Assignment 2

This assignment is choose one of Canny edge, Watershed algorithm, Hough transform and SLIC super pixel algorithm apply on 4 to 8 pictures. **Canny Edge** will be performed in my assignment.

The techniques will be used on this assignment:

1. Gaussian Filter:

To filter out noise and maintain the weight(original image) of the image, also can improve the Sobel filter performance. Gaussian filter use gaussian distribution to keep and distribute the original pixel information to others and filter noise at the same time.

2. Sobel Filter:

To filter out the edge of image by 2 direction Sobel filter. Gx = [[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]] as horizontal filter to detect the horizontal pixels is the same or not , if the horizontal pixels is the same, the sum of detected area will be 0. Otherwise, it will detected how different of those pixel, that is, to present the edge of image by intensity and also the gradient direction. For vertical pixels, Gy = [[-1, -2, -1], [0, 0, 0], [1, 2, 1]] will be used.

3. Canny detector

    1. Non-maximum suppression

    According to the gradient direction/angle calculated by Sobel Filter, a direction of intensity will be known, therefore, the width of edge will be selected to compare and keep the maximum one by compare that gradient direction intensity one by one.

    2. Connect Weak Edge by select correct threshold

    For the real case, we cannot easily distinguish noise by one threshold since there must be some noises or fake edges in the real image after filter, a low and high threshold were choose to reduce most of noises and fake edges, also keep the real significant edge. However, the pixels between high and low threshold will be kept if there are two higher threshold pixels aside the pixel need to be determined, Or it will be count as noises or fake edges.

Try to improve canny edge method:

1. Histogram Equalization:

To count the number of pixel in the image and accumulate it before multiplication of 255 to let image distribute much more better.

2. Add 2 more direction of Sobel filter.

3. Use larger gaussian filter.

4. Using Sharpen filter instead of Sobel filter.

5. Normalization after Sobel Calculation

Part 1 Gaussian Filter:

To keep the weight of original pixel and reduce the noise, also color image is not good for edge detection, not easy to read, may confuse by observation and may generate some noises, that is, I use gray image as original image.

Figure 1-1 original image

Figure 1-2 image after gaussian filter

In here, I selected 3x3 gaussian filter by below equation where σ = square root of 0.5 for easy implement.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

We can obtain the picture become blur and the pixel value will much more average as original one.

Part 2 Sobel Filter:

To filter out the edge and present by horizontal and vertical edge, then combine it as image edge detection.

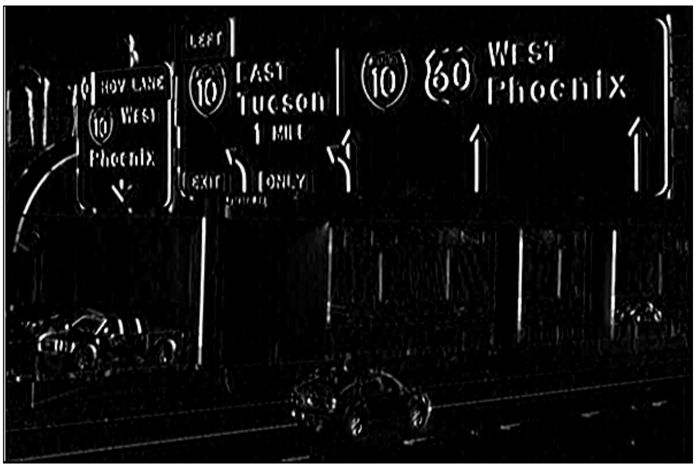

Figure 2-1 Image after Sobel Filter in Horizontal Direction

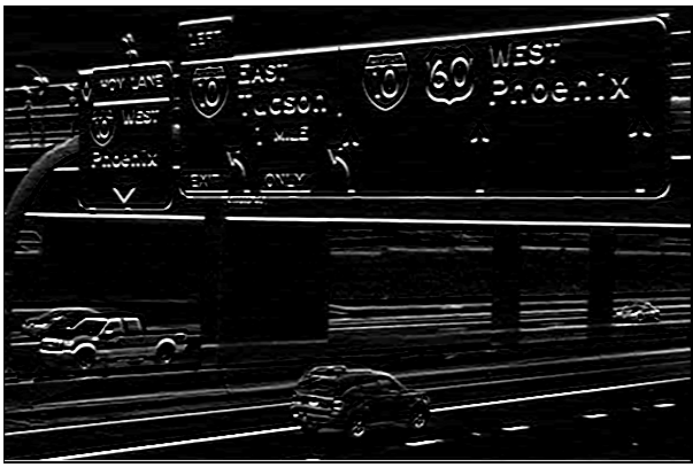

Figure 2-2 Image after Sobel Filter in vertical Direction

As we can see, Sobel filter Gx is actually a vertical edge detector (Figure2-1) and Gy is horizontal edge detector(Figure2-2). After careful consideration of Sobel filter Gx and Gy from the beginning introduction, you will understand it. For more explanation, horizontal filter will compare left and right pixel, that is, it will produce a result of vertical edge if left and right pixels are not the same. And finally, we will have the image edge by combined both direction edge together as Figure 2-4.

Figure 2-3 Original gray image

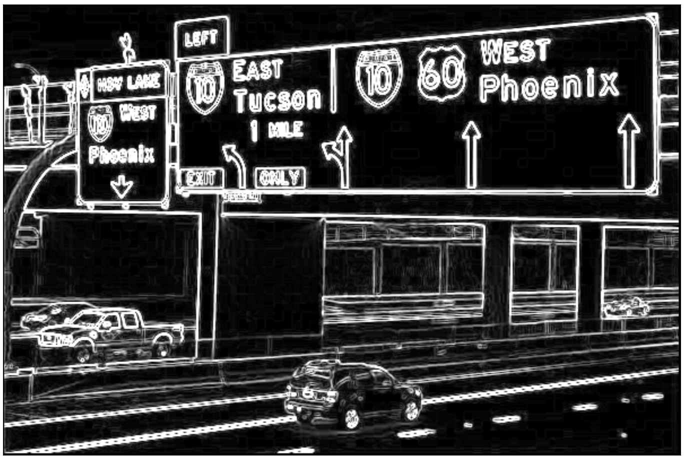

Figure 2-4 Combination of Sobel filter in both horizontal and vertical direction

Actually, I believe this edge detection is good enough for drawing the edge of input image, but there may have application need to have more precise edge such as medical application - tumor detection. As a result we need to reduce the edge to have more accurate edge.

Part 3 Canny detection:

Part 3-1:
Non-maximum suppression:
To compare with adjacent pixels and select the maximum intensity by the intensity (G) and direction(θ) calculated by Sobel filter as shown as below. We can obviously obtain the different between Sobel edge detector and Figure 3-1, it is significantly thinner.

$$G = \sqrt{G_x^2 + G_y^2} \qquad \theta = arc\tan\left(G_y / G_x\right)$$

Part 3-2:
Connect weak edge:
Set a high and low thresholds to maintain the real edge and drop the noise. For the real edge, set it to 255. For the noise, set it to be 0. In the meantime, check pixels between high and low threshold are around the real edge pixel, if so, connect it with real edge pixel, otherwise, drop it.

From the observation, we can found that the edge of image is much more easier to view, especially for the vehicle in the middle bottom. The original curve of vehicle is almost disappear, but it reappear again after connect weak edge. Somehow, the weak edge connectivity is not good enough.

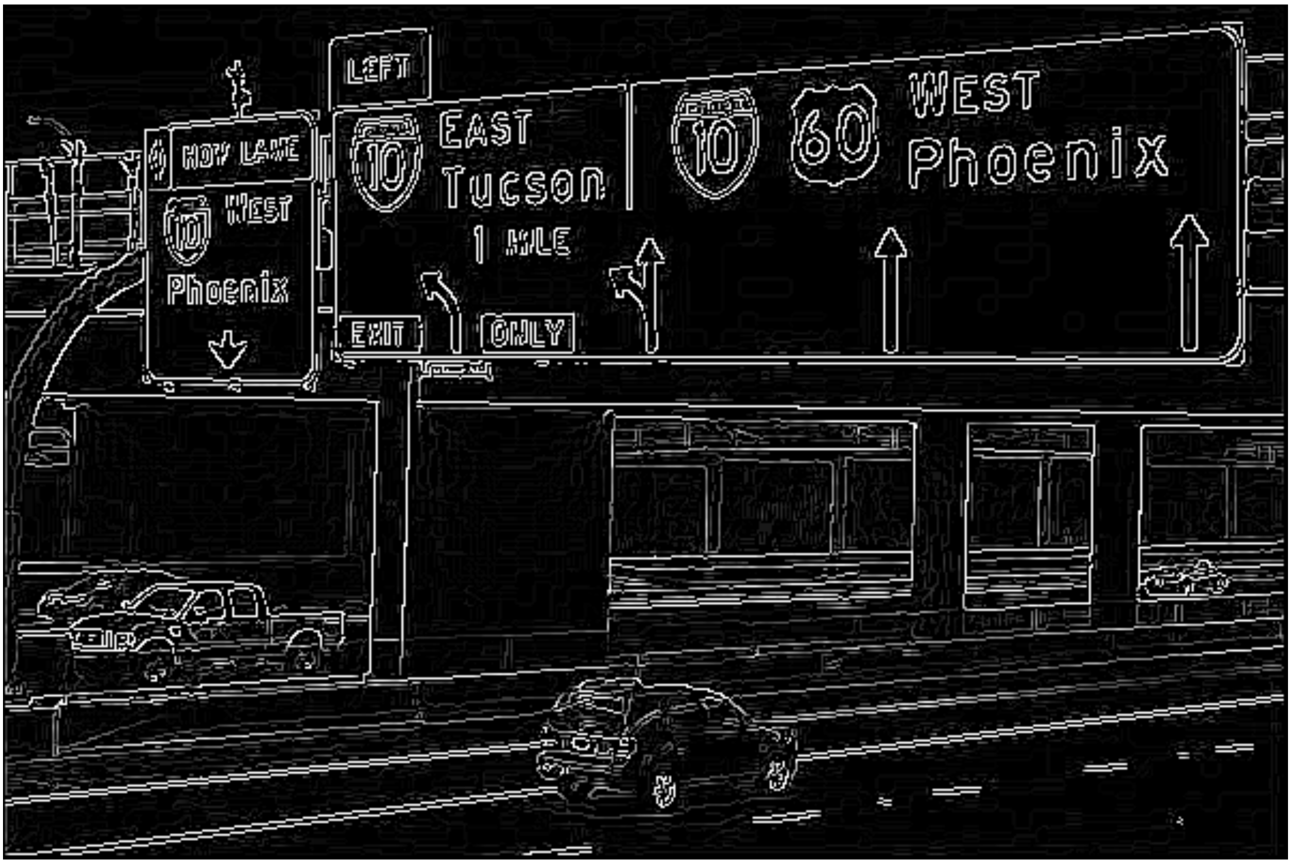

Figure 3-1 After Non-max suppression to minimize the edge

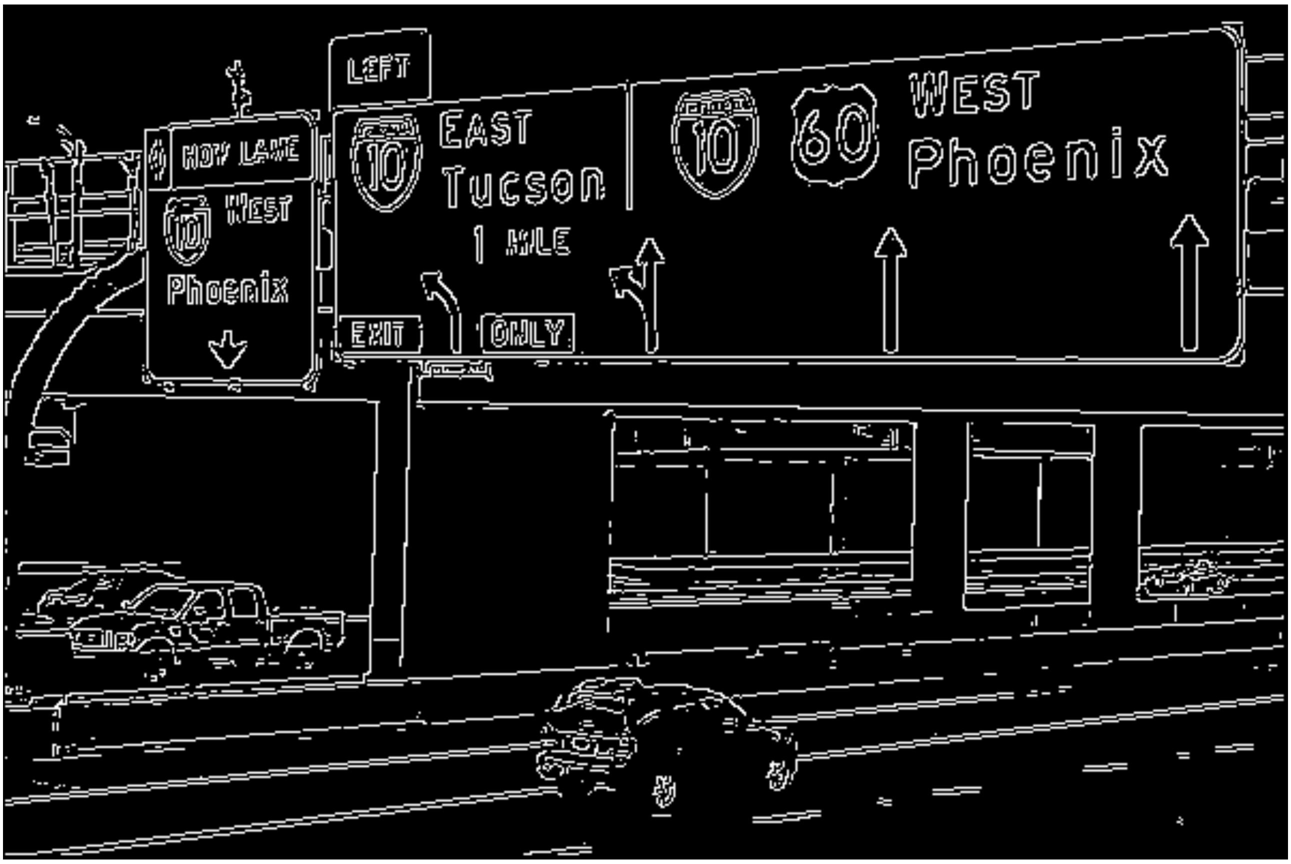

Figure 3-2 After weak edge detection to maximize the edge and drop the noise

Part 4 Comparison with canny library of internet:

Figure 4-1 Canny Detection from Internet
(High threshold = 130, Low threshold = 60)

Figure 4-2 Canny Detection of my coding
(High threshold = 130, Low threshold = 60)

From the figure 4-1, I found that for the same threshold, my image generate more noise and edges detect also not shown completely after process. May be it is because of image pre-process and weak detection.

Improvement Trials

Trial 1: (Not work!)

Pre-process image before Sobel edge detection by Histogram Equalization(HE) and compare with original one at Canny Edge detection stage. The detail of HE design can review to my HW1.

Figure 4-3 Original Sobel filter edge detector

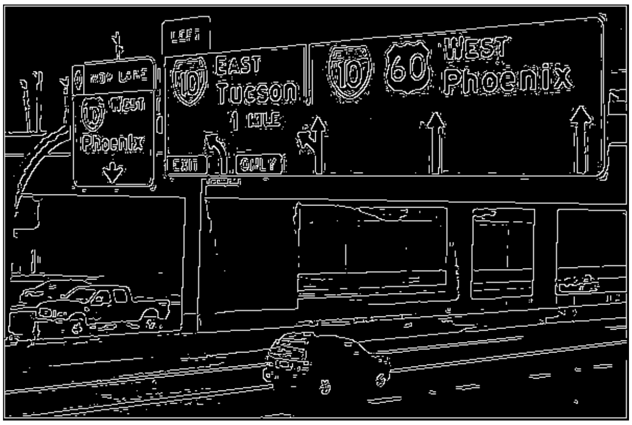

Figure 4-4 Pre-process original by Histogram Equalization

Obviously, Histogram Equalization is not benefit to edge detection, the result even become worst than previous one since some of edge missing and generate more noise.

Trial 2: (Not work!)

Add detected direction of Sobel filter see if we can improve the Sobel edge detection.

As I added 45 degree and 135 degree Sobel filter, the result seems not improvement. Some of edges even become lighter.

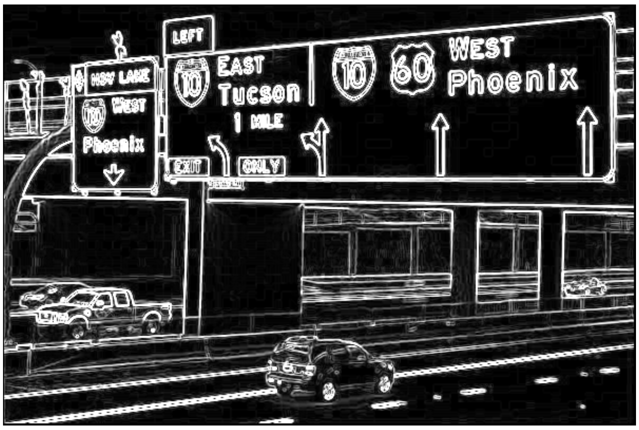

Figure 4-5 Original Sobel filter edge detector

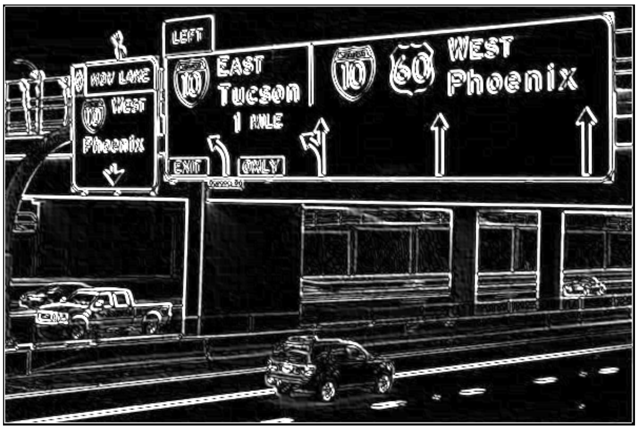

Figure 4-6 Using another 2 angle Sobel edge detection

Trial 3: (Not work!)

Use larger size of Gaussian filter.

From the observation, it is difficult to distinguish different between two images.

Figure 4-7 Original Sobel filter edge detector with 3x3 Gaussian filter

Figure 4-8 Sobel filter edge detector with 7x7 Gaussian filter

Trial 4: (Not work!)

Use sharpen filter to detect more edges before doing canny detection.

Figure 4-9 Original Sobel image

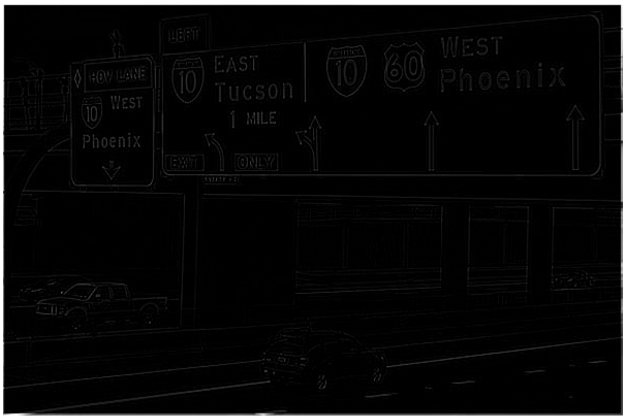

Figure 4-10 Sharpen filter edge detection

As a result, Sharpen filter also cannot detect more edge from image and seems worst than Sobel.

Trial 5: (seems work?)

Normalization after Sobel filter calculate magnitude. As the magnitude of Sobel calculation is square root of (Gx^2 + Gy^2), it may exceed possible value of image(255), so I normalized it as divided all pixels by maximum pixel value and multiple 255. Thus, a result shown as below.

Figure 4-11 Comparison between Internet Canny Edge library with threshold 130 and 60 vs my normalized calculation after Sobel with threshold 20 and 5

I am not sure this normalization improve the edge detection of my original although it produces less noise than previous one, but more edge on the image. However, it change the among of threshold values for canny detection, maybe it is because of threshold selection or it reduces noise.

Part 5 Application of different image:

Figure 5-1 Reference Canny detection using High threshold 230 and Low threshold 100

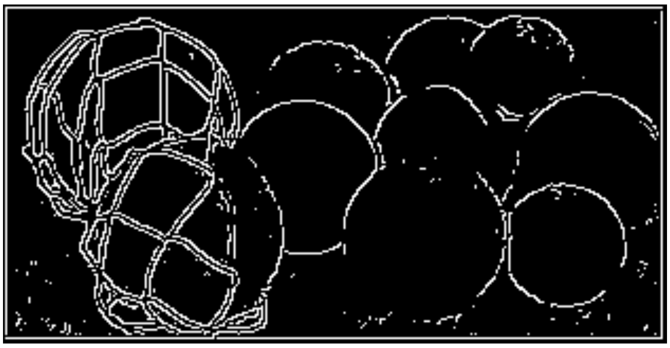

Figure 5-2 My Canny detection using High threshold 120 and Low threshold 45.

Figure 5-3 Reference Canny detection using High threshold 130 and Low threshold 80

Figure 5-4 My Canny detection using High threshold 130 and Low threshold 80.

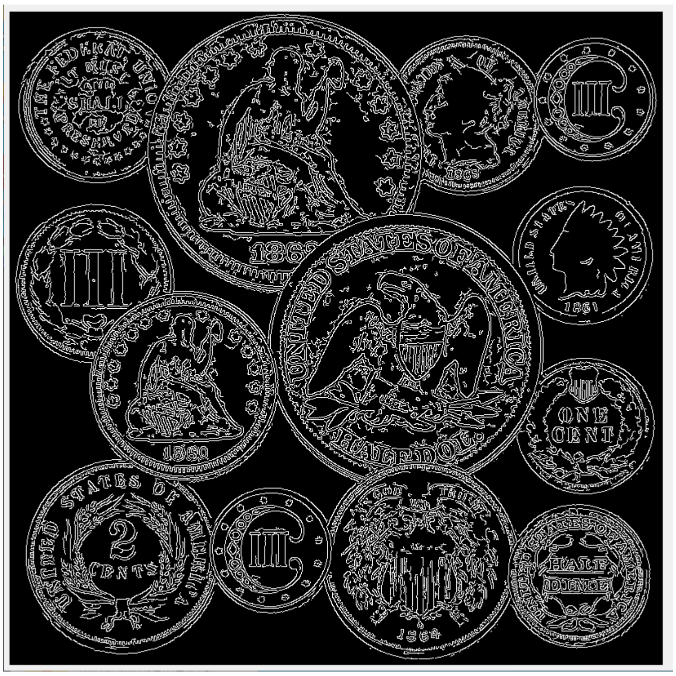

Figure 5-5 Reference Canny detection using High threshold 160 and Low threshold 70

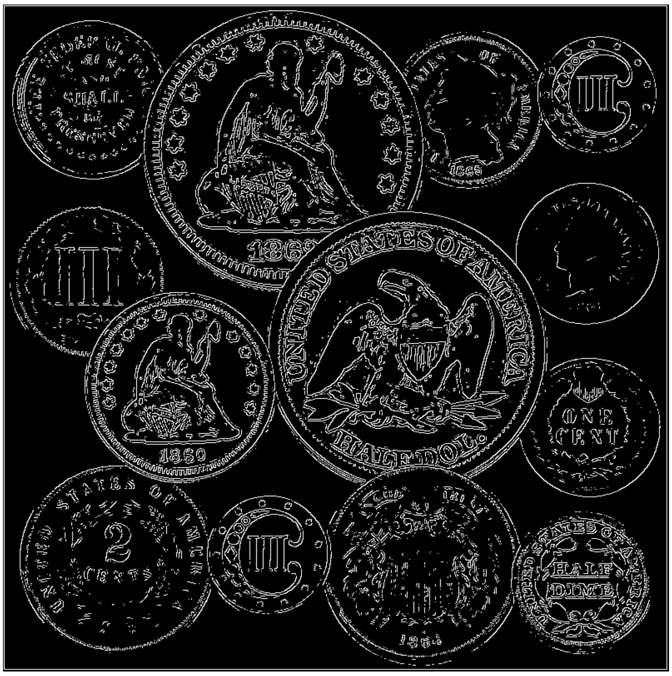

Figure 5-6 My Canny detection using High threshold 160 and Low threshold 70.

2nd testing picture(balls) was shown as below Figure 5-1 and 5-2 by using reference canny detection compare with my canny detection.
3rd testing picture(basketball) was shown as below Figure 5-3 and 5-4 by using reference canny detection compare with my canny detection.
4th testing picture(coins) was shown as below Figure 5-5 and 5-6 by using reference canny detection compare with my canny detection.

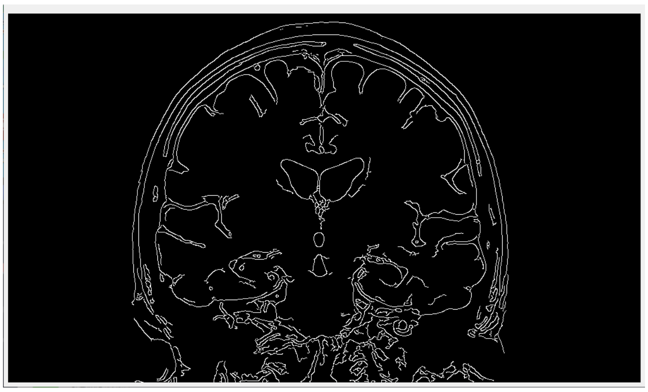

Figure 5-7 Reference Canny detection using High threshold 160 and Low threshold 60

Figure 5-8 My Canny detection using High threshold 160 and Low threshold 60.

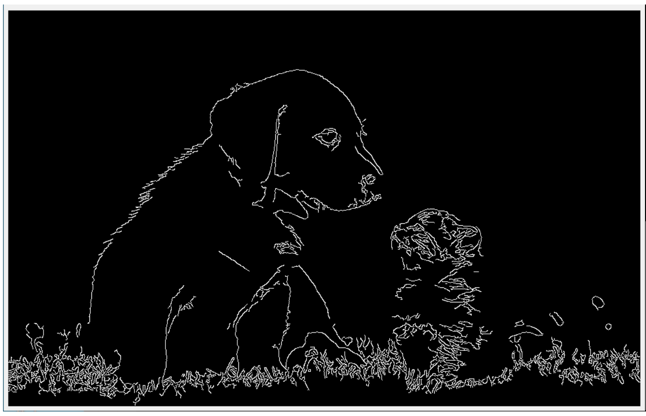

Figure 5-9 Reference Canny detection using High threshold 180 and Low threshold 70

Figure 5-10 My Canny detection using High threshold 180 and Low threshold 70.

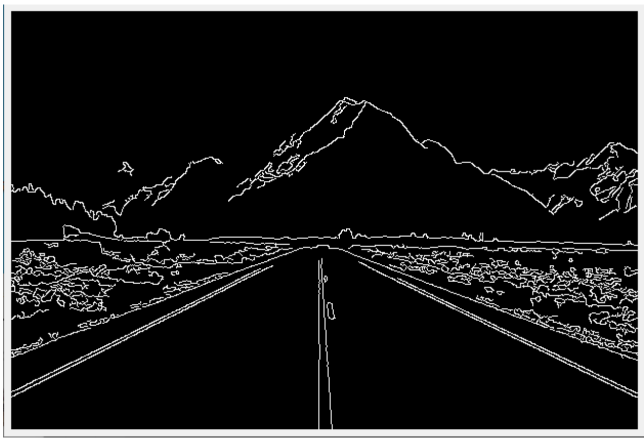

Figure 5-11 Reference Canny detection using High threshold 210 and Low threshold 70

Figure 5-12 My Canny detection using High threshold 130 and Low threshold 50.

5th testing picture(MRI) was shown as below Figure 5-7 and 5-8 by using reference canny detection compare with my canny detection.

6th testing picture(pets) was shown as below Figure 5-9 and 5-10 by using reference canny detection compare with my canny detection.

7th testing picture(scene) was shown as below Figure 5-11 and 5-12 by using reference canny detection compare with my canny detection.

8th testing picture(CT) was shown as below Figure 5-13 and 5-14 by using reference canny detection compare with my canny detection.

Figure 5-13 Reference Canny detection using High threshold 210 and Low threshold 70

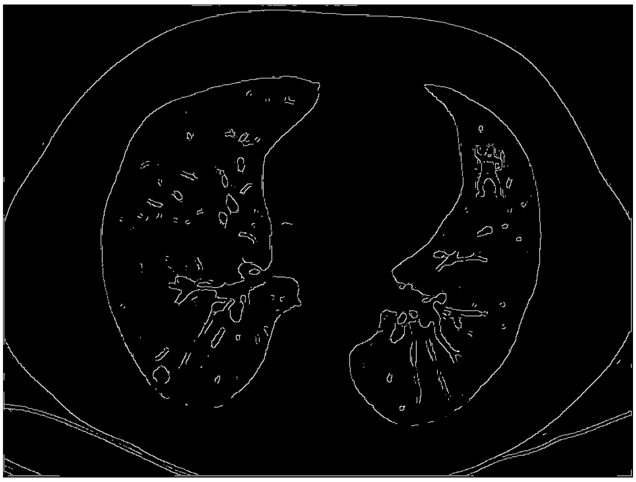

Figure 5-14 My Canny detection using High threshold 130 and Low threshold 40.

<u>Part 6 Conclusion:</u>

This assignment let me more understand canny edge detection operation, it can be adaptable to various environment. However, my canny edge detector is not as good as website library reference. By the method and observation, I believe it is because of pre-processing related problems - noise reduction. In advance, my weak edge detection seems not good enough.

As the reference canny detection can have much more edge. I believe it is related to weak edge detection method, I think my weak edge detection is not the best and remain many improvement. For example, if there are several weak edge pixels connected together and one of them connected to real edge, my weak edge detection cannot consider it as real edge, but actually, it can be a real edge.

In my trials, I failed to improve my image processing program, but actually, I found that normalization during Sobel calculation may improve edge detection, it may be related to threshold selection, but also may related to noise reduction after my consideration.

As a result, if I would like to have better result, maybe I should try below direction:
1. To enlarge the kernel of weak edge detection and improve the detection method.
2. To find a better way to determine a better threshold.
3. Image pre-processing as noise reduction or better edge detection rather than Sobel.

At last, I will briefly explain my coding. First, I create a GUI for easy implementation, and I use Library code for comparison, also I had included my trial code in below like Histogram Equalization, Sharpen filter, etc.
I try to name the function more easier to read and add some explanation, please let me know if you curious or you have any questions about this assignment.import itertools

```python
import math
import tkinter as tk
from PIL import Image, ImageTk
import math as m
import numpy as np
import copy
import cv2
from tkinter import filedialog


# create a layout
def define_layout(obj, cols=1, rows=1):
    def method(trg, col, row):

        for c in range(cols):
            trg.columnconfigure(c, weight=1)
        for r in range(rows):
            trg.rowconfigure(r, weight=1)

    if type(obj) == list:
        [method(trg, cols, rows) for trg in obj]
    else:
        trg = obj
        method(trg, cols, rows)


# open file
def openfn():
    filename = filedialog.askopenfilename(title='open')
    return filename


# open image
def open_img_left():
```

```
    x = openfn()
    img = Image.open(x)
    img = img.resize((img_before_size[1], img_before_size[0]), Image.ANTIALIAS)
    img = ImageTk.PhotoImage(img)
    panel = tk.Label(div2, image=img)
    panel.image = img
    panel.grid(column=1, row=1, sticky=align_mode)


def open_img_right():
    x = openfn()
    img = Image.open(x)
    img = img.resize((img_after_size[1], img_after_size[0]), Image.ANTIALIAS)
    img = ImageTk.PhotoImage(img)
    panel = tk.Label(div3, image=img)
    panel.image = img
    panel.grid(column=1, row=1, sticky=align_mode)


# test input scale function including gaussian size and thresholds
def s_print(text):  # print scale variable
    print('Gaussian Filter:', value1.get(), 'High threshold:', value3.get(), 'Low thresh-
old:', value4.get())


# create a window
window = tk.Tk()
window.title('Canny Edge')
align_mode = 'nswe'
pad = 5

# create a scale bar value for initial
value1 = tk.IntVar()
value2 = tk.IntVar()
value3 = tk.IntVar()
value4 = tk.IntVar()

# input image path
```

```
read_image_before = './pic4PR2/CT.jpg'  # read image you would like to process, can be changed
write_image_before = "CT_gray.png"  # write image you would like to process, can be changed
read_image_after = './CT_gray.png'  # preview gray image you would like to process, can be changed

# read image
image_before = cv2.imread(read_image_before)
img_before_gray = cv2.cvtColor(image_before, cv2.COLOR_RGB2GRAY)
arr = np.array(img_before_gray)
img_before_size = arr.shape
cv2.imwrite(write_image_before, img_before_gray)

image_after = copy.deepcopy(image_before)
img_after_temp = copy.deepcopy(image_before)
img_after_size = img_before_size

# grid the testing window layout
div_size = 200
div1 = tk.Frame(window, width=img_before_size[1], height=div_size, bg='blue')  # bg='blue'
div4 = tk.Frame(window, width=img_before_size[1], height=div_size)  # bg='yellow'
div2 = tk.Frame(window, width=img_before_size[1], height=img_before_size[0], bg='orange')
div3 = tk.Frame(window, width=img_after_size[1], height=img_after_size[0], bg='green')

window.update()
win_size = min(window.winfo_width(), window.winfo_height())
# print(win_size)

div1.grid(column=0, row=0, padx=pad, pady=pad, sticky=align_mode)
div4.grid(column=1, row=0, padx=pad, pady=pad, sticky=align_mode)
div2.grid(column=0, row=1, padx=pad, pady=pad, rowspan=2, sticky=align_mode)
div3.grid(column=1, row=1, padx=pad, pady=pad, rowspan=2, sticky=align_mode)

define_layout(window, cols=2, rows=2)
define_layout([div1, div2, div3, div4])

im = Image.open(write_image_before)  # read original image
imTK_L = ImageTk.PhotoImage(im.resize((img_before_size[1], img_before_size[0])))
```

```
image_main = tk.Label(div2, image=imTK_L)
image_main['height'] = img_before_size[0]
image_main['width'] = img_before_size[1]

image_main.grid(column=0, row=1, sticky=align_mode)
image_main.grid(column=1, row=1, sticky=align_mode)

define_layout(window, cols=3, rows=3)
define_layout(div1)
define_layout(div4)
define_layout(div2, rows=2)
define_layout(div3, rows=2)

# read image in right hand side
im = Image.open(read_image_after)
imTK_R = ImageTk.PhotoImage(im.resize((img_after_size[1], img_after_size[0])))

image_main = tk.Label(div3, image=imTK_R)
image_main['height'] = img_after_size[0]
image_main['width'] = img_after_size[1]
image_main.grid(column=0, row=1, sticky=align_mode)
image_main.grid(column=1, row=1, sticky=align_mode)


# filter design
def initial_filter(fNum):
    filter_size = []
    for i in range(fNum):
        filter_size.append([0.0] * fNum)
    fil = np.array(filter_size)
    return fil


# create a gaussian filter
def gaussian_filter_creator():
    sum_of_filter = 0
    gaussian_size = int(value1.get())  # gaussain filter size(3) can be changed to val-
ue1.get()
```

```
    matrix_var = int(gaussian_size / 2)
    initial_gaussian_filter = initial_filter(gaussian_size)
    for i, j in itertools.product(range(gaussian_size), range(gaussian_size)):
        initial_gaussian_filter[i][j] = m.exp(-((-matrix_var + i) ** 2 + (-matrix_var + j) ** 2))
        sum_of_filter = initial_gaussian_filter[i][j] + sum_of_filter
    normalize_gaussian_filter = initial_gaussian_filter / sum_of_filter
    print('done Gaussian')
    return normalize_gaussian_filter


def conv_full_image(img, fil):
    temp = []
    img_temp = copy.deepcopy(img)
    img_new_c = np.zeros(img.shape[0] * img.shape[1])
    img_new_c = img_new_c.reshape(img.shape[0], img.shape[1])
    img_size = img.shape
    fil_size = fil.shape
    for i in range(img_size[0]):
        for j in range(img_size[1]):
            img_new_c = conv(img_new_c, img_temp, img_size, fil, fil_size, temp, i, j)

    return img_new_c


def conv(img_new_c, img_temp_c, img_size_c, fil, fil_size, temp, i, j):
    if fil_size[0] % 2 != 0:
        for ii in range(fil_size[0]):
            if i + fil_size[0] < img_size_c[0]:
                for jj in range(fil_size[1]):
                    if j + fil_size[1] < img_size_c[1]:
                        temp.append(img_temp_c[i + ii][j + jj] * fil[ii][jj])
                    else:
                        break
            else:
                break

        temp_sum = np.sum(temp)
```

```
        cen_pointx = int(fil_size[0] / 2)
        cen_pointy = int(fil_size[1] / 2)
        num = int(temp_sum)
        if (i + fil_size[0] < img_size_c[0]) and (j + fil_size[1] < img_size_c[1]):
            img_new_c[i + cen_pointx][j + cen_pointy] = num
        temp.clear()
    return img_new_c


def sobel_filter(phase):
    if phase > 112.5:  # 135 degree
        sobel_fil = [[2, 1, 0], [1, 0, -1], [0, -2, -1]]
    elif phase > 67.5:  # 90 degree
        sobel_fil = [[1, 2, 1], [0, 0, 0], [-1, -2, -1]]
    elif phase > 22.5:  # 45 degree
        sobel_fil = [[0, 1, 2], [-1, 0, 1], [-2, -1, 0]]
    else:  # 0 degree
        sobel_fil = [[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]]

    sobel_fil = np.array(sobel_fil)
    return sobel_fil


def sobel_calculation(image_x, image_y):
    image_combine = np.zeros(image_x.shape[0] * image_x.shape[1])
    image_combine = image_combine.reshape(image_x.shape[0], image_x.shape[1])
    theta = np.zeros(image_x.shape[0] * image_x.shape[1])
    theta = theta.reshape(image_x.shape[0], image_x.shape[1])
    for i, j in itertools.product(range(image_x.shape[0]), range(image_x.shape[1])):
        theta[i][j] = math.atan2(image_y[i][j], image_x[i][j]) * 180 / math.pi
        theta[i][j] = theta[i][j] % 180
        if theta[i][j] > 112.5:
            theta[i][j] = 135
        elif theta[i][j] > 67.5:
            theta[i][j] = 90
        elif theta[i][j] > 22.5:
            theta[i][j] = 45
        else:
```

```
            theta[i][j] = 0
        image_combine[i][j] = int(math.sqrt(image_x[i][j] ** 2 + image_y[i][j] ** 2))
    # for i, j in itertools.product(range(image_x.shape[0]), range(image_x.shape[1])):
    #     image_combine[i][j] = int(image_combine[i][j] / np.max(image_combine) * 255)   # normalize
    print('np.max(image):', np.max(image_combine))
    print('done sobel_calculation')
    return image_combine, theta


def smooth_filter_creator():
    average_of_filter = value1.get() * value1.get()
    smooth_size = int(value1.get())  # gaussain filter size(3) can be changed to value1.get()
    matrix_var = int(smooth_size / 2)
    initial_smooth_filter = initial_filter(smooth_size)
    final_smooth_filter = initial_filter(smooth_size)
    for i, j in itertools.product(range(smooth_size), range(smooth_size)):
        final_smooth_filter[i][j] = (initial_smooth_filter[i][j] + 1) / average_of_filter
    print('done Smooth')
    return final_smooth_filter


def sharpen_filter(source_image):
    smooth_filter = smooth_filter_creator()
    smooth_filter_size = smooth_filter.shape
    print('smooth_filter:', smooth_filter_size, 'sum:', np.sum(smooth_filter))
    img_after_smooth = conv_full_image(source_image, smooth_filter)
    image_after_sharpen = np.zeros(source_image.shape[0] * source_image.shape[1])
    image_after_sharpen = image_after_sharpen.reshape(source_image.shape[0], source_image.shape[1])
    for i in range(source_image.shape[0]):
        for j in range(source_image.shape[1]):
            image_after_sharpen[i][j] = int(source_image[i][j]) - int(img_after_smooth[i][j])
            if image_after_sharpen[i][j] < 0:  # confirm the value will not less than zero
                image_after_sharpen[i][j] = 0

    return image_after_sharpen
```

```
def canny_edge_detector(image_source, canny_theta):
    canny_image = copy.deepcopy(image_source)
    for i, j in itertools.product(range(canny_theta.shape[0]),
range(canny_theta.shape[1])):
        if (i + 1) < canny_image.shape[0] and (j + 1) < canny_image.shape[1]:
            if canny_theta[i][j] > 112.5:  # 135 degree, compare two dimension value with
core one
                if image_source[i][j] < image_source[i - 1][j - 1] or \
                        image_source[i][j] < image_source[i + 1][j + 1]:
                    canny_image[i][j] = 0
            elif canny_theta[i][j] > 67.5:  # 90 degree
                if image_source[i][j] < image_source[i - 1][j] or \
                        image_source[i][j] < image_source[i + 1][j]:
                    canny_image[i][j] = 0
```

```
            elif canny_theta[i][j] > 22.5:  # 45 degree
                if image_source[i][j] < image_source[i + 1][j - 1] or \
                        image_source[i][j] < image_source[i - 1][j + 1]:
                    canny_image[i][j] = 0
```

```
            else:  # 0 degree
                if image_source[i][j] < image_source[i - 1][j - 1] or \
                        image_source[i][j] < image_source[i + 1][j + 1]:
                    canny_image[i][j] = 0
    print('done canny_edge_detector')
    return canny_image
```

```
def canny_edge_threshold(source_image, high_threshold, low_threshold, filter_in):
    canny_image = np.zeros(source_image.shape[0] * source_image.shape[1])
    canny_image = canny_image.reshape(source_image.shape[0], source_image.shape[1])
    print('high_threshold:', high_threshold, 'low_threshold', low_threshold)
    count = 0
    for i, j in itertools.product(range(source_image.shape[0]),
range(source_image.shape[1])):
        if source_image[i][j] >= high_threshold:
            canny_image[i][j] = 255
        elif source_image[i][j] <= low_threshold:
```

```
            canny_image[i][j] = 0
        else:
            for ii, jj in itertools.product(range(-1), range(2)):
                if (0 < (i + ii) < source_image.shape[0]) and (0 < (j + jj) < source_image.shape[1]):
                    if source_image[i + ii][j + jj] >= high_threshold:
                        count += 1
                        if count >= 2:  # if 2 real edge aside
                            canny_image[i][j] = 255
                            count = 0
                            break
                    else:
                        canny_image[i][j] = 0
    print('done canny_edge_threshold')
    return canny_image


# Histogram Equalization
def CountPixel(original_image, ArrayG, imageH, imageW):
    for i in range(imageH - 1):  # use i as index of image to determine which pixels need to be filtered
        for j in range(imageW - 1):  # use j as index of image to determine which pixels need to be filtered
            ArrayG[original_image[i][j]] += 1  # count the corresponding pixel number in the image from 0 to 255
    return ArrayG


def Accumulate(Array):
    AccumArray = [0] * 256
    j = 0
    for i in range(256):
        AccumArray[i] = Array[i] + j  # accumulate the appear pixel
        j = AccumArray[i]
    return AccumArray


def ImageHeq(original_image, ACPAG, imageH, imageW):
    img_tmp = copy.deepcopy(original_image)
    for i in range(imageH):
```

```
        for j in range(imageW):
            img_tmp[i][j] = ACPAG[img_tmp[i][j]] * 255  # do the histogram equalization

    return img_tmp


def Histogram_Equalization(source_image):
    image_temp = copy.deepcopy(source_image)
    IGray = [0] * 256  # initial the RGB value for counting the corresponding number of 
image
    AcPAG = [0] * 256

    CountPixel(image_temp, IGray, source_image.shape[0], source_image.shape[1])
    EveryPAG = np.array(IGray) / (source_image.shape[0] * source_image.shape[1])
    AcPAG = Accumulate(EveryPAG)  # accumulate the number of appear pixel for each chan-
nel

    image_temp = ImageHeq(image_temp, AcPAG, source_image.shape[0],
                          source_image.shape[1])  # do the histogram equalization
    print('Histogram Done!')
    return image_temp


# main body
def main():

    gaussian_filter = gaussian_filter_creator()
    gaussian_filter_size = gaussian_filter.shape
    print('gaussian_filter_size:', gaussian_filter_size, 'sum', np.sum(gaussian_filter))

    img_after_temp = conv_full_image(img_before_gray, gaussian_filter)
    cv2.imwrite("img_after_gaussian.png", img_after_temp)

    # img_after_hq = Histogram_Equalization(img_after_temp.astype(int))
    # cv2.imwrite("img_after_hq.png", img_after_hq)

    img_after_sobel_x = conv_full_image(img_after_temp, sobel_filter(0))
    img_after_sobel_y = conv_full_image(img_after_temp, sobel_filter(90))
    img_after_sobel_combine_x_y, img_after_sobel_combine_theta = 
sobel_calculation(img_after_sobel_x, img_after_sobel_y)
```

```
    cv2.imwrite("img_after_sobel_x.png", img_after_sobel_x)
    cv2.imwrite("img_after_sobel_y.png", img_after_sobel_y)
    cv2.imwrite("img_after_sobel_combine_x_y.png", img_after_sobel_combine_x_y)

    img_after_sharpen = sharpen_filter(img_before_gray)  # method 2 to get edge
    cv2.imwrite("img_after_sharpen.png", img_after_sharpen)

    # img_after_sobel_yx = conv_full_image(img_after_temp, sobel_filter(45))
    # img_after_sobel_xy = conv_full_image(img_after_temp, sobel_filter(135))
    # img_after_sobel_combine_yx_xy, img_after_sobel_combine_theta = 
sobel_calculation(img_after_sobel_yx, img_after_sobel_yx)
    # cv2.imwrite("img_after_sobel_yx.png", img_after_sobel_yx)
    # cv2.imwrite("img_after_sobel_xy.png", img_after_sobel_xy)
    # cv2.imwrite("img_after_sobel_combine_yx_xy.png", img_after_sobel_combine_yx_xy)

    # img_after_sobel_combine_final = np.zeros(img_after_temp.shape[0] * img_after_tem-
p.shape[1])
    # img_after_sobel_combine_final = 
img_after_sobel_combine_final.reshape(img_after_temp.shape[0] * img_after_temp.shape[1])
    # img_after_sobel_combine_final = sobel_calculation(img_after_sobel_combine_yx_xy, 
img_after_sobel_combine_x_y)
    # cv2.imwrite("img_after_sobel_combine_final.png", img_after_sobel_combine_final)

    img_after_canny = canny_edge_detector(img_after_sobel_combine_x_y, img_after_sobel_-
combine_theta)
    cv2.imwrite("img_after_canny.png", img_after_canny)

    img_after_canny_threshold = canny_edge_threshold(img_after_canny, value3.get(), val-
ue4.get(), gaussian_filter)
    cv2.imwrite("img_after_canny_threshold.png", img_after_canny_threshold)

    print('done, please open image')


def library_canny():
    image_source = cv2.imread(read_image_before)
    gray = cv2.cvtColor(image_source, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (value1.get(), value1.get()), 0)
    img_after_library_canny = cv2.Canny(blurred, value3.get(), value4.get())
    cv2.imwrite("img_after_library_canny.png", img_after_library_canny)
```

```
    print('done library canny')


# scale bar function
lbl_title1 = tk.Scale(div1, label='Gaussian Filter', from_=3, to=9, orient=tk.HORIZONTAL,
                      resolution=1, variable=value1, command=s_print)

lbl_title3 = tk.Scale(div1, label='High threshold', from_=0, to=255, orient=tk.HORIZON-
TAL,
                      resolution=5, variable=value3, command=s_print)

lbl_title4 = tk.Scale(div1, label='Low threshold', from_=0, to=250, orient=tk.HORIZONTAL,
                      resolution=5, variable=value4, command=s_print)

lbl_title1.grid(column=0, row=1, sticky=align_mode)

lbl_title3.grid(column=0, row=3, sticky=align_mode)
lbl_title4.grid(column=0, row=4, sticky=align_mode)

# create button
btn1 = tk.Button(div4, text='Run', command=main).grid(column=0, row=1, sticky=align_mode)
btn2 = tk.Button(div4, text='Run library_canny', command=library_canny).grid(column=0,
row=2, sticky=align_mode)
btn3 = tk.Button(div4, text='open before image', command=open_img_left).grid(column=0,
row=3, sticky=align_mode)
btn4 = tk.Button(div4, text='open after image', command=open_img_right).grid(column=0,
row=4, sticky=align_mode)

print('initial')

window.mainloop()
```