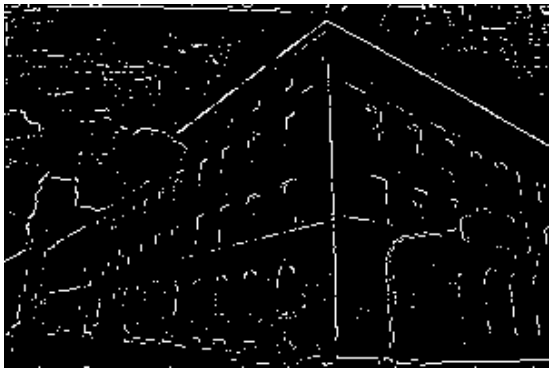
After applying the Sobel operator, we can use non maximum suppression to find the local maximum in the image. The results are shown in Figure 4. We can see kernel size of the non maximum suppression is very important. If the kernel size is too small, the non maximum suppression will be sensitive to noise. If the kernel size is too large, we might lose too much detail in the image.



kernel size = 4



kernel size = 10



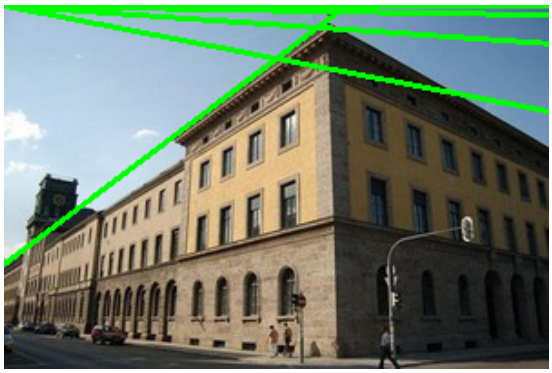
kernel size = 20

kernel size = 30

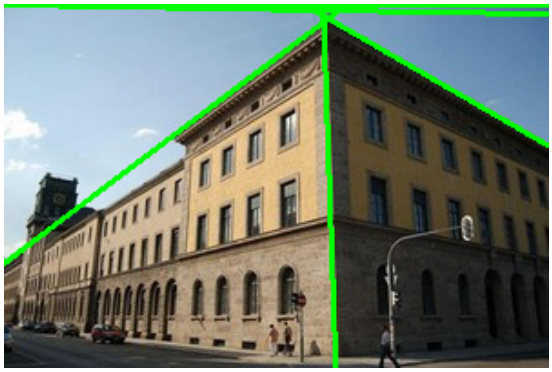
Figure 4: Results of non maximum suppression

Now we can apply the Hough Transformation on the edges. We also have to tune the parameter of threshold and kernel size of the non maximum suppression. The results are shown in Figure 5.

Hsin-Feng Ho
03770686


kernel size = 4, threshold = 100


kernel size = 10, threshold = 100


kernel size = 20, threshold = 80


kernel size = 25, threshold = 40

Figure 5: Results of Hough Transformation

We can see that the Hough Transformation is very sensitive to the parameter of the non maximum suppression. If the kernel size is too small, the Hough Transformation will detect too many lines. If the kernel size is too large, the Hough Transformation will not detect enough lines. The threshold is also very important. If the threshold is too small, the Hough Transformation will detect too many lines. If the threshold is too large, the Hough Transformation will not detect enough lines. In this example, the best result is achieved with a kernel size of 20 and a threshold of 80. It's able to detect 3 lines in the given building image.

Hsin-Feng Ho
03770686

Code

```
import matplotlib.pyplot as plt
import numpy as np
import cv2
from hough import hough_lines_accumulator, hough_peaks


# test points image 02
image_4pts = np.zeros((40,40))
image_4pts[20,7] = 1
image_4pts[7,20] = 1
image_4pts[20,33] = 1
image_4pts[33,20] = 1

# test points image 01
image_ptsInRow = np.zeros((40,40))
image_ptsInRow[5,0] = 1
image_ptsInRow[15,10] = 1
image_ptsInRow[25,20] = 1
image_ptsInRow[35,30] = 1

img=image_4pts
accumulator, thetas, rhos = hough_lines_accumulator(img)

plt.figure()
plt.imshow(img,cmap='gray')
plt.figure()
plt.imshow(accumulator)
accumulator_copy = accumulator.copy()
peaks = hough_peaks(accumulator_copy, 20,1)


d_idx, theta_idx = peaks[8]
d= rhos[d_idx]
alpha = thetas[theta_idx]
print('Line 1: d=',d,'alpha=',alpha/np.pi*180,'accumulated
   votes=',accumulator[93,135])
a=np.cos(alpha)
b=np.sin(alpha)
x0=a*d
y0=b*d
x1=int(x0+1000*(-b))
y1=int(y0+1000*(a))
x2=int(x0-1000*(-b))
y2=int(y0-1000*(a))
cv2.line(img,(x1,y1),(x2,y2),(255,255,255),1)
d_idx, theta_idx = peaks[14]
d= rhos[d_idx]
alpha = thetas[theta_idx]
```

```
print('Line 2: d=',d,'alpha=',alpha/np.pi*180, 'accumulated votes=',accumulator[d_idx,theta_idx])
a=np.cos(alpha)
b=np.sin(alpha)
x0=a*d
y0=b*d
x1=int(x0+1000*(-b))
y1=int(y0+1000*(a))
x2=int(x0-1000*(-b))
y2=int(y0-1000*(a))
cv2.line(img,(x1,y1),(x2,y2),(255,255,255),1)
plt.figure()
plt.imshow(img,cmap='gray')
plt.show()
```

code/task1.py

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
from hough import hough_lines_accumulator, hough_peaks
from IP_ex3_func import sobel, non_max_suppresion, colorTransform, convolution

img=cv.imread('../images/building.jpg')
gray=cv.cvtColor(img,cv.COLOR_BGR2GRAY)
Sx,Sy,Sdir,Smag=sobel(gray)
plt.imsave('../plots/sobel_mag.png',Smag,cmap='gray')
plt.imsave('../plots/sobel_dir.png',Sdir,cmap='gray')
edges=non_max_suppresion(Sdir,Smag,size=30)
plt.imsave('../plots/edges_30.png',edges,cmap='gray')
# Hough Transform
accumulator, alpha, d = hough_lines_accumulator(edges)
peaks = hough_peaks(accumulator, num_peaks=5, threshold=40)

# Draw lines
for peak in peaks:
    rho = d[peak[0]]
    theta = alpha[peak[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))
    cv.line(img, (x1, y1), (x2, y2), (0, 255, 0), 2)

# Save the result
cv.imwrite('../plots/houghlines_25_40.png', img)
```

code/task2.py

```python
import numpy as np
import cv2 as cv
def hough_lines_accumulator(edges):
    # Implement a Hough transform that takes in an edge image and outputs a Hough
    height, width = edges.shape
    max_d = int(np.sqrt(height**2 + width**2))
    d = np.arange(-max_d, max_d, 1)
    alpha = np.deg2rad(np.arange(-90,90,1))

    accumulator = np.zeros((len(d), len(alpha)), dtype=int)
    y_idxs, x_idxs = np.nonzero(edges)

    for i in range(len(x_idxs)):
        x = x_idxs[i]
        y = y_idxs[i]

        for j in range(len(alpha)):
            d = int(x * np.cos(alpha[j]) + y * np.sin(alpha[j]))
                + max_d
            accumulator[d, j] += 1

    return accumulator, alpha, d

def hough_peaks(accumulator, num_peaks, threshold=100):
    # Sort the accumulator to find the maxima
    peaks = []
    for _ in range(num_peaks):
        idx = np.argmax(accumulator)
        d_idx, alpha_idx = np.unravel_index(idx, accumulator.shape)
        if accumulator[d_idx, alpha_idx] > threshold:
            peaks.append((d_idx, alpha_idx))
            accumulator[d_idx, alpha_idx] = 0
        else:
            break
    return peaks
```

code/hough.py