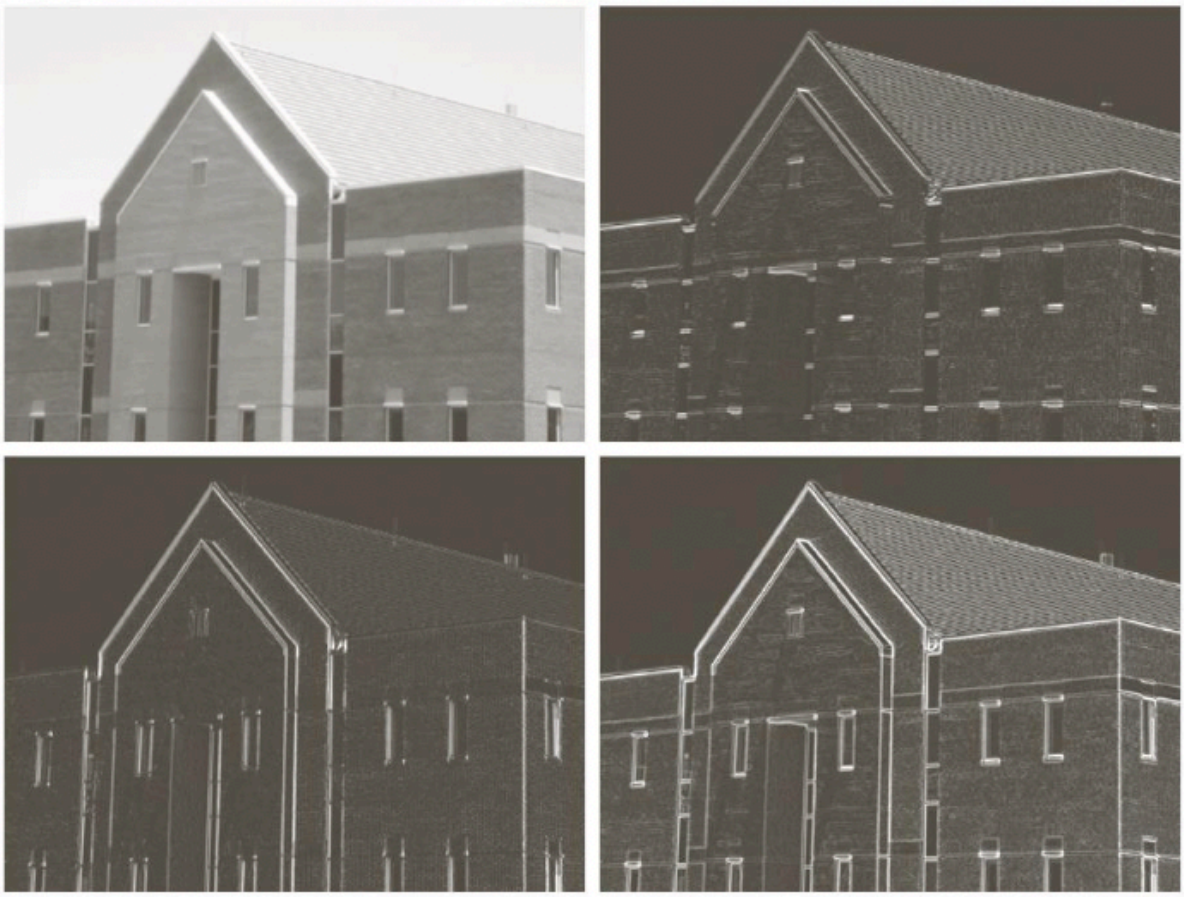
309551122 曹芳驊

Image processing
Project2 report

Introduction

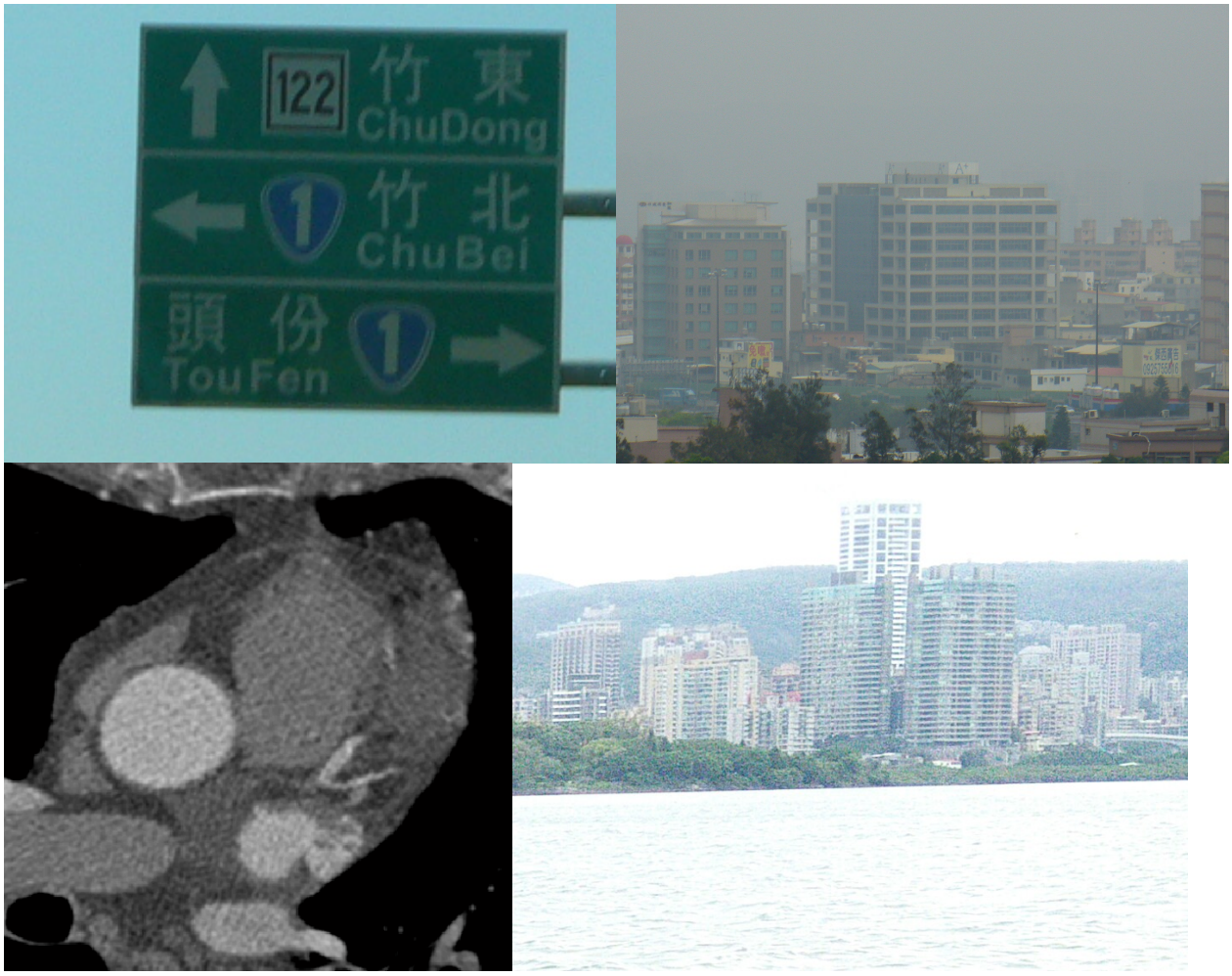
The project goal is about implement edge detection and one of canny, houth, watershed algorithm. In my work, I choose to implement canny edge detection algorithm.



In my work, I choose to test on the 4 images below, because these 4 images consist of many edges and obvious contour can be captured.

I implement the prewitt, sobel gradient filter and LOG. Then try smoothing (median filter) and threshold before and after, observing and compare the result with and without these adjustment.

In the end I implement the canny edge detection algorithm, compare the result with the above filter.



Review of methods

image gradient

$$M(x,y) = |\nabla f| \approx |g_x| + |g_y|$$

Use first-derivatives to detect the variation of neighbor pixel (edge)

Prewitt Filters:

-1	-1	-1
0	0	0
1	1	1

-1	0	1
-1	0	1
-1	0	1

Sobel Filters:

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

LOG filter

$$-\nabla^2 h(r) = -\left(\frac{r^2 - 2\sigma^2}{\sigma^4}\right)\exp(-r^2/2\sigma^2)$$

Smooth the image with a Gaussian smoothing filter first, then apply Laplacian filter to obtain the second derivative information.

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

Canny edge detector

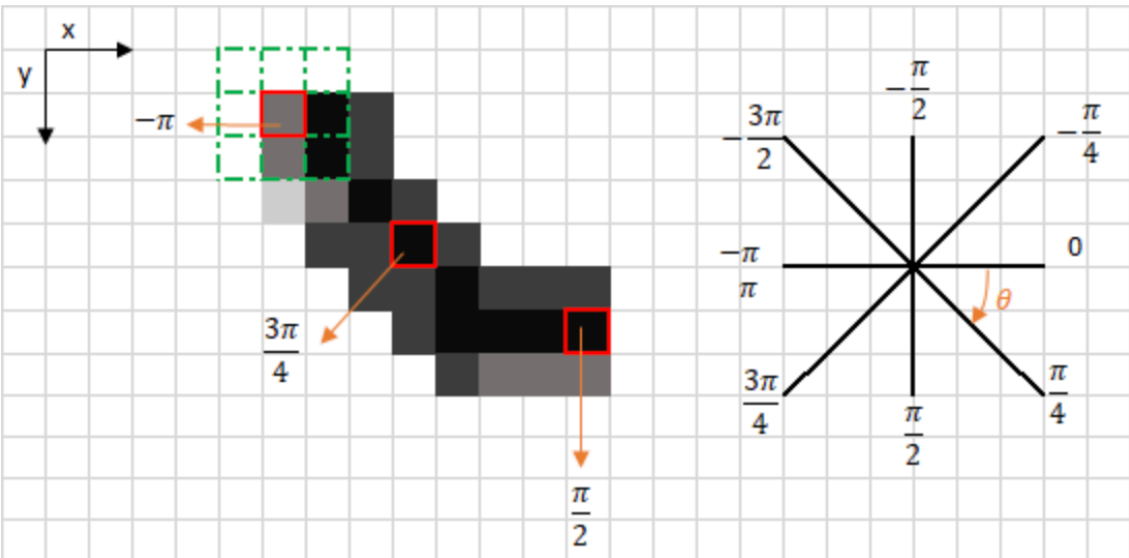
Step

1. smoothing. The reason is similar to LoG.
2. Gradient magnitude and angle computation using a pair of directional first-derivative operators (e.g., Sobel).
3. Non-maxima suppression (thinning of edges): Keep only the "strongest" edge point along the gradient direction.
4. Double thresholding: Retain all the "strong" edge points as well as the "weak" edge points that are connected to the "strong" edge points.

Non-maxima suppression

Keep only the "strongest" edge point along the gradient direction.

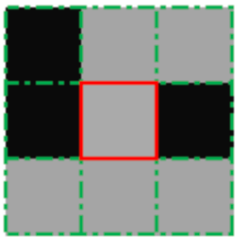
For each pixel, in the direction fo gradient, if there are some “stronger” pixel, erase the current pixel (suppression).



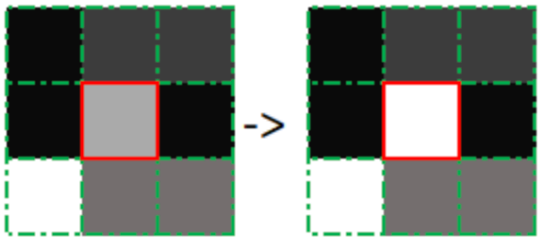
Double thresholding

Retain all the "strong" edge points as well as the "weak" edge points that are connected to the "strong" edge points.

Keep doing until there is no weak edge points connect to any strong edge points.



No strong pixels around



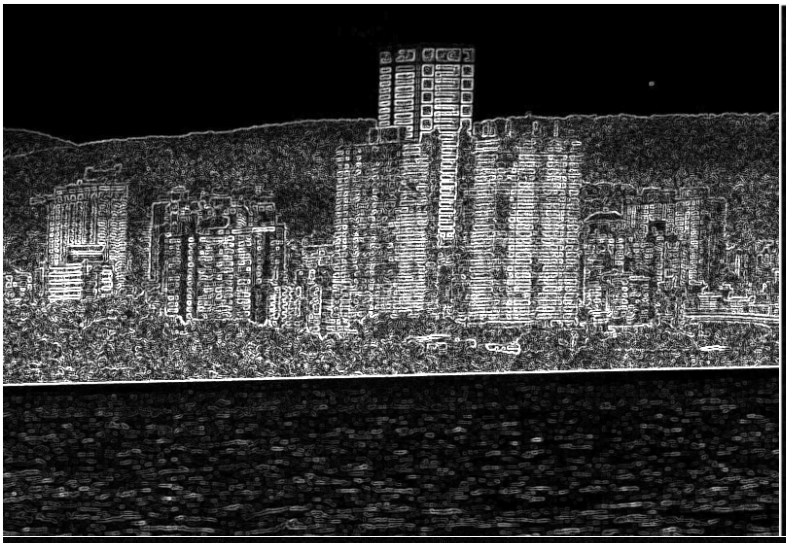
One strong pixel around

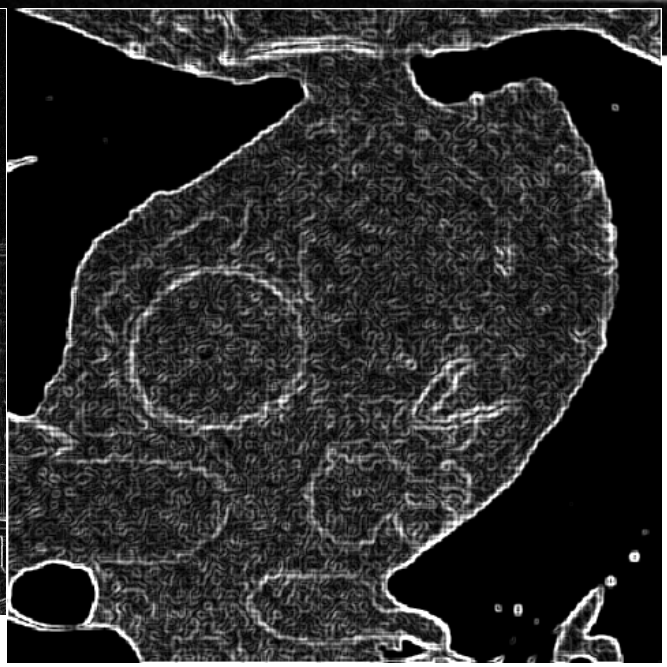
Experiment

Standard gradient filter

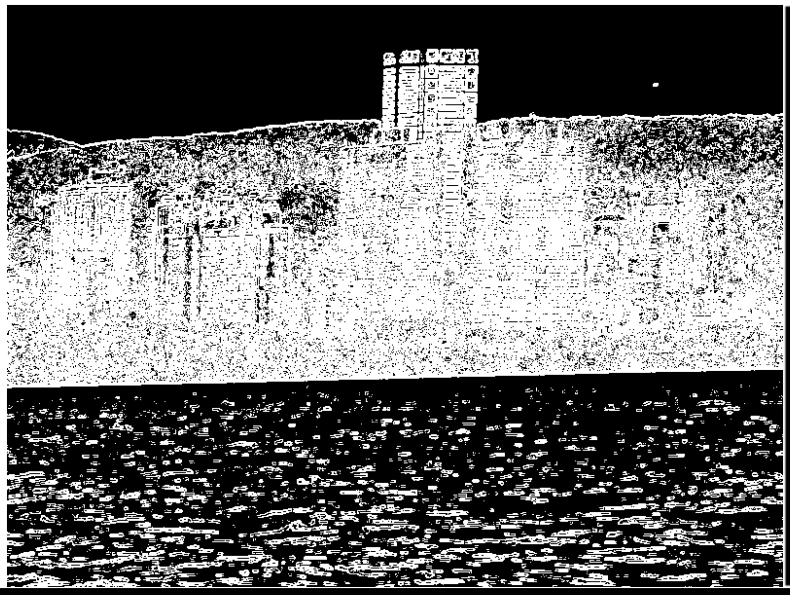
Because the result and experiment of prewitt filter Is almost the same with sobel filter, so in this part I only show the result of sobel.

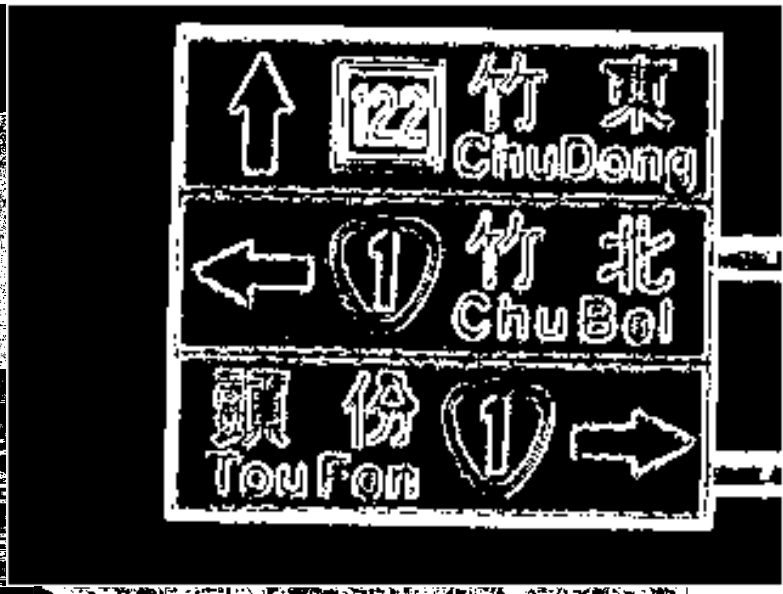
Only gradient filter (|gx|+|gy|):

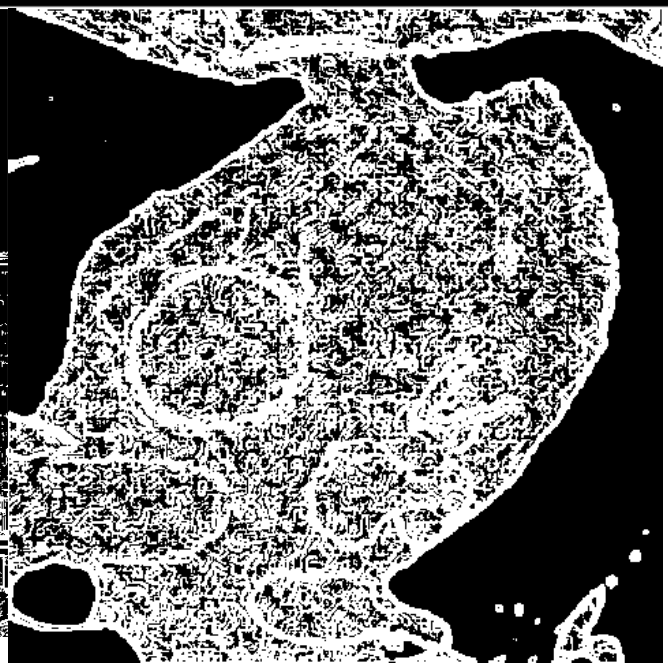




Gradient + threshold :

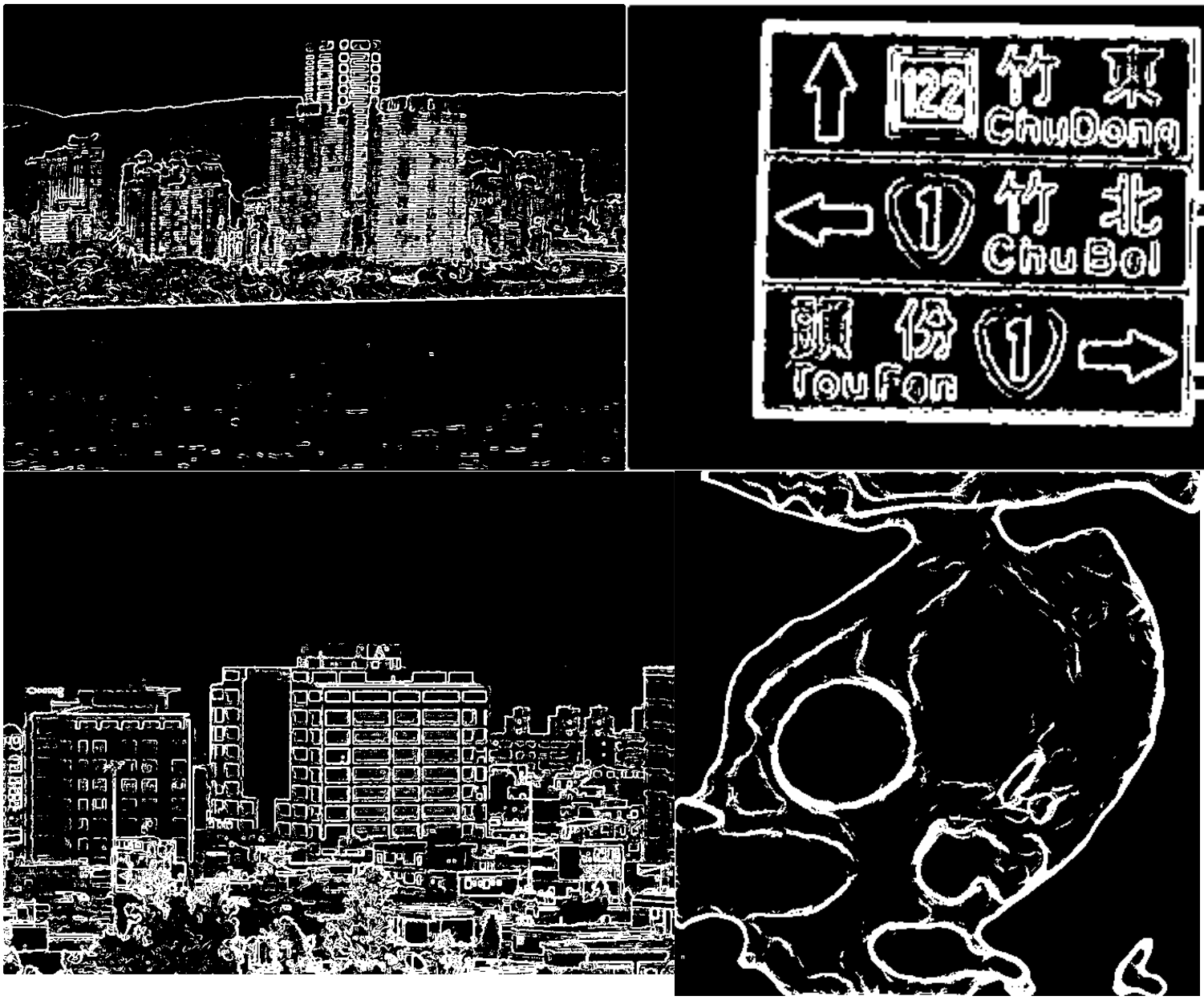






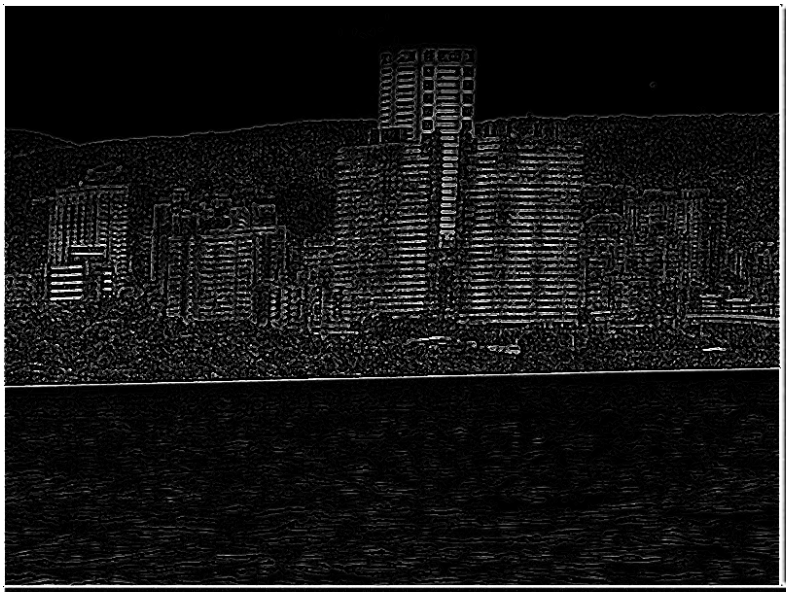
(suffer from noise)

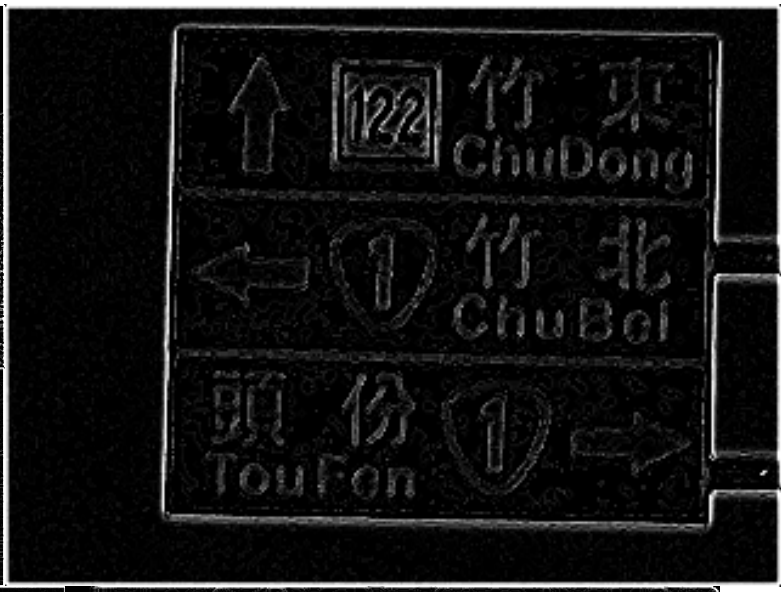
smooth(median filter) + gradient + threshold:

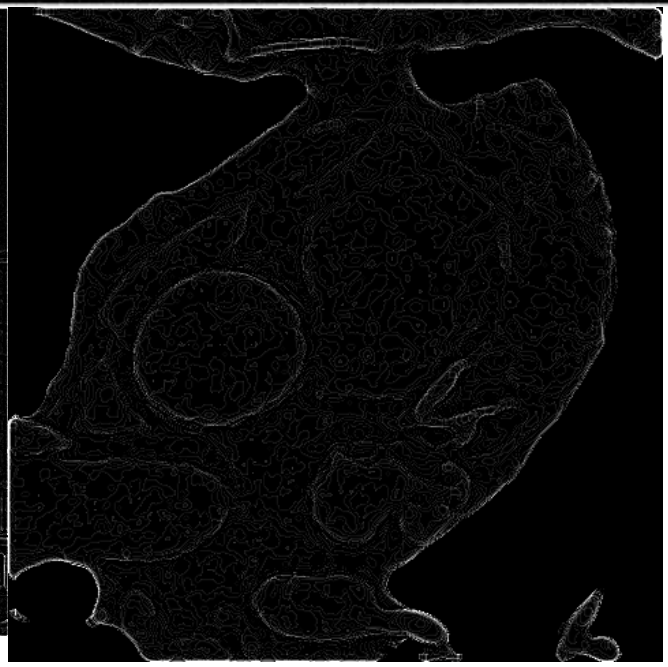


LOG

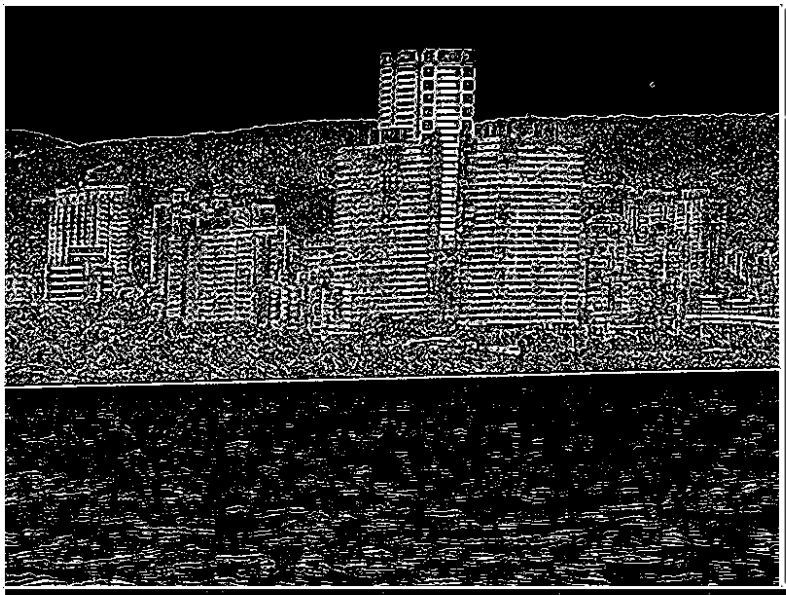
Smooth + log :

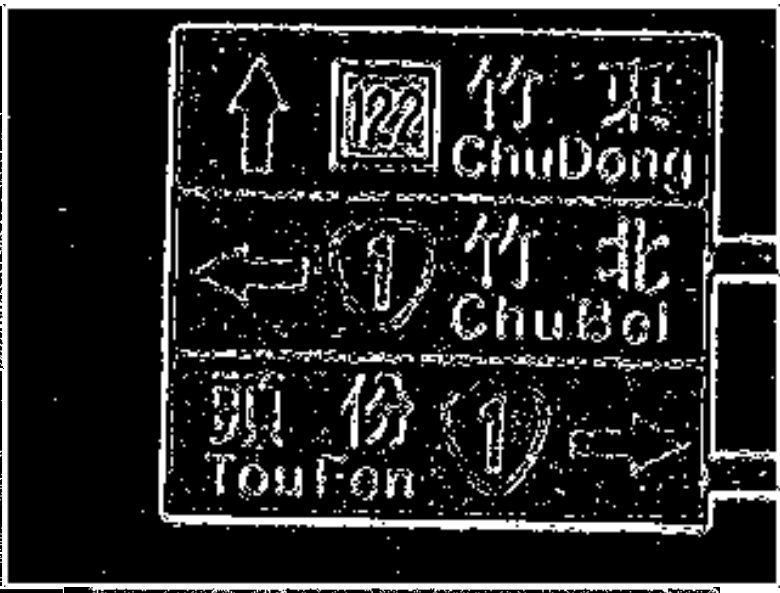


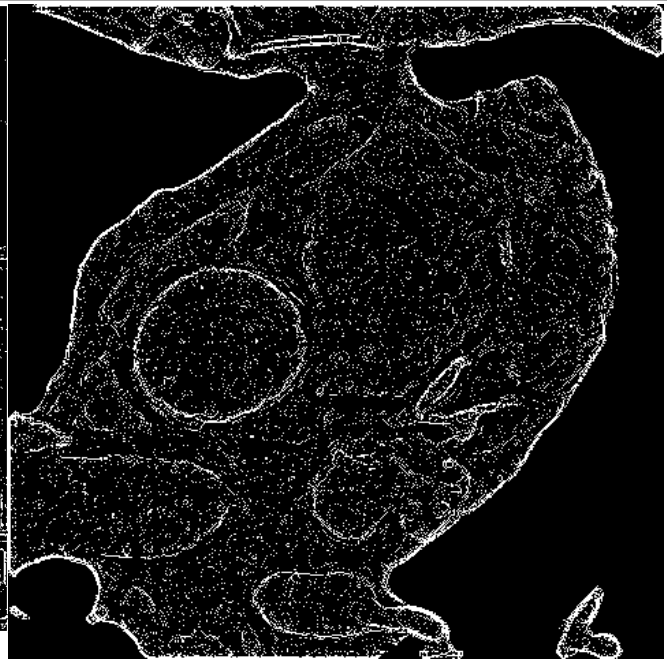




Smooth + log + threshold



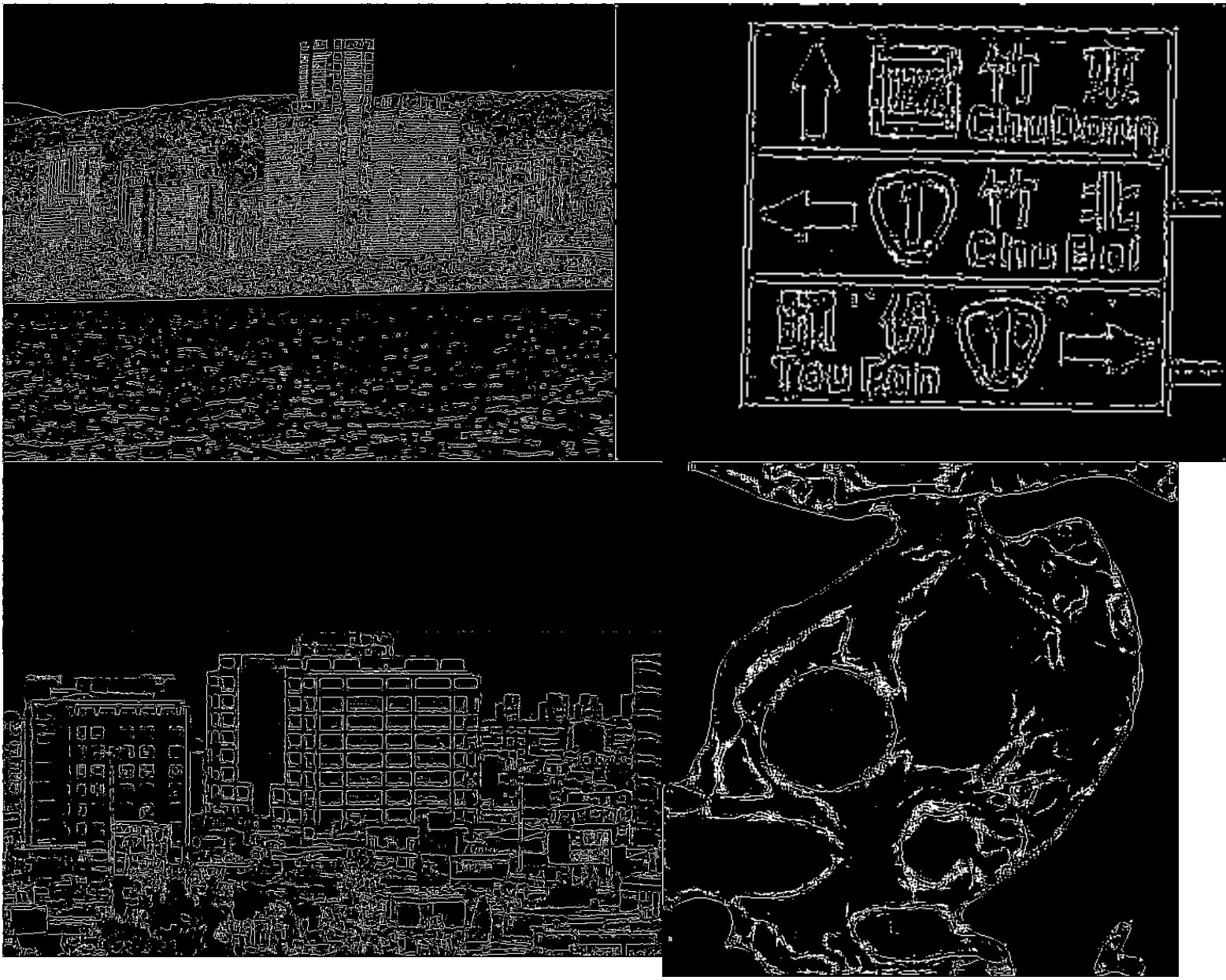




Even though after median and gaussian filter denosing, result still suffer from unignorable noise

But laplacian filter can actually capture more information about edge.

Canny edge detecter



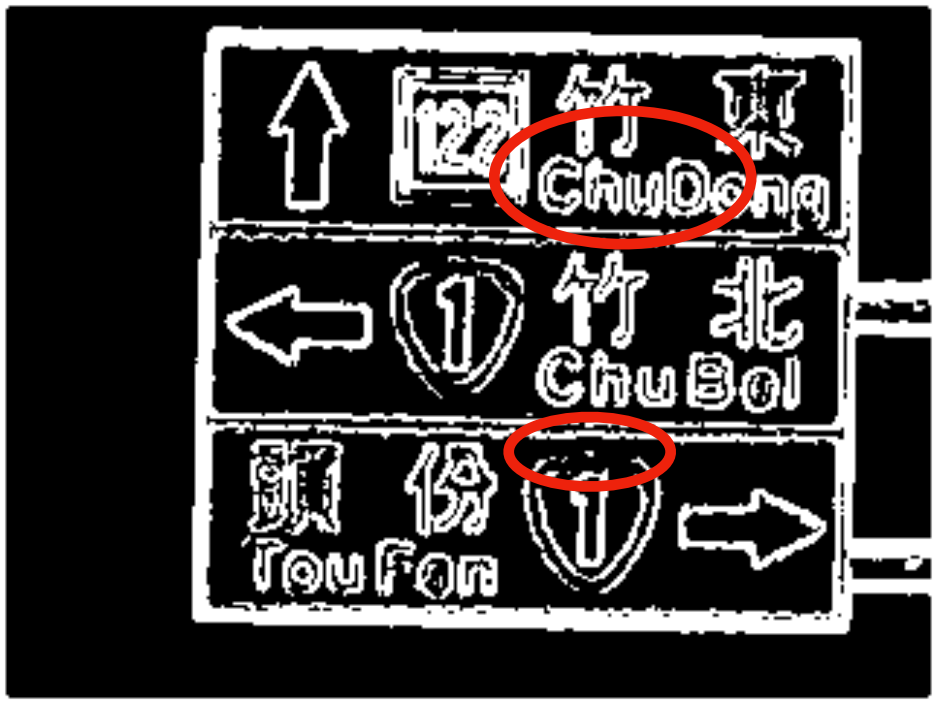
The local edge points are connected, but also increase the impact of noises.

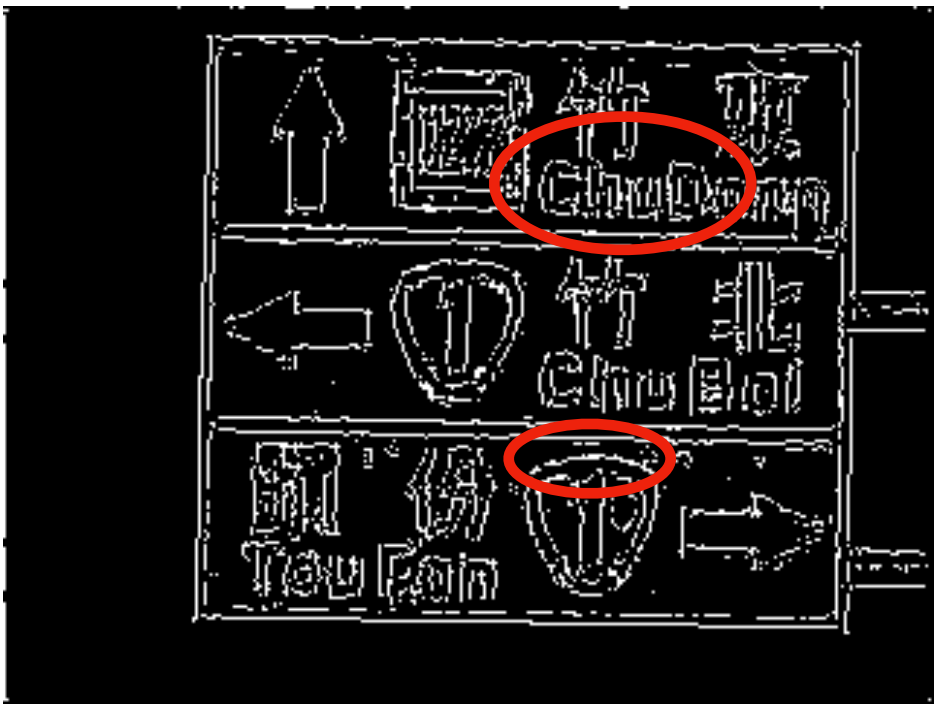
Comparison

Standard gradient filter v.s Canny algorithm :

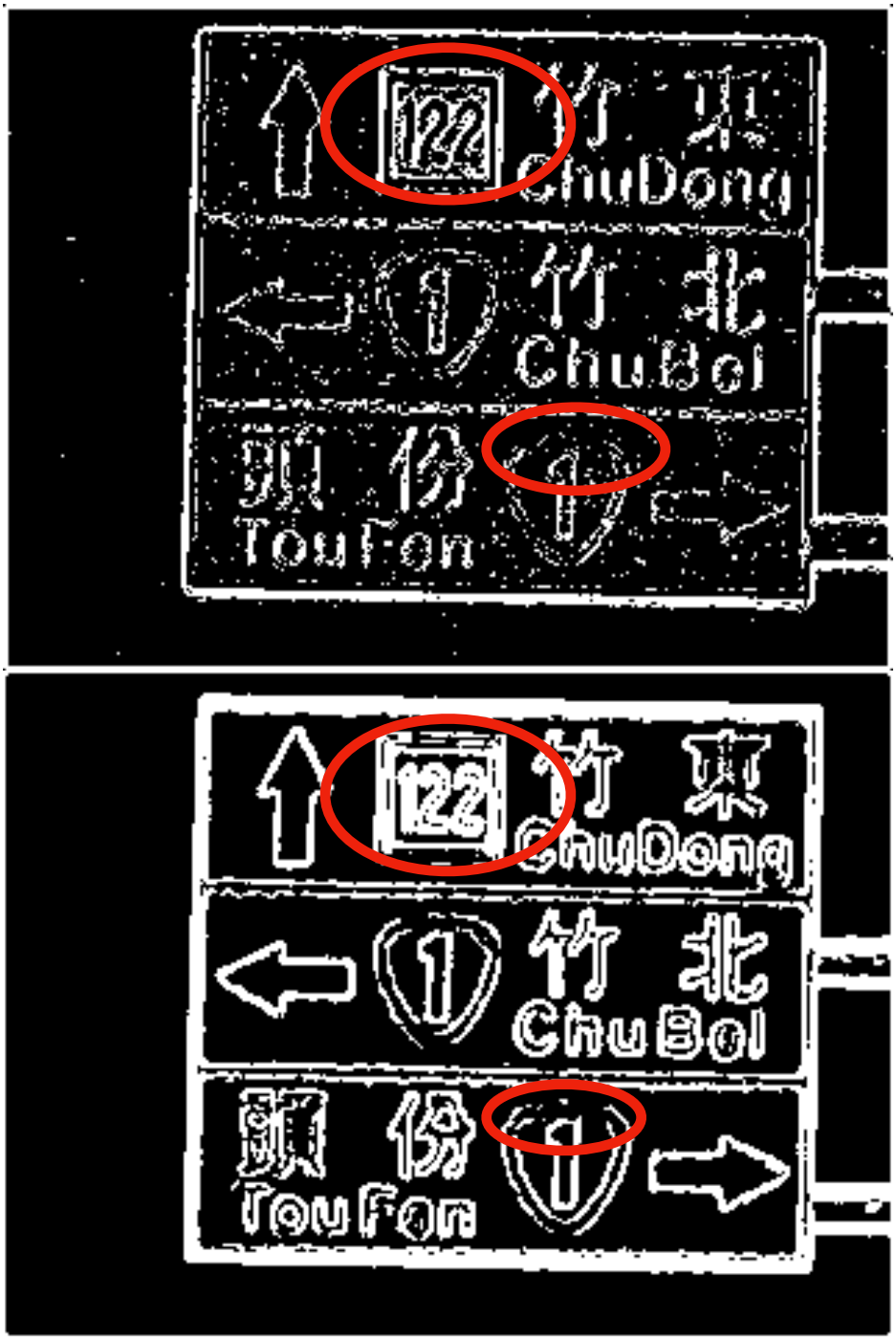
Because of the non-maixma suppression, canny detector lost some information of oblique edges

But for some non-connected edge in the standard filter, canny detector can properly connect them.





LOG v.s Standard gradient filter



Discussion

According to the result of my experiments, I think the noise cancelling is still a important problem of the edge detection. If there are lots of noises exist, the result would be very terrible after thresholding. The more precise we want to detect the edge, the problem will be more serious.

Additionally, the result of the canny edge detector in my experiments shows that it's actually have better ability to detect some blur edge, but after the non-maximal suppression there might be some information lost. And when there are lots of complex overlapping edges in the image (img1, img6) canny detector might not have better result.

Code implementation

Conv : use for filter convolution

```
def conv(kernel, img, size):
    '''doing padding and conv operation to img base on given kernel and size'''
    if img.ndim == 2 : gray = True
    else : gray = False
    pad = size // 2
    img_pad = cv2.copyMakeBorder(img, pad, pad, pad, pad, cv2.BORDER_CONSTANT, 0)
    fil_img = np.zeros(img.shape)
    for x in range(img.shape[0]):
        for y in range(img.shape[1]):
            if gray :
                val = np.sum(img_pad[x:x+size, y:y+size] * kernel)
                fil_img.itemset((x,y), val)
            else :
                for c in range(3):
                    val = np.sum(img_pad[x:x+size, y:y+size, c] * kernel)
                    fil_img.itemset((x,y,c), val)
    return fil_img
```

Filters :

```
def prewitt(img, size = 3):
    '''prewitt gradient filter'''
    kernel_y = np.array([[-1,-1,-1],
                         [0,0,0],
                         [1,1,1]])
    kernel_x = np.array([[-1,0,1],
                         [-1,0,1],
                         [-1,0,1]])
    grad_x = conv(kernel_x, img, size)
    grad_y = conv(kernel_y, img, size)
    new_img = np.add(np.abs(grad_x), np.abs(grad_y))
    return new_img
```

```python
def sobel(img, size = 3):
    '''sobel gradient filter'''
    kernel_y = np.array([[-1,-2,-1],
                         [0,0,0],
                         [1
                          1: int
    kernel_x = np.array([[-1,0,1],
                         [-2,0,2],
                         [-1,0,1]])
    grad_x = conv(kernel_x, img, size)
    grad_y = conv(kernel_y, img, size)
    new_img = np.add(np.abs(grad_x), np.abs(grad_y))
    return new_img
```

```python
def log(img):
    kernel = 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]])
    filted = conv(kernel, img, 5)
    return filted
```

Canny detector:

```
def canny(img):
    '''img: the gradient magnitude map'''
    kernel_y = np.array([[-1,-2,-1],
                         [0,0,0],
                         [1,2,1]])
    kernel_x = np.array([[-1,0,1],
                         [-2,0,2],
                         [-1,0,1]])
    grad_x = conv(kernel_x, img, size = 3)
    grad_y = conv(kernel_y, img, size = 3)
    mag = np.add(np.abs(grad_x), np.abs(grad_y))
    theta = np.arctan2(grad_y, grad_x) * 180 /np.pi
    mag = non_max_suppression(mag, theta)
    res = doub_threshold(mag)
    return res
```

```
def non_max_suppression(img, theta):
    res = np.zeros(img.shape)
    theta[theta < 0] += 180
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            try :
                if (0 <= theta[i,j] < 22.5) or (157.5 <= theta[i,j] <= 180):
                    q = img[i, j+1]
                    r = img[i, j-1]
                elif (22.5 <= theta[i,j] < 67.5):
                    q = img[i+1, j-1]
                    r = img[i-1, j+1]
                elif (67.5 <= theta[i,j] < 112.5):
                    q = img[i+1, j]
                    r = img[i-1, j]
                elif (112.5 <= theta[i,j] < 157.5):
                    q = img[i-1, j-1]
                    r = img[i+1, j+1]
                if q > img[i,j] or r > img[i,j]: res[i,j] = 0
                else: res[i,j] = img[i,j]
            except IndexError:
                pass
    return res
```

```python
def doub_threshold(img):
    T_h = 40
    T_l = 0
    new_img = np.zeros(img.shape)
    s_x, s_y = np.where(img >= T_h)
    w_x, w_y = np.where((img >= T_l) & (img < T_h))
    new_img[s_x, s_y] = 255
    change = True
    while(change):
        change = False
        for i in range(len(w_x)):
            for j in range(len(w_y)):
                x = w_x[i]
                y = w_y[j]
                if x != -1 and y != -1:
                    if x - 1 >= 0 and new_img[x-1,y] == 255:
                        change = True
                    if x + 1 < img.shape[0] and new_img[x+1,y] == 255:
                        change = True
                    if y - 1 >= 0 and new_img[x,y-1] == 255:
                        change = True
                    if y + 1 < img.shape[1] and new_img[x,y+1] == 255:
                        change = True
                    if x - 1 >= 0 and y - 1 >=0 and new_img[x-1,y-1] == 255:
                        change = True
                    if x + 1 < img.shape[0] and y - 1 >=0 and new_img[x+1,y-1] == 255:
                        change = True
                    if x - 1 >= 0 and y + 1 < img.shape[1] and new_img[x-1,y+1] == 255:
                        change = True
                    if x + 1 < img.shape[0] and y + 1 < img.shape[1] and new_img[x+1,y+1] == 255:
                        change = True
                    if change:
                        new_img.itemset((x,y), 255)
                        w_x[i] = -1
                        w_y[j] = -1
    return new_img
```