Image Processing - Programming Assignment #2

310553056 陳威齊

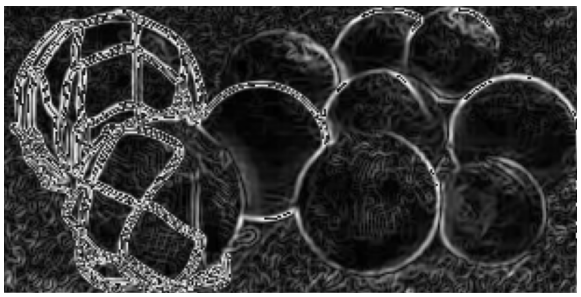
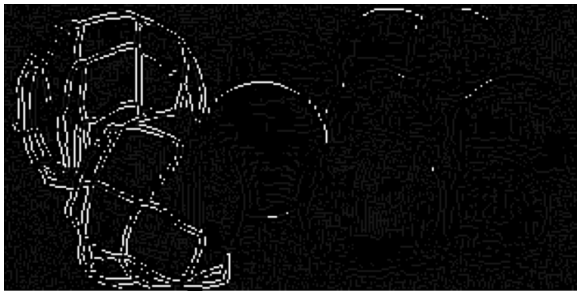
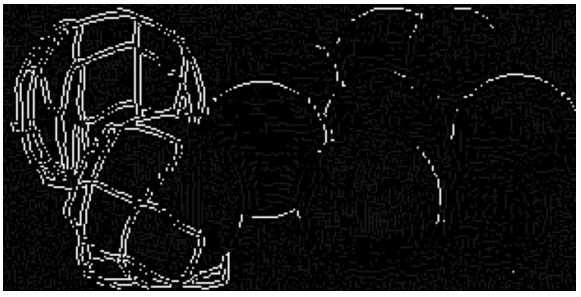
Introduction / Objectives

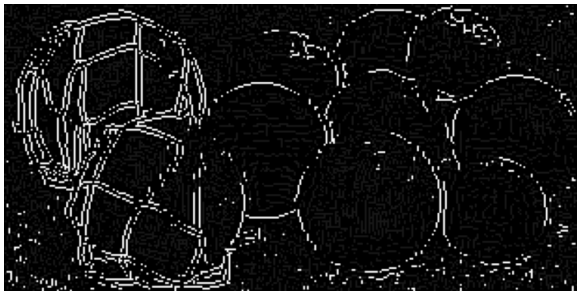
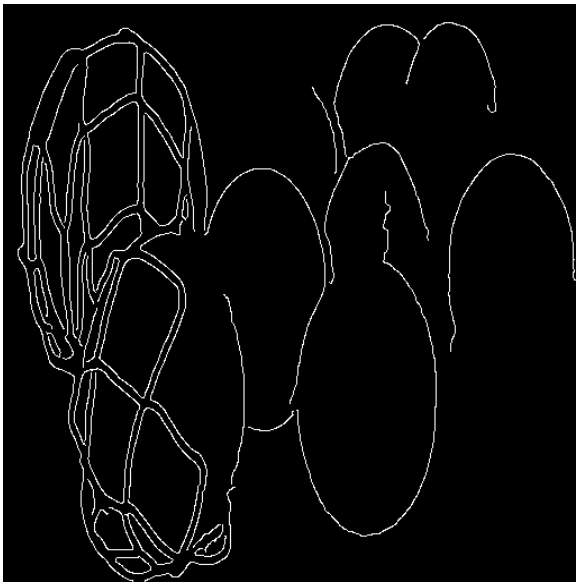
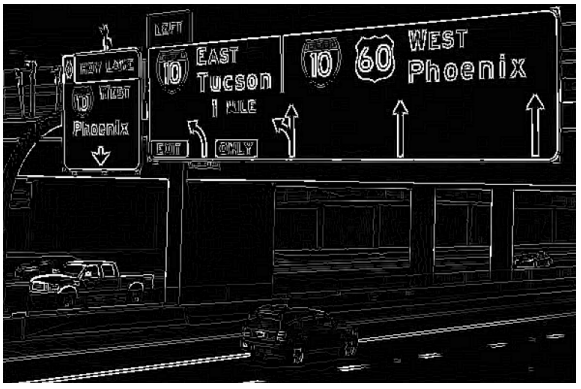
This project is to experiment with the image segmentation technique, Canny edge detector, mentioned in the class.

Methods

- Program Language : Python3.8
- Image Processing :
    - Intensity Transform (from hw01)
    - Noise Filter (from hw01)
    - Edge Detection (Canny)

Result

1. balls	
	 (1) = GRAYSCALE + Gaussian
 (2) = (1) + Sobel	 (3) = (2) + Non-maxima suppression
 (4) = (3) + Thresholding	 (5) = (3) + Thresholding

(low=0.05, high=0.3)	(low=0.05, high=0.2)
 (6) = (3) + Thresholding (low=0.05, high=0.1)	 (7) result from web demo
2. signs	
	 (1) = GRAYSCALE + Gaussian
 (2) = (1) + Sobel + ConstrastAdjust	 (3) = (2) + Non-maxima suppression

(4) = (3) + Thresholding
(low=0.05, high=0.3)

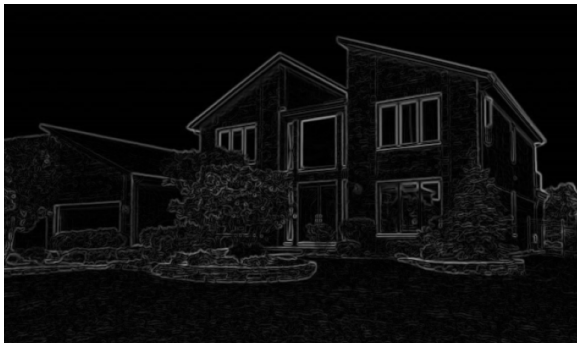
(5) = (3) + Thresholding
(low=0.05, high=0.2)

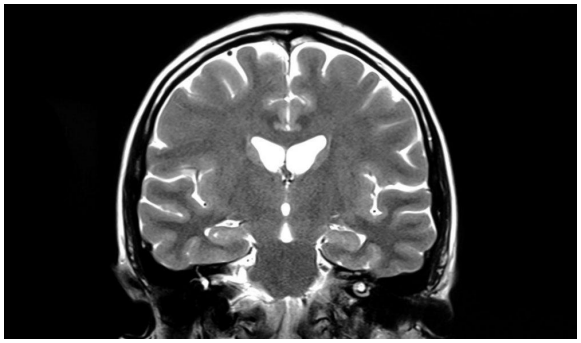
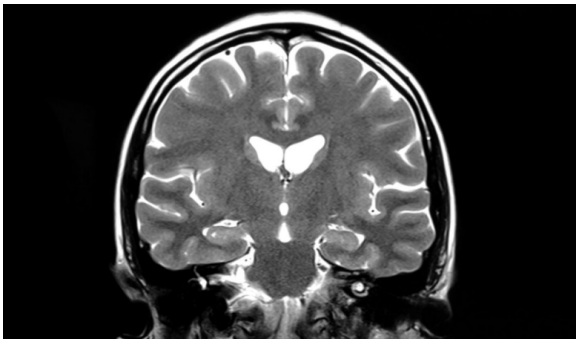
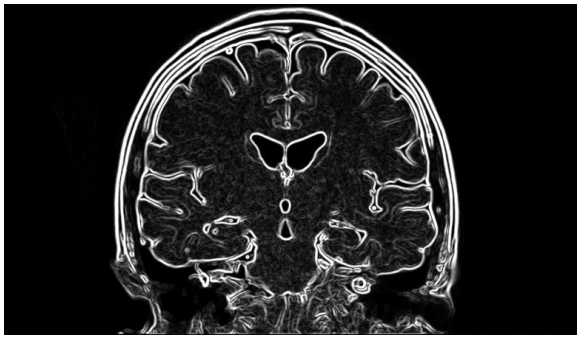
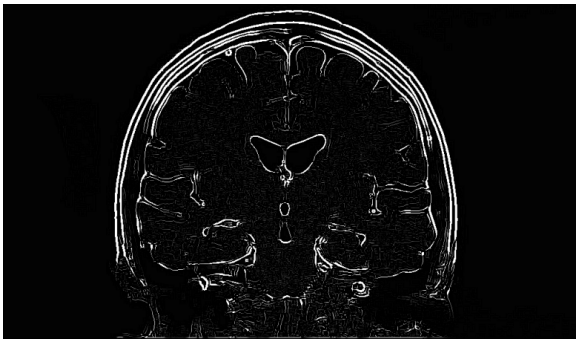
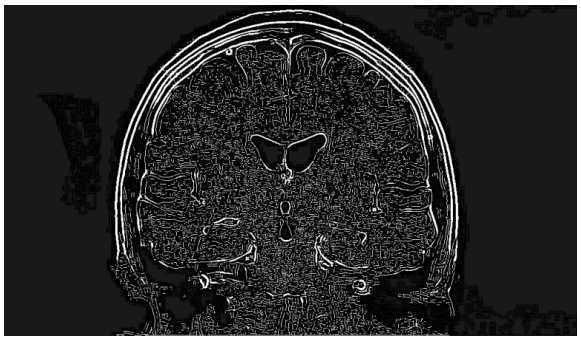
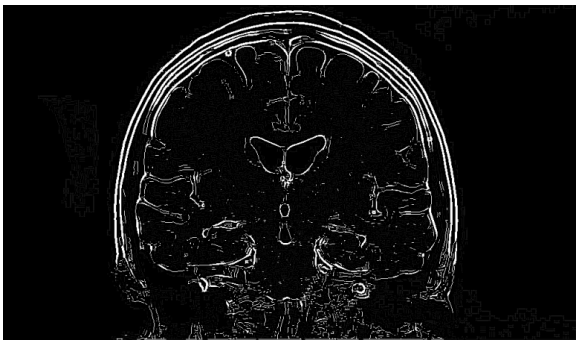
(6) = (3) + Thresholding
(low=0.05, high=0.1)

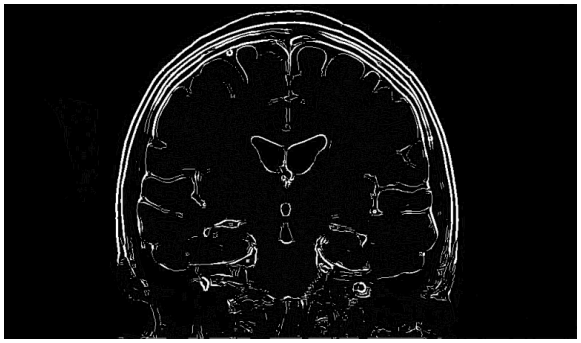
(7) result from web demo

3. house

(1) = GRAYSCALE + Gaussian

<table>
<tr><td> 
(2) = (1) + Sobel</td><td> 
(3) = (2) + Non-maxima suppression + ConstrastAdjust</td></tr>
<tr><td> 
(4) = (3) + Thresholding (low=0.05, high=0.3)</td><td> 
(5) = (3) + Thresholding (low=0.1, high=0.5)</td></tr>
<tr><td> 
(6) = (3) + Thresholding (low=0.3, high=0.7)</td><td> 
(7) result from web demo</td></tr>
<tr><td>4. brain</td><td></td></tr>
</table>

	 (1) = GRAYSCALE + Gaussian
 (2) = (1) + Sobel + ConstrastAdjust	 (3) = (2) + Non-maxima suppression
 (4) = (3) + Thresholding (low=0.05, high=0.1)	 (5) = (3) + Thresholding (low=0.05, high=0.3)



(6) = (3) + Thresholding
(low=0.05, high=0.5)

(7) result from web demo

5. coins

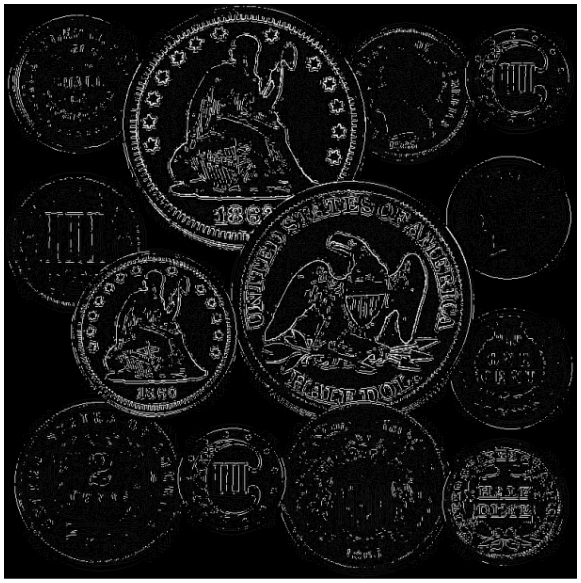
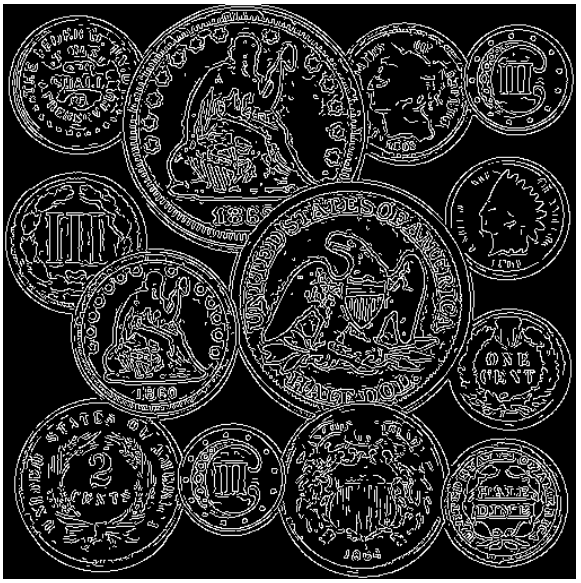
(1) = GRAYSCALE + Gaussian

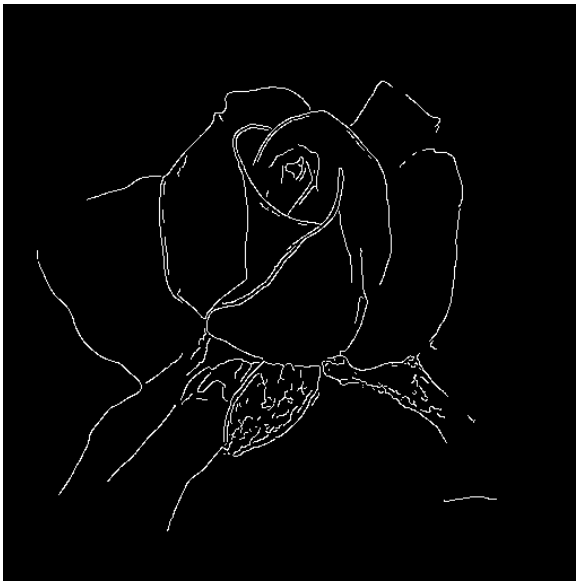
(2) = (1) + Sobel + ConstrastAdjust

(3) = (2) + Non-maxima suppression

(4) = (3) + Thresholding
(low=0.1, high=0.7)

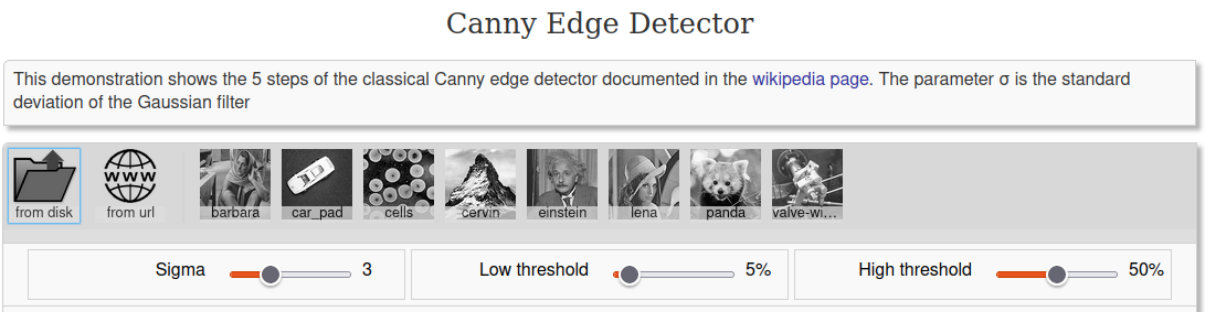
(5) = (3) + Thresholding
(low=0.1, high=0.6)

 (6) = (3) + Thresholding (low=0.1, high=0.5)	 (7) result from web demo
6. rose	
	 (1) = GRAYSCALE + Gaussian

 (2) = (1) + Sobel + ConstrastAdjust	 (3) result from web demo+ ConstrastAdjust

Discussion

1. 每一張在做 Canny edge detection 之前都會先做 Gaussian filter 以消除雜訊
2. 每一張接著做 Sobel 處理, 有些圖片亮度會變很暗, 所以會做對比度調整
3. 有些圖片經過 Sobel 處理之後, 即可得到不錯的 edge 效果, 如 rose.jpg
4. 每一張接著做 non-maxima suppression 和設定 threshold, 在設定 low threshold 時, 當值越低，則白色線條越多；相反地，high threshold 越高，則白色線條越少
5. 有些圖片經 non-maxima suppression 處理後亮度會變很暗, 所以會做對比度調整
6. 與現有Canny edge detection平台相比 (http://bigwww.epfl.ch/demo/ip/demos/edgeDetector/), 它成果的線條較連續且 noise 較少, 原因在於它可以隨時調整 sigma, low threshold, high threshold 以求得理想的 edge detection 效果



Code

main.py

```
import numpy as np
import sys
import cv2

from matplotlib import pyplot as plt
from intensity_transform import *
from noise_reduction import *
from bilateral_filter import *
from canny_edge import *
from edge_detect_array import *

path = sys.argv[1]
img = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
row, col= img.shape

dst = img.copy()
# dst = dst[:,:,::-1] # bgr to rgb
# dst = np.where(dst > np.mean(dst), 255, 0)

# === noise_reduction ===
# mean_filter = GetMeanFilter()
gaussian_filter = GetGaussianFilter(1)
dst = Convolution(dst, row, col, gaussian_filter)
dst_y = Convolution(dst, row, col, sobel)
dst_x = Convolution(dst, row, col, sobel.T)
dst = np.hypot(dst_y, dst_x)
dst = dst / dst.max() * 255

# === intensity_transform ===
dst = ConstrastAdjust(dst, 3, 3)
# dst = GammaCorrection(dst, 2.2)
# dst = LogTransform(dst)
# pixelVal_vec = np.vectorize(pixelVal)
# dst = pixelVal_vec(dst, 70, 30, 180, 230)
# dst = HistogramEqual(dst, row, col)

# === Canny edge detection ===
theta = np.arctan2(dst_y, dst_x)
```

```
dst = non_max_suppression(dst, theta)

# dst = ConstrastAdjust(dst, 3, 3)

dst = threshold(dst, 0.1, 0.6)

# === intensity_transform ===
dst = ConstrastAdjust(dst, 3, 3)
# dst = GammaCorrection(dst, 2.2)
# dst = LogTransform(dst)
# pixelVal_vec = np.vectorize(pixelVal)
# dst = pixelVal_vec(dst, 70, 30, 180, 230)
# dst = HistogramEqual(dst, row, col)

# dst = Convolution(dst, row, col, LoG3)


dst = np.round(dst)
dst = dst.astype(np.uint8)

# cv2.imshow('img', img)
# cv2.imshow('dst', dst)

# cv2.waitKey(0)

cv2.imwrite("images/test.png", dst)
# cv2.destroyAllWindows()

plt.figure(figsize=(10, 8))
plt.imshow(dst, cmap="gray")
plt.show()
```

intensity_transform.py

```
import numpy as np

def ConstrastAdjust(dst, contrast, brightness):
    a = contrast
    b = brightness

    dst = dst * a + b
    dst[dst > 255] = 255
    dst[dst < 0] = 0
```

```
    return dst

def GammaCorrection(dst, gamma):
    dst = (dst / 255) ** (1 / gamma)
    dst = dst * 255
    dst[dst > 255] = 255
    dst[dst < 0] = 0

    return dst

def LogTransform(dst):
    c = 255/(np.log(1 + np.max(dst)))
    dst = c * np.log(1 + dst)

    return dst

def pixelVal(dst, r1, s1, r2, s2):
    if (0 <= dst and dst <= r1):
        return (s1 / r1)*dst
    elif (r1 < dst and dst <= r2):
        return ((s2 - s1)/(r2 - r1)) * (dst - r1) + s1
    else:
        return ((255 - s2)/(255 - r2)) * (dst - r2) + s2

def HistogramEqual(dst, row, col):
    P = dst

    P_cdf = np.zeros((256,))
    P_sum = 0
    for i in range(256):
        P_count = np.count_nonzero(P == i)
        if P_count != 0:
            P_sum = P_sum + P_count
            P_cdf[i] = P_sum

    P_eq = P_cdf / (row * col * 255)

    for i in range(row):
        for j in range(col):
            P_color = dst[i][j]
            dst[i][j] = P_eq[P_color]
```

```
    return dst
```

noise_reduction.py

```
import numpy as np
from numba import jit

def GetMeanFilter():
    filter = np.zeros((3, 3))
    filter = filter + 1
    filter = filter / filter.sum()

    return filter

def GetGaussianFilter(sigma):
    x, y   = np.mgrid[-1:2, -1:2]
    gaussion_filter = np.exp(-(x**2 + y**2) / 2 * pow(sigma, 2))
    gaussion_filter = gaussion_filter / gaussion_filter.sum() #
normalize

    return gaussion_filter

@jit
def Convolution(dst, row, col, ftr):
    tmp = np.zeros((row, col))

    for i in range(2, row-2):
        for j in range(2, col-2):
            for k in range(i-1, i+2):
                for l in range(j-1, j+2):
                    tmp[i][j] += dst[k][l] * ftr[k-i+1][l-j+1]

    return tmp


def GetMedianFilter(dst, row, col, img_extend):
    for i in range(row):
        for j in range(col):
            tmp = img_extend[i:i+3, j:j+3]
            tmp = tmp.reshape(9, 3)
            median = np.median(tmp, axis=0)
            dst[i][j] = median
```

```
    return dst

def GetAlphaTrimmedMeanFilter(dst, row, col, img_extend, d):
    d2 = int(d/2)

    for i in range(row):
        for j in range(col):
            tmp = img_extend[i:i+3, j:j+3]
            tmp = tmp.reshape(9, 3)
            tmp = tmp[d2:9-d2, 0:3]
            mean = np.mean(tmp, axis=0)
            dst[i][j] = mean

    return dst
```

bilateral_filter.py

```
import numpy as np

def distance(x, y, i, j):
    return np.sqrt(((x-i)**2 + (y-j)**2))

def gaussian(x, sigma):
    return np.exp(- (x ** 2) / (2 * sigma ** 2))

def BilateralFilter(dst, row, col, img_extend, sigma_d=10,
sigma_r=10):
    for x in range(row):
        for y in range(col):
            pixel_filter = np.zeros((3,))
            w_sum = np.zeros((3,))
            for i in range(x, x+3):
                for j in range(y, y+3):
                    d = distance(i, j, x+1, y+1)
                    Id = img_extend[i][j] - img_extend[x+1][y+1]
                    space_w = gaussian(d, sigma_d)
                    color_w = gaussian(Id, sigma_r)
                    w = space_w * color_w
                    pixel_filter += img_extend[i][j] * w
                    w_sum += w
            pixel_filter = pixel_filter / w_sum
            dst[x][y] = pixel_filter
```

```
    return dst
```

canny_edge.py

```
import numpy as np


def non_max_suppression(img, D):
    row, col = img.shape
    res = np.zeros((row, col))
    angle = D * 180. / np.pi
    angle[angle < 0] += 180


    for i in range(1, row-1):
        for j in range(1, col-1):
            try:
                q = 255
                r = 255

               #angle 0
                if (0 <= angle[i,j] < 22.5) or (157.5 <= angle[i,j] <=
180):
                    q = img[i, j+1]
                    r = img[i, j-1]
                #angle 45
                elif (22.5 <= angle[i,j] < 67.5):
                    q = img[i+1, j-1]
                    r = img[i-1, j+1]
                #angle 90
                elif (67.5 <= angle[i,j] < 112.5):
                    q = img[i+1, j]
                    r = img[i-1, j]
                #angle 135
                elif (112.5 <= angle[i,j] < 157.5):
                    q = img[i-1, j-1]
                    r = img[i+1, j+1]

                if (img[i,j] >= q) and (img[i,j] >= r):
                    res[i,j] = img[i,j]
                else:
                    res[i,j] = 0
```

```
            except IndexError as e:
                pass

    return res

def threshold(img, lowThresholdRatio, highThresholdRatio):

    highThreshold = img.max() * highThresholdRatio
    lowThreshold = highThreshold * lowThresholdRatio

    row, col = img.shape
    res = np.zeros((row, col))

    weak = 25
    strong = 255

    strong_i, strong_j = np.where(img >= highThreshold)
    zeros_i, zeros_j = np.where(img < lowThreshold)

    weak_i, weak_j = np.where((img <= highThreshold) & (img >=
lowThreshold))

    res[strong_i, strong_j] = strong
    res[weak_i, weak_j] = weak

    return res
```

edge_detect_array.py

```
import numpy as np

prewitt = np.array([
    [-1, -1, -1],
    [ 0,  0,  0],
    [ 1,  1,  1]
])

prewitt45 = np.array([
    [-1, -1,  0],
    [-1,  0,  1],
    [ 0,  1,  1]
])
```

```
prewitt135 = np.array([
   [ 0,  1,  1],
   [-1,  0,  1],
   [-1, -1,  0]
])

sobel = np.array([
   [-1, -2, -1],
   [ 0,  0,  0],
   [ 1,  2,  1]
])

sobel45 = np.array([
   [-2, -1,  0],
   [-1,  0,  1],
   [ 1,  1,  2]
])

sobel135 = np.array([
   [ 0,  1,  2],
   [-1,  0,  1],
   [-2,  -1, 0]
])

LoG3 = np.array([
   [ 0, -1,  0],
   [-1,  4, -1],
   [ 0, -1,  0]
])

LoG38 = np.array([
   [-1, -1, -1],
   [-1,  8, -1],
   [-1, -1, -1]
])

LoG5 = np.array([
   [0, 0, -1, 0, 0],
   [0, -1, -2, -1, 0],
   [-1, -2, 16, -2, -1],
   [0, -1, -2, -1, 0],
```

```
    [0, 0, -1, 0, 0]
])
```