Assignment 8: Edge detection

1. Develop and apply the compass filter to an image and discuss the results.

The compass filter employs eight Sobel filters with an orientation spaced at 45° starting from

$$H_0^{ES} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, H_1^{ES} = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}, \dots, H_7^{ES} = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}$$

Where the edge strength at position (x, y) can be calculated from the maximum output after applying the convolution between the original input and the eight filters

$$E^{ES}(x,y) = \max\left(D_0(x,y), D_1(x,y), \dots, D_7(x,y)\right) \text{ where } D_i^{ES} = I * H_i^{ES}$$

CODE (1)

```
# Compass operator
def convert_color(BGR):
    converted = cv2.cvtColor(BGR, cv2.COLOR_BGR2RGB)
    plt.imshow(converted)

img = cv2.resize(cv2.imread("/Users/nopparuj/ZSM-00/IPM/zuto.png",0),
(1000,1000))

robin0 = np.multiply(np.array([[-1,0,1],[-2,0,2],[-1,0,1]]), (1/8))
robin1 = np.multiply(np.array([[-2,-1,0],[-1,0,1],[0,1,2]]), (1/8))
robin2 = np.multiply(np.array([[-1,-2,-1],[0,0,0],[1,2,1]]), (1/8))
robin3 = np.multiply(np.array([[0,-1,-2],[1,0,-1],[2,1,0]]), (1/8))
robin4 = np.multiply(np.array([[1,0,-1],[2,0,-2],[1,0,-1]]), (1/8))
robin5 = np.multiply(np.array([[2,1,0],[1,0,-1],[0,-1,-2]]), (1/8))
robin6 = np.multiply(np.array([[1,2,1],[0,0,0],[-1,-2,-1]]), (1/8))
robin7 = np.multiply(np.array([[0,1,2],[-1,0,1],[-2,-1,0]]), (1/8))

def sizing(image,filters):
    out_size = (image.shape[0] - filters.shape[0]) + 1
    return out_size

def apply_filter(image,kernel,outputsize):
    filtered = np.zeros((outputsize,outputsize))
    for i in range (outputsize):
        for j in range (outputsize):
            pre_res = (kernel * image[i:i+3, j:j+3]).sum()
            if (pre_res < 50):
                pre_res = 0
            else:
                pre_res = 255
            filtered[i][j] = pre_res
    return filtered.astype(np.uint8)

def
compass_filter(input,kernel0,kernel1,kernel2,kernel3,kernel4,kernel5,kernel6,kern
el7):

    results = np.zeros((resolution,resolution))
```

```
    compass0 = apply_filter(input,kernel0,resolution)
    compass1 = apply_filter(input,kernel1,resolution)
    compass2 = apply_filter(input,kernel2,resolution)
    compass3 = apply_filter(input,kernel3,resolution)
    compass4 = apply_filter(input,kernel4,resolution)
    compass5 = apply_filter(input,kernel5,resolution)
    compass6 = apply_filter(input,kernel6,resolution)
    compass7 = apply_filter(input,kernel7,resolution)

    for u in range (resolution):
        for v in range (resolution):
            pre_out =
np.array([compass0[u][v],compass1[u][v],compass2[u][v],compass3[u][v],compass4[u]
[v],compass5[u][v],compass6[u][v],compass7[u][v]])
            output = pre_out.max()
            if (output < 50):
                output = 0
            else:
                output = 255
            results[u][v] = output

    return results.astype(np.uint8)

resolution = sizing(img,robin0) #All sizes are the same
compass_res =
compass_filter(img,robin0,robin1,robin2,robin3,robin4,robin5,robin6,robin7)

plt.figure(1)
plt.subplot(1,2,1)
plt.title("Original image")
convert_color(img)
plt.subplot(1,2,2)
plt.title("Compass operator")
convert_color(compass_res)
plt.show()
```

Result

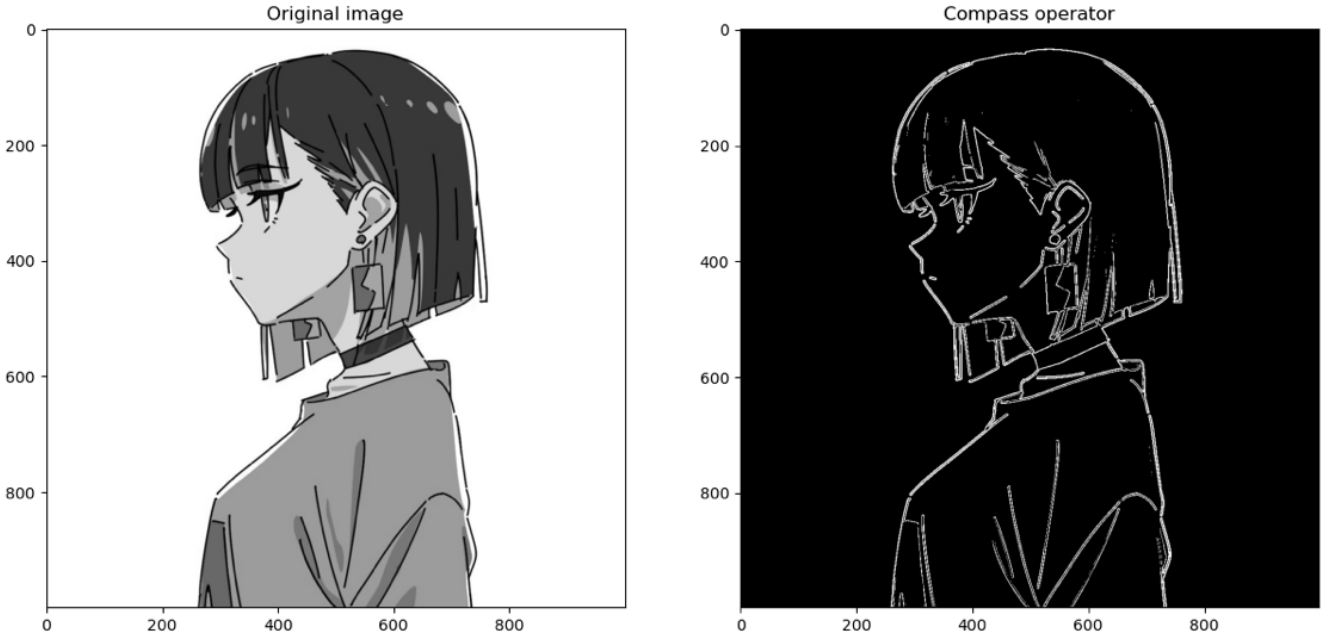


Figure 1: The result compares input image (grayscale) and output from compass operator

The edges obtained from using compass operator seem to have good outline, the threshold is selected as 50. The output (right) is compared to the image obtained from using Sobel operator (left), the edges are more detailed and obvious using compass operator, especially at the hair of the character in the image, apart from that the result is almost identical maybe due to the filters used is the Sobel filters with rotated orientations.

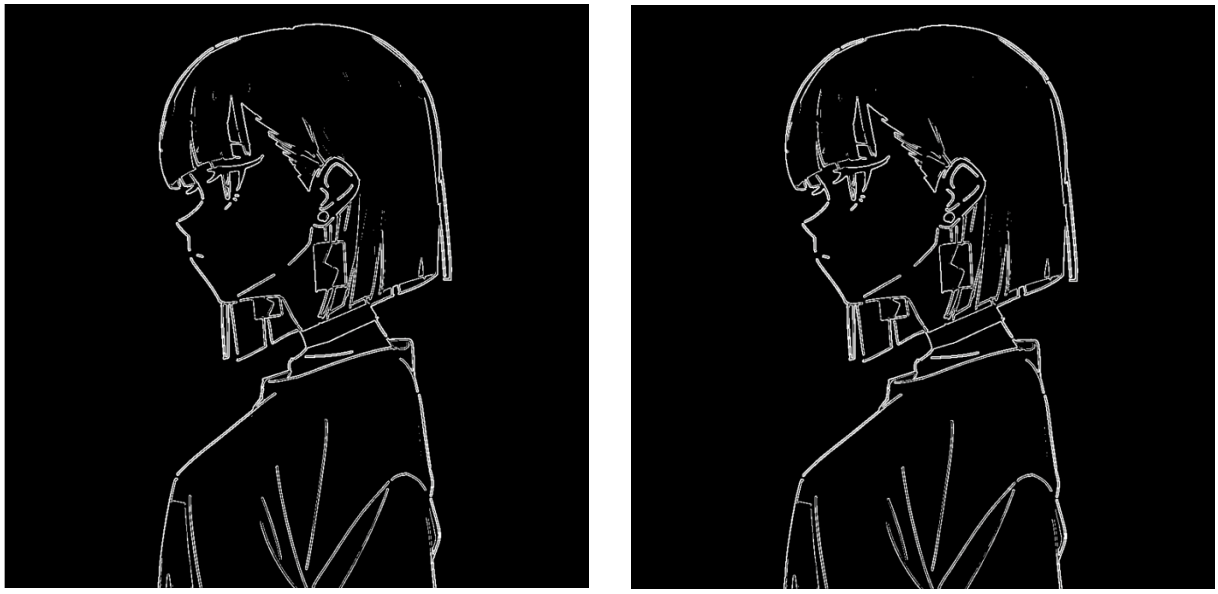


Figure 2: Result compare between using Sobel and compass operator

2. Study Canny Operator and apply to an image.

Canny operator is one of the best methods when performing an edge detection, the advantage of this method is it can minimize the false positive result (false edges) and have good edge localization, below is the processes in performing Canny edge detection.

Pre-processing – first we smooth the image using a gaussian filter with its width σ, then the general edge detection is performed using Sobel filter, hence, gradient magnitude E_{mag} and local edge orientation ϕ.

$$\text{Where } E_{mag} = \sqrt{I_x^2 + I_y^2} \text{ and } \phi = tan^{-1}\left(\frac{I_y}{I_x}\right)$$

The gradient magnitude and edge orientation sample is shown in Figure X below.

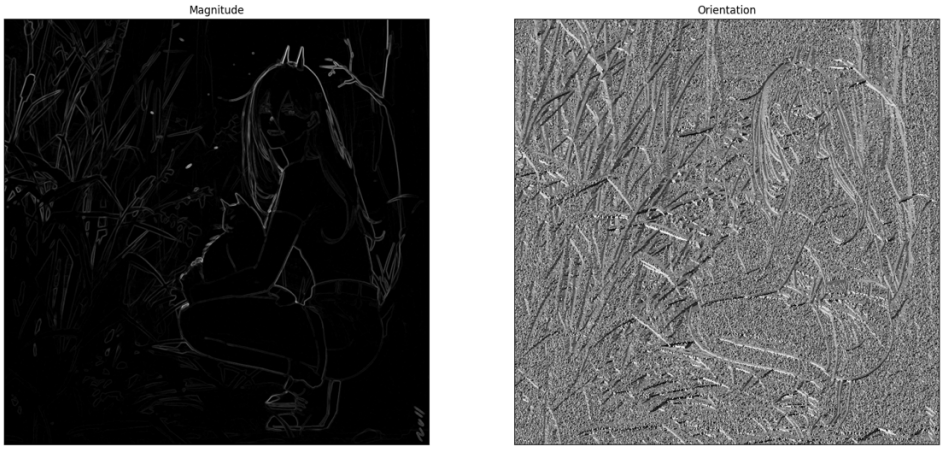
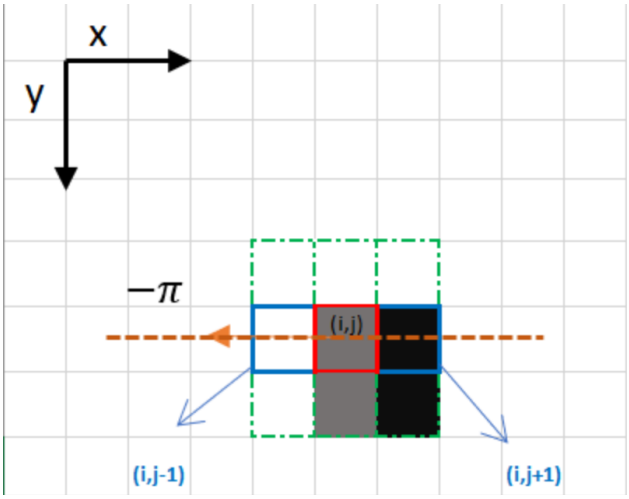


Figure 3: Gradient magnitude (left) and edge orientation (right)

Edge localization – the ideal image result should have thin edges, thus, we need 'non-maximum suppression' method to reduce the edge size, where the principle is simply by going through all the points from the previous step result and find the maximum value from the edge direction while suppressing the others. Using Sobel filter will obtain the edge orientation, we may use these values to determine which direction the edge goes. For instance, based on the image below the edge direction goes from left to right, hence, we check whether the left or right pixel is larger or not, as we can see that pixel $(i, j-1)$ has more intensity hence the pixel (i, j) is suppressed.



The coding principle of non-maximum suppression is according to the angle of the directions, it has 4 cases, as we can see from the octant.

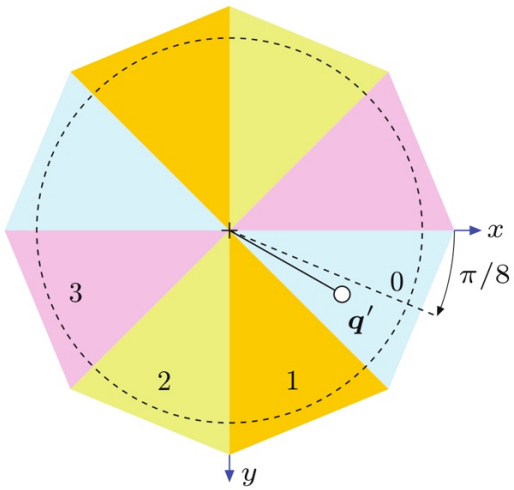


Figure 4: The octant of orientation vectors

Hence 4 conditions can be made, assume pixel (i, j) is observed:

1. If orientation rotate in 0 octant: check pixel $(i-1, j)$ and $(i+1, j)$
2. If orientation rotate in 1 octant: check pixel $(i-1, j-1)$ and $(i+1, j+1)$
3. If orientation rotate in 2 octant: check pixel $(i, j-1)$ and $(i, j+1)$
4. If orientation rotate in 3 octant: check pixel $(i-1, j+1)$ and $(i+1, j-1)$

The result from this process is the same with thinner edges with certain amount of pixel suppressed, the image is shown in Figure X, using written code in python.

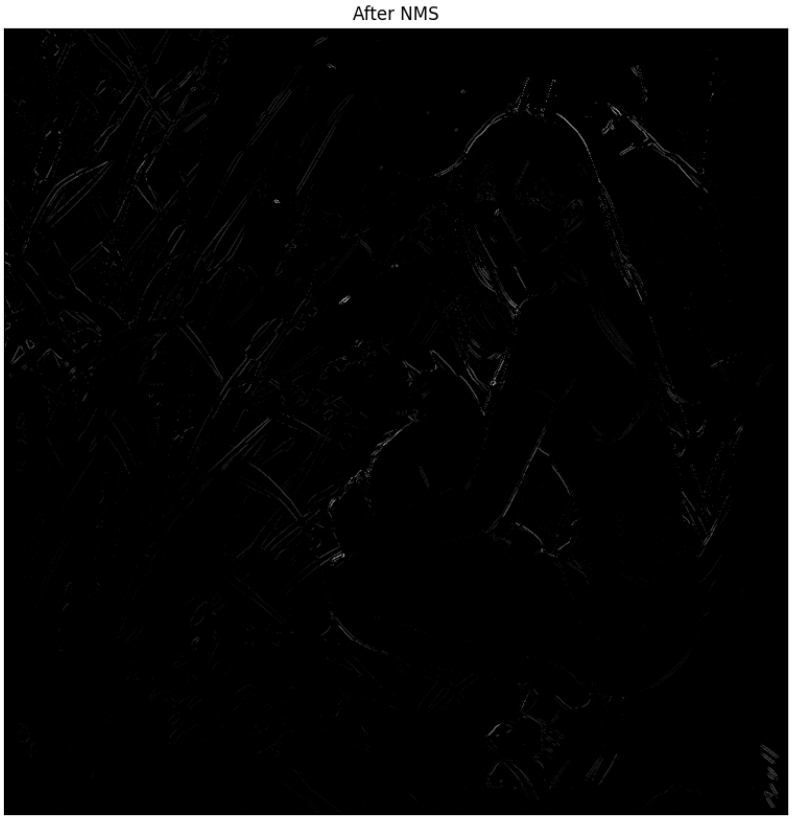


Figure 5: The result after non-maximum suppression is performed

Hysteresis thresholding – after all edge points are collect (the pixel that remains unsuppressed), double thresholding is performed, we set of two values for high and low threshold, any value that is higher than high threshold will be changed to 255, and below low threshold will be changed to 0. While any value that is between low and high threshold ($low \leq x \leq high$) will remain 'only if' that pixel is connect to the pixel that exceed the high threshold.

The result after all processes is perform is shown in Figure X, where different values of threshold is used. Using low values of threshold will lead to unnecessary edges detected while using high values will lead to missing to some significant edges, as we can see, the result from using 50 and 150 provides the most appropriate result.

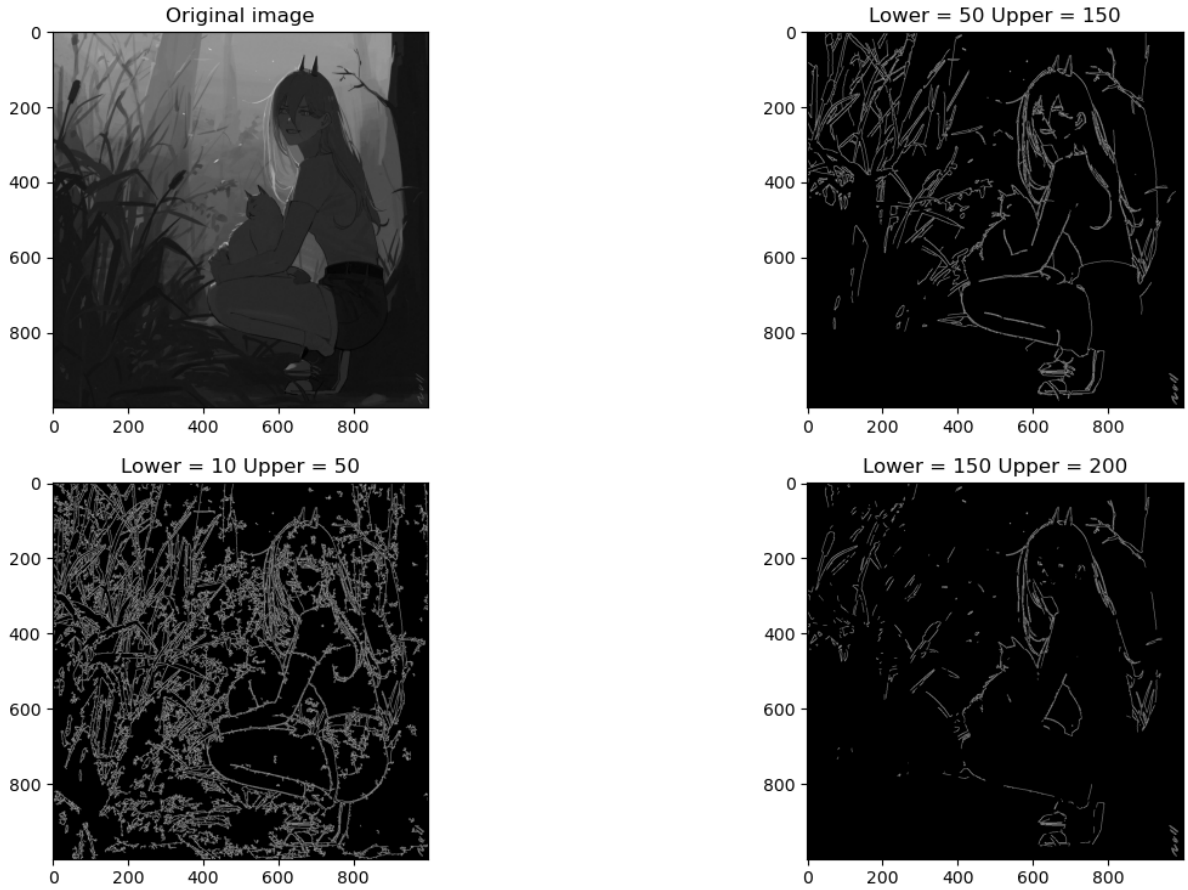


Figure 6: The result after non-maximum suppression is performed

CODE (2) (built-in)(Own written code for NMS and general detection is in appendix)

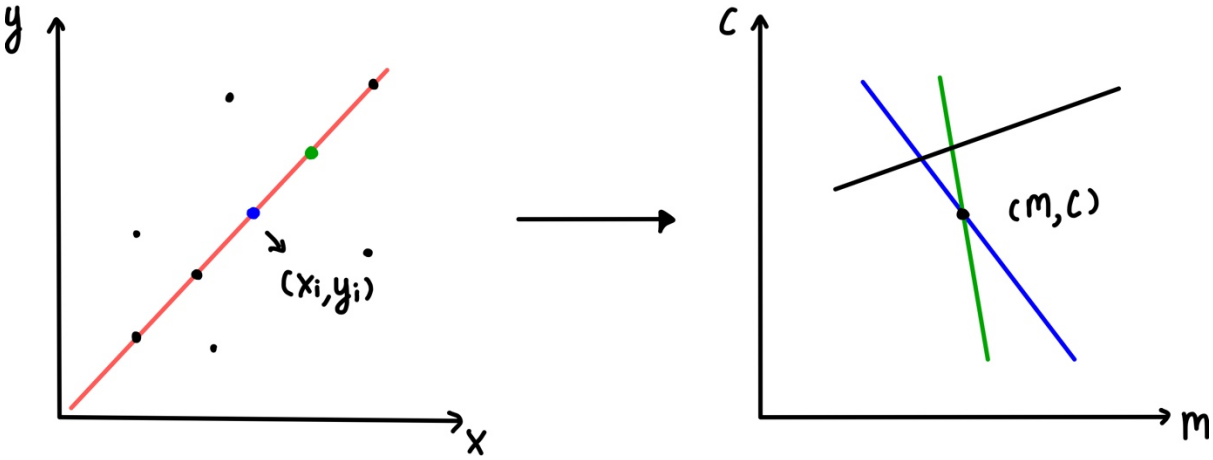
```
def plot_images(image1,image2,image3,image4):
    plt.figure(1)
    plt.subplot(2,2,1)
    plt.title("Original image")
    convert_color(image1)
    plt.subplot(2,2,2)
    plt.title("Lower = 50 Upper = 150")
    convert_color(image2)
    plt.subplot(2,2,3)
    plt.title("Lower = 10 Upper = 50")
    convert_color(image3)
```

```
    plt.subplot(2,2,4)
    plt.title("Lower = 150 Upper = 200")
    convert_color(image4)
    plt.show()

def apply_canny(image,t_upper,t_lower):
    canny_edges = cv2.Canny(image,t_upper,t_lower)
    return canny_edges.astype(np.uint8)

result1 = apply_canny(img,50,150)
result2 = apply_canny(img,10,50)
result3 = apply_canny(img,150,200)

plot_images(img,result1,result2,result3)
```

3. Study Hough Transform and show how to use it step by step.

Hough transform is a feature extraction method that is used for detecting particular shape such as line or circle, with a goal of avoiding an extraneous data and noises. (There will only have an basic illustration of line detection only)

If we consider the point (x_i, y_i) the linear equation can be written which is $y_i = mx_i + c$



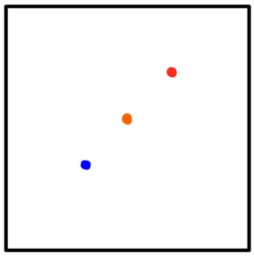
The linear equation takes place in image space; however, the equation can be rewritten to change its form into parameter space which is $c = -mx_i + y_i$

Any specific point that made up a straight line, if changes to parameter space it will become a line that pass-through coordinate (m, c) (blue and green), the there is a point that does not related to a line, it will not pass through the point when changed to parameter space.

The algorithm of line detection composed of 4 steps:

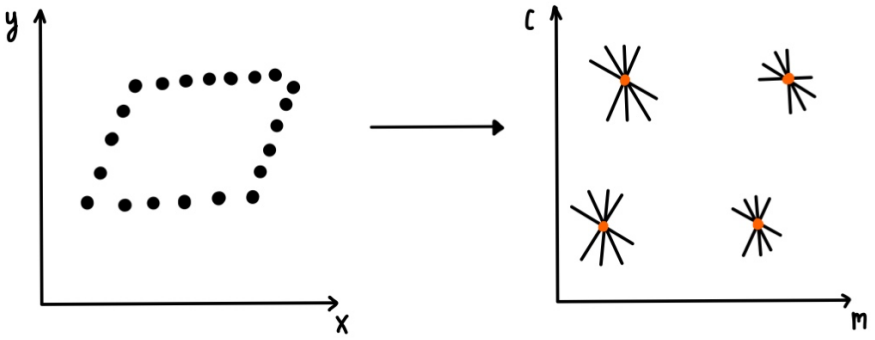
1. Quantize parameter space (m, c)
2. Create an accumulator array $A(m, c)$
3. Assign zero for all in $A(m, c)$
4. For each point (x_i, y_i), $A(m, c) = A(m, c) + 1$ (only if (m, c) lies on the line in parameter space ($c = -mx_i + y_i$)

For instance, based on the left image below, there is 3 points that can be connected to create a line, in accumulator array is written with different color except for the intersection point which is cumulative.



1	0	0	0	1
0	1	0	1	0
1	1	**3**	1	1
0	1	0	1	0
1	0	0	0	1

For another example, suppose we have certain number of points that can be connected to create a shape, in parameter space it will look like this



However, when parametrize the accumulator array using slope (m) and intercept (c), it needs a large accumulator size also the slope can go from $-\infty$ to ∞. Hence, we instead use the parameterization equation of the straight line

$$xcos\theta + ycos\theta = -\rho$$

Where θ is finite: $0 \leq \theta \leq \pi$
And ρ is the distance of the line from origin which cannot be bigger than the image itself.

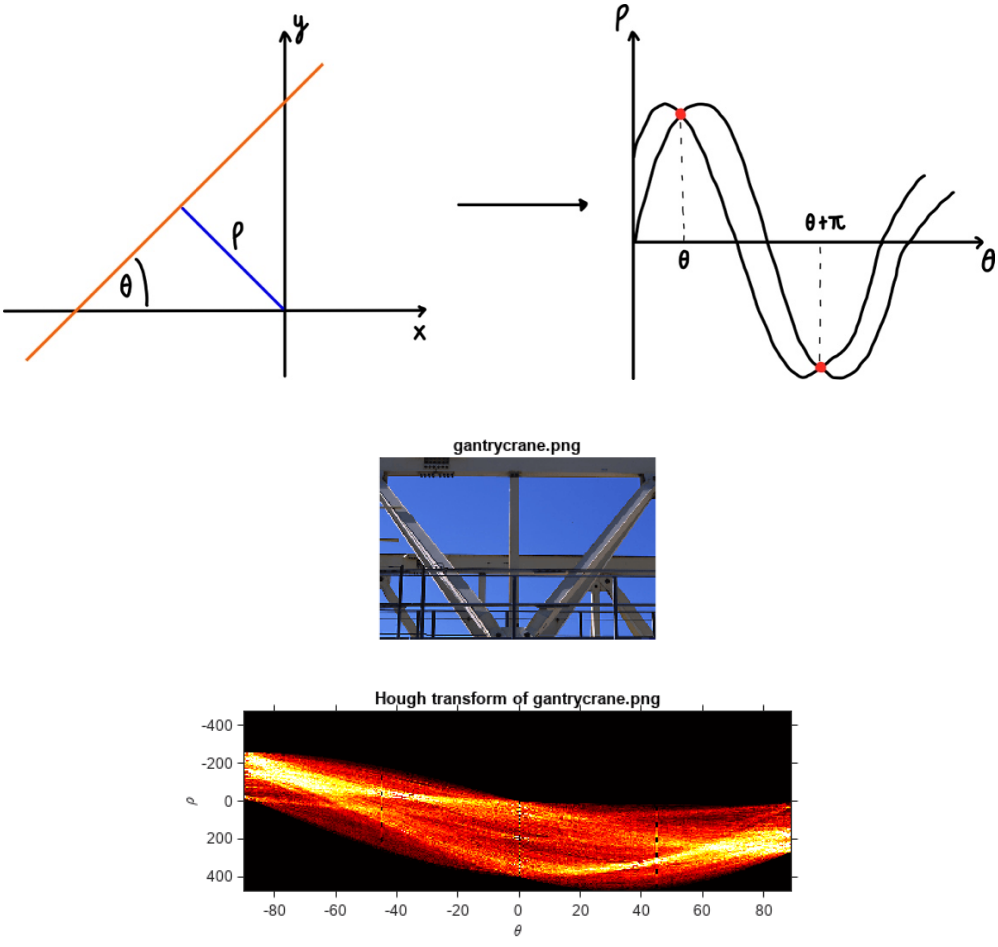


Figure 7: Line detection using Hough transform from image space to parameter space

Python implementation

```
#Hough line transform (OpenCV-based)
img = cv2.resize(cv2.imread("/Users/nopparuj/ZSM-00/IPM/room.png",0),(1000,1000))

def hough_linePdetect(image):
    gray = img.copy()
    get_edges = cv2.Canny(image,50,150)
    get_lines = cv2.HoughLinesP(get_edges,1,np.pi/180,threshold=150,
minLineLength=10, maxLineGap=100)
    for i in get_lines:
        coor1, coor2, coor3, coor4 = i[0]
        res = cv2.line(gray,(coor1,coor2),(coor3,coor4),(0,0,0),2)
    return res

result = hough_linePdetect(img)

plt.subplot(1,2,1)
plt.title("Original")
convert_color(img)
plt.subplot(1,2,2)
plt.title("Hough transform : Line detection")
convert_color(result)
plt.show()
```

Result

Figure 8: Line detection using Hough transform

4. Apply edge sharpening with the Laplace filter to an image.
Edge sharpening technique employs Laplacian filter, which is related to a second derivative of the image function, by using the original image function to subtract with a certain fraction w of the second derivative of original function.

$$\hat{f}(x) = f(x) - w \cdot f''(x)$$

Where $f''(x)$ is second derivative of original function and $w \geq 0$.

$$I' = I - w \cdot (H^L * I)$$

$$H^L = H_x^L + H_y^L = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

(Result will be shown together in the same section with **(5)** and full **code(4&5)** will be shown in appendix section)

5. Apply unsharp mask technique to sharpening an image.
The first thing to do when performing the unsharp masking is to subtract a smooth filtered image from original image, we called this 'mask',M. Then the output is added to the original image again, hence, the image edges are enhanced.

$$M = I - (I * \widehat{H}) \text{ (1st step)}$$

$$\check{I} = I + a \cdot M \text{ (2nd step)}$$

Noted that $\widehat{H} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$

Where $\check{I}$ is the sharper image and a is a weight factor that control the amount of sharpening.

Results (4)

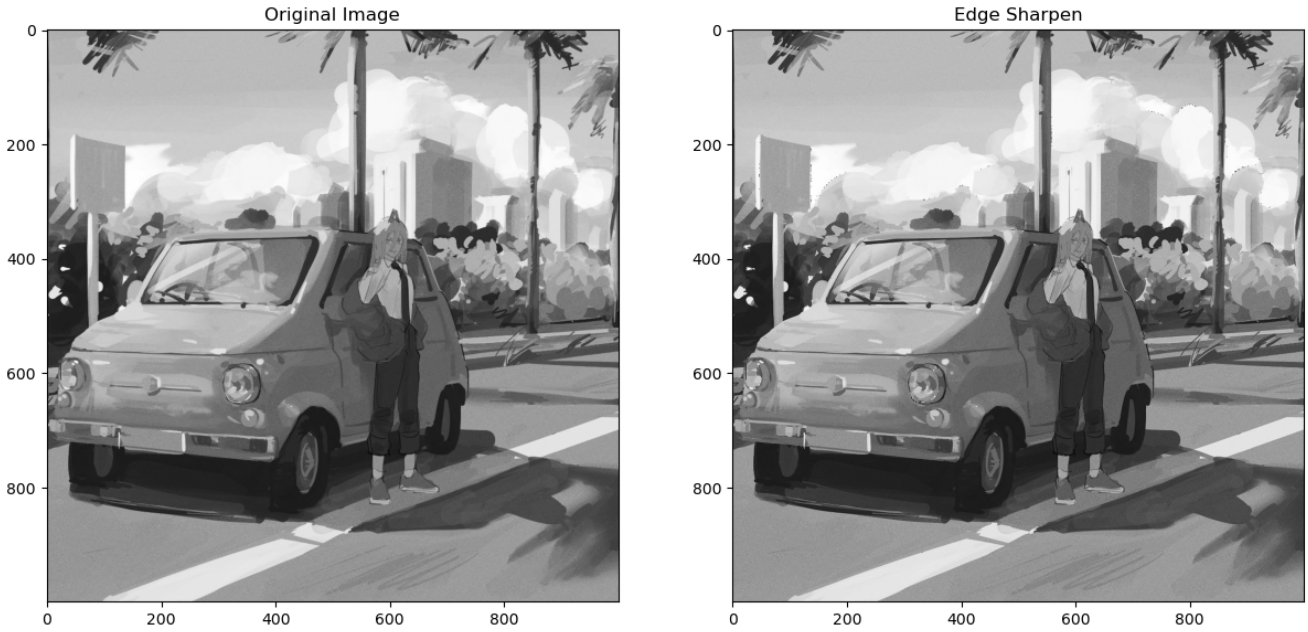


Figure 9: Result from using edge sharpening

After edge sharpening is performed, if focused, it can be seen that some edges have been enhanced (clouds and trees). Below is the intensity histogram comparing before and after being processed, some spikes can be detected throughout the intensity peaks.

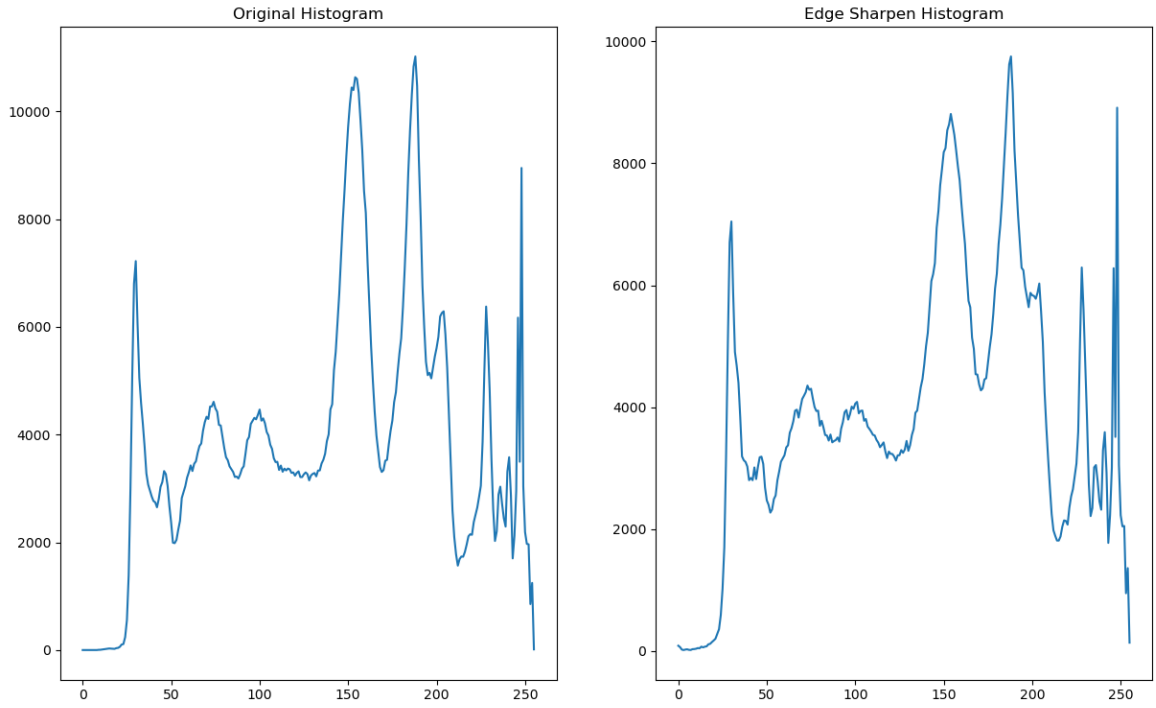


Figure 10 Histogram comparing before and after using edge sharpening technique

Another adjustable variable that we can change when performing this method is sharpening factor w, Figure 11, is the results from using different sharpening factor

between 2 and 0.8, increasing a factor 2 enhance the edge visibility to be more obvious while using 0.8 results in only clear visibility of cloud edges.

Figure 11 : Result from using sharpening factor of 2 (left) and 0.8 (right)

Result (5)

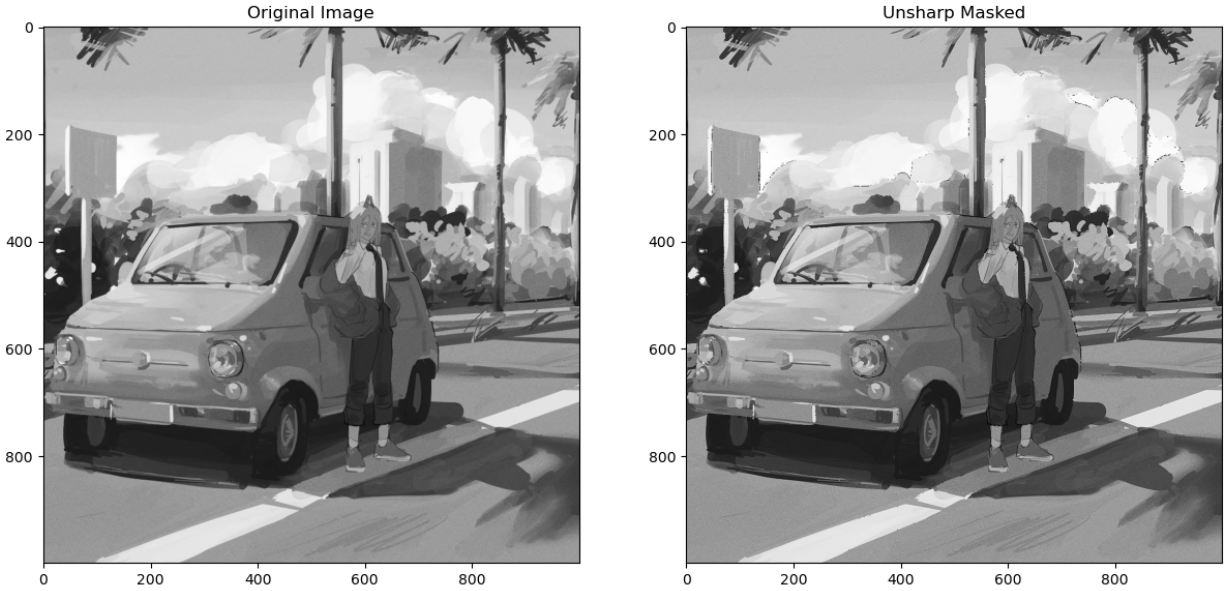


Figure 12: Result from using edge sharpening

Using unsharp masking technique leads to more visibility of edges comparing with edge sharpening, edges of cloud and trees is clearly noticeable, Figure 13 shows the histogram of pixel intensity between original image and processed image, comparing with the previous technique the graph becomes more oscillated along the curve. If we increase the weight factor to 2, the edges found will even become more visible and it is shown in Figure 14.

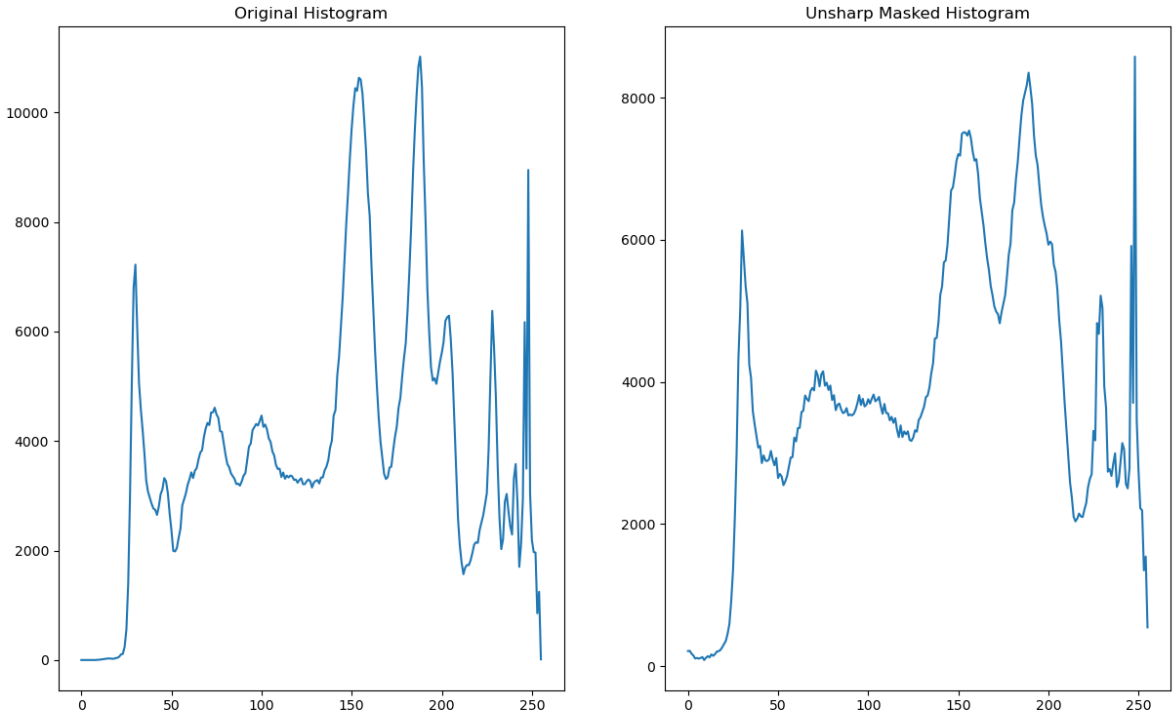


Figure 13: Histogram comparing before and after using unsharp masking technique

Figure 14: Unsharp masking technique result using weight factor of 2

Appendix 1 (Full code of (1))

```
import numpy as np
import cv2
import matplotlib.pyplot as plt

# Compass operator
def convert_color(BGR):
    converted = cv2.cvtColor(BGR, cv2.COLOR_BGR2RGB)
    plt.imshow(converted)

img = cv2.resize(cv2.imread("/Users/nopparuj/ZSM-00/IPM-ninja/zuto.png",0),
(1000,1000))

robin0 = np.multiply(np.array([[-1,0,1],[-2,0,2],[-1,0,1]]), (1/8))
robin1 = np.multiply(np.array([[-2,-1,0],[-1,0,1],[0,1,2]]), (1/8))
robin2 = np.multiply(np.array([[-1,-2,-1],[0,0,0],[1,2,1]]), (1/8))
robin3 = np.multiply(np.array([[0,-1,-2],[1,0,-1],[2,1,0]]), (1/8))
robin4 = np.multiply(np.array([[1,0,-1],[2,0,-2],[1,0,-1]]), (1/8))
robin5 = np.multiply(np.array([[2,1,0],[1,0,-1],[0,-1,-2]]), (1/8))
robin6 = np.multiply(np.array([[1,2,1],[0,0,0],[-1,-2,-1]]), (1/8))
robin7 = np.multiply(np.array([[0,1,2],[-1,0,1],[-2,-1,0]]), (1/8))

def sizing(image,filters):
    out_size = (image.shape[0] - filters.shape[0]) + 1
    return out_size

def apply_filter(image,kernel,outputsize):
    filtered = np.zeros((outputsize,outputsize))
    for i in range (outputsize):
        for j in range (outputsize):
            pre_res = (kernel * image[i:i+3, j:j+3]).sum()
            if (pre_res < 50):
                pre_res = 0
            else:
                pre_res = 255
            filtered[i][j] = pre_res
    return filtered.astype(np.uint8)

def
compass_filter(input,kernel0,kernel1,kernel2,kernel3,kernel4,kernel5,kernel6,kern
el7):

    results = np.zeros((resolution,resolution))
    compass0 = apply_filter(input,kernel0,resolution)
    compass1 = apply_filter(input,kernel1,resolution)
    compass2 = apply_filter(input,kernel2,resolution)
    compass3 = apply_filter(input,kernel3,resolution)
    compass4 = apply_filter(input,kernel4,resolution)
    compass5 = apply_filter(input,kernel5,resolution)
    compass6 = apply_filter(input,kernel6,resolution)
    compass7 = apply_filter(input,kernel7,resolution)
```

```
    for u in range (resolution):
        for v in range (resolution):
            pre_out =
np.array([compass0[u][v],compass1[u][v],compass2[u][v],compass3[u][v],compass4[u]
[v],compass5[u][v],compass6[u][v],compass7[u][v]])
            output = pre_out.max()
            if (output < 50):
                output = 0
            else:
                output = 255
            results[u][v] = output

    return results.astype(np.uint8)

resolution = sizing(img,robin0) #All sizes are the same
compass_res =
compass_filter(img,robin0,robin1,robin2,robin3,robin4,robin5,robin6,robin7)

plt.figure(1)
plt.subplot(1,2,1)
plt.title("Original image")
convert_color(img)
plt.subplot(1,2,2)
plt.title("Compass operator")
convert_color(compass_res)
plt.show()
```

Appendix 2.1 (Written code for general edge detection and NMS)

```
import numpy as np
import cv2
import matplotlib.pyplot as plt

# Try cunny operator without using built-in

# Convert BGR to RGB for matplotlib functions
def convert_color(BGR):
    converted = cv2.cvtColor(BGR, cv2.COLOR_BGR2RGB)
    plt.imshow(converted)

# Load image in grayscale
img = cv2.resize(cv2.imread("C:\\Users\\MBComputer\\Downloads\\powercat.png",0),
(1000,1000))

# Needed filter
gaussian = np.multiply(np.array([[1,2,1],[2,4,2],[1,2,1]]), (1/16))
sobel_x = np.multiply(np.array([[-1,0,1],[-2,0,2],[-1,0,1]]), (1/8))
sobel_y = np.multiply(np.array([[-1,-2,-1],[0,0,0],[1,2,1]]), (1/8))

# Find outputsize after convolution with filters
def sizing(image,filters):
    out_size = (image.shape[0] - filters.shape[0]) + 1
    return out_size

# Normalization
def normal(image):
    image = image / np.max(image)
    return image

# Apply gaussian then perform edge padding (from 998px --> 1000px)
def apply_gaussian(image,g_kernel):
    outputsize = sizing(image,g_kernel)
    smoothed = np.zeros((outputsize,outputsize))
    pre_smoothed = np.zeros((outputsize,outputsize))

    for i in range (outputsize):
        for j in range (outputsize):
            conv = (g_kernel * image[i:i+g_kernel.shape[0],
j:j+g_kernel.shape[1]]).sum()
            if (conv < 0):
                conv = 0
            pre_smoothed[i][j] = conv

    smoothed = np.pad(pre_smoothed, pad_width =1, mode ='edge')

    return smoothed.astype(np.uint8)

#Apply general edge detction (Sobel is used)
def general_detect(input_img,xaxis,yaxis):
```

```
    smooth_img = apply_gaussian(input_img,gaussian)
    resolution = sizing(smooth_img,xaxis)
    magnitude = np.zeros((resolution,resolution))
    horizontal = np.zeros((resolution,resolution))
    vertical = np.zeros((resolution,resolution))
    theta = np.zeros((resolution,resolution))

    for u in range (resolution):
        for v in range (resolution):
            res_x = (xaxis * smooth_img[u:u+xaxis.shape[0],
v:v+xaxis.shape[1]]).sum()
            horizontal[u][v] = res_x
            res_y = (yaxis * smooth_img[u:u+yaxis.shape[0],
v:v+yaxis.shape[1]]).sum()
            vertical[u][v] = res_y

            pre_magnitude = np.sqrt(((np.power(horizontal[u][v],2))
+(np.power(vertical[u][v],2))))
            # if (pre_magnitude < 5):
            #     pre_magnitude = 0
            # else:
            #     pre_magnitude = 255
            magnitude[u][v] = pre_magnitude

            theta[u][v] = np.degrees(np.arctan2(horizontal[u][v],vertical[u][v]))
    return magnitude.astype(np.uint8), theta

# Apply Non-Max-Suppression
def local_maximum(magni,dir):
    pre_canny = np.zeros(magni.shape)
    for i in range(dir.shape[0]-1):
        for j in range(dir.shape[1]-1):

            if ((dir[i][j] >= -22.5 and dir[i][j] <= 22.5) or (dir[i][j] <= -
157.5 and dir[i][j] >= 157.5)):
                if ((magni[i][j] > magni[i][j+1]) and (magni[i][j] > magni[i][j-
1])):
                    pre_canny[i][j] = magni[i][j]
                else:
                    pre_canny[i][j] = 0

            elif ((dir[i][j] >= 22.5 and dir[i][j] <= 67.5) or (dir[i][j] <= -
112.5 and dir[i][j] >= -157.5)):
                if ((magni[i][j] > magni[i+1][j+1]) and (magni[i][j] > magni[i-
1][j-1])):
                    pre_canny[i][j] = magni[i][j]
                else:
                    pre_canny[i][j] = 0

            elif ((dir[i][j] >= 67.5 and dir[i][j] <= 112.5) or (dir[i][j] <= -
67.5 and dir[i][j] >= -112.5)):
```

```
                if ((magni[i][j] > magni[i+1][j]) and (magni[i][j] > magni[i-
1][j])):
                    pre_canny[i][j] = magni[i][j]
                else:
                    pre_canny[i][j] = 0

            elif ((dir[i][j] >= 112.5 and dir[i][j] <= 157.5) or (dir[i][j] <= -
22.5 and dir[i][j] >= -67.5)):
                if ((magni[i][j] > magni[i+1][j-1]) and (magni[i][j] > magni[i-
1][j+1])):
                    pre_canny[i][j] = magni[i][j]
                else:
                    pre_canny[i][j] = 0
    return pre_canny.astype(np.uint8)

check1, direction = general_detect(img,sobel_x,sobel_y)
check2 = direction.astype(np.uint8)
ans = local_maximum(check1,direction)

plt.subplot(121),plt.imshow(check1,cmap='gray')
plt.title('After Sobel filter')
plt.xticks([])
plt.yticks([])
plt.subplot(122),plt.imshow(ans,cmap='gray')
plt.title('After NMS')
plt.xticks([])
plt.yticks([])
plt.show()
```

Appendix 2.2 (Full code of (2)) (Built-in, OpenCV)

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def convert_color(BGR):
    converted = cv2.cvtColor(BGR, cv2.COLOR_BGR2RGB)
    plt.imshow(converted)

img = cv2.resize(cv2.imread("/Users/nopparuj/ZSM-00/IPM/powercat.png",0),
(1000,1000))

def plot_images(image1,image2,image3,image4):
    plt.figure(1)
    plt.subplot(2,2,1)
    plt.title("Original image")
    convert_color(image1)
    plt.subplot(2,2,2)
    plt.title("Lower = 50 Upper = 150")
    convert_color(image2)
    plt.subplot(2,2,3)
    plt.title("Lower = 10 Upper = 50")
    convert_color(image3)
    plt.subplot(2,2,4)
    plt.title("Lower = 150 Upper = 200")
    convert_color(image4)
    plt.show()

def apply_canny(image,t_upper,t_lower):
    canny_edges = cv2.Canny(image,t_upper,t_lower)
    return canny_edges.astype(np.uint8)

result1 = apply_canny(img,50,150)
result2 = apply_canny(img,10,50)
result3 = apply_canny(img,150,200)

plot_images(img,result1,result2,result3)
```

Appendix 3 (Full code of (3))

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import math

def convert_color(BGR):
    converted = cv2.cvtColor(BGR, cv2.COLOR_BGR2RGB)
    plt.imshow(converted)

#Hough line transform (OpenCV-based)
img = cv2.resize(cv2.imread("/Users/nopparuj/ZSM-00/IPM/room.png",0),
(1000,1000))

def hough_linePdetect(image):
    gray = img.copy()
    get_edges = cv2.Canny(image,50,150)
    get_lines = cv2.HoughLinesP(get_edges,1,np.pi/180,threshold=150,
minLineLength=10, maxLineGap=100)
    for i in get_lines:
        coor1, coor2, coor3, coor4 = i[0]
        res = cv2.line(gray,(coor1,coor2),(coor3,coor4),(0,0,0),2)
    return res

result = hough_linePdetect(img)

plt.subplot(1,2,1)
plt.title("Original")
convert_color(img)
plt.subplot(1,2,2)
plt.title("Hough transform : Line detection")
convert_color(result)
plt.show()
```

Appendix 4 (Full code of (4) and (5))

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import warnings


def created_by_nopparuj():
    print("   __  ___     __    _           ")
    print("  /  |/  /__ _/ /__ (_)_ _  ___ _")
    print(" / /|_/ / _ `/  '_// /  ' \/ _ `/")
    print("/_/  /_/\_,_/_/\_\/_/_/_/_/\_,_/ ")
    print("\n<--- By Nopparuj J. 6213460 --->")
    print("\n❄️ This python program is able to perform both edge sharpening and
unsharp masking ❄️")

# Convert BGR to RGB for matplotlib functions
def convert_color(BGR):
    converted = cv2.cvtColor(BGR, cv2.COLOR_BGR2RGB)
    plt.imshow(converted)

img = cv2.resize(cv2.imread("/Users/nopparuj/ZSM-00/IPM-ninja/powercar.png",0),
(1000,1000))
warnings.filterwarnings('ignore')

# Required filters
# 1) Laplacian filter
laplacian = np.multiply(np.array([[0,1,0],[1,-4,1],[0,1,0]]), (1/8))
# 2) Gaussian filter
gaussian = np.multiply(np.array([[1,2,1],[2,4,2],[1,2,1]]), (1/16))

# Find outputsize after convolution with filters
def sizing(image,filters):
    out_size = (image.shape[0] - filters.shape[0]) + 1
    return out_size

# Convolution function: result in lesser pixels
def apply_filter(image,kernel):
    outputsize = sizing(image,kernel)
    img_filtered = np.zeros((outputsize,outputsize))
    for i in range (outputsize):
        for j in range (outputsize):
            pre_res = (kernel * image[i:i+kernel.shape[0],
j:j+kernel.shape[1]]).sum()
            img_filtered[i][j] = pre_res
    return img_filtered.astype(np.uint8)

# Edge sharpening function with edge padding
def edge_sharpening(input,kernel_filter):
    results = np.zeros((input.shape[0],input.shape[1]))
    post_laplace = apply_filter(input,kernel_filter)
```

```
    post_laplace_padded = np.multiply(np.pad(post_laplace, pad_width =1, mode
='edge'),1)
    for u in range (input.shape[0]):
        for v in range (input.shape[1]):
            # Use sharpening factor = 1
            sharped = input[u][v] - post_laplace_padded[u][v]
            if (sharped < 0):
                sharped = 0
            results[u][v] = sharped
    return results.astype(np.uint8)

# Unsharp masking function with edge padding
def unsharp_masking(input,kernel_filter):
    mask = np.zeros((input.shape[0],input.shape[1]))
    unsharped = np.zeros((input.shape[0],input.shape[1]))
    post_gaussian = apply_filter(input,kernel_filter)
    post_gaussian_padded = np.pad(post_gaussian, pad_width =1, mode ='edge')
    for q in range(input.shape[0]):
        for r in range(input.shape[1]):
            pre_mask = input[q][r] - post_gaussian_padded[q][r]
            # Use weight factor = 2 to enhance visibility of the edges
            mask[q][r] = np.multiply(pre_mask,2)

            final = input[q][r] + mask[q][r]
            if (final < 0):
                final = 0
            unsharped[q][r] = final
    return unsharped.astype(np.uint8)

# Comparing histogram
def compare_histogram(image_one,image_two):
    hist1 = cv2.calcHist([image_one], [0], None, [256], [0, 256])
    hist2 = cv2.calcHist([image_two], [0], None, [256], [0, 256])
    return hist1, hist2

def
plot_results(image_one,image_two,histogram1,histogram2,my_str1,my_str2,my_str3,my
_str4):
    plt.figure(1)
    plt.subplot(1,2,1)
    plt.title(my_str1.title())
    convert_color(image_one)
    plt.subplot(1,2,2)
    plt.title(my_str2.title())
    convert_color(image_two)

    plt.figure(2)
    plt.subplot(1,2,1)
    plt.title(my_str3.title())
    plt.plot(histogram1)
    plt.subplot(1,2,2)
```

```
    plt.title(my_str4.title())
    plt.plot(histogram2)

    plt.show()

while True:
    created_by_nopparuj()
    print(">> 1 << : for 'EDGE SHARPENING'")
    print(">> 2 << : for 'UNSHARP MASK'")
    case = int(input("\nSelect yout function >> "))

    if (case == 1):
        print("Applying Edge sharpening")
        edge_sharped = edge_sharpening(img,laplacian)
        before_sharp, after_sharp = compare_histogram(img,edge_sharped)
        plot_results(img,edge_sharped,before_sharp,after_sharp,"Original
image","Edge sharpen","Original histogram","Edge sharpen histogram")
        quit()

    elif (case == 2):
        print("Applying Unsharp masking")
        unsharp_mask = unsharp_masking(img,gaussian)
        before_mask, after_mask = compare_histogram(img,unsharp_mask)
        plot_results(img,unsharp_mask,before_mask,after_mask,"Original
image","Unsharp masked","Original histogram","Unsharp masked histogram")
        quit()

    else:
        break
```