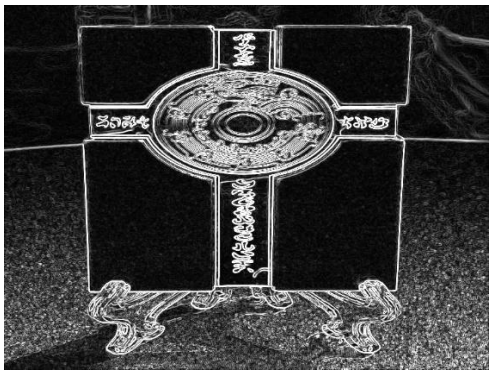
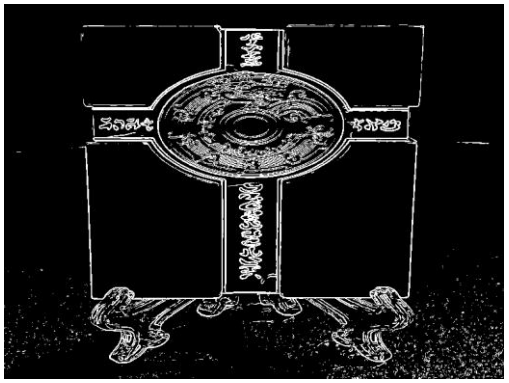
Problem 1: EDGE DETECTION

(a) (10 pt) Apply Sobel edge detection to sample1.png. Output the gradient image and its corresponding edge map as result1.png and result2.png, respectively. Please also describe how you select the threshold and how it affects the result.

Original image: Sample1



Output image: result1



Output image: result2
(binary 白表示邊)

Motivation and approach (include parameters): Sobel edge detection

- **Discrete case**
  - **Approximation III – 9points**

A_0	A_1	A_2
A_7	$F(j,k)$	A_3
A_6	A_5	A_4

    - **Row gradient**

$$G_R(j,k)=\frac{1}{K+2}\left[(A_2+KA_3+A_4)-(A_0+KA_7+A_6)\right]$$

    - **Column gradient**

$$G_C(j,k)=\frac{1}{K+2}\left[(A_0+KA_1+A_2)-(A_6+KA_5+A_4)\right]$$

$$\Rightarrow G(j,k)=\sqrt{G_R^2(j,k)+G_C^2(j,k)} \qquad \theta(j,k)=\tan^{-1}\left(\frac{G_C(j,k)}{G_R(j,k)}\right)$$

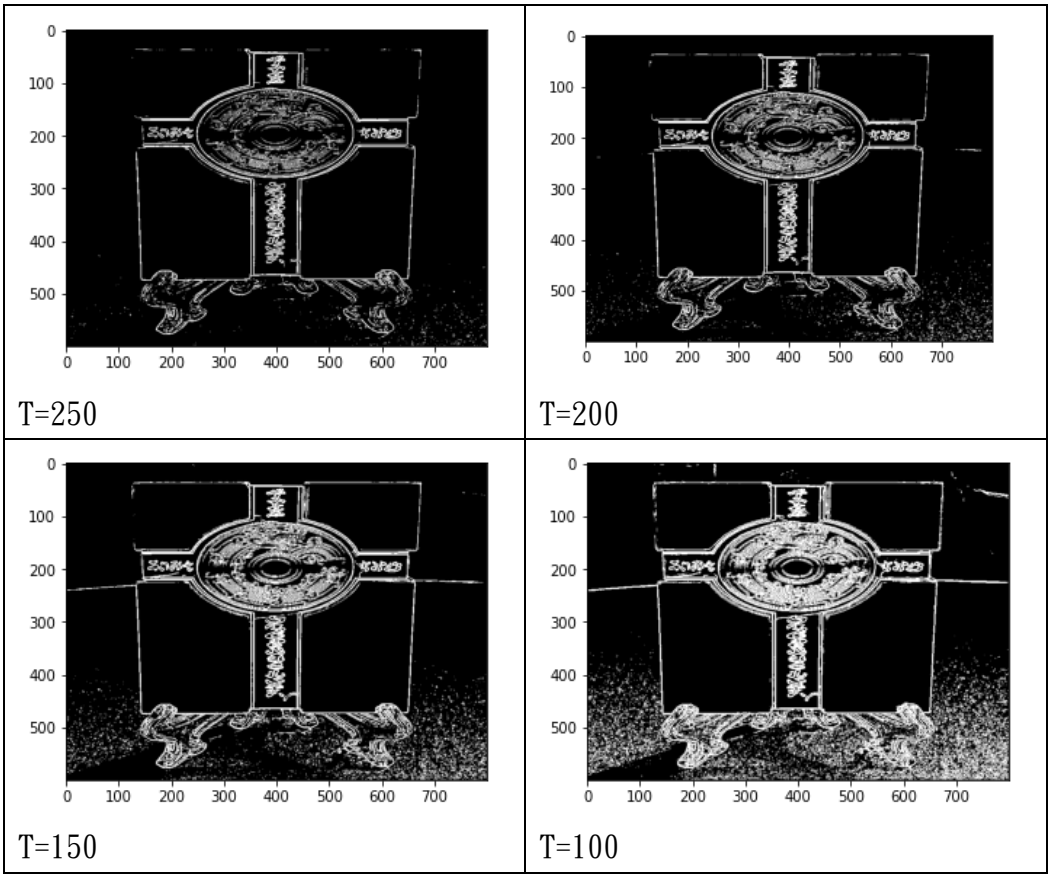
    - **K=1: Prewitt Mask ; K=2: Sobel Mask**

15

- 經由一階微分(差分)來判斷邊界，斜率越大代表數值變動劇烈，越有可能是邊界
- 因為使用 Sobel 所以取 K=2

Discussion of results:

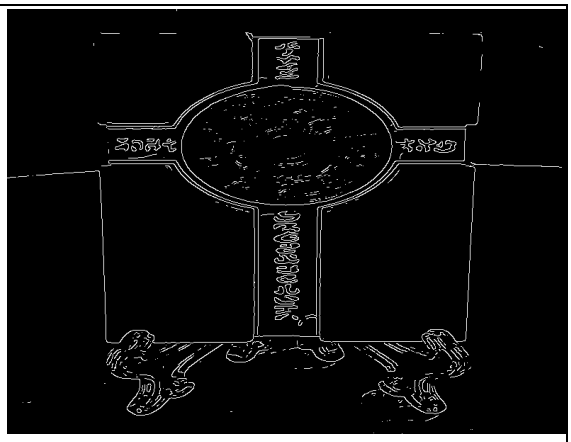
T 越小，被偵測為邊界的點就越多



最後我選擇了 T=200，因為在保留盒子與腳的外觀的同時，地面的雜訊比較少

(b) (10 pt) Perform Canny edge detection on sample1.png and output the edge map as result3.png. Please also describe how you select the parameters and how they affect the result.

Original image: Sample1



Output image: result3

Motivation and approach (include parameters):

Canny Edge Detector

- **Five Steps:**
  - Noise reduction
  - Compute gradient magnitude and orientation
  - Non-maximal suppression
  - Hysteretic thresholding
  - Connected component labeling method

Noise reduction

Smooth the image with a Gaussian filter

Example:
5x5 Gaussian filter with σ = 1.4

$$F_{NR} = \frac{1}{159}\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix} * F$$

使用 5x5 的高斯濾波器，參數如上圖所示

Compute gradient magnitude and orientation

$$G(j,k)=\sqrt{G_R^2(j,k)+G_C^2(j,k)}$$

$$\theta(j,k)=\tan^{-1}\left(\frac{G_C(j,k)}{G_R(j,k)}\right)$$

Magnitude

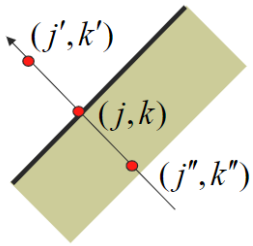
由上圖公式計算 gradient 的大小與方向
GR、GC 使用 Sobel 計算而來(K=2)

Non-maximal suppression

藉由 orientation 算出方向後，找出那個方向中最大的點
如果自己是最大值就保留，如果不是就設為 0

Search the nearest neighbors (j',k') and (j'',k'') along the edge normal

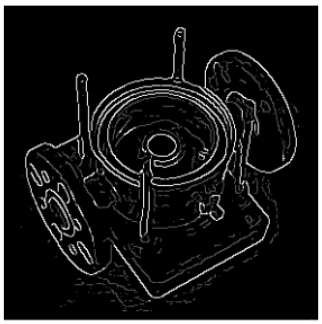
$$G_N(j,k)=\begin{cases}G(j,k) & if\ G(j,k)>G(j',k')\\ & and\ G(j,k)>G(j'',k'')\\ 0\end{cases}$$



Hysteretic thresholding

Label each pixels according to two threshold: T_H, T_L

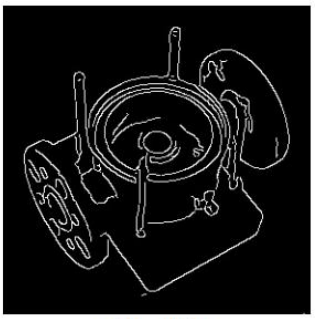
$G_N(x,y)\geq T_H$	Edge Pixel
$T_H>G_N(x,y)\geq T_L$	Candidate Pixel
$G_N(x,y)<T_L$	Non-edge Pixel



藉由 TH、TL 來標記 edge、non-edge 和 candidate pixel
為了讓地板的雜訊盡量減少還有保持盒子外觀
我將 TH 設為 60、TL 設為 100

Connected component labeling method

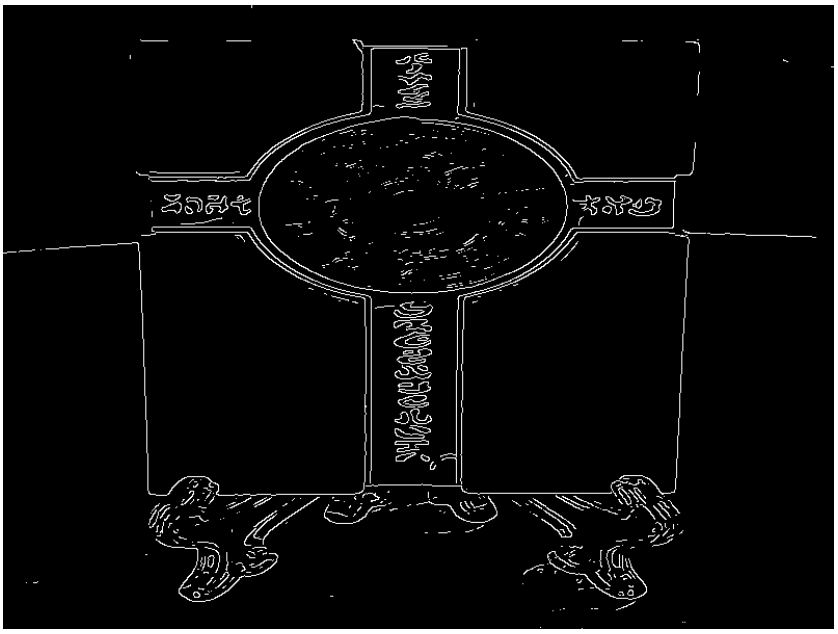
If a candidate pixel is connected to an edge pixel directly or via another candidate pixel then it is declared as an edge pixel



Edge Map

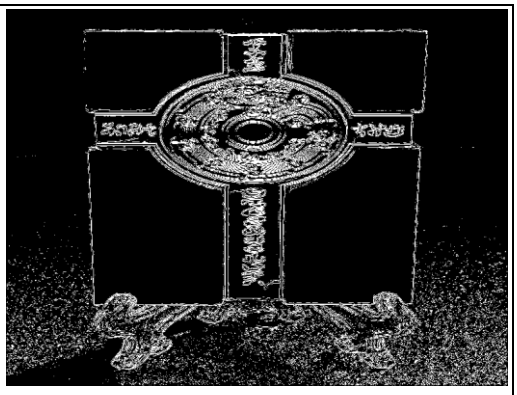
如果 candidate pixel 直接或間接的連到 edge pixel 就把它設為 edge pixel

Discussion of results:



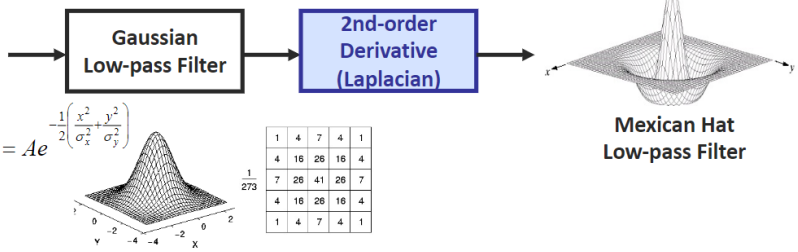
Edge 還蠻俐落的

(c) (10 pt) Use the Laplacian of Gaussian edge detection to generate the edge map of sample1.png and output it as result4.png. Compare result2.png result3.png and result4.png and discuss on these three results.

	
Original images	Output image

Motivation and approach (include parameters):

- **Laplacian of Gaussian (LOG)**

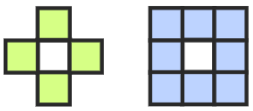


$$g(x,y) = Ae^{-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2}+\frac{y^2}{\sigma_y^2}\right)}$$

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

- **Laplacian impulse response**
  - **four-neighbor**

$$H = \frac{1}{4}\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$



  - **eight-neighbor**

$$H = \frac{1}{8}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

non-separable

$$H = \frac{1}{8}\begin{bmatrix} -2 & 1 & -2 \\ 1 & 4 & 1 \\ -2 & 1 & -2 \end{bmatrix}$$

separable

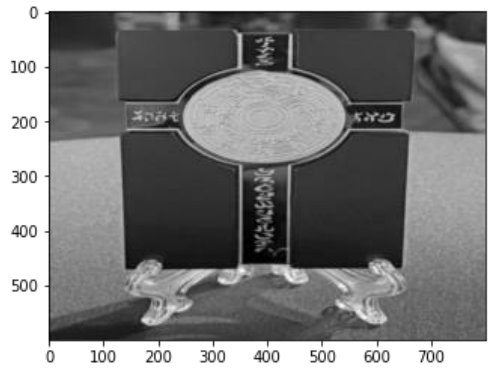
$$H_1 = \frac{1}{8}\begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}$$

$$H_2 = \frac{1}{8}\begin{bmatrix} -1 & -1 & -1 \\ 2 & 2 & 2 \\ -1 & -1 & -1 \end{bmatrix}$$

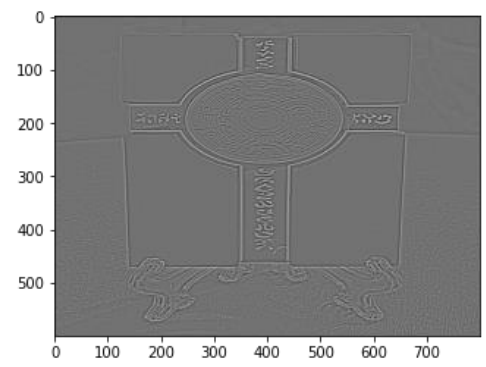
Zero-crossing

3 steps:

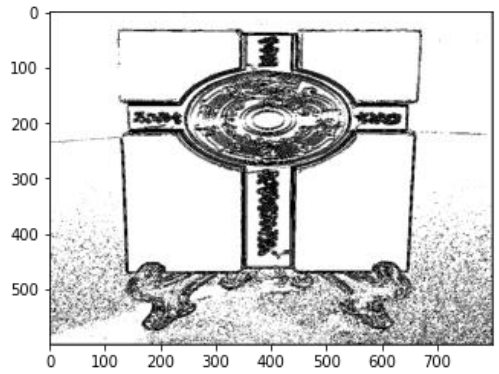
- **Generate the histogram of G**
- **Set up a threshold to separate zero and non-zero, output as G'**
- **For G'(j,k)=0, decide whether (j,k) is a zero-crossing point**



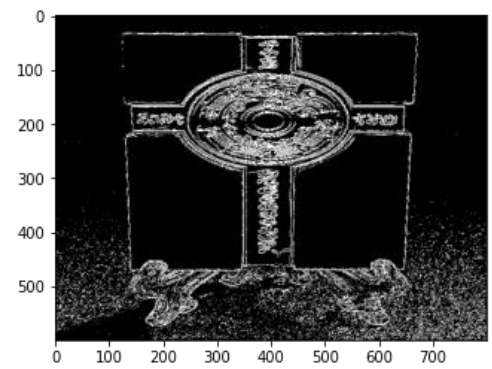
高斯模糊去噪



eight-neighbor 的拉普拉斯算子



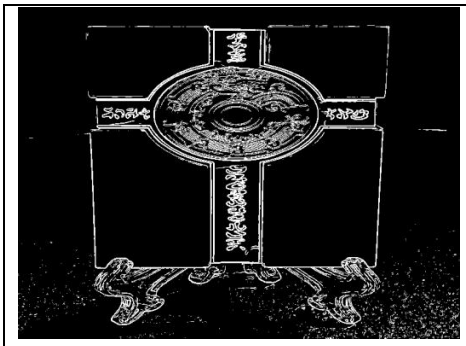
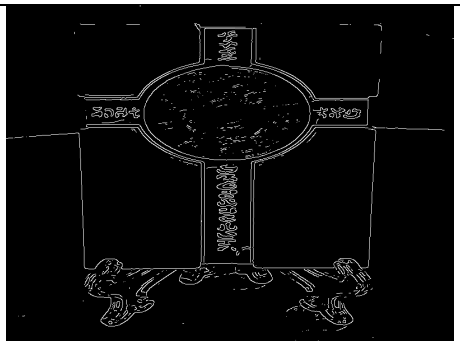
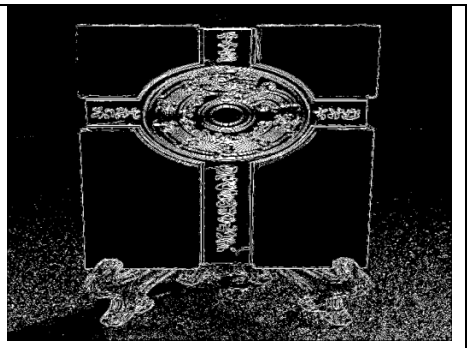
Thresholded = 3



Zero-crossing

Discussion of results:

Compare result2.png result3.png and result4.png:

		
result2	result3	result4

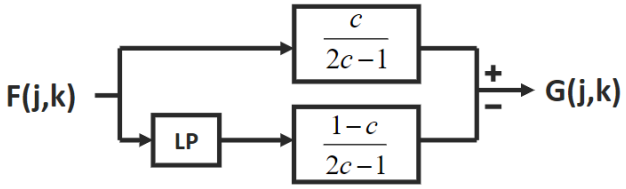
result3(Canny edge detection)的邊最為俐落，地板上的雜訊也最少，但步驟最為複雜耗時，result4(Laplacian of Gaussian)則是細節很多，但地板的雜訊也多，result2(Sobel)結合了上述兩者的優點，既俐落步驟也不複雜，所以我最喜歡的方法是 Sobel

(d) (10 pt) Perform edge crispening on sample2.png and output the result as result5.png. What difference can you observe from sample2.png and result5.png? Please specify the parameters you choose and discuss how they affect the result.

	
Original image	Output image

Motivation and approach (include parameters):

- **Unsharp Masking**
  - **Appropriate combination of all-pass and low-pass filters**



$$G(j,k)=\frac{c}{2c-1}F(j,k)-\frac{1-c}{2c-1}F_L(j,k),\ \ where\ \ \frac{3}{5}\le c\le\frac{5}{6}$$

LP 的部分使用了高斯濾波

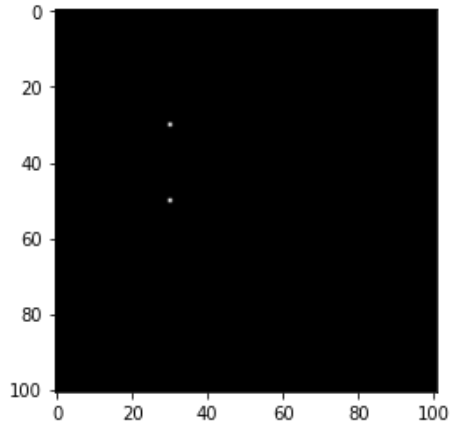
```
edge_crispening(sample2, gauss(77, 25) ,  4 / 6)
```

Kernel size = 77, sigma = 25, c=4/6
經過多次測試發現數字 Kernel size 和 sigma 設大一點、c 設小一點，效果比較明顯

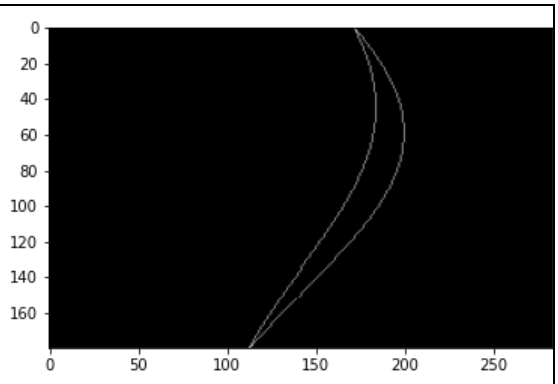
Discussion of results:
LP 濾掉了高頻，只留下了低頻
將原圖減掉低頻，可說是去掉低頻，突顯高頻，因此圖像變得更加銳利了
不知道是不是因為這張圖本身就有點糊糊的，結果看起來其實跟原圖是差不多的，為了突顯效果我故意將數字設很大(對比很強，屋子周圍有光圈)

(e) (Bonus) Apply the Hough transform to result3.png and output the Hough space as result6.png. What lines can you detect by this method?

Motivation and approach (include parameters):
先以一個簡單一點的例子實作
首先將 angle(0~179 度), x, y 帶入 $\rho = xcos\theta + ysin\theta$ 算出 ρ 存成 Hough image，因為角度大於 90 度時 ρ 可能為負值，所以有往右平移 ρ max(142, 圖的對角線距離)



image



Hough image

Y 軸為 0~179 度

X 軸為 ρ

找到交集最多的點為 angle=0, ρ=172-142=30

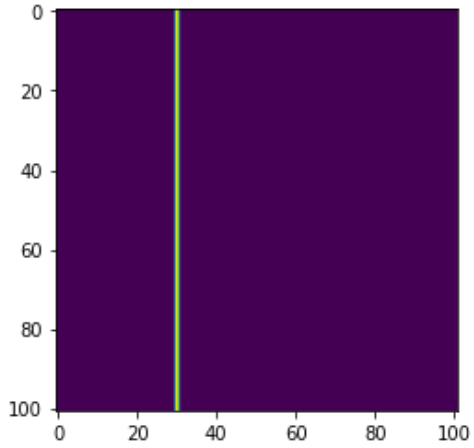
最後利用這個公式

$$\rho = xcos\theta + ysin\theta$$

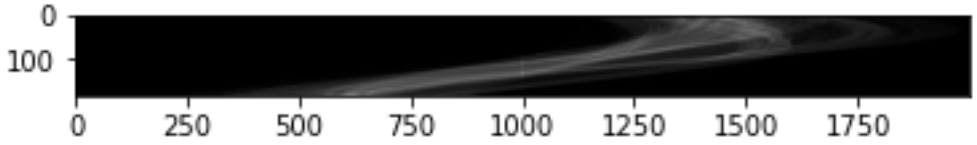
隨便找 x0,x1 帶入以上公式將 y0,y1 推出來

要注意除以 0 的 case（ cos(90°）or sin(0°））

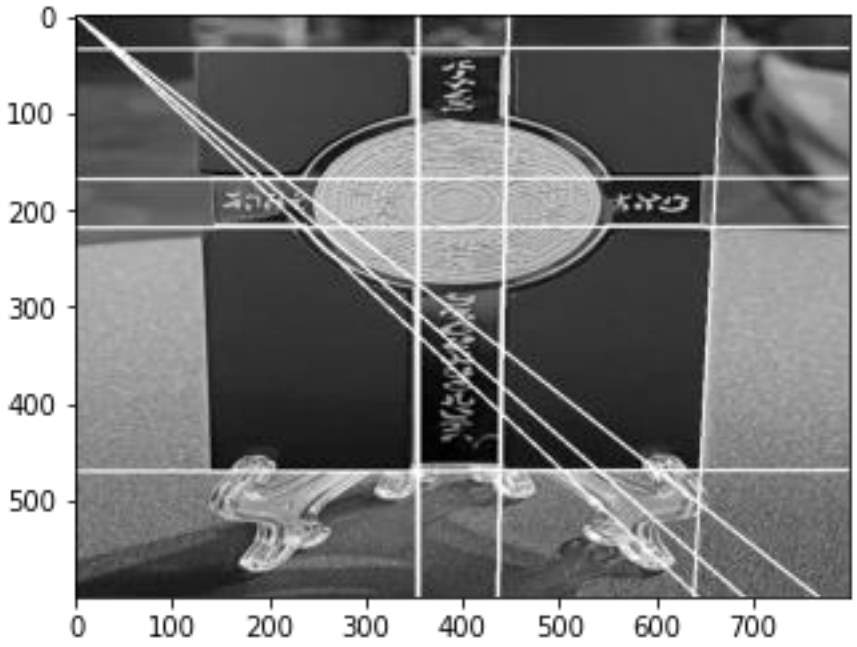
找到座標後將這兩點連起來畫成直線即可



Discussion of results:



Hough space



我只選了前 13 條交點最多的線，再多就會有很多斜對角的線(如上圖)

Problem 2: GEOMETRICAL MODIFICATION

(a) (10 pt) Please design a method to improve sample3.png.
Describe your method in detail and specify all the parameters.

1. 貓有白邊(需移除) 2. 頭太黑，腳太亮

	
Original image	Output image

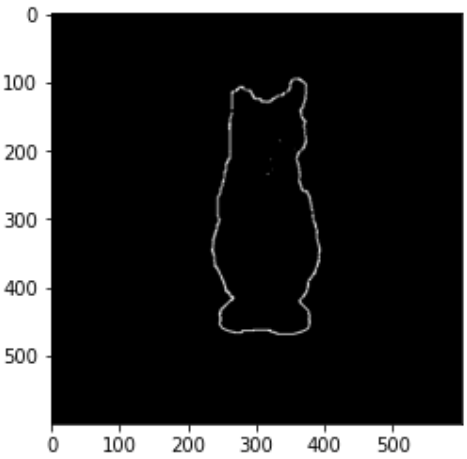
Motivation and approach (include parameters):

1. 去除貓的白邊

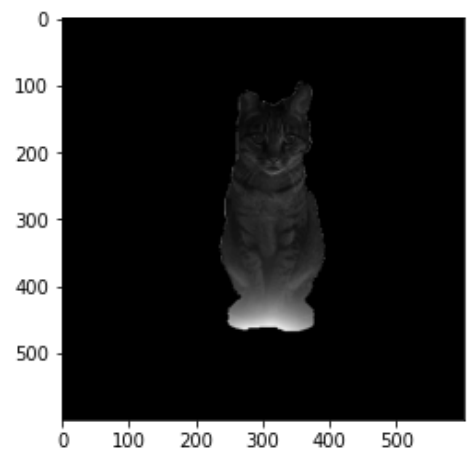
藉由 Sobel 的 edge detection 來偵測邊緣，之後將邊緣設為黑色，但因為偵測邊緣時會把貓臉也一起偵測進去，所以要設個 threshold 將貓臉濾

掉，但將非 255 的值濾掉後，眼睛的光還是在，因此我還特別把眼睛的區塊濾掉(170~185, 330~370)

```
T = 255
res = result1.copy()
res[res>=T] = 255
res[res<T] = 0
res[170:185,330:370] = 0
plt.imshow(res, cmap='gray')
```



白邊偵測



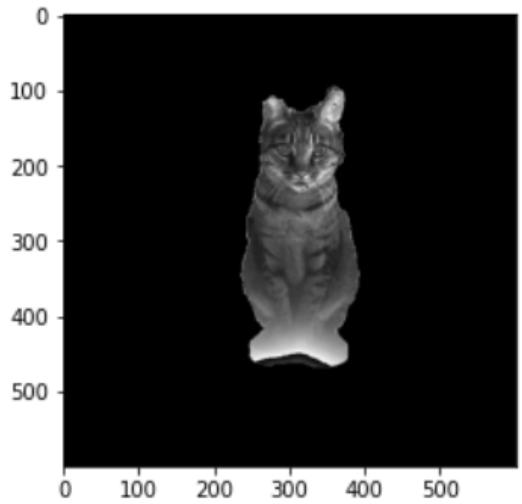
去掉白邊

2. 頭太黑，腳太亮

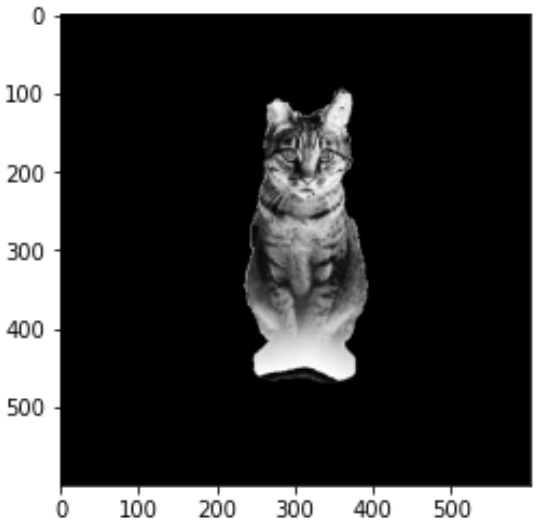
   經過觀察，我發現圖片隨著 row 越大，亮度就越大，因此我使用除法，隨著 row 變大，除的數值也變大

```
res3 = sample3.copy()
for i in range(res3.shape[0]):
    res3[i] = res3[i]/((i+1)/600)
plt.imshow(res3, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x1
```



最後為了增強對比，我使用了 hw1 學到的 global HE



Discussion of results:

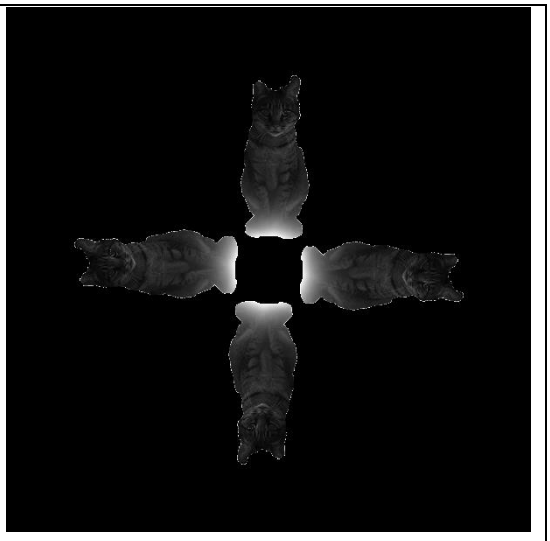
貓的上半部變得相當清楚，可惜的是腳的部分無法復原，可能是因為原圖的腳白到被 clip 掉了，因此無法還原數值;白邊的部分座標偵測可能沒有對到，因此無法將白邊完全去掉。

(b) (25 pt) The cat needs friends! Please design an algorithm to help her find new friends by making sample3.png become

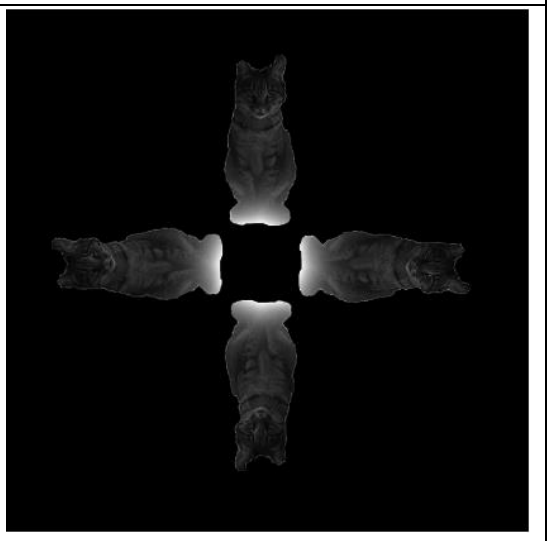
sample4.png. Output the result as result7.png with the same dimension as sample3.png. Please describe your method and implementation details clearly. (hint: you may perform rotation, scaling, translation, etc.)

Translation, scale, rotation

Original image



Output image



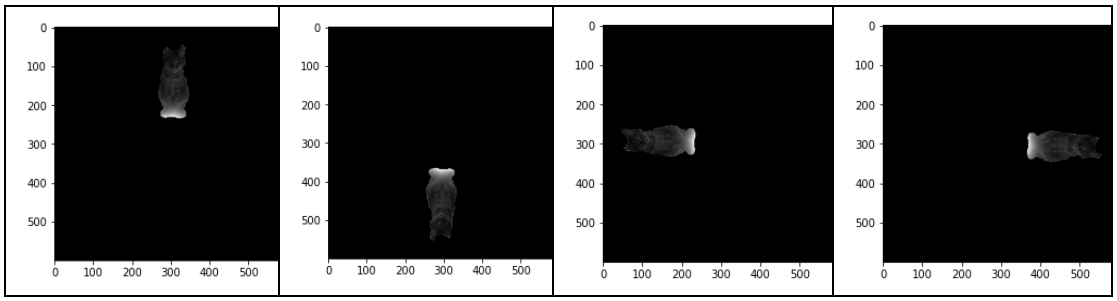
Sample4.png

Motivation and approach (include parameters):

- **translation** $\begin{bmatrix} x_k \\ y_j \end{bmatrix} = \begin{bmatrix} u_q \\ v_p \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$
- **scaling** $\begin{bmatrix} x_k \\ y_j \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} u_q \\ v_p \end{bmatrix}$
- **rotation** $\begin{bmatrix} x_k \\ y_j \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} u_q \\ v_p \end{bmatrix}$

先分別寫出 Translate, scale, rotate 的函數，產生上下左右的貓，再將四張圖相加

```
L = translate(scale(rotate(sample3,-90),1/2),0,-150)
R = translate(scale(rotate(sample3,90),1/2),300,-150)
U = translate(scale(rotate(sample3,0),1/2),150,0)
D = translate(scale(rotate(sample3,180),1/2),150,-300)
plt.imshow(L+R+U+D, cmap='gray')
```

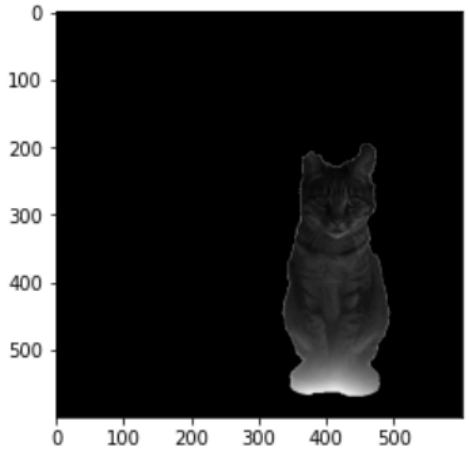


以第四張圖為例(R)，我先往右旋轉 90 度，再縮小 1/2，再右移 300，下移 150

一開始想像 ppt 的做法一樣用成座標矩陣相乘，但一直弄不好如何使用一個座標陣列座標選取 numpy 陣列，後來還是用 for 迴圈遍歷整個陣列，一個一個轉換，並且使用 Backward treatment，不用處理內插的問題，例如以下 translate 的功能。

```python
def translate(img,dx,dy):
    img2 = np.zeros(img.shape)
    h,w = img.shape
    for j in range(h):
        for k in range(w):
            u = j + dy
            v = k - dx
            if u<0 or u>=h or v<0 or v>=w: continue
            img2[j][k] = img[u][v]

    return img2
plt.imshow(translate(sample3,100,-100), cmap='gray')
```

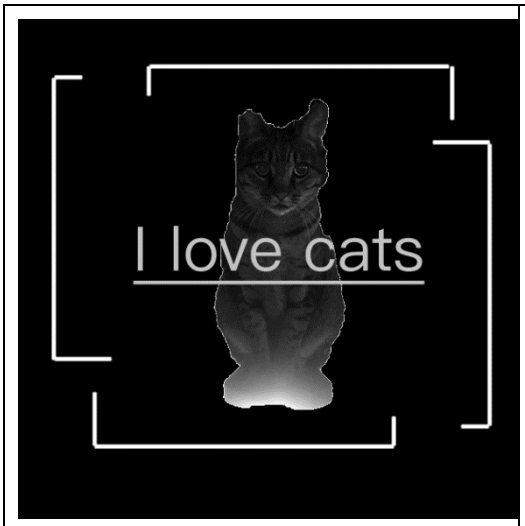
```
<matplotlib.image.AxesImage at 0x1865caf1a60>
```



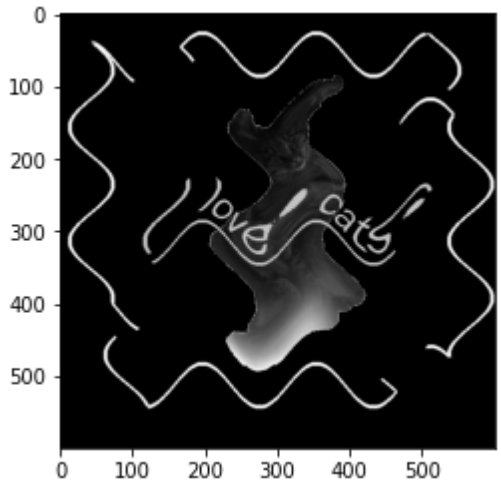
Discussion of results:

將每個功能分開寫再組合在一起，成功讓貓貓有很多朋友!

(c) (25 pt) Legend says that cats are a kind of liquid. Let's perform some magic tricks on the lovely cat to confirm this rumor. By observing the wave shown in sample6.png, please design an algorithm to make sample5.png look like it as much as possible and save the output as result8.png. Please describe the details of your method and also provide some discussions on the designed method, the result, and the difference between result8.png and sample6.png, etc.



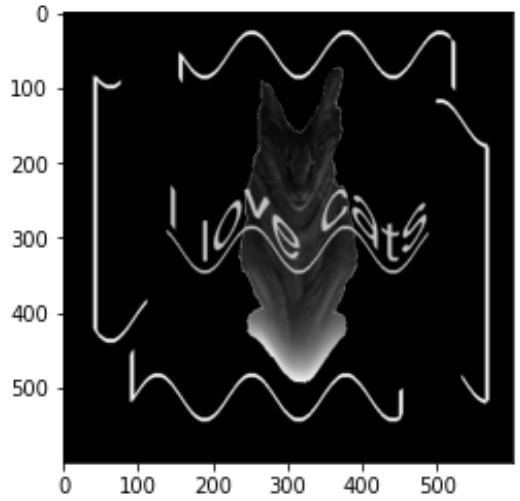
Original image



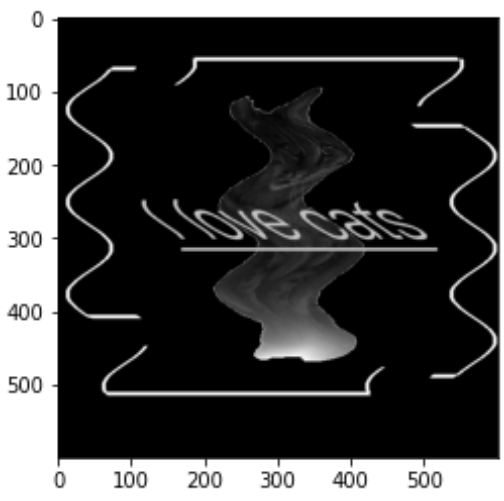
Output image

Motivation and approach (include parameters):

經由觀察，圖片的變形像是 cos or sin 波，因此我使用 cos 和 sin 去做計算



垂直方向變形



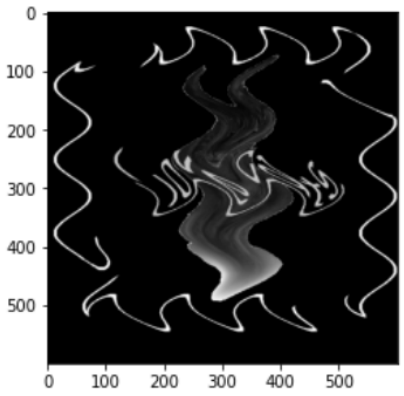
水平方向變形

Discussion of results:

將兩者結合後，變形會跌加

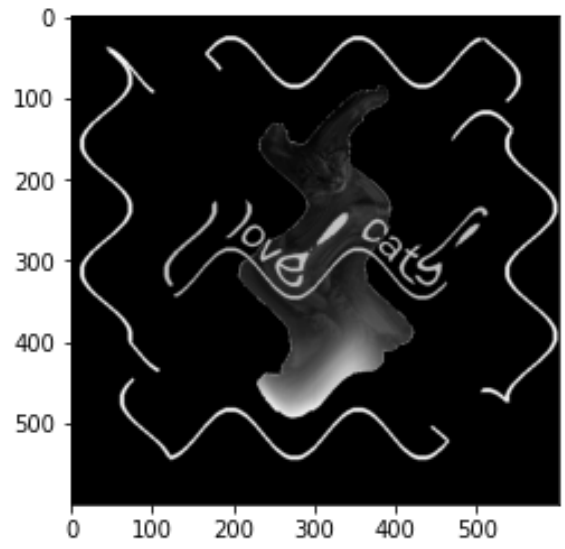
```
plt.imshow(liquid_cat_horizontal(liquid_cat_verticle(sample5,20,30),20,30), cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x1866a857f70>
```

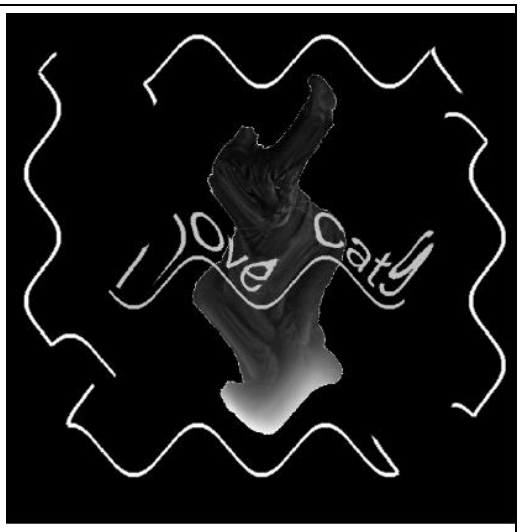


所以我最後改成把兩個方向同時做變換

```
def liquid_cat(img, wave_length, Wave_crest):
    h,w = img.shape
    img2 = np.zeros((h,w))

    for j in range(h):
        for k in range(w):
            shif_v = np.cos(j/wave_length)*Wave_crest
            shif_u = np.sin(k/wave_length)*Wave_crest
            u = j + shif_u
            v = k + shif_v
            u = int(u)
            v = int(v)
            if u<0 or u>=h or v<0 or v>=w: continue
            img2[j][k] = img[u][v]

    return img2

plt.imshow(liquid_cat(sample5,25,30), cmap='gray')
```



result8.png



sample6.png