HW7

*Date: 2021. 11. 3*

*Student ID: 21600635*

*Name: Bomoon JUNG*

1. **Homework 7**

1) Develop a program which performs various kinds of thresholding

* Use ‘finger\_print.png’, ‘adaptive\_1.jpg’, ‘adaptive.png’.
* Read all images as gray scale images.
* Select and apply a threshold method for each image.
* For ‘finger\_print.png’, set fingerprint region to 0 and background region to 255.
* For ‘adaptive\_1.jpg’, and ‘adaptive.png’, set character region to 0 and background region to 255.
* Display results on three windows
* Windows name: ‘finger\_print’, ‘adaptive\_1’, and ‘adaptive’
* In your report, include explanations of why, for each image, you used those algorithms, argument values, and so forth.

2) explanation

1. finger\_print.png: In this picture, it is easy to distinguish between the background and the object. Therefore, thresholding can be performed easily with one threshold. Therefore, the Basic method of Global Thresholding was used. Basic method sets thresh\_T first. And after making two groups using thresh\_T, find the average of each group. Then, this step is repeated until a suitable thresh\_T is reached. Finally, input and output are assigned to the threshold function, and set as THRESH\_BINARY.

텍스트이(가) 표시된 사진

자동 생성된 설명

Figure 1. result of finger\_print

2. adaptive\_1: If you look at the bottom right of this picture, you can see that it is a little dark. Therefore, using local thresholding, the threshold is determined by considering adjacent pixels. Therefore, much better thresholding can be implemented. Therefore, I used the adaptiveThreshold function, and input image, output image, max value, method, threshold type, block size of an appropriate size, and constant are inserted in order from the first argument. At this time, the threshold type is set to ‘THRESH\_BINARY’, so the pixel value is set as the max value for the background that does not exceed the threshold.

텍스트이(가) 표시된 사진

자동 생성된 설명

Figure 2. result of adaptive\_1

3. adaptive: Banding noise is present in this figure. Therefore, local thresholding should be used for each column. In that case, much better thresholding can be implemented because the threshold is determined by considering the intensity for each column. Therefore, the adaptiveThreshold function is used in the same way as 'adaptive\_1', but the difference is that thresholding is performed for each column of the image. And by inserting the block size as much as the number of rows, thresholding is taken precisely in column unit. Therefore, banding noise can be solved.

텍스트이(가) 표시된 사진

자동 생성된 설명

Figure 3. result of adaptive

3) Source code

*// g++ $(pkg-config --cflags --libs opencv) HW7\_21600635.cpp -o ./a*

*#include* <opencv2/opencv.hpp>

*#include* <iostream>

*#include* <string>

using *namespace* cv ;

using *namespace* std ;

Mat

basic (Mat src) {

*int* thresh\_T = 200 ;

*int* th = 10 ;

*int* low\_cnt = 0 ;

*int* high\_cnt = 0 ;

*int* low\_sum = 0 ;

*int* high\_sum = 0 ;

Mat image = imread("./resources/Lena.png", 0) ;

*while*(1) {

*for*( *int* j = 0 ; j < src.rows ; j++ ) {

*for*( *int* i = 0 ; i < src.cols ; i++ ) {

*if*( src.at<uchar>(j, i) < thresh\_T) {

low\_sum += src.at<uchar>(j, i) ;

low\_cnt++ ;

}

*else* {

high\_sum += src.at<uchar>(j, i) ;

high\_cnt++ ;

}

}

}

*if*( abs(thresh\_T - (low\_sum / low\_cnt + high\_sum / high\_cnt) / 2.0f) < th) {

*break* ;

}

*else* {

thresh\_T = (low\_sum / low\_cnt + high\_sum / high\_cnt) / 2.0f ;

low\_cnt = high\_cnt = low\_sum = high\_sum = 0 ;

}

}

Mat thresh ;

threshold(src, thresh, thresh\_T, 255, THRESH\_BINARY) ;

*return* thresh ;

}

Mat

local\_adaptive (Mat src) {

Mat adaptive\_binary ;

adaptiveThreshold(src, adaptive\_binary, 255, ADAPTIVE\_THRESH\_MEAN\_C, THRESH\_BINARY, 85, 15) ;

*return* adaptive\_binary ;

}

Mat

local\_adaptive\_partition (Mat src) {

*int* mask = src.rows ;

*for* ( *int* i = 0 ; i < src.cols ; i ++ ) {

Rect rect(i, 0, 1, src.rows) ;

adaptiveThreshold(src(rect), src(rect), 255, ADAPTIVE\_THRESH\_MEAN\_C, THRESH\_BINARY, mask, 15) ;

}

*return* src ;

}

*int*

main() {

Mat finger\_print = imread("HW7\_resources/finger\_print.png", 0) ;

Mat adaptive\_1 = imread("HW7\_resources/adaptive\_1.jpg", 0) ;

Mat adaptive = imread("HW7\_resources/adaptive.png", 0) ;

*if* ( finger\_print.empty() | adaptive\_1.empty() | adaptive.empty() ) {

cout << "no such file" << endl ;

*return* 0 ;

}

finger\_print = basic(finger\_print) ;

adaptive\_1 = local\_adaptive(adaptive\_1) ;

adaptive = local\_adaptive\_partition(adaptive) ;

imshow("finger\_print", finger\_print) ;

imshow("adaptive\_1", adaptive\_1) ;

imshow("adaptive", adaptive) ;

waitKey(0) ;

*return* 0 ;

}