

## DIP Homework #2

ID #: B04902028

Department & grade: CSIE 3

Name: 洪浩翔

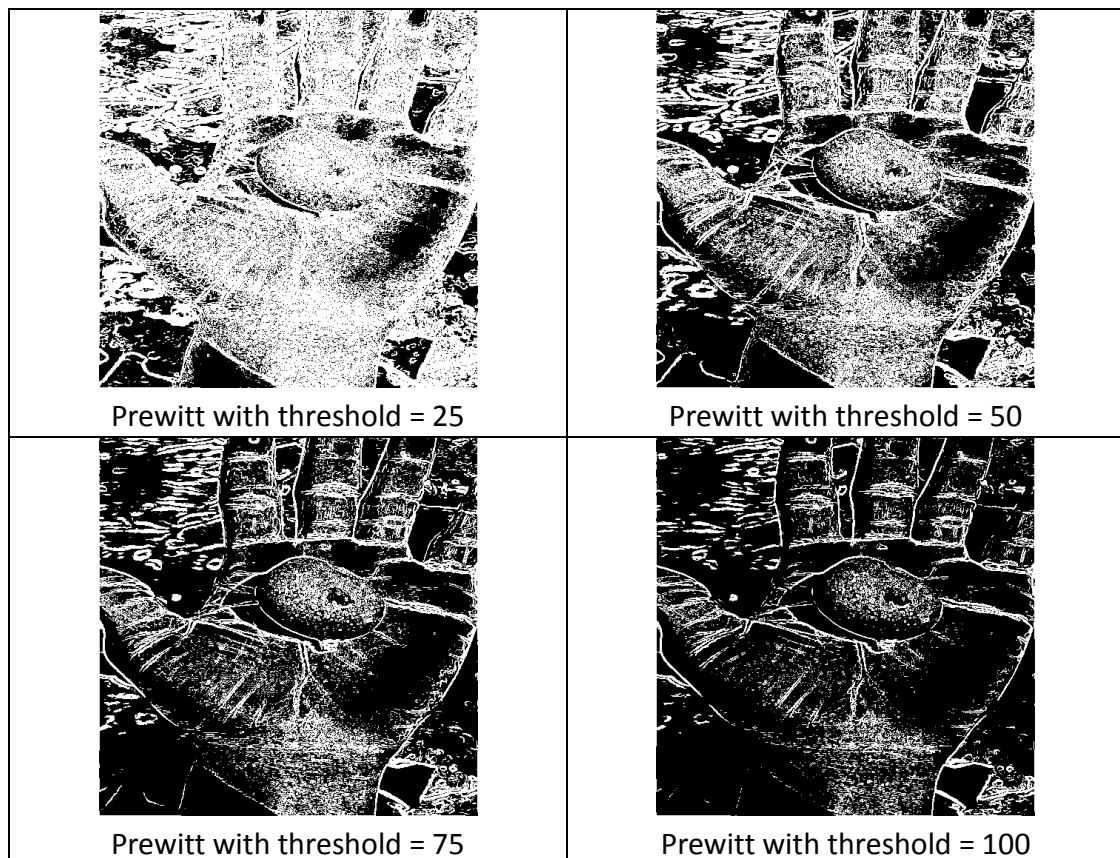
Email: b04902028@ntu.edu.tw



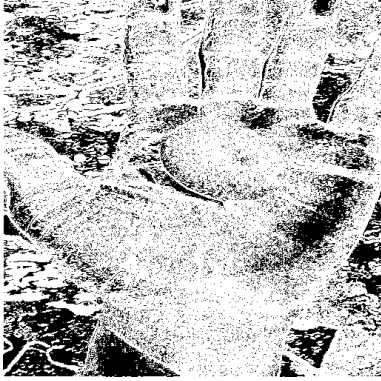
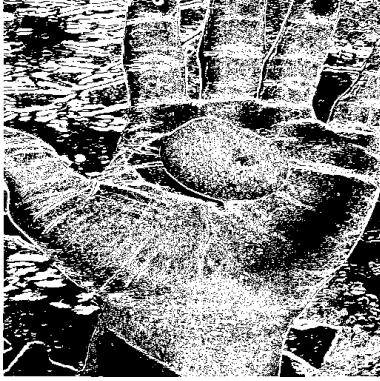




Submit time: 04/11/2018

### PROBLEM 1: EDGE DETECTION

(a) Given an image I1 as show in Fig. 1(a), please perform 1<sup>st</sup> order edge detection, 2<sup>nd</sup> order edge detection, and Canny edge detection to obtain corresponding edge maps. Please describe each method in detail, specify each parameter clearly and discuss how each of them affects the resultant edge map. What are pros and cons of each method?

#### 1<sup>st</sup> order edge detection:



 <p>Prewitt with threshold = 125</p>	 <p>Prewitt with threshold = 150</p>
 <p>Sobel with threshold = 25</p>	 <p>Sobel with threshold = 50</p>
 <p>Sobel with threshold = 75</p>	 <p>Sobel with threshold = 100</p>
 <p>Sobel with threshold = 125</p>	 <p>Sobel with threshold = 150</p>

For 1<sup>st</sup> order edge detection, I implement Prewitt and Sobel to display the results. Prewitt use the gradient like:

-1	-1	-1
1	1	1

$p_1$

-1		1
-1		1
-1		1

$p_2$

to implement the 1<sup>st</sup> order edge detection, while Sobel use the gradient like:

-1	-2	-1
1	2	1

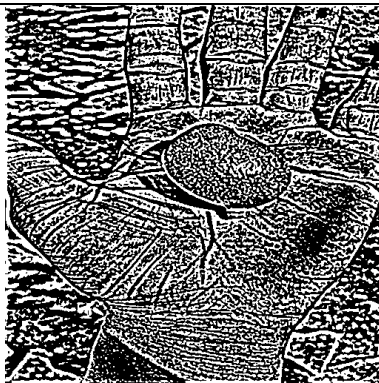
$s_1$

-1		1
-2		2
-1		1

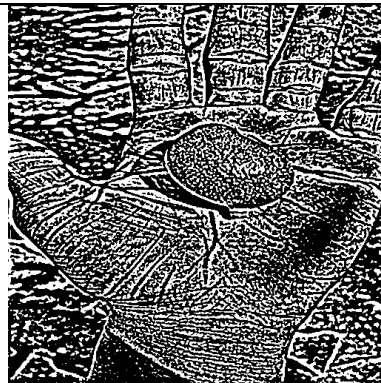
$s_2$

to implement the edge detection. I calculate the gradient result of each pixel and determine it is edge or not by the threshold. The use of threshold is very simple. If the gradient value is larger than the threshold, then it is possible to be edge. Therefore, the threshold value is the only one effect the edge detection result. If just compare these two methods with the same threshold, it seems that they are similar when the threshold is small. However, when the threshold goes high, Prewitt seems to remove more little details, or saying, noises, comparing with Sobel. This causes the result of Prewitt cleaner, or on the other hand, less little edge details in the end. I think the reason is because of the method of gradient. Sobel takes more weight of the central cross. As a result, the same threshold may filter less pixels to the dark side. If comparing just thresholds, it is obvious that larger threshold causes the result cleaner and less details meanwhile. The small threshold causes the detailed edge result, but full of little pieces of edge, or saying noises some part.

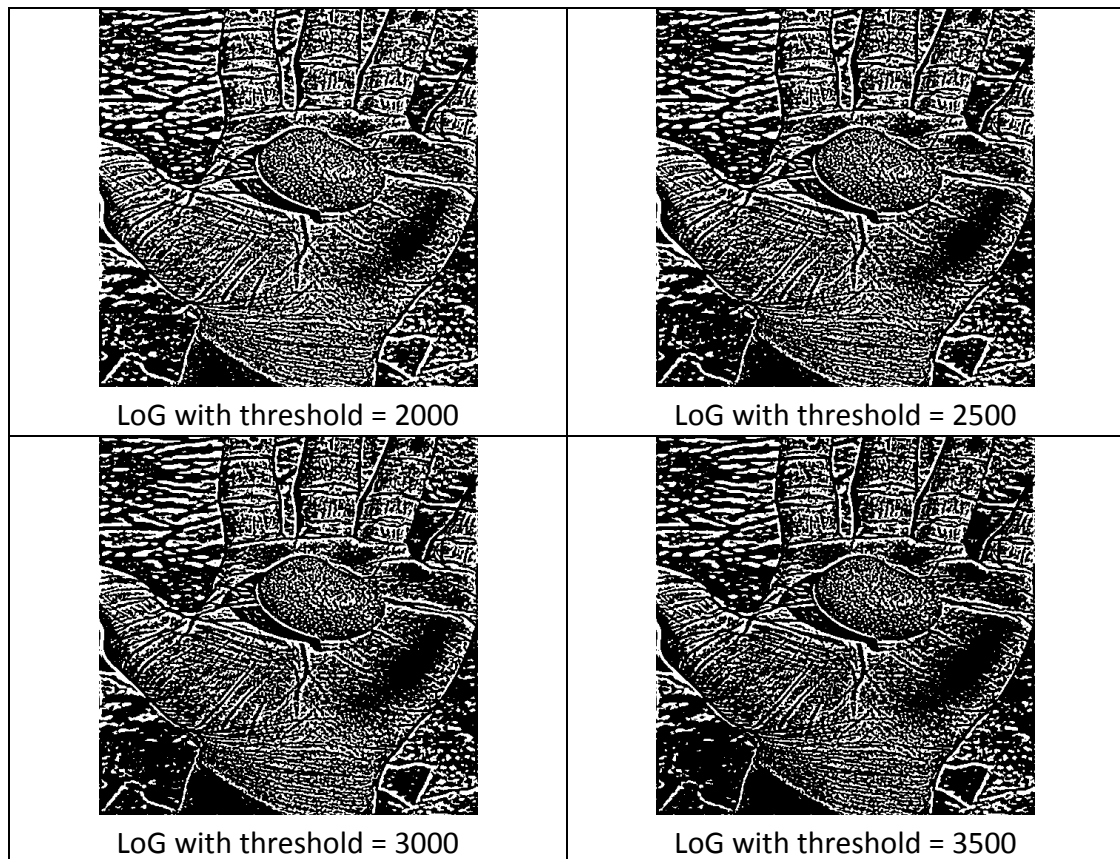
## 2<sup>nd</sup> order edge detection:



LoG with threshold = 1000



LoG with threshold = 1500







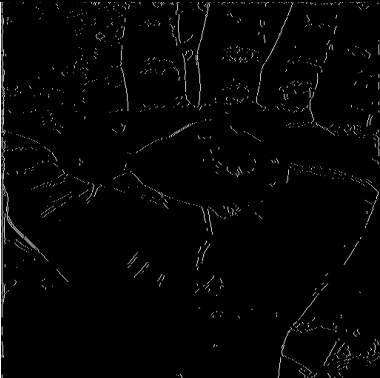
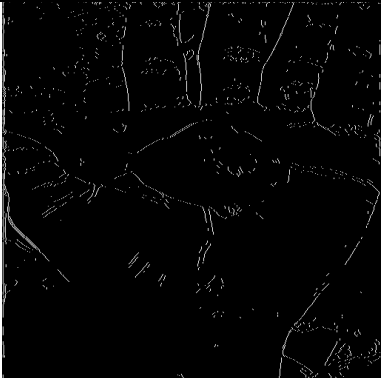
For 2<sup>nd</sup> order edge detection, I implement LoG (Laplacian of Gaussian) mainly. LoG uses the kernel like:

0	0	0	-1	-1	-2	-1	-1	0	0	0
0	0	-2	-4	-8	-9	-8	-4	-2	0	0
0	-2	-7	-15	-22	-23	-22	-15	-7	-2	0
-1	-4	-15	-24	-14	-1	-14	-24	-15	-4	-1
-1	-8	-22	-14	52	103	52	-14	-22	-8	-1
-2	-9	-23	-1	103	178	103	-1	-23	-9	-2
-1	-8	-22	-14	52	103	52	-14	-22	-8	-1
-1	-4	-15	-24	-14	-1	-14	-24	-15	-4	-1
0	-2	-7	-15	-22	-23	-22	-15	-7	-2	0
0	0	-2	-4	-8	-9	-8	-4	-2	0	0
0	0	0	-1	-1	-2	-1	-1	0	0	0

to implement the second order edge detection. I calculate the gradient result of each pixel and check if the zero-crossing happened, and determine whether it is edge or not. The threshold is used to check the distance of determining the zero crossing. For example, two pixels A and B are neighbor. If pixel A is larger than the threshold, and the negative of pixel B is larger than threshold as well, then it is possible that there is an edge between A and B. Because the kernel is the same in these pictures, the only fact to affect the results is the threshold. When threshold is small, it seems that the image is full of edges. There are too many details. When threshold goes high, there

are much more dark pixels, and the edge map seems to be better because of the removing of some unneeded details. This is easy to find the reason if consider the meaning of threshold: when the threshold becomes larger, it is not easy to pass the restricted distance because of the threshold.

**Canny edge detection:**

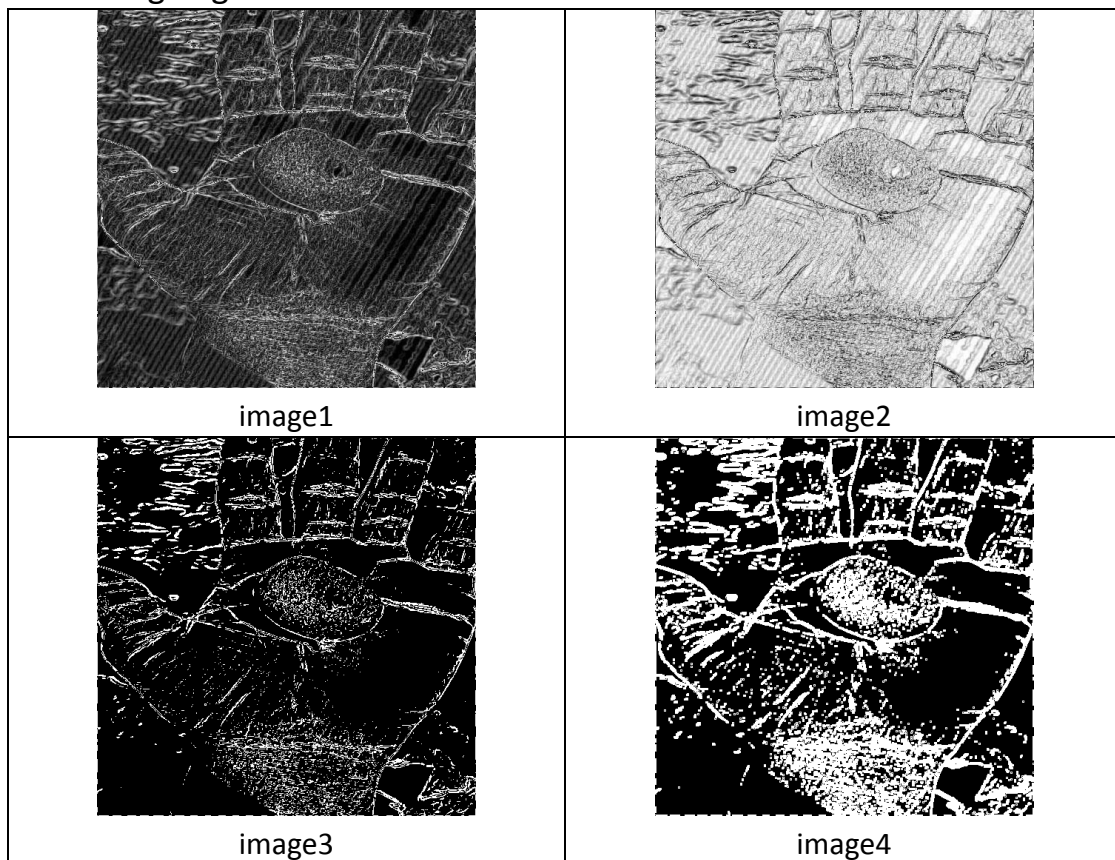
 <p>Canny with high threshold = 30 and low threshold = 20</p>	 <p>Canny with high threshold = 50 and low threshold = 40</p>
 <p>Canny with high threshold = 70 and low threshold = 10</p>	 <p>Canny with high threshold = 70 and low threshold = 60</p>
 <p>Canny with high threshold = 100 and low threshold = 10</p>	 <p>Canny with high threshold = 100 and low threshold = 60</p>

The implementation of Canny edge detection is following the method in class with the gradient method being the same as the Prewitt method. Therefore, the only parameter is the threshold, high threshold and low threshold. These two thresholds are used for detection as well. The value higher than high threshold will be labeled '2', higher than low threshold but lower than high threshold will be labeled '1', and lower than low threshold will be labeled nothing. If the label is '2', then the pixel is a part of edge. If label is '1' but there is a '2' next to it, the pixel is edge as well. The rest is not the edge. To find the effect the thresholds can produce, I try two kind of resultant images: two thresholds are closed and are far. If the not closed thresholds compared with the closed one, we can find that the effect is very small in this case. The difference is only appearing on some little parts which are not important. Therefore, two thresholds with large period seem to have very little effect if comparing with the closed one. If just comparing the thresholds, the smaller threshold pairs maintain more small details, while the larger one produce a smoother resultant image with less details. The reason is also very trivial. Larger threshold pairs will filter much more pixels. That is why different threshold pairs can have this result.

### **Pros and cons:**

	1 <sup>st</sup> order	2 <sup>nd</sup> order	canny
pros	1. easy to implement 2. fastest 3. still have other methods to implement (variety)	1. better performance on main edges 2. much bearable to noise than 1 <sup>st</sup> order 3. still not hard to implement	1. the best Performance of edge detection 2. noise tolerance
cons	If the input image is not smooth, it is easy to catch unneeded details (noises).	1. maintain too many unneeded small edges 2. consuming much time than 1 <sup>st</sup> order	1. slowest 2. complex 3. hard to optimize thresholds

(b) Given an image I2 with periodic noise as shown in Fig. 1(b), please design your own method to generate the edge map by avoiding obtaining edges of the noise.

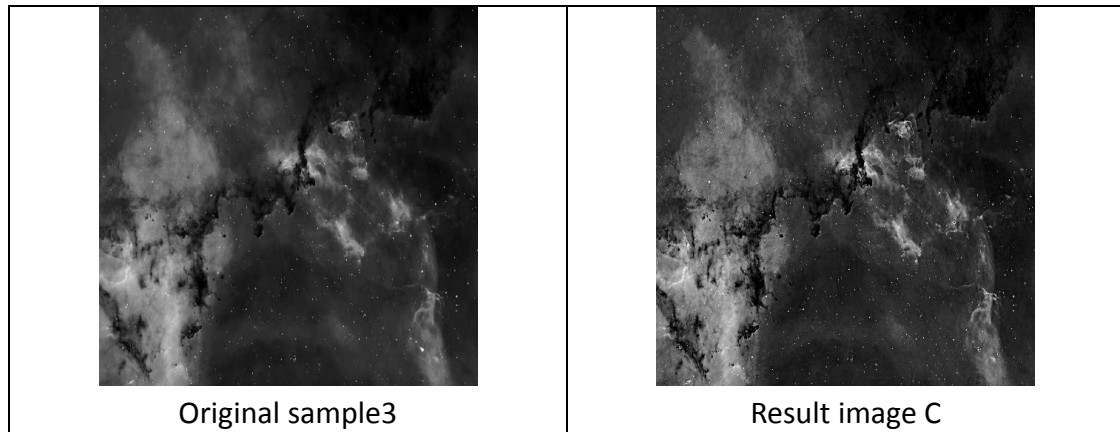


If just focus on the result, it seems that Canny edge detection is a good choice. However, it is still an interesting thing to find a good way by myself.

My method is very simple. I use gradient result to generate edge map directly. It can be divided into 4 parts. First, I find the 1<sup>st</sup> order gradient of the image and get the result image1. Then, use  $255 - \text{image1}$  to produce image2. Third, take a threshold = 135 to produce a binary image and use 255 to subtract it again to convert it back. The result of this step is image3. The last step is dilation operation to make the edge much clear, and the result is image4. This step is optional. It is nearly no effect of periodic noise in image4/image3. It some part produces a good edge map in the end.

## PROBLEM 2: GEOMETRICAL MODIFICATION

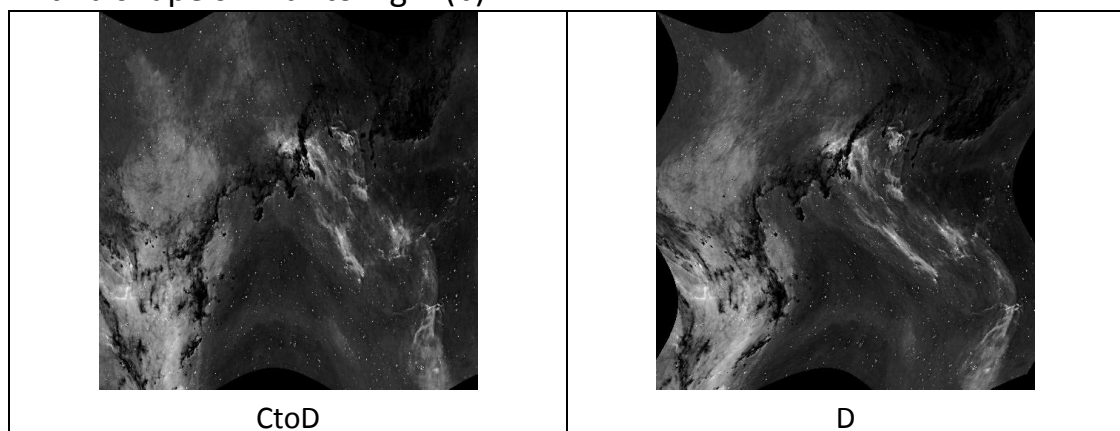
(a) Please perform edge crispening on I3 and denote the result as C. Show the parameters adopted and provide some discussions on the result as well.



During sharpening, I use 9x9 box filter as the low-pass filter. In addition, I use  $c = 0.7$  as my coefficient. The rest steps are the same as what mentioning in class.

If comparing the result with the original image. The details in C are promoted to be much clear, including the edge of every object. Everything in C seems to be very distinct, so does some unnecessary details. The objects in original image sample3 seems to have a level of front and back, but in C, everything is just like being put on the same plane. Although C is much clear, the feeling of it is not as real as that of sample3. However, C indeed makes some small but important parts much distinct.

(b) Please design a warping function to convert the image C to image D with a shape similar to Fig. 2(b).

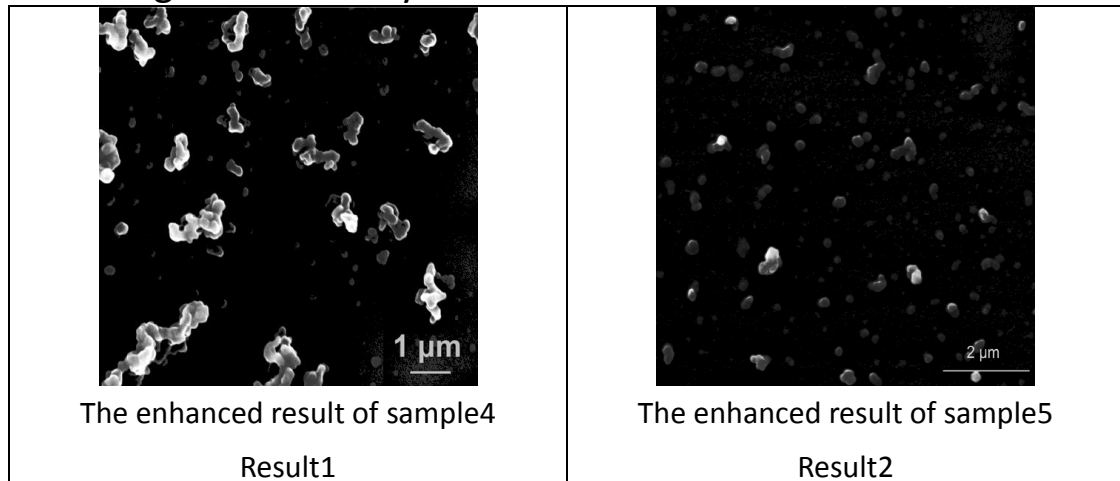


First, I use the formula  $\text{new\_x} = x + \sin(y/50) * 30$  to update the x-axis of every pixel and get the result CtoD. After complete the 1<sup>st</sup> step, I use the formula  $\text{new\_y} =$



$y + \sin(x/50) * 30$  to update the y-axis of every pixel and get the result D. Then, the warping is done.

[Bonus] Please design an algorithm to enhance the following two images as best as you can.



To enhance sample4, first I use exponential transform on the image. The transform formula is  $f(x) = x^5$ ,  $0 \leq x \leq 1$ . Then, I set threshold = 15 to let the pixel value lower than 15 become 0. This step is to remove some pixel with lower intensity to 0. The second step is to do a log transform. The formula of the transform is  $f(x) = 255 / \log(1+r) * \log(1+x)$ ,  $0 \leq x \leq 1$  and  $r$  is the max value in the image after exponential transform. This step is to enhance the image, especially the pixels with higher intensity. As for the pixels being set to 0 in previous step, they will not change because their value is 0. As a result, these two steps remove some unnecessary pixels. The third step is histogram equalization. This step is set to enhance the image as well. The forth step is add the result image and the original image sample4, and divide the result by 2. Then, set two thresholds to remove some unnecessary pixels again. The first threshold is for intensity, while the second one is for column. If  $y < 384$  and the value  $< 40$ , then set the pixel to 0. If  $y \geq 384$  and value  $< 45$ , then set the pixel to 0. This step is to make the background black, remaining the details we want. The reason why set a threshold for  $y$  is because the right side of the original image is brighter than the left side. That is why I need a  $y$  threshold to increase the power of the original intensity threshold. In the end, result1 is produced.

The method to promote sample5 is almost the same. The threshold in the first step is set to 10 here because average intensity of sample5 is lower than that of sample4. In the last step,  $y$  threshold is not necessary here because sample5 doesn't

have the condition like sample4. In addition, intensity threshold is set to 21 here. The rest is the same. In the end, result2 is produced.

## **README**

# DIP Homework Assignment #2

# 04/11/2018

# Name: 洪浩翔

# ID: B04902028

# email: b04902028@ntu.edu.tw

# compiled on ubuntu 16.04.3 LTS with g++11

# output name and file is decided, so no need to change

# image path should be "../raw/\*.raw" and output image will be here as well

# type make -f README to compile and run

# below is for makefile

.PHONY: all

CC=g++

LN=g++

All: prob1 prob2

prob1 :

    @echo "hw2"

    @echo "compiling and linking the code"

    \$(CC) -std=c++11 -c hw2.cpp

    \$(LN) -std=c++11 -o hw2 hw2.o

    @echo "running the program"

    ./hw2

prob2 :

    @echo "hw2-1"

    @echo "compiling and linking the code"

    \$(CC) -std=c++11 -c hw2-1.cpp

    \$(LN) -std=c++11 -o hw2-1 hw2-1.o

    @echo "running the program"

    ./hw2-1