NATIONAL UNIVERSITY OF SINGAPORE

COLLEGE OF DESIGN AND ENGINEERING

CA - AY 2022/2023, SEMESTER 1

# Assignment for ME5405 Machine Vision

# Computing Project

BY:

Group 33

Jiang Juncheng, Wang Longfei, Wang Renjie

Date: 31/10/2022

# Contents

## Project Description

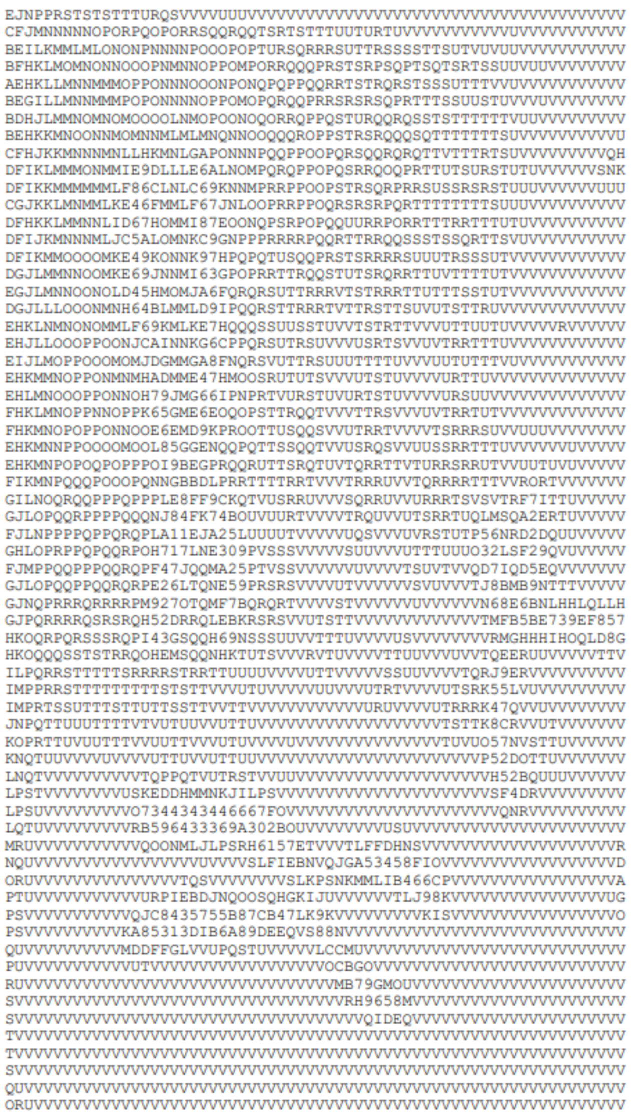
This is a computing project of ME5405 Machine Vision, and the software is required to be developed using MATLAB. We are encouraged to rely on our own implementations. Otherwise, MATLAB's Imaging Toolbox is also allowed.

This project involves processing two images, each consisting of multiple subtasks. Thus, this report will introduce the processing of each image and the completion of its subtasks, respectively.

## Image 1: Chromosomes

### 1. Introduction

Image 1 is a 64x64 32-level image. It is a coded array that contains an alphanumeric character for each pixel. The range of these characters is 0-9 and A-V, which corresponds to 32 levels of gray. The text file of Image 1 is shown in Figure 1.1



```
EJNPPRSTSTSTTTURQSVVVVUUUVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
CFJMNNNNNOPORPQOPORRSQQRQQTSRTSTTTUUTURTUVVVVVVVVVVVUVVVVVVVVVVV
BEILKMMLMLONONPNNNNPOOOPOPTURSQRRRSUTTRSSSSTTSUTVUVUUVVVVVVVVVVV
BFHKLMOMNONNOOOPNMNNOPPOMPORRQQQPRSTSRPSQPTSQTSRTSSUUVUUVVVVVVVV
AEHKLLMNNMMMOPPONNNOOONPONQPQPPQQRRTSTRQRSTSSSUTTTVVUVVVVVVVVVVV
BEGILLMNNMMMPOPONNNNNOPPOMOPQRQQPRRSRSRSQPRTTTSSUUSTUVVVUVVVVVVV
BDHJLMMNOMNOMOOOOLNMOPOONOQORRQPPQSTURQQRQSSTSTTTTTTVUUVVVVVVVVV
BEHKKMNOONNMOMNNMLMLMNQNNOOQQQROPPSTRSRQQQSQTTTTTTTSUVVVVVVVVVVU
CFHJKKMNNNMNLLHKMNLGAPONNNPQQPPOOPQRSQQRQRQTTVTTTRTSUVVVVVVVVVQH
DFIKLMMMONMMIE9DLLLE6ALNOMPQRQPPOPQSRRQOQPRTTUTSURSTUTUVVVVVVSNK
DFIKKMMMMMMLF86CLNLC69KNNMPRRPPOOPSTRSQRPRRSUSSRSRSTUUUVVVVVVUUU
CGJKKLMNMMLKE46FMMLF67JNLOOPRRPPOQRSRSRPQRTTTTTTTTSUUUVVVVVVVVVV
DFHKKLMMNNLID67HOMMI87EOONQPSRPOPQQUURRPORRTTTRRTTTUTUVVVVVVVVVV
DFIJKMNNNMLJC5ALOMNKC9GNPPPRRRRPQQRTTRRQQSSSTSSQRTTSVUVVVVVVVVVV
DFIKMMOOOOMKE49KONNK97HPQPQTUSQQPRSTSRRRRSUUUTRSSSUTVVVVVVVVVVVV
DGJLMMNNOOMKE69JNNMI63GPOPRRTTRQQSTUTSRQRRTTUVTTTTUTVVVVVVVVVVVV
EGJLMNNOONOLD45HMOMJA6FQRQRSUTTRRRVTSTRRRTTUTTTSSTUTVVVVVVVVVVVV
DGJLLLOOONMNH64BLMMLD9IPQQRSTTRRRTVTTRSTTSUVUTSTTRUVVVVVVVVVVVVV
EHKLNMNONOMMLF69KMLKE7HQQQSSUUSSTUVVTSTRTTVVVUTTUUTUVVVVVRVVVVVV
EHJLLOOOPPOONJCAINNKG6CPPQRSUTRSUVVVUSRTSVVUVTRRTTTUVVVVVVVVVVVV
EIJLMOPPOOOMOMJDGMMGA8FNQRSVUTTRSUUUTTTTUVVVUUTUTTTVUVVVVVVVVVVV
EHKMMNOPPONMNMHADMME47HMOOSRUTUTSVVVUTSTUVVVVURTTUVVVVVVVVVVVVVV
EHLMNOOOPPONNOH79JMG66IPNPRTVURSTUVURTSTUVVVVURSUUVVVVVVVVVVVVVV
FHKLMNOPPNNOPPK65GME6EOQOPSTTRQQTVVVTTRSVVVUVTRRTUTVVVVVVVVVVVVV
FHKMNOPOPPONNOOE6EMD9KPROOTTUSQQSVVUTRRTVVVVTSRRRSUVVUUUVVVVVVVV
EHKMNNPPOOOOMOOL85GGENQQPQTTSSQQTVVUSRQSVVUUSSRRTTTUVVVVVVUVVVVV
EHKMNPOPOQPOPPPOI9BEGPRQQRUTTSRQTUVTQRRTTVTURRSRRUTVVUUTUVUVVVVV
FIKMNPQQQPOOOPQNNGBBDLPRRTTTTRRTVVVTRRRUVVTQRRRRTTTVVRORTVVVVVVV
GILNOQRQQPPPQPPPLE8FF9CKQTVUSRRUVVVSQRRUVVURRRTSVSVTRF7ITTUVVVVV
GJLOPQQRPPPPQQQNJ84FK74BOUVUURTVVVVTRQUVVUTSRRTUQLMSQA2ERTUVVVVV
FJLNPPPPQPPQRQPLA11EJA25LUUUUTVVVVVUQSVVVUVRSTUTP56NRD2DQUUVVVVV
GHLOPRPPQPQQRPOH717LNE309PVSSSVVVVVSUUVVVUTTTUUUO32LSF29QVUVVVVV
FJMPPQQPPPQQRQPF47JQQMA25PTVSSVVVVVVUVVVVTSUVTVVQD7IQD5EQVVVVVVV
GJLOPQQPPQQRQRPE26LTQNE59PRSRSVVVVUTVVVVVVSVUVVVTJ8BMB9NTTTVVVVV
GJNQPRRRQRRRRPM927OTQMF7BQRQRTVVVVSTVVVVVVUVVVVVVN68E6BNLHHLQLLH
GJPQRRRRQSRSRQH52DRRQLEBKRSRSVVUTSTTVVVVVVVVVVVVVTMFB5BE739EF857
HKOQRPQRSSSRQPI43GSQQH69NSSSUUVVTTTUVVVVUSVVVVVVVVRMGHHHIHOQLD8G
HKOQQQSSTSTRRQOHEMSQQNHKTUTSVVVRVTUVVVVTTUUVVVUVVTQEERUUVVVVVTTV
ILPQRRSTTTTTSRRRRSTRRTTUUUUVVVVUTTVVVVVSSUUVVVVTQRJ9ERVVVVVVVVVV
IMPPRRSTTTTTTTTTSTSTTVVVUTUVVVVVUUVVVUTRTVVVVUTSRK55LVUVVVVVVVVV
IMPRTSSUTTTSTTUTTSSTTVVTTVVVVVVVVVVVVURUVVVVUTRRRK47QVVUVVVVVVVV
JNPQTTUUUTTTTVTVUTUUVVUTTUVVVVVVVVVVVVVVVVVVVTSTTK8CRVVUTVVVVVVV
KOPRTTUVUUTTTVVUUTTVVVUTUVVVVUVVVVVVVVVVVVVVVTUVUO57NVSTTUVVVVVV
KNQTUUVVVVUVVVVUTTUVVUTTUUVVVVVVVVVVVVVVVVVVVVVVP52DOTTUVVVVVVVV
LNQTVVVVVVVVVVVTQPPQTVUTRSTVVUUVVVVVVVVVVVVVVVVVVH52BQUUUVVVVVVV
LPSTVVVVVVVVUSKEDDHMMNKJILPSVVVVVVVVVVVVVVVVVVVVSF4DRVVVVVVVVVVV
LPSUVVVVVVVVVO7344343446667FOVVVVVVVVVVVVVVVVVVVVQNRVVVVVVVVVVVV
LQTUVVVVVVVVVRB596433369A302BOUVVVVVVVVUSUVVVVVVVVVVVVVVVVVVVVVV
MRUVVVVVVVVVVQOONMLJLPSRH6157ETVVVTLFFDHNSVVVVVVVVVVVVVVVVVVVVVR
NQUVVVVVVVVVVVVVVVVVUVVVVSLFIEBNVQJGA53458FIOVVVVVVVVVVVVVVVVVVD
ORUVVVVVVVVVVVVVVVTQSVVVVVVVVSLKPSNKMMLIB466CPVVVVVVVVVVVVVVVVVA
PTUVVVVVVVVVVVURPIEBDJNQOOSQHGKIJUVVVVVVTLJ98KVVVVVVVVVVVVVVVVUG
PSVVVVVVVVVVVQJC8435755B87CB47LK9KVVVVVVVVVKISVVVVVVVVVVVVVVVVVO
PSVVVVVVVVVVKA85313DIB6A89DEEQVS88NVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
QUVVVVVVVVVVMDDFFGLVVUPQSTUVVVVVLCCMUVVVVVVVVVVVVVVVVVVVVVVVVVVV
PUVVVVVVVVVVVUTVVVVVVVVVVVVVVVVVVOCBGOVVVVVVVVVVVVVVVVVVVVVVVVVV
RUVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVMB79GMOUVVVVVVVVVVVVVVVVVVVVVV
SVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVRH9658MVVVVVVVVVVVVVVVVVVVVVV
SVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVQIDEQVVVVVVVVVVVVVVVVVVVVVV
TVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
TVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
SVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
QUVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
ORUVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
```

Figure 1.1 chromosome image in the text file

## 2. Display the Original Image on Screen. (Task 1)

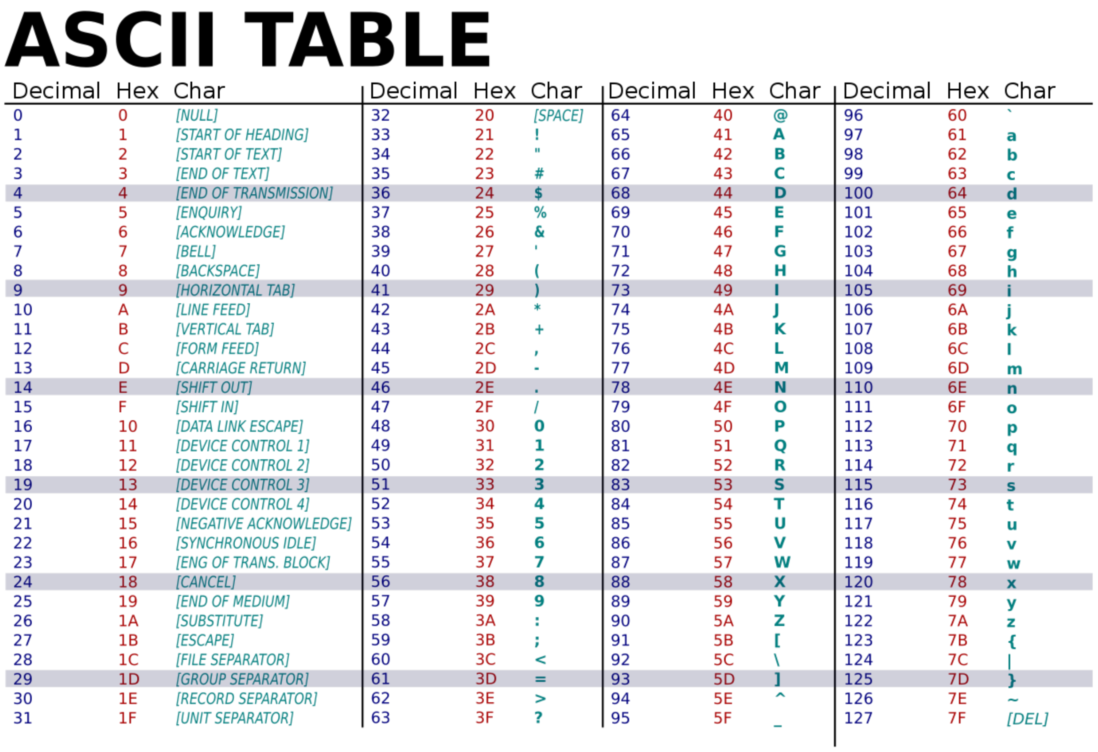
The main idea of this task is to convert the content of a specific number and letter in the text file into a grayscale value matrix and turn this matrix into a picture output.

### A. Processing Step and Algorithm

The first thing to do is turn the string text into a 64x64 matrix of 32 gray-level values. According to the introduction above, we know that this is a 64x64 image with 32 levels, and the text corresponds to these 32 gray levels from 0 to 9 and A to V, respectively.

The numeric part can easily correspond to the grayscale. We need to convert the string to a double value and add 1. And we get the corresponding gray level 1-10.

The problem is the conversion of letters. We know that the value of the letter in MATLAB corresponds to his ASCII value, and by observing the corresponding value of the number and letter in ASCII, it is possible to convert the letter to the ASCII value and minus 55 and get the corresponding gray level 10-32.



ASCII TABLE

| Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char | Decimal | Hex | Char |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | [NULL] | 32 | 20 | [SPACE] | 64 | 40 | @ | 96 | 60 | ` |
| 1 | 1 | [START OF HEADING] | 33 | 21 | ! | 65 | 41 | A | 97 | 61 | a |
| 2 | 2 | [START OF TEXT] | 34 | 22 | " | 66 | 42 | B | 98 | 62 | b |
| 3 | 3 | [END OF TEXT] | 35 | 23 | # | 67 | 43 | C | 99 | 63 | c |
| 4 | 4 | [END OF TRANSMISSION] | 36 | 24 | $ | 68 | 44 | D | 100 | 64 | d |
| 5 | 5 | [ENQUIRY] | 37 | 25 | % | 69 | 45 | E | 101 | 65 | e |
| 6 | 6 | [ACKNOWLEDGE] | 38 | 26 | & | 70 | 46 | F | 102 | 66 | f |
| 7 | 7 | [BELL] | 39 | 27 | ' | 71 | 47 | G | 103 | 67 | g |
| 8 | 8 | [BACKSPACE] | 40 | 28 | ( | 72 | 48 | H | 104 | 68 | h |
| 9 | 9 | [HORIZONTAL TAB] | 41 | 29 | ) | 73 | 49 | I | 105 | 69 | i |
| 10 | A | [LINE FEED] | 42 | 2A | * | 74 | 4A | J | 106 | 6A | j |
| 11 | B | [VERTICAL TAB] | 43 | 2B | + | 75 | 4B | K | 107 | 6B | k |
| 12 | C | [FORM FEED] | 44 | 2C | , | 76 | 4C | L | 108 | 6C | l |
| 13 | D | [CARRIAGE RETURN] | 45 | 2D | - | 77 | 4D | M | 109 | 6D | m |
| 14 | E | [SHIFT OUT] | 46 | 2E | . | 78 | 4E | N | 110 | 6E | n |
| 15 | F | [SHIFT IN] | 47 | 2F | / | 79 | 4F | O | 111 | 6F | o |
| 16 | 10 | [DATA LINK ESCAPE] | 48 | 30 | 0 | 80 | 50 | P | 112 | 70 | p |
| 17 | 11 | [DEVICE CONTROL 1] | 49 | 31 | 1 | 81 | 51 | Q | 113 | 71 | q |
| 18 | 12 | [DEVICE CONTROL 2] | 50 | 32 | 2 | 82 | 52 | R | 114 | 72 | r |
| 19 | 13 | [DEVICE CONTROL 3] | 51 | 33 | 3 | 83 | 53 | S | 115 | 73 | s |
| 20 | 14 | [DEVICE CONTROL 4] | 52 | 34 | 4 | 84 | 54 | T | 116 | 74 | t |
| 21 | 15 | [NEGATIVE ACKNOWLEDGE] | 53 | 35 | 5 | 85 | 55 | U | 117 | 75 | u |
| 22 | 16 | [SYNCHRONOUS IDLE] | 54 | 36 | 6 | 86 | 56 | V | 118 | 76 | v |
| 23 | 17 | [ENG OF TRANS. BLOCK] | 55 | 37 | 7 | 87 | 57 | W | 119 | 77 | w |
| 24 | 18 | [CANCEL] | 56 | 38 | 8 | 88 | 58 | X | 120 | 78 | x |
| 25 | 19 | [END OF MEDIUM] | 57 | 39 | 9 | 89 | 59 | Y | 121 | 79 | y |
| 26 | 1A | [SUBSTITUTE] | 58 | 3A | : | 90 | 5A | Z | 122 | 7A | z |
| 27 | 1B | [ESCAPE] | 59 | 3B | ; | 91 | 5B | [ | 123 | 7B | { |
| 28 | 1C | [FILE SEPARATOR] | 60 | 3C | < | 92 | 5C | \ | 124 | 7C | \| |
| 29 | 1D | [GROUP SEPARATOR] | 61 | 3D | = | 93 | 5D | ] | 125 | 7D | } |
| 30 | 1E | [RECORD SEPARATOR] | 62 | 3E | > | 94 | 5E | ^ | 126 | 7E | ~ |
| 31 | 1F | [UNIT SEPARATOR] | 63 | 3F | ? | 95 | 5F | _ | 127 | 7F | [DEL] |

Figure 1.2 ASCII table [1]

Thus, after we get the matrix, we can then convert this matrix to gray image output, and here we use the "*mat2gray*" function to convert the matrix to the image.

### B. Flowchart and Output Image

The output image is shown in Figure 1.3, and the flowchart of the entire process is shown in Figure 1.4.

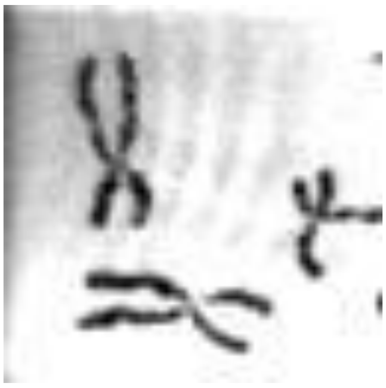


Figure 1.3 Output image

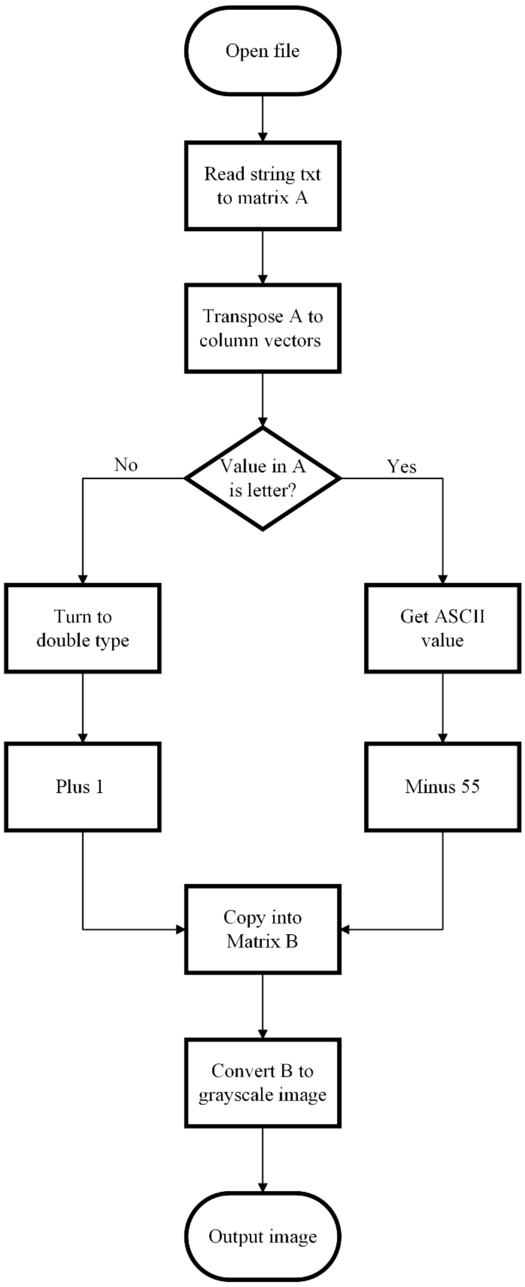


Figure 1.4 Flowchart of output chromosome image

## C. Discussion and Conclusion

Here we directly convert the string text to a numeric matrix, then turn it into a gray image and output the picture. This method of processing the conversion of the ASCII code value of the letter is very effective and convenient. From here, we can learn the technique of converting letters to the corresponding numbers, converting the text file to the numeric matrix, and finally, successfully displaying the image.

## 3. Threshold the Image and Convert it into Binary Image. (Task 2)

The task here is converting the gray image into a binary image to facilitate later processing. Each pixel of a gray image has 32 levels of grayscale, while the values of a binary graph of pixels are composed of only 1 and 0, representing objects and backgrounds, respectively.

## A. Processing Step and Algorithm with Resulting Image

The main purpose of converting to a binary diagram is to identify objects. Our most common method is to perform a Thresholding-Labelling operation to produce a binary image [2]:

- Determine a threshold value.
- Pixels having gray values higher than the Threshold value are given the value 1.
- Pixels having gray values lower than the Threshold value are given the value 0.

However, getting this threshold is a big problem. The professor describes several reasons in class, such as there is no apparent valley in the histogram as the gray levels of the object and the background are pretty close.

In addition, there is a problem with this chromosome image: a shadow appears in the upper left corner (uneven lighting problem), which makes some standard thresholding methods not work very well.

We first tried three thresholding methods: Global thresholding, OTSU thresholding, and Iterative thresholding. They're not very good at dealing with shadow problems. Here we briefly introduce these three methods and give the resulting image:

- ➢ Global thresholding

This method is a simple technique introduced in Chapter 2 of the course. It is helpful to segment an image by dividing the gray levels into bands and using thresholds to determine regions or to obtain object boundaries. [2] The resulting image is shown in Figure 1.5.

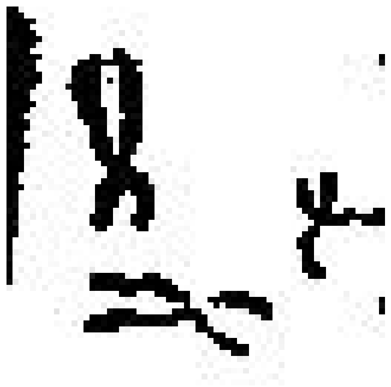


Figure 1.5 Global thresholding's resulting image

- OTSU thresholding

Here we use the function “*graythresh*” of MATLAB. This algorithm is a dynamic threshold segmentation algorithm proposed by the Japanese Otsu. Its main idea is to divide the image into background and target according to the grayscale characteristics and divide it according to the threshold value. After that, the variance between the background and the target is maximized. (From a blog on the website CSDN). The resulting image is shown in Figure 1.6.

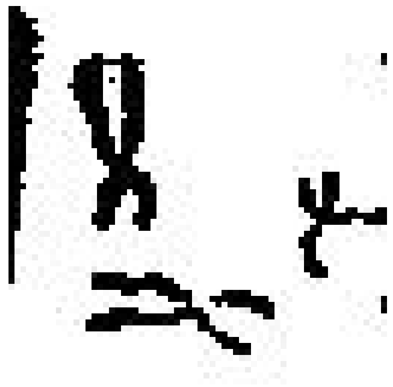


Figure 1.6 OTSU thresholding’s resulting image

- Iterative thresholding

The iterative method starts with a threshold as the initial estimate and then continuously improves that estimate according to a strategy until the given criterion is met. The resulting image is shown in Figure 1.7.

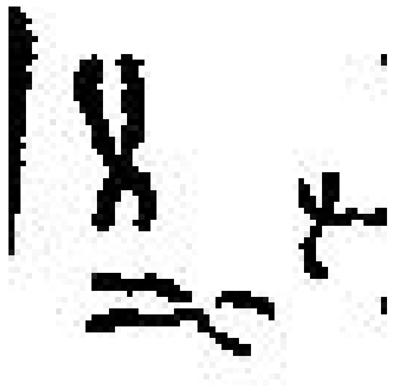


Figure 1.7 Iterative thresholding’s resulting image

Now we found the problems. One is that the shadow in the upper left corner causes the grayscale of that area to be different from other areas, so the shadow cannot be eliminated entirely, and the object in the upper left corner is not the same as the original image. Second, noise points appear on the right side.

The above methods belong to the global threshold algorithm, so the global threshold segmentation effect is not very good for the uneven lighting image here. We need an adaptive approach to solve this problem, preferably with threshold segmentation based on dividing the image into different areas.

After looking up various threshold segmentation methods, we found one called the adaptive threshold method, which was very suitable for our requirements and worked well in dealing with the problem of uneven lighting. Here we refer to the paper from Derek Bradley & Gerhard Roth paper 2011. [3]

- Adaptive thresholding

Its idea is not to calculate the threshold of the global image but to calculate the local threshold according to the brightness distribution of different areas of the image. So for different image regions, different thresholds can be calculated adaptively, which is called the adaptive threshold method. (Also called local threshold method) (Explanation from a blog on the website CSDN)

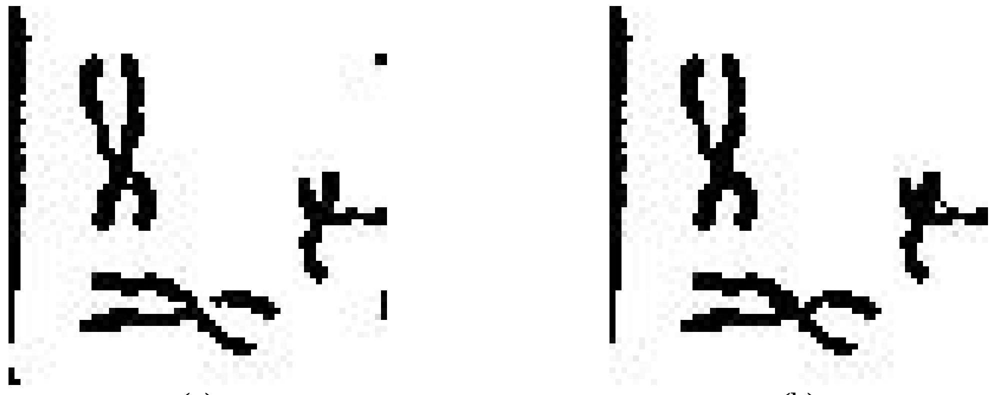
Firstly, we take an approach called "Adaptive Thresholding using the Integral Image" [3] and reprogram it by ourselves. Our implementation is in the function named "*Adaptive_thresholding*".

Here, by using the integral image, we try to calculate the average of the $s \times s$ pixel window centered on each pixel. It is a good average because it considers adjacent pixels on all sides. The time for averaging calculations using integral images is linear.

The processes of the algorithm are as follows.

In the first loop, we calculate the integral image when we traverse the image. In the second loop, we use the integral image to calculate the average of the $s \times s$ of each pixel, using the time constant and then comparing. If the value of the current pixel is less than $m$, multiply this average, and it is set to black. Otherwise, it is set to white. (We put a sensibility value $m$ to help tune the result).

The resulting rendering is shown in Figure 1.8 (a). Compared with the previous global threshold method, the shadow processing here is greatly improved, but simple morphological operations are still required to remove noise points and connect breakpoints. We use the function "*bwmorph*" to join the disconnected pixel (with "*bridge*") and fill the isolated inner pixel (with "fill"), and the function "*bwareaopen*" to delete the slight area pixel noise. The resulting image is shown in Figure 1.8 (b), and the flowchart is shown in Figure 1.9.



(a) (b)

Figure 1.8 Adaptive Thresholding using the Integral Image's resulting image

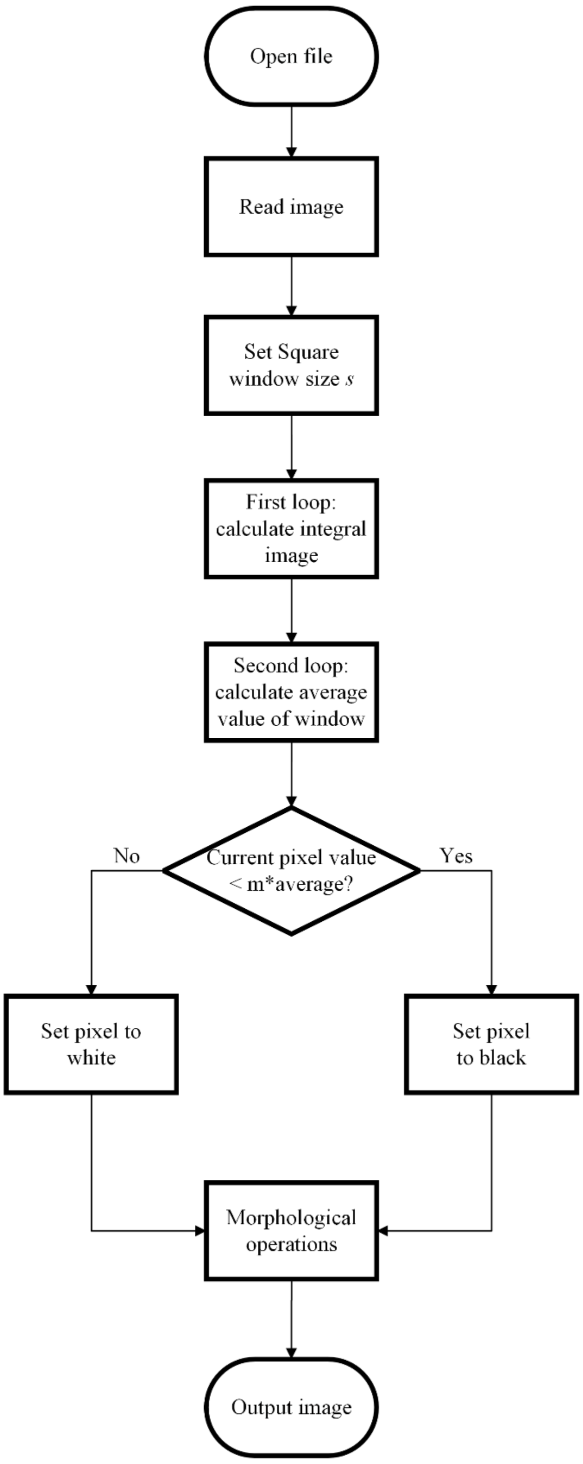

Open file
Read image
Set Square window size *s*
First loop: calculate integral image
Second loop: calculate average value of window
Current pixel value < m*average?
No
Yes
Set pixel to white
Set pixel to black
Morphological operations
Output image

Figure 1.9 Flowchart of Adaptive Thresholding using the Integral Image

However, we still found that the left shadow here still has some residues. After trying different adaptive thresholding methods, we discovered that MATLAB's function "*adaptthresh*" works best. It is an "Adaptive image threshold using local first-order statistics" function and comes with appropriate morphological operations ("*bwmorph*" can connect breakpoints, and "*bwareaopen*" can delete small areas of noise). Hence, we decided to use the resulting image out of this method, as shown in Figure 1.10.

It is worth noting that we still try to write the implementation program of "Adaptive Thresholding using the Integral Image" ourselves. Please check it in the file "Adaptive_thresholding.m".

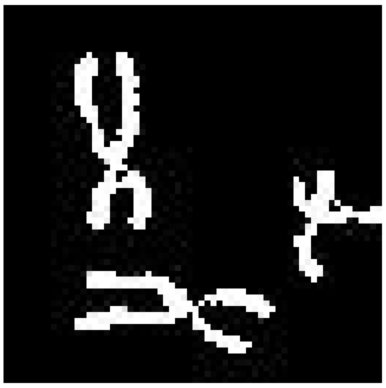


Figure 1.10 Adaptive image threshold using local first-order statistics' resulting image

### B. Discussion and Conclusion

Here we convert the gray image into a binary image. The image mainly has the problem of uneven lighting, we first used several global thresholding methods, but the effect was not good. Subsequently, the adaptive thresholding method is selected, the above problems are solved, and the binary value image with a relatively good result is obtained.

We understand the different thresholding methods for obtaining a binary image and how to deal with problems in the original image, such as uneven lighting, noise point, and connecting breakpoints in objects.

## 4. Determine a One-Pixel Thin Image of the Objects. (Task 3)

Task 3 focuses on thinning the image obtained in task 2 to get the skeleton of the image. We can finally get a one-pixel thin image of the objects by thinning the image.

### A. Processing Step and Algorithm with Resulting Image

In this task, our target is thinning the original image to get a one-pixel thin image.

We mainly used the following methods to get the thinning image:

- Hilditch Thinning Algorithm

Firstly, we use the original image with black pixels one and white pixels zero. The algorithm is operated on each black pixel P1, which has eight neighbors P2-P9.

| | | |
|---|---|---|
| P9 | P2 | P3 |
| P8 | P1 | P4 |
| P7 | P6 | P5 |

Figure 1.11 Black Pixel with 8 Neighbours

After that, we define two functions:

$$\mathrm{A(P1) = number\ of\ 0{,}1\ patterns\ in\ the\ sequence\ of\ P2 - P9 - P2}$$

$$\mathrm{B(P1) = number\ of\ non - zero\ neighbors\ of\ P1}$$

| | | |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |
| 0 | 1 | 0 |

Figure 1.12 A(P1) and B(P1) Example

For example, A(P1) of the picture above is 3, and B(P1) is also 3.

The next step is iterating through each white pixel and changing it to white if the black pixel satisfies the following conditions.

1. $2 \leq \mathrm{B(P1)} \leq 6$
2. $\mathrm{A(P1)} = 1$
3. $\mathrm{P2} * \mathrm{P4} * \mathrm{P8} = 0$ or $\mathrm{A(P2)!} = 0$
4. $\mathrm{P2} * \mathrm{P4} * \mathrm{P6} = 0$ or $\mathrm{A(P4)!} = 0$

We should stop the iteration until no more pixels change to black.

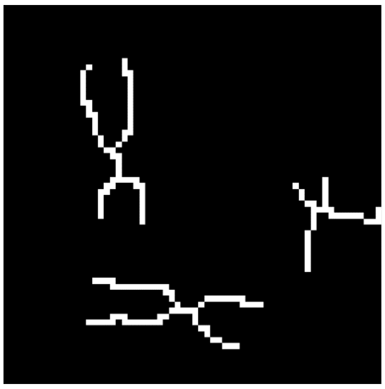


Figure 1.13 Hilditch Algorithm resulting image

- Zhang-Suen Thinning Algorithm

Firstly, same in Zhang-Suen Algorithm, each pixel has eight neighbors P2-P9.

The next step is iterating through each white pixel and changing it to white if the black pixel satisfies the following conditions.

1. $2 \leq \mathrm{B(P1)} \leq 6$
2. $\mathrm{A(P1)} = 1$
3. $\mathrm{P2} * \mathrm{P4} * \mathrm{P6} = 0$
4. $\mathrm{P4} * \mathrm{P6} * \mathrm{P8} = 0$

The second iterating step is roughly the same as the first iterating step.

1. $2 \leq \mathrm{B(P1)} \leq 6$
2. $\mathrm{A(P1)} = 1$
3. $\mathrm{P2} * \mathrm{P4} * \mathrm{P8} = 0$
4. $\mathrm{P2} * \mathrm{P6} * \mathrm{P8} = 0$

Those two steps compose a complete thinning step, and we should iterate this thinning algorithm until no more pixels are changed to black.

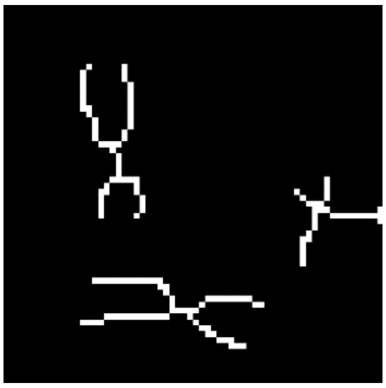


Figure 1.14 Zhang-Suen Thinning Algorithm resulting image

- Rosenfeld Thinning Algorithm

Firstly, same in Zhang-Suen Algorithm, each pixel has eight neighbors P2-P9. After that, we define that if P2=0, then P1 is the north border point; if P4=0, P1 is the east border point; if P6 =0, P1 is the south border point; and if P8=0, P1 is the west border point.

| | | |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 0 |
| 0 | 1 | 0 |

Figure 1.15 Border Point Example

For example, P1 of the above picture is both the north and east border point.

Besides, if all eight neighbor pixels of P1 are 0, P1 is an isolated point. If only one neighbor point is 1, P1 is an endpoint.

Another concept is 8-simple. If changing P1 to 0 will not affect the connectivity of 8 neighbors of P1, then P1 is 8-simple.

The next step is iterating the following four steps

1. Scanning each pixel, if the pixel is a north border point and 8-simple, and it is not an isolated point or endpoint, change this pixel to black.
2. Scanning each pixel, if the pixel is a south border point and 8-simple, and it is not an isolated point or endpoint, change this pixel to black.
3. Scanning each pixel, if the pixel is an east border point and 8-simple, and it is not an isolated point or endpoint, change this pixel to black.
4. Scanning each pixel, if the pixel is a west border point and 8-simple, and it is not an isolated point or endpoint, change this pixel to black.

These four steps compose an iteration, and we repeat the above iteration until no more pixels change to black.

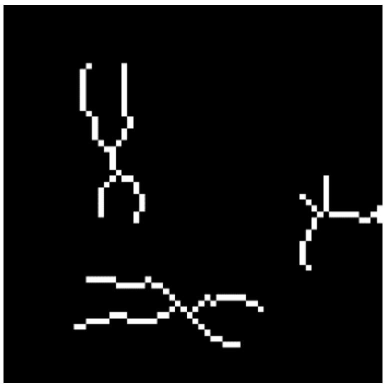


Figure 1.16 Rosenfeld Thinning Algorithm resulting image

## 5. Determine the Outline(s). (Task 4)

The main target of Task 4 is finding the boundary between the background and the object by edge detection. And the most common method

In this part, we introduced some standard edge detection methods used in our project. And convolution is the basis of those methods.

- Canny Edge Detection Algorithm

1. Noise Reduction

The first step of the Canny Edge Detection Algorithm is Noise Reduction. In this step, we apply image convolution to the original image with a Gaussian Kernel (3*3, 5*5, 7*7). The different kernel sizes will affect the effect of blurring.

And The equation for the Gaussian filter kernel of size (2k-1)*(2k-1) is given as follows:

$$\mathrm{H}_{ij} = \frac{1}{2\Pi\sigma^2} e^{\left(-\frac{\left(i-(k+1)\right)^2+\left(j-(k+1)\right)^2}{2\sigma^2}\right)}; 1 \le i,j \le (2k+1)$$

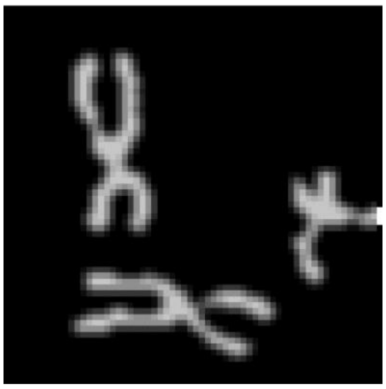


Figure 1.17 Noise Reduction resulting image

2. Gradient Calculation

According to applying image convolution to blurred images with edge detection operators, the Gradient Calculation step can detect the edge intensity and direction of the image.

$$K_x = \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix}, \; K_y = \begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$$

These two edge detection operators correspond to the change of pixels' intensity of horizontal Ix and vertical Iy. After that, we can calculate the magnitude G and the slope θ as follow.

$$|G| = \sqrt{I_x{}^2 + I_y{}^2}$$

$$\theta(x, y) = \tan^{-1}(\frac{I_y}{I_x})$$

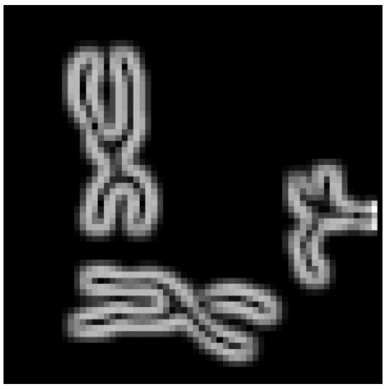


Figure 1.18 Gradient Calculation resulting image

3. Non-maximum suppression

The target of Non- maximum suppression is thinning out the edges. The algorithm will go through all the points on the pixels' slope matrix and finds the pixels with the maximum value in edge directions.

And the steps of Non- maximum suppression are as follows.

1. Create a zero matrix of the same size as the original magnitude matrix.
2. Derivate the edge direction based on the slope matrix.
3. Check that compared with the currently processed pixel, the pixel in the same direction has a higher intensity. And return a non-maximum suppression image.

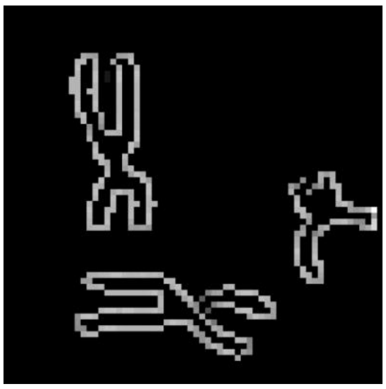


Figure 1.19 Non-maximum suppression resulting image

4. Double Threshold

The Double Threshold aims to identify three kinds of pixels: strong, weak, and non-relevant(zero).

Strong pixels have a high intensity and contribute to the final edge image. As for weak pixels, we still need to consider whether they will contribute to the last edge image. And the other pixels are non-relevant for the edge image.

Consequently, we are supposed to make a double threshold to the image, a high threshold is used to identify the strong pixels, and a low threshold can differentiate the weak and non-relevant pixels.

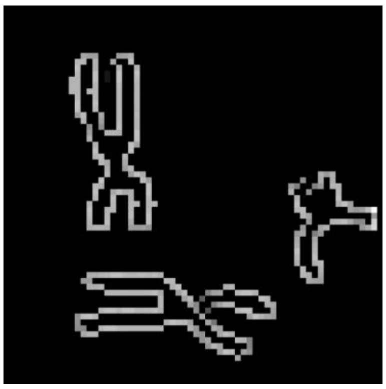


Figure 1.20 Double Threshold resulting image

5. Edge Tracking by Hysteresis

The final step is Edge Tracking by Hysteresis, which can transform the weak pixels into strong pixels or non-relevant pixels.

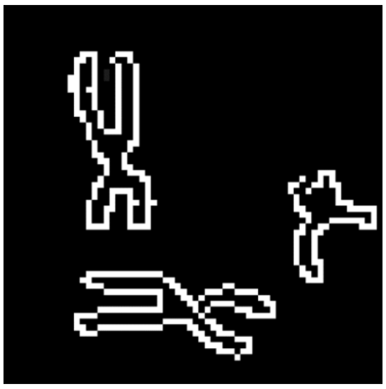


Figure 1.21 Edge Tracking by Husteresis resulting image

- Prewitt Edge Detection Algorithm

The main steps of the Prewitt Edge Detection Algorithm are the same as Canny Edge Detection Algorithm. The difference between those two algorithms is that they have different edge detection operators, and the edge detection operators are as follows.

$$\mathrm{K}_x = \begin{matrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{matrix}, \quad \mathrm{K}_y = \begin{matrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{matrix}$$

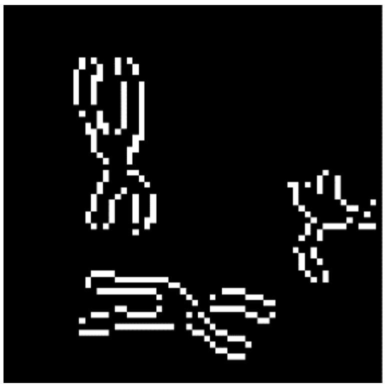


Figure 1.22 Prewitt Edge Detection Algorithm resulting image

➢ Roberts Edge Detection Algorithm

The Robert Edge Detection Algorithm's main steps are the same as Canny Edge Detection Algorithm. The difference between those two algorithms is that they have different edge detection operators, and the edge detection operators are as follows.

$$K_x = \begin{matrix} 1 & 0 \\ 0 & -1 \end{matrix}, \quad K_y = \begin{matrix} 0 & -1 \\ 1 & 0 \end{matrix}$$

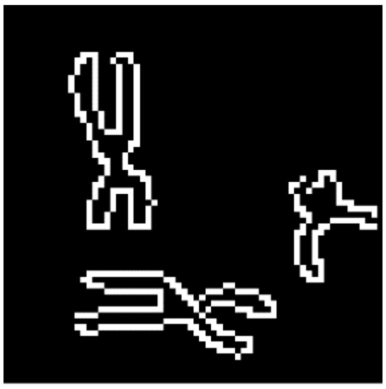


Figure 1.23 Rosenfeld Edge Detection Algorithm resulting image

## 6. Label the Different Objects. (Task 5)

For this task, the main idea is to finish the binary image's connected component labeling. For some specific images, the preprocessing of the image is necessary before it. And next segment them by different colures. Finally, draw bounding boxes for different objects with numbers and crop them to separate images.

### A. Preprocessing the Binary Image

In this project, the labeling task is based on the binary image. However, due to the selection of binary thresholding and some particular characteristic in the original image, the binary often cannot satisfy our desire (one object separated into many regions (Figure 1.24)). Hence, the preprocessing for the binary image is necessary for the task to finish.

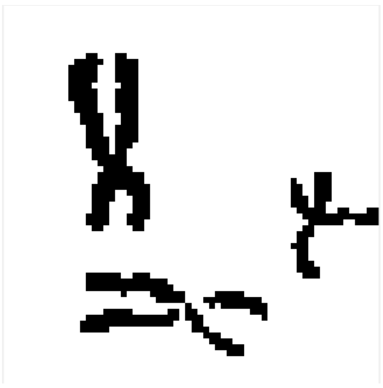


Figure 1.24 Binary image not good

- Firstly, a Gaussian low pass filter makes the image smoother (Figure 1.25).

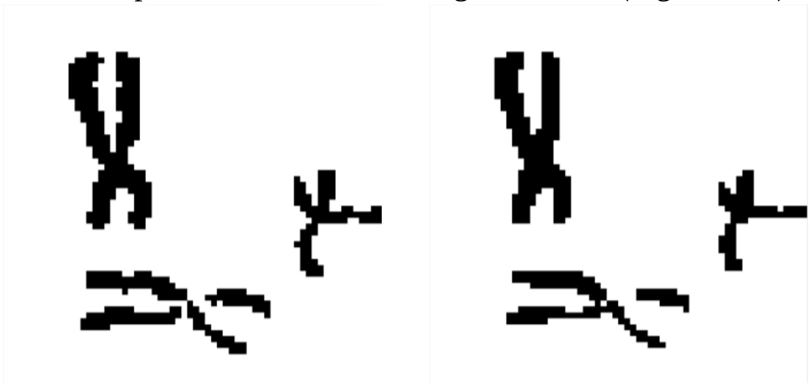


Figure 1.25 Binary image after Gaussian low pass filter

The result does not satisfy the expectation, while the closed part becomes closer and the isolated part becomes more isolated.

- Secondly, the image morphology operations are applied to it. The role of dilatation is to increase the target, fill the small holes in the object, and smooth the object's boundary, and the border expands to the outside (Figure 1.26).

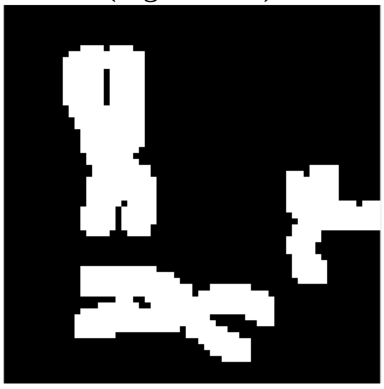


Figure 1.26 Binary image after dilating

The bridge operation is another morphology operation (Figure 1.27). Bridges disconnected pixels, i.e., if a zero-valued pixel has two disconnected non-zero neighbors, set those zero-valued pixels to one. And then do the fill operation to fill isolated interior pixels.

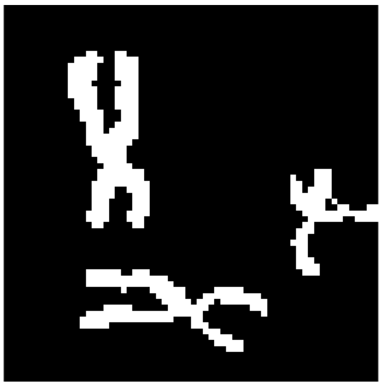


Figure 1.27 Binary image after the bridge and fill

### B. Connected Component Labelling

After an appropriate binary image is created, the labeling can start. Here, the adaptive image thresholding result is used to compare connectivity 4 and 8 better. We select three ways to segment the image:1. MATLAB function 2. Classical algorithm 3. Region growing algorithm.

- There are two functions in MATLAB to label the connected binary components. One is the "*bwconncomp*" (Figure 1.28). Another one is "*bwlabel*". Both of them use the 8-connectivity principle. The results are the same.

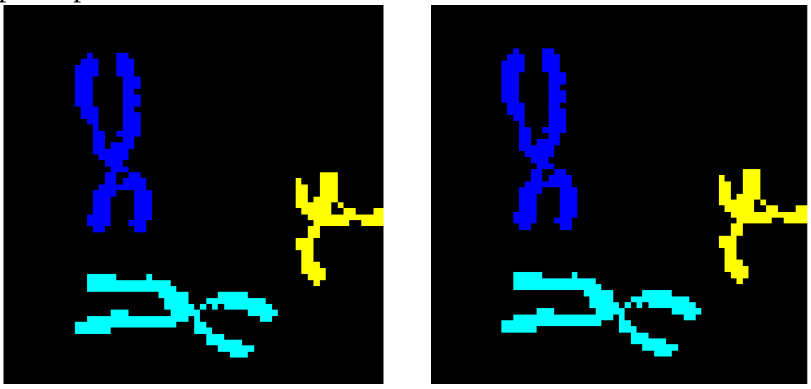


Figure 1.28 Label image with "*bwconncomp*" and "*bwlabel*"

- Classical Algorithm
  This algorithm is taught in class. The main idea is two passes. The first pass is a scan from left to right, top to down, assigning the object pixels with labels. The following job is to find the equivalence table (some labels are equivalent). The second pass is to perform a transition through the image to assign pixels with a minimum label. The 4_connectivity flow chart is shown below (Figure 1.29). For the 8_connectivity one, add the left and up pixels with the left-up corner pixel.

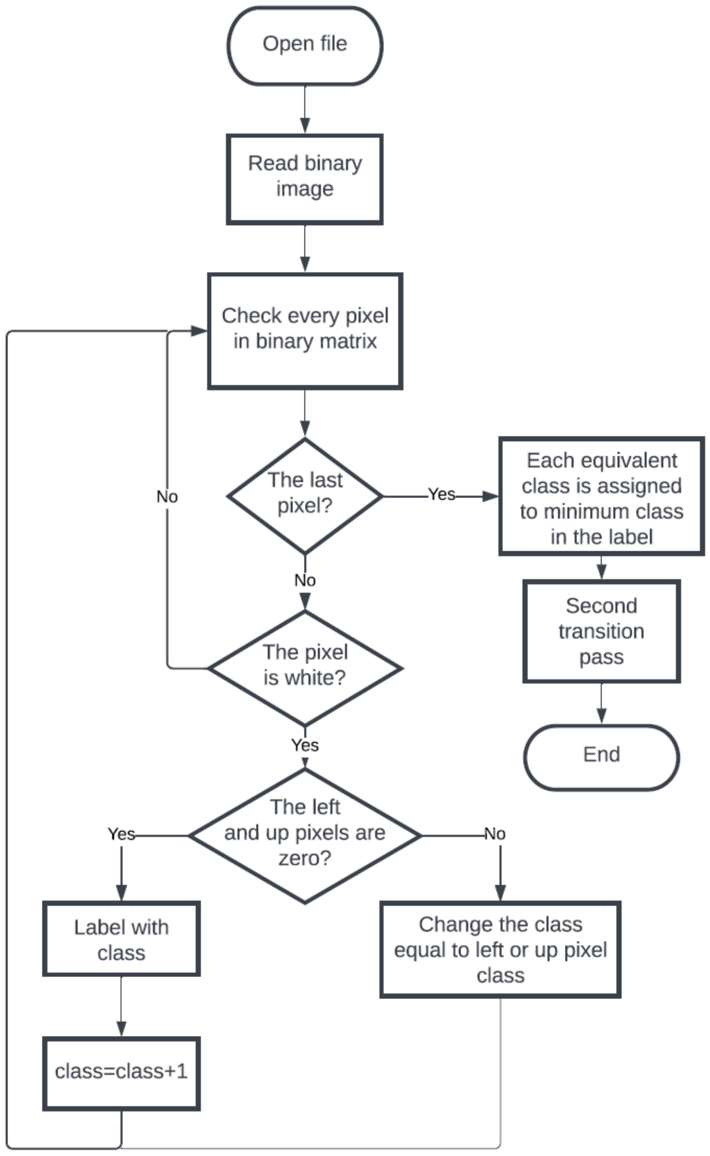

Open file
Read binary image
Check every pixel in binary matrix
The last pixel?
Yes
No
Each equivalent class is assigned to minimum class in the label
Second transition pass
End
No
The pixel is white?
Yes
The left and up pixels are zero?
Yes
No
Label with class
class=class+1
Change the class equal to left or up pixel class

Figure 1.29 Figure Flowchart of the classical algorithm

After running from MATLAB code (Figure 1.30). We can see that 4_connectivity cannot connect the 4-connected component to one component. And the connectivity_8 one has a good effect.

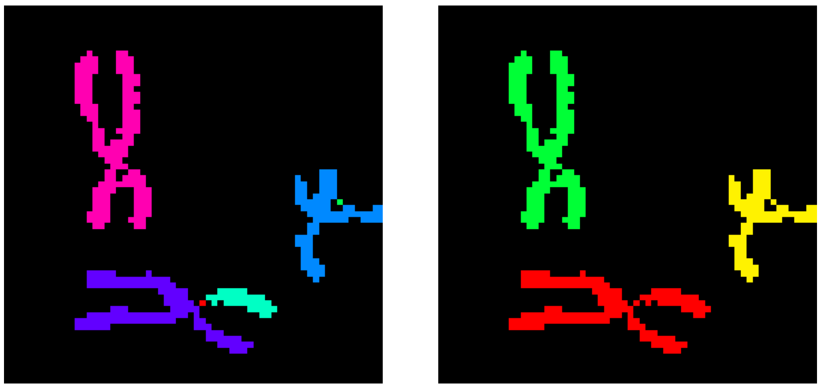


Figure 1.30 Results of the classical algorithm with connectivity 4 and 8

- Region Growing Method.
  The main idea of the Region Growing algorithm is first to find the seed point of the image and extend from one point to multiply points connected by the law of connectivity 4 or 8 [4]. The flow chart is shown below (Figure 1.31).

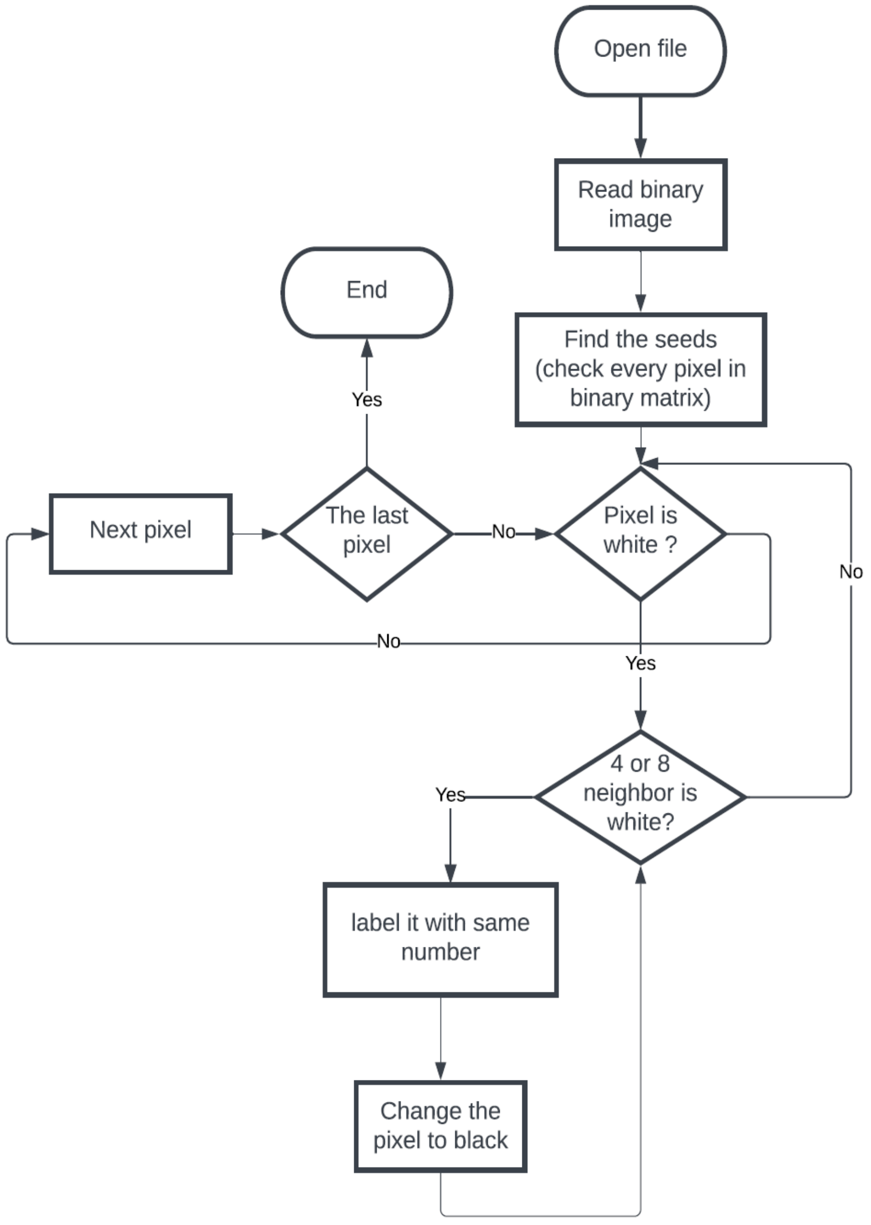

Open file
Read binary image
Find the seeds (check every pixel in binary matrix)
End
Yes
Next pixel
The last pixel
No
Pixel is white ?
No
No
Yes
4 or 8 neighbor is white?
Yes
label it with same number
Change the pixel to black

Figure 1.31 Flowchart of region growing algorithm

After running from MATLAB code (Figure 1.32), we can see it also faces the same problem in the difference of connectivity 4 and 8 as in the classical algorithm.

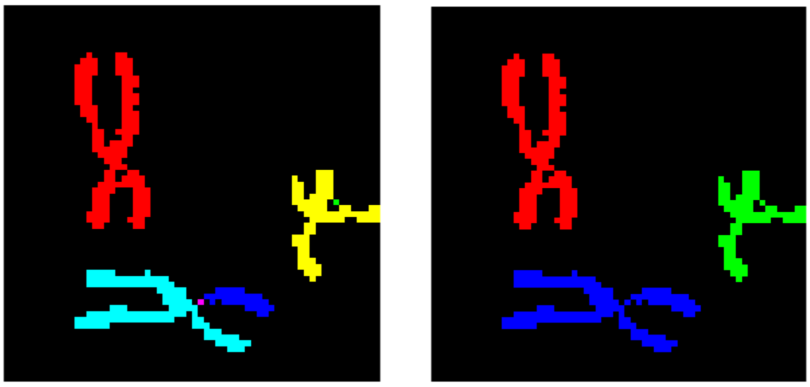


Figure 1.32 Results of region growing algorithm with connectivity 4 and 8

- Comparison
  The results of these three methods are the same as shown in the above three figures. However, their running speeds are different. After the observation in MATLAB, the fastest one is the MATLAB function, while the classical algorithm is the slowest. The reason may be that there are two loops in the classical algorithm. The coding way still has a vast area to improve for better performance.

## B. Bounding Box

With the objects attached to different labels, we can use MATLAB's "*regionprops*" function to find objects' left-up corner coordinates, width, and length. Then we can draw the bounding box for different objects and text with labels at the center of each object (Figure 1.33).

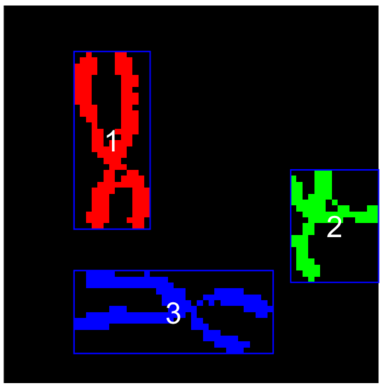


Figure 1.33 Results of the bounding box

## C. Crop

Using the parameters given in the bounding box, we can crop the image to separate objects and do the rotation separately. Then subplot them due to their relative position. The flow chart (Figure 1.34) and result (Figure 1.35) are shown below. The default rotation angle in my crop function is 0.

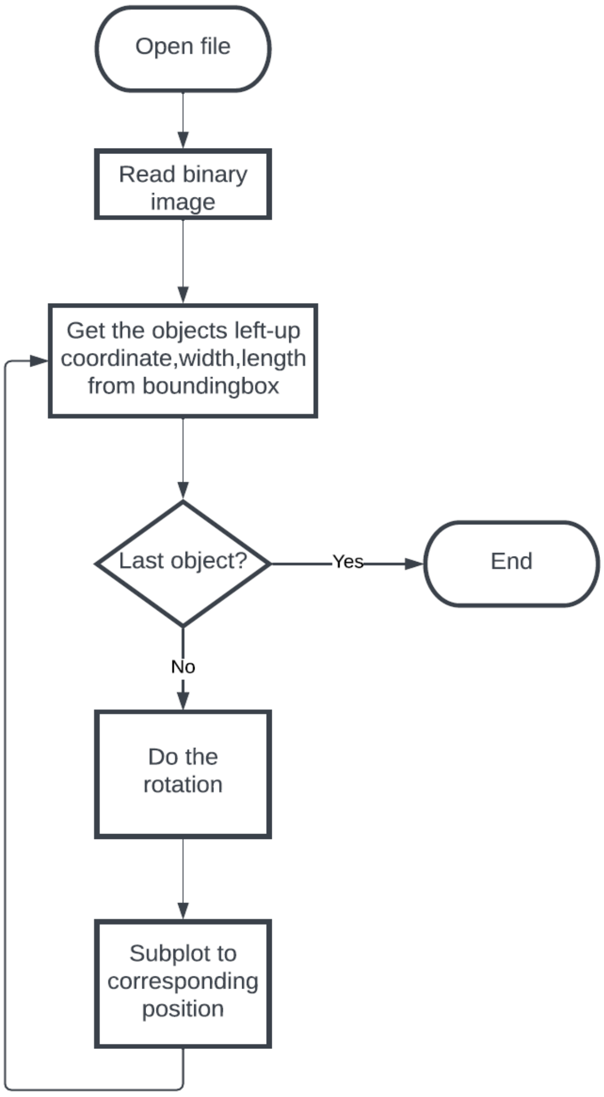

Open file
Read binary image
Get the objects left-up coordinate,width,length from boundingbox
Last object?
Yes
End
No
Do the rotation
Subplot to corresponding position

Figure 1.34 Flowchart of crop

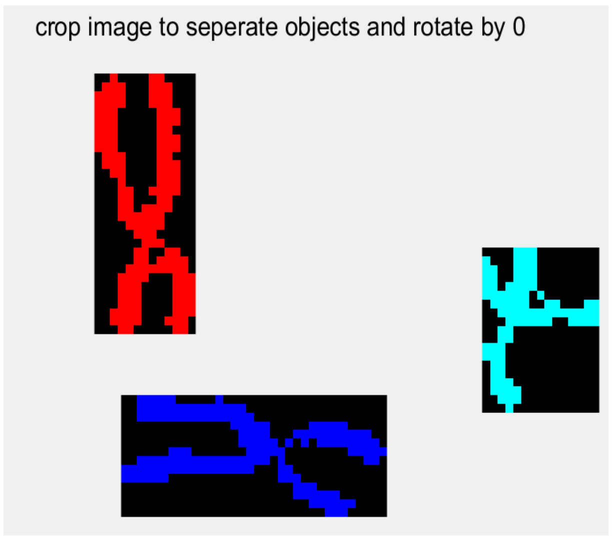


Figure 1.35 Results of the crop without rotation

## 7. Rotate the Original Image by 30 Degrees, 60 Degrees and 90 Degrees Respectively. (Task 6)

First of all, the MATLAB function "*imrotate*" has been used. The interpolation method can be chosen as nearest, bilinear, and bicubic. Below are the original images rotated by 30 degrees by three different methods (Figure 1.36).

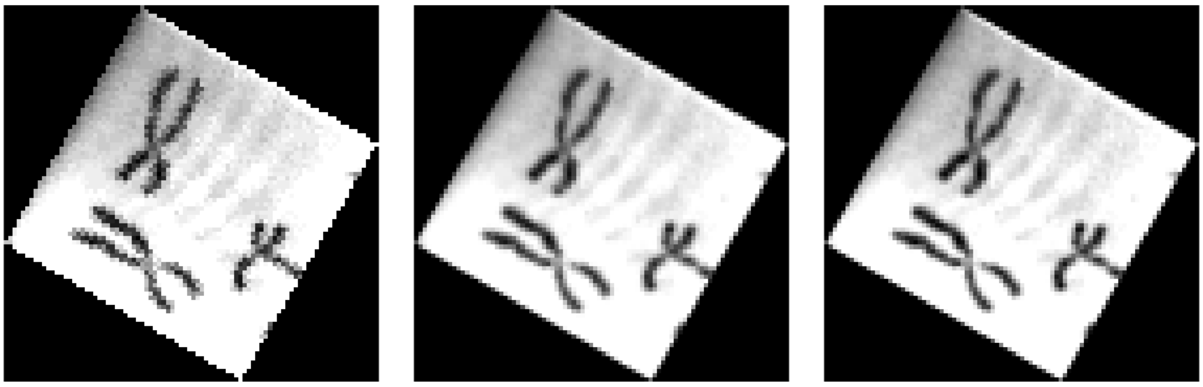


Figure 1.36 Results of rotation by "*imrotate*" with three methods

We also want to finish this job with our implementations. The flow chart for rotation is shown below (Figure 1.37). Firstly, for the coordinate transform, the centroid of the rotation image and the original image need to be calculated. The centroid calculation equation is shown here:

$$\mathrm{A} = \frac{1}{A}\sum_{(r,c)\in R} 1$$

$$\text{Centroid} = (\bar{r}, \bar{c}) \quad \text{where} \quad \bar{r} = \frac{1}{A}\sum_{(r,c)\in R} r$$

$$\bar{c} = \frac{1}{A}\sum_{(r,c)\in R} c$$

For each pixel in the rotated matrix, it is first necessary to change its coordinates to be relative to the centroid of its matrix. And do the inverse of rotation. Change the new coordinate to the left-up corner frame. Affine transformation matrixes can represent this transform procedure:

The first step:
$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -0.5\bar{r} \\ 0 & -1 & 0.5\bar{c} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

The second step:
$$\begin{bmatrix} x_3 \\ y_3 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$

The third step:
$$\begin{bmatrix} x_4 \\ y_4 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0.5\bar{r} \\ 0 & -1 & 0.5\bar{c} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_3 \\ y_3 \\ 1 \end{bmatrix}$$

Overall transformation:
$$\begin{bmatrix} x_4 \\ y_4 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0.5\bar{r} \\ 0 & -1 & 0.5\bar{c} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -0.5\bar{r} \\ 0 & -1 & 0.5\bar{c} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

This new coordinate often cannot fall on the discrete pixel value. So, interpolation is used to determine the pixel value. The flow chart is shown here (Figure 1.37). And here are three interpolation methods. They are nearest-neighborhood, bilinear, and bicubic.

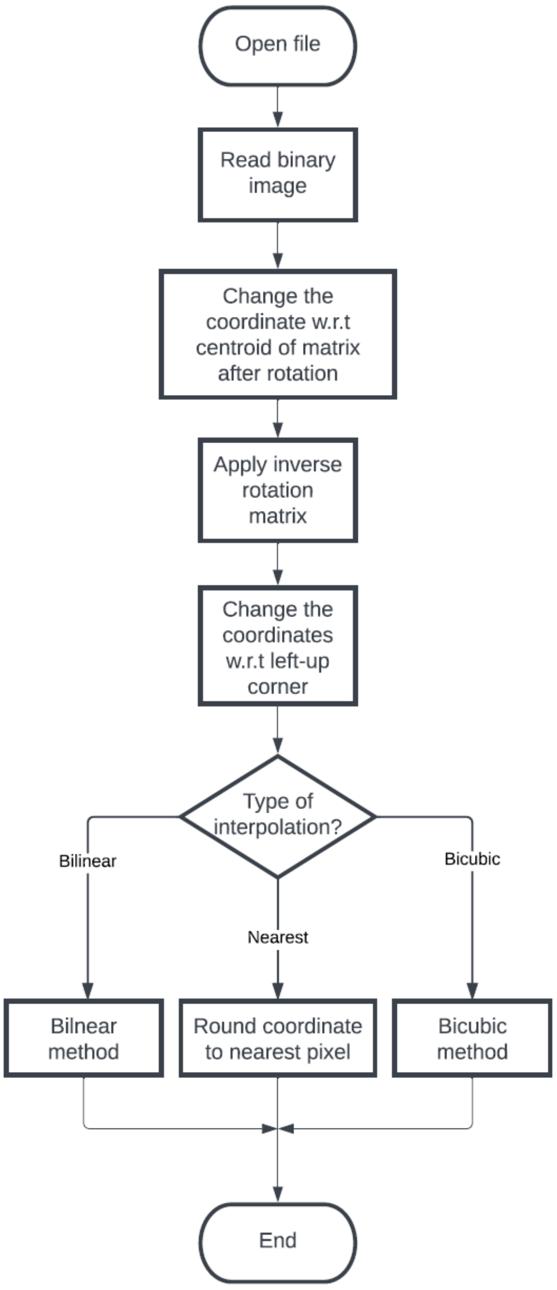

Open file
Read binary image
Change the coordinate w.r.t centroid of matrix after rotation
Apply inverse rotation matrix
Change the coordinates w.r.t left-up corner
Type of interpolation?
Bilinear
Nearest
Bicubic
Bilnear method
Round coordinate to nearest pixel
Bicubic method
End

Figure 1.37 Flowchart of image rotation by three different methods

## A. Interpolated by Nearest-neighborhood

The case when new coordinates often cannot fall on the discrete pixel value is shown below (Figure 1.38).

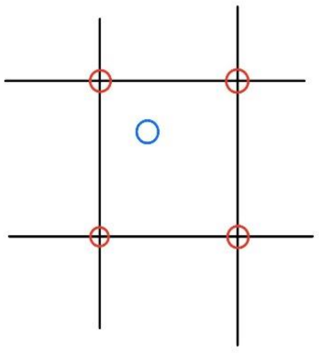


Figure 1.38 Interpolated by nearest

This method assigns the blue pixel with the nearest pixel (left-up one) red pixel value to it. The function is shown here:

$$f(x,y)=g(round(x),round(y))$$

## B. Interpolated by Bilinear

The bilinear way assumes the brightness function is linear in this neighborhood (Figure 1.39).

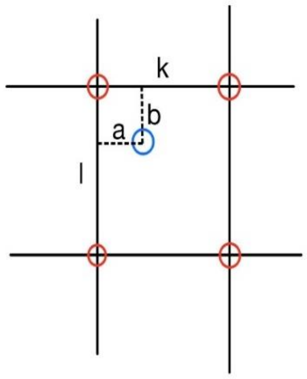


Figure 1.39 Interpolated by bilinear

The interpolation function:

$$f(x,y)=(1-a)(1-b)g(l,k)+a(1-b)g(l+1,k)+b(1-a)g(l,k+1)+abg(l+1,k+1)$$

where l=floor(x), a=x-l, k=floor(y), b=y-k.

## C. Interpolated by Bicubic

The bicubic method uses 16 neighborhood values with a nonlinear weight (Figure 1.40). The inner pixel has more effect on the value.

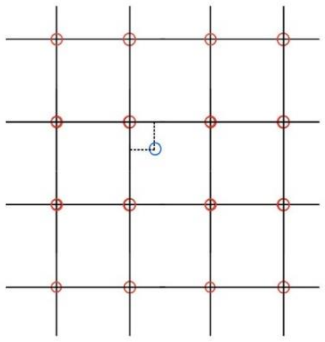


Figure 1.40 Interpolated by bicubic

The function of bicubic:

$$f(x,y)=\sum h_3(x)h_3(y)g(x,y)$$

where

$$h_3=\begin{cases} 1-2|x|^2+|x|^3 & ,0\le |x|<1 \\ 4-8|x|+5|x|^2-|x|^3 & ,1\le |x|<2 \\ 0 & \text{otherwise} \end{cases}$$

## D. Result Comparison

All the rotation degrees by three methods are shown here (Figure 1.41).

The advantage of the nearest neighbor interpolation method is that the calculation amount is small, and the algorithm is simple, so the operation speed is faster. However, it only uses the grey value of the pixel closest to the sampling point to be measured as the grey value of the sampling point, without considering the influence of other adjacent pixels, so the grey value after resampling has an apparent discontinuity. The loss of image quality is large, resulting in evident mosaic and aliasing.

The effect of bilinear interpolation is better than that of nearest neighbor interpolation. Still, the calculation amount is slightly larger, the algorithm is more complicated, and the program running time is somewhat longer. However, the image quality after scaling is high, which overcomes the discontinuous grey value of nearest neighbor interpolation because it considers the correlation influence of the four immediate neighbors around the sampling point to be measured. However, this method only considers the impact of the grey value of the four direct adjacent points around the sample to be measured. It does not consider the influence of the grey value change rate between the adjacent points, so it has the property of a low-pass filter, which leads to scaling the high-frequency components of the post-image are lost. The edges of the image become blurred to a certain extent. Compared with the input image, the output image scaled by this method still has the problems of image quality damage and low calculation accuracy due to the poor design of the interpolation function.

The cubic convolution interpolation has the most significant amount of calculation and the most complex algorithm. In geometric operations, the smoothing effect of bilinear interpolation may degrade the details of the image, and this effect is more evident when zooming in. In other applications, slope discontinuities in bilinear interpolation can produce undesirable results. The cubic convolution interpolation considers the influence of the grey value of the four directly adjacent pixels and the impact of the change rate of their grey value. Therefore, it overcomes the shortcomings of the first two methods and can generate smoother edges than bilinear interpolation.

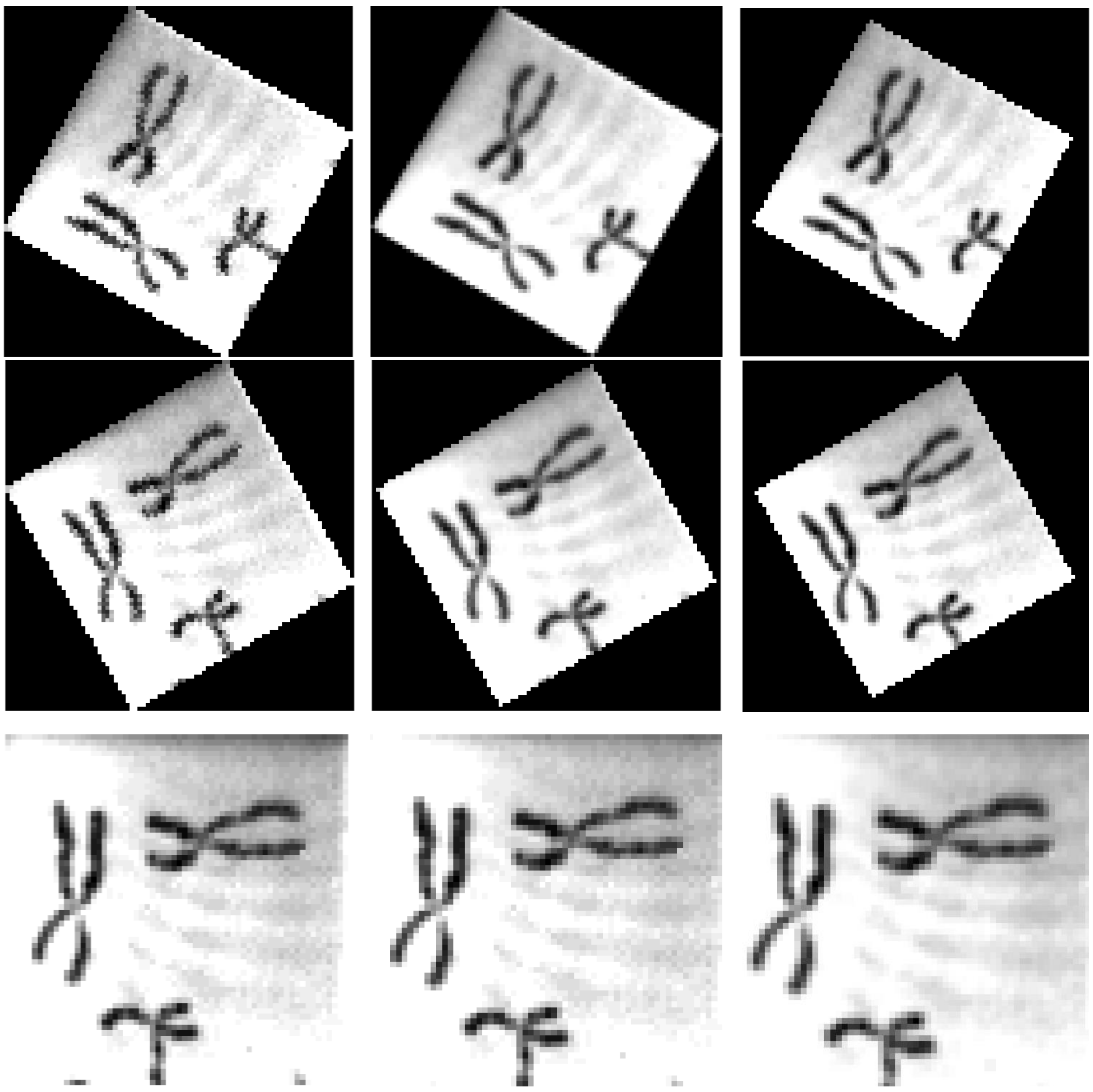


Figure 1.41 Results of rotation in three degrees by three methods

Below is the image that rotates the objects separately by 30 degrees (Figure 1.42).

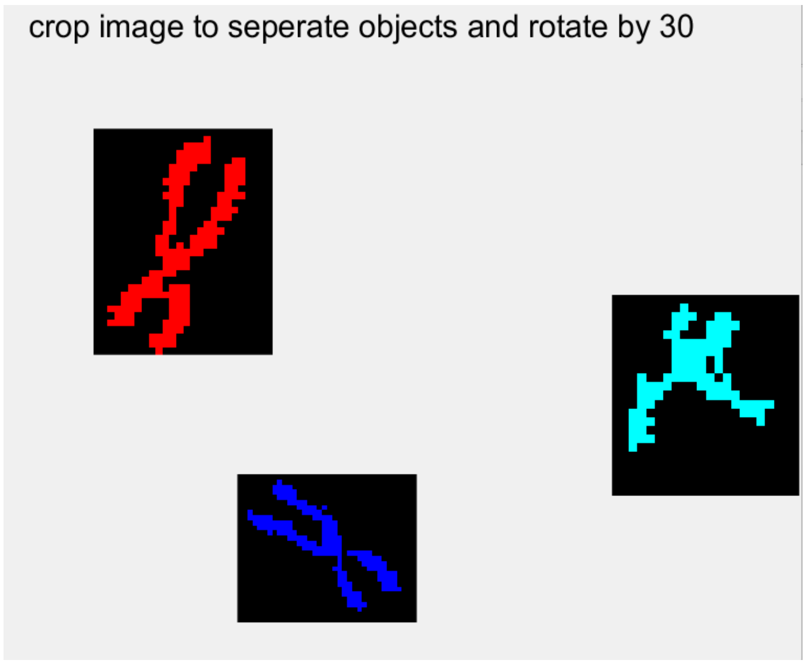
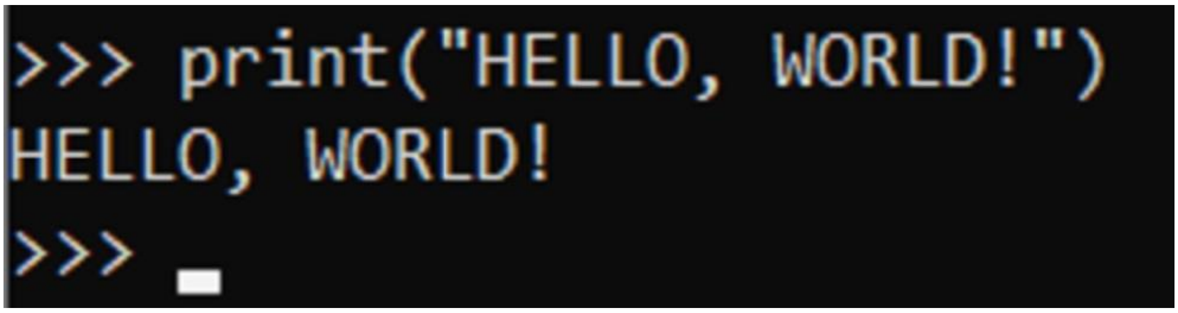


Figure 1.42 Rotate the objects separately by 30 degrees

## Image 2: Characters

### 1. Introduction

Image 2 is a JPEG color image comprising three lines of characters. The image is shown in Figure 2.1.



```
>>> print("HELLO, WORLD!")
HELLO, WORLD!
>>> _
```

Figure 2.1 JPEG color image: Characters

### 2. Display the Original Image on Screen. (Task 1)

The source file is a jpg image file. We need to read and display it.

### A. Processing Step

First, read the jpg file with function "*imread*", then display the image on the screen with the function "*imshow*" and write it to the "/result" folder with the function "*imwrite*". The display image is the same as shown in Figure 2.1.

### B. Discussion

The image file here is different from the first image file. Its format is jpg. No need to convert the change can be read and displayed as a picture. Besides, because it is a JEPG color picture, it is read to an RGB three-dimensional matrix.

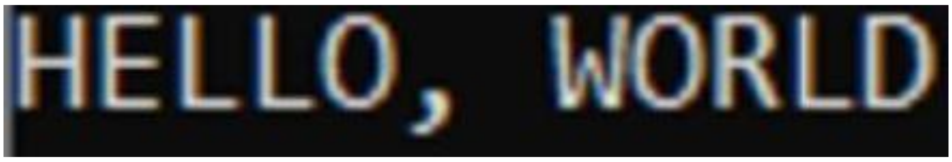
### 3. Create an Image Which is a Sub-Image of the Original Image Comprising the Middle Line – HELLO, WORLD (Task 2)

This task is to capture a region of interest in the image as a sub-image.

We have two ideas. One is to directly use the "*ginput*" function to select the image region by us with the mouse. The second is to try to get the coordinates of the region boundary of interest and use the "*imcrop*" function to capture according to the coordinates.

### A. Processing Step

The first step is to read the picture, then use the "*imcrop*" function to capture the region of interest, adjust the corresponding region boundary coordinates, and finally display the output picture. The resulting image is shown in Figure 2.2.



```
HELLO, WORLD
```

Figure 2.2 Sub-image "HELLO, WORLD"

### B. Discussion

Capturing the region of interest to get the sub-image is a relatively simple operation after trying to get its coordinates. But it is also possible to use other methods, such as automatically identifying and capturing the text region. Because of this image's simple structure, we can capture it here.

## 4. Create a Binary Image from Step 2 Using Thresholding (Task 3)

The main idea of this task is to convert the sub-image obtained above from an RGB image to a gray image and then to a binary image.

### A. Processing Step and Algorithm with Resulting Image

Note that situation of the image is pretty good. There are no problems, such as the uneven lighting in the first image. The construction of the letters is also straightforward, so we do not need to use too complex thresholding methods.

First, we directly use the "*rgb2gray*" function to convert the image to a gray image, and then we use the global thresholding method introduced in Chapter 2 of the course to carry out the binarization.

- Global thresholding

This simple technique helps segment an image by dividing the gray levels into bands and using thresholds to determine regions or obtain object boundaries. [2]

Since the original figure only has a black background and the foreground of the light letters, it is judged that there are only two peaks in the grayscale histogram. Thus, we should first set an initial threshold value and divide it into two bands. And then, count the gray values and the average gray value of the two bands. After that, average the average gray values of the two bands to obtain a new threshold to replace the original initial threshold.

Finally, the obtained threshold is used to segment the image. It is found that the obtained threshold will have two letters with a bit of connection, so add a correction value to the threshold.

### B. Flowchart and Output Image

The final image obtained is shown in the Figure 2.3, and the flowchart is shown in the Figure 2.4.

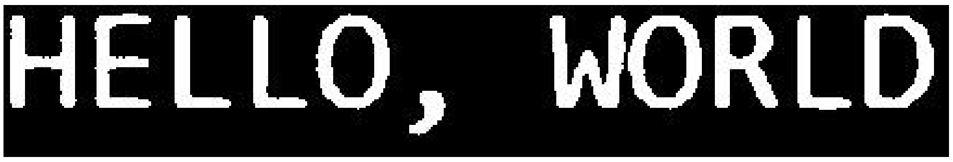


Figure 2.3 Resulting image using global thresholding

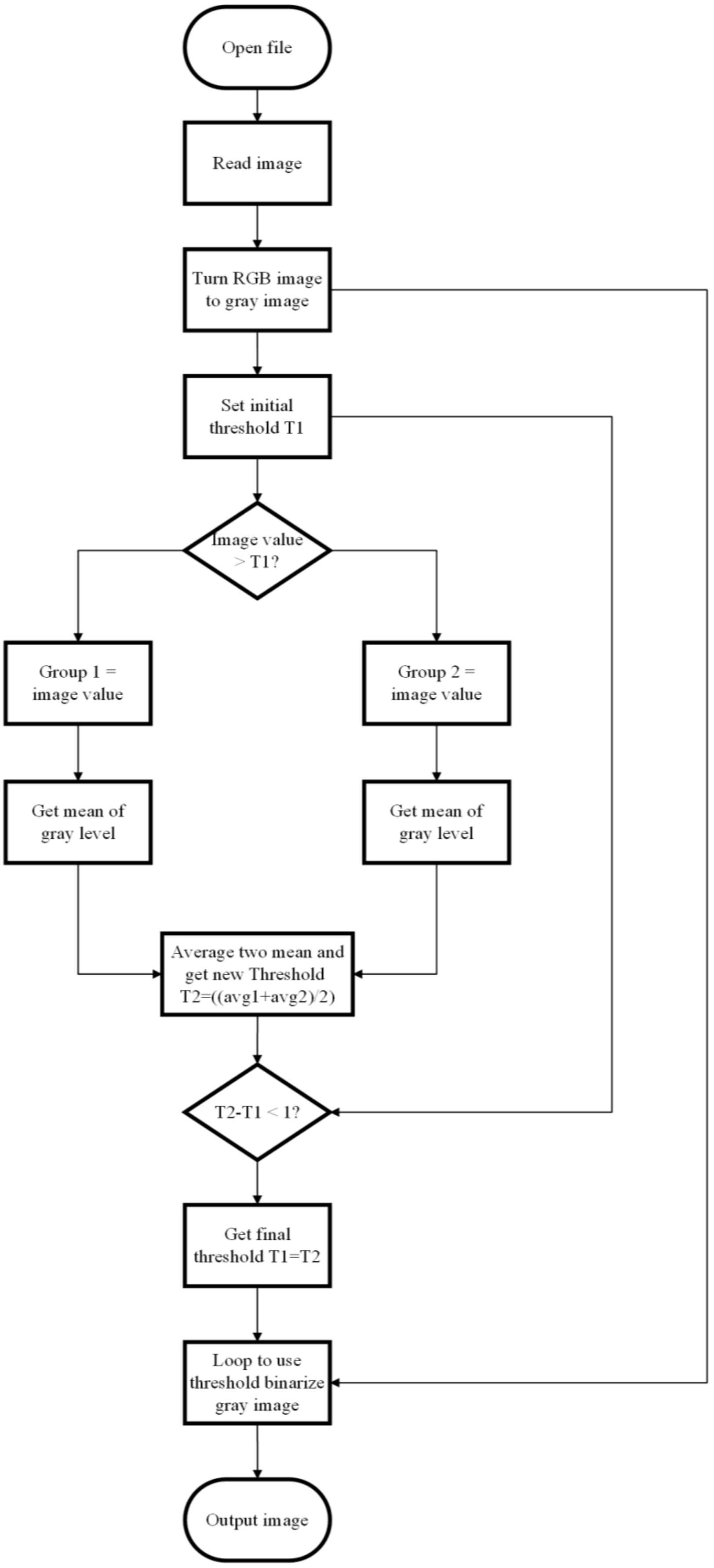


Figure 2.4 Flowchart of Global thresholding processing

### C. Conclusion and Discussion

Here we use the global threshold for binarization. The global threshold can already get good results for this kind of image without too much noise or shadow.

## 5. Determine a One-Pixel Thin Image of the Characters. (Task 4)

From Image 1, we already present the concept method of thinning algorithm, and we just carry out those algorithms in Image 2 and shows the result of those thinning algorithm.

- Hilditch Thinning Algorithm

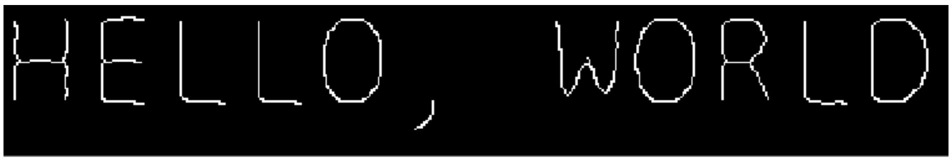


Figure 2.5 Hilditch Algorithm resulting image

- Zhang-Suen Thinning Algorithm

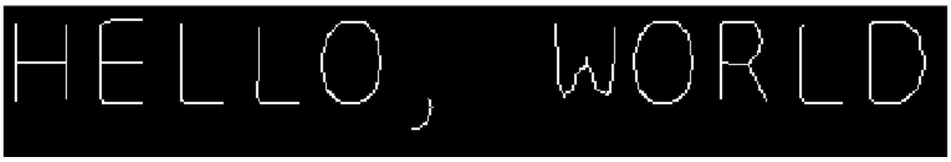


Figure 2.6 Zhang-Suen Thinning Algorithm resulting image

- Rosenfeld Thinning Algorithm

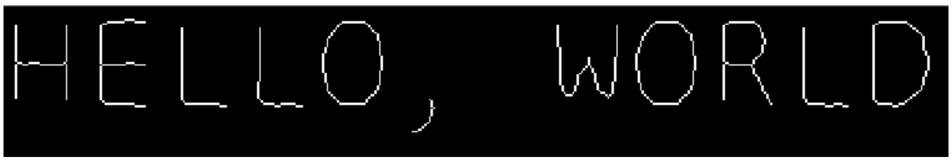


Figure 2.7 Zhang-Suen Thinning Algorithm resulting image

## 6. Determine the Outline(s). (Task 4)

From image 1, we already present the concept method of the edge detection algorithm. We carry out those algorithms in Image 2 and show the result of those edge detection algorithms.

- Canny Edge Detection Algorithm

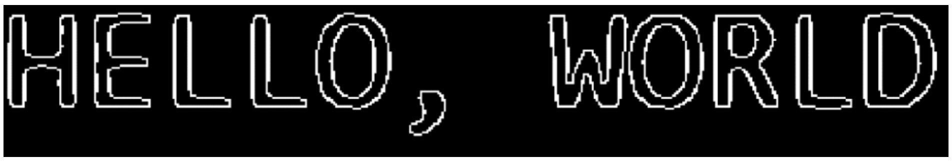


Figure 2.8 Canny Edge Detection Algorithm resulting image

- Prewitt Edge Detection Algorithm

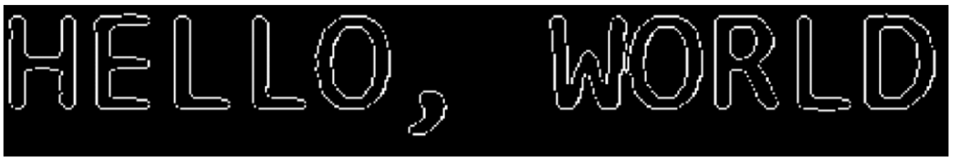


Figure 2.9 Prewitt Edge Detection Algorithm resulting image

- Roberts Edge Detection Algorithm

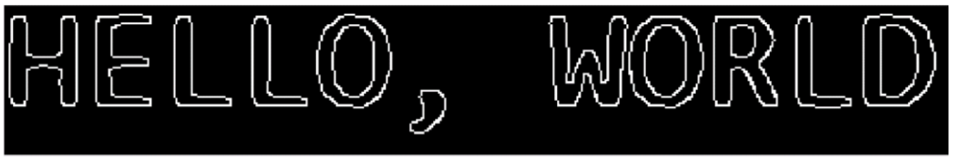


Figure 2.10 Prewitt Edge Detection Algorithm resulting image

**7. Segment the Image to Separate and Label the Different Characters. (Task 6)**

From the Image 1 part, we have already developed the algorithm for segmentation for binary images, and here shows the result of the classical algorithm (Figure 2.11), region growing method (Figure 2.12), and the bounding box (Figure 2.13).

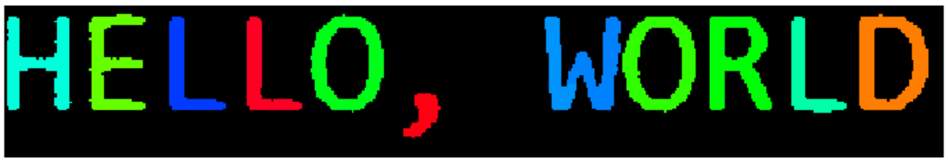


Figure 2.11 Results of the classical algorithm with connectivity 4 and 8

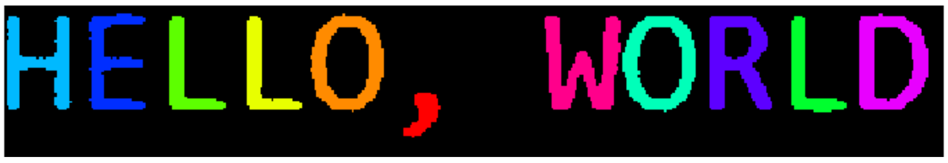


Figure 2.12 Results of region growing algorithm with connectivity 4 and 8

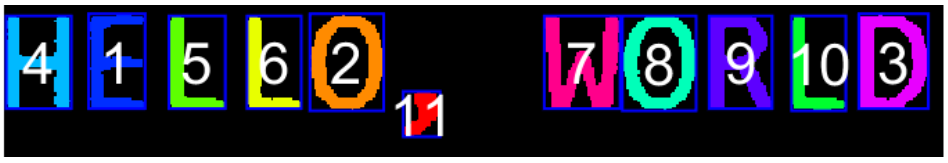


Figure 2.13 Results of bounding box and labeling the image

We can crop them separately (Figure 2.14) and rotate them individually (Figure 2.15).

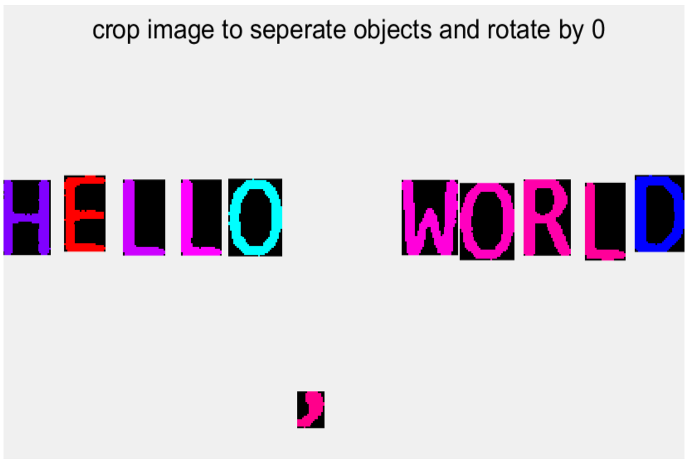


Figure 2.14 Results of cropping the image without rotation

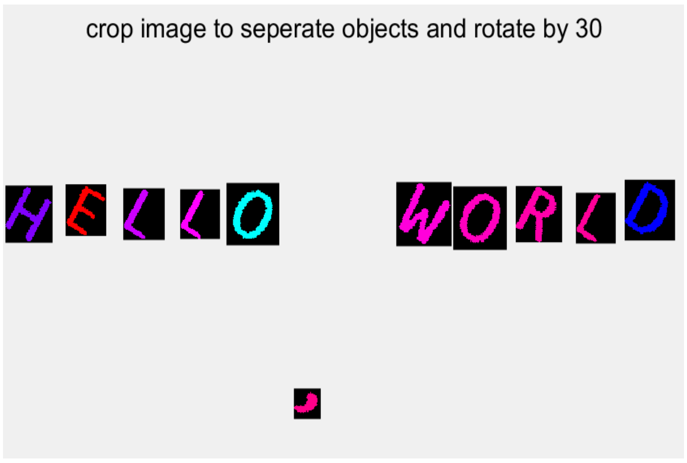


Figure 2.15 Results of cropping the image and rotating separately by 30 degrees

## 8. KNN Training and Accuracy Testing (Task 7)

Here we mainly use K-Nearest Neighbor (KNN) to train the dataset and identify the characters (“H”, “E”, “L”, “O”, “W”, “R”, “D”) and report the accuracy of the results.

### A. Introduction to Algorithms

The K-Nearest Neighbor algorithm (KNN, K-Nearest Neighbor) can be used for classification and regression [4]. The K-nearest neighbor algorithm means that each sample can be represented by its nearest K neighbors, and the characteristics of the sample are represented by the characteristics of most neighbors and then classified. Its advantages are prominent: the idea is simple, easy to understand, easy to implement, and does not require parameter estimation.

The calculation process of the KNN algorithm is as follows: a sample of unknown classification enters the data set, then which category is the most similar to it (the closest neighbor in the feature space) of the K samples, then which category is it. The primary process of the K-nearest neighbor algorithm is as follows:

- There is a training sample set, each sample in the training set is known to belong to the classification, and the number of nearest neighbors K is specified.

- After inputting a new sample without classification, compare each attribute of the new sample with the attribute corresponding to the sample in the training set and then extract the classification of the most similar K samples.
- The classification with the most occurrence of these K samples is the classification of new data

### B. Processing Step

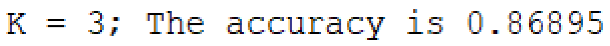
Here we first need to preprocess the dataset (in “train_preparation.m”). I binarize the images in the dataset, convert them to 32x32 size, and then convert them to the “.mat” file format and save them in the folder “train_preparation” classified in different characters.

Then I also need to preprocess the original test image (Figure 2.1), cut out each letter and binarize and convert to a mat format file, then save them in the folder “KNN_Test\Test_character”.

After completing these two steps, we try to train the dataset and examine the test images.

We first get 75% of the images in the dataset to train and 25% for validation. Then we test the characters from the original image and output the accuracy.

Here we choose $k = 3$, and the result accuracy is shown in Figure 2.16 as follows:



```
K = 3; The accuracy is 0.86895
```

Figure 2.16 KNN accuracy result (k=3)

Since it takes a long time to run a one-time KNN here (about once a minute), we comment on the code in MATLAB first, and if you need to test, please uncomment the code at the bottom of “Project_image2.m”.

### C. The Method of SOM

Self-Organizing Map (SOM) generates a low-dimensional, discrete map (Map) by learning the data in the input space, which can also be regarded as a dimensionality reduction algorithm to some extent.

SOM is an unsupervised artificial neural network (Figure 2.17). Unlike the general neural network training based on the backward transfer of the loss function, it uses a competitive learning strategy, which relies on the competition between neurons to optimize the network gradually. And use the neighbor’s relationship function (neighborhoods’ function) to maintain the topology of the input space.

Maintain the topology of the input space: this means that the 2D map contains the relative distances between data points. Adjacent samples in the input space are mapped to adjacent output neurons.

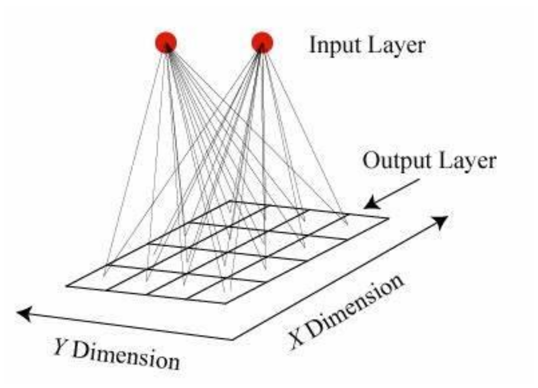


Figure 2.17 The architecture for the SOM

We can understand the working principle from the below flowchart (Figure 2.18).

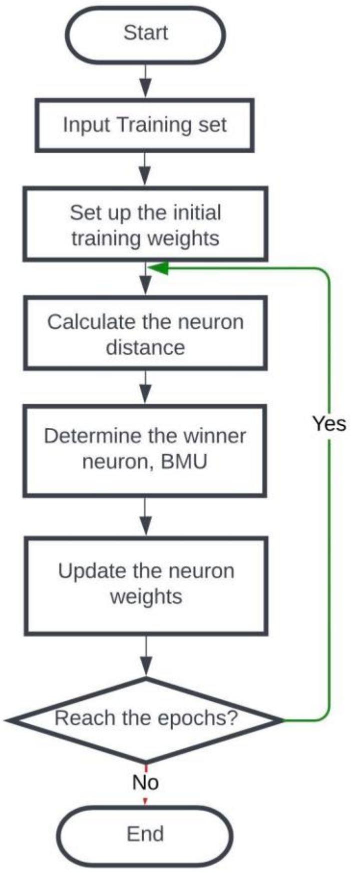

Start
Input Training set
Set up the initial training weights
Calculate the neuron distance
Determine the winner neuron, BMU
Update the neuron weights
Reach the epochs?
Yes
No
End

Figure 2.18 Flowchart of SOM

After every epoch, the weight of neurons will be updated by learning rate and topological distance to BMU. The equations are shown below:

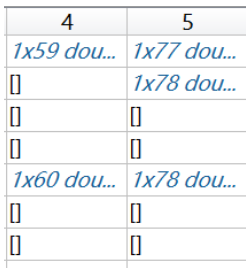
$$w(n+1) = w(n) + learn\_rate(n)*distance\ (n)*(x - w(n));$$

where learning rate decreases with time =learning _rate(0)*exp($-\frac{n}{parameter1}$)

topological distance to BMU= exp($-\frac{d*d}{2parameter2}$) d is calculated by Euclidean distance.

The users can set the number of neurons, iteration times, and two parameters. However, the minimum number of nodes in the competition layer = $5*\sqrt{N}$, where N is the number of training samples. If it is a square output layer, the side length equals the number of nodes in the competition layer. Open the square again and round it up.

After the experiments in MATLAB, we found SOM is very time-consuming for calculating neuron weights. For the better convergence of neuron weights, we have to let the computational layer be at least 20x20 (due to the conventional methods), and the iteration epochs are at least 1000 times. Our code runs for about 3540s (1 hour) in that situation. And the neurons are not so good at segmenting different characters (Figure 2.19). The figure below shows that different columns represent different neurons while rows mean different characters. However, neurons cannot distinguish them too well. For example, for neuron 5, characters 'E' and 'O' have the same number of training data due to the BMU. That is because improper initialization and similar clusters occur in different areas.



| 4 | 5 |
|---|---|
| 1x59 dou... | 1x77 dou... |
| [] | 1x78 dou... |
| [] | [] |
| [] | [] |
| 1x60 dou... | 1x78 dou... |
| [] | [] |
| [] | [] |

Figure 2.19 One neuron consists of many characters

After adjusting the initial parameters, the computational layer is set to 5x5 with 1000 iterations. The final accuracy rate is 0.8521.

accuracy 0.8521

Figure 2.20 SOM accuracy result

## 9. Parameter Tuning and Test for KNN

Here we try to adjust the parameters and preprocessing process from the previous task and observe the effect.

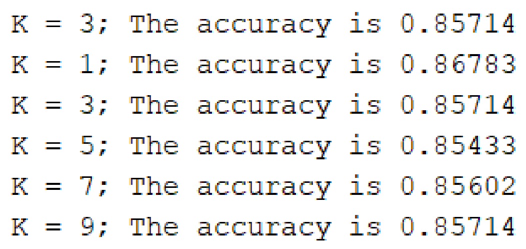
First, we try to adjust the value of $k$, and we get different accuracy values, shown in Figure 2.21. We found that the smaller the value of k, the higher the recognition accuracy.

```
K = 3; The accuracy is 0.86895
K = 1; The accuracy is 0.88076
K = 3; The accuracy is 0.86895
K = 5; The accuracy is 0.86614
K = 7; The accuracy is 0.86164
K = 9; The accuracy is 0.85489
```

Figure 2.21 KNN accuracy results in different k value

Then we try to adjust the size of the processed image when preprocessing the dataset.

The above figure is the accuracy of preprocessing the dataset image to 32x32. We try to preprocess the dataset image size to 16x16 and run again to obtain the accuracy shown in the figure below. We can see that the accuracy decreases, which indicates that the smaller the image size of the preprocessed dataset, the less accurate the training.



```
K = 3; The accuracy is 0.85714
K = 1; The accuracy is 0.86783
K = 3; The accuracy is 0.85714
K = 5; The accuracy is 0.85433
K = 7; The accuracy is 0.85602
K = 9; The accuracy is 0.85714
```

Figure 2.22 KNN accuracy result in preprocessing image size 16x16

## APP Design

After coding in MATLAB, we design an app in the MATLAB app design toolbox to better perform the image processing results and facilitate customer interaction. Firstly, the app's main interface is separated into two menu bars. One is for Image 1. The other is for image 2 in this project. And for every problem in each image, buttons have been designed to show the corresponding image as the user presses them (Figure 3.1).

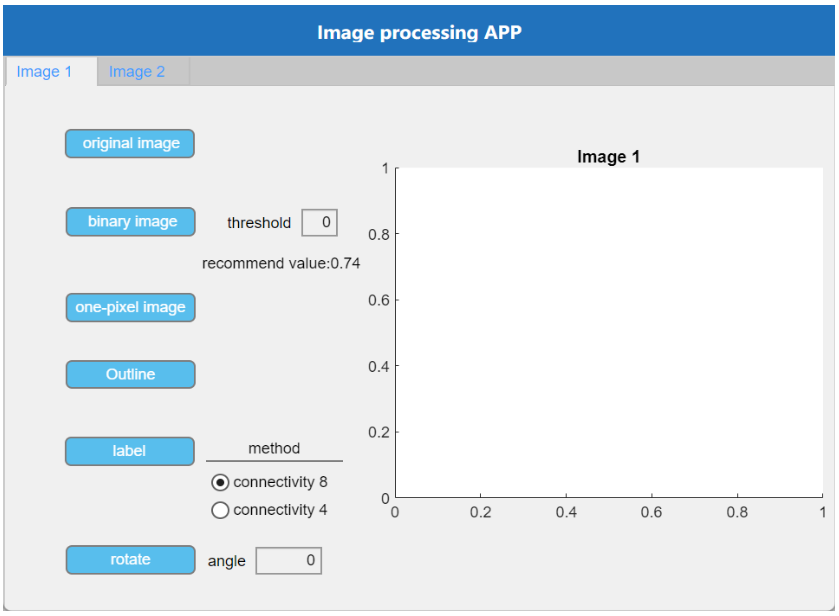
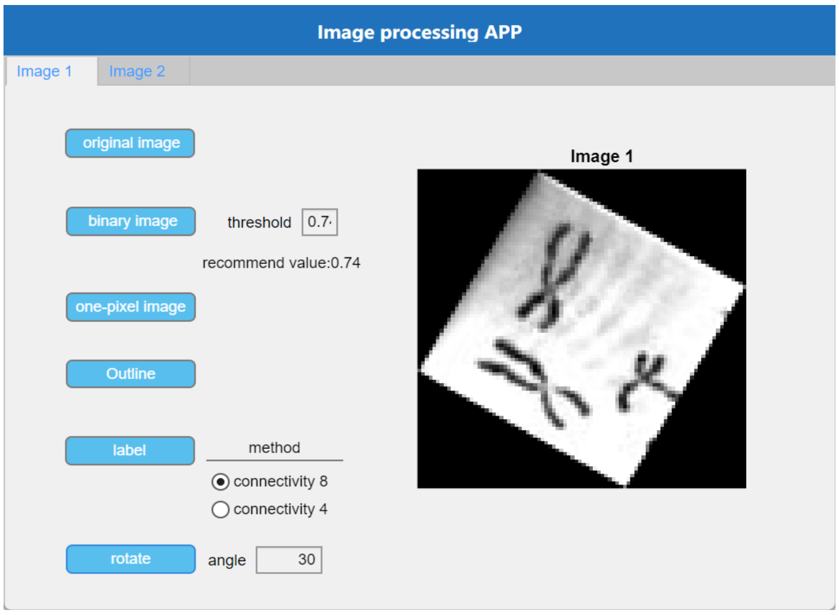


Figure 3.1 Interface of the app

The user can adjust the threshold value for the binary image determination, while the recommended value after several experiments is 0.74. for the connectivity way in the determination of image segmentation, there are two ways available: connectivity 4 and 8. They will show different results in Image 1. And after the users choose the rotation value they decide to apply, the rotated image with the corresponding value will display in the axes area. The main function for image 2 is the same, with one more button for problem 2 (sub the image to the middle part).



threshold 0.7·

recommend value:0.74

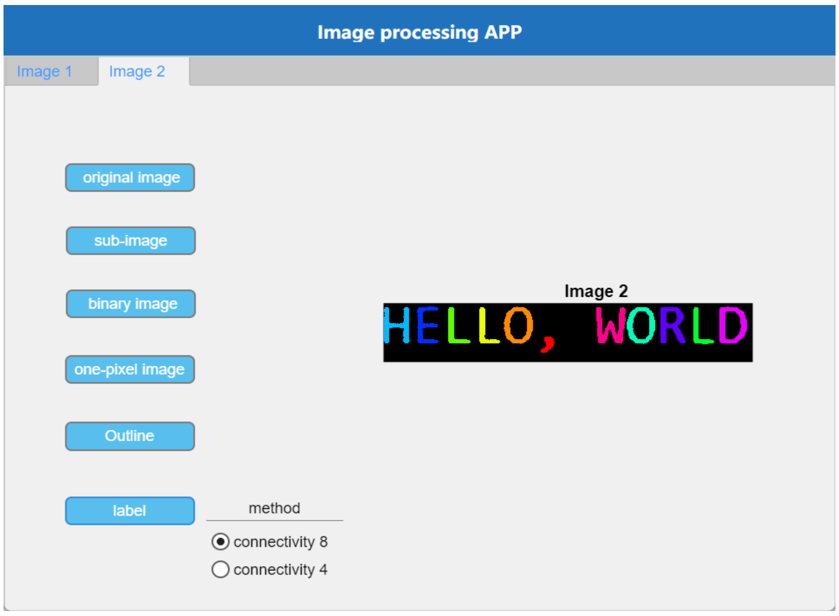
method

(●) connectivity 8

( ) connectivity 4

angle 30

Figure 3.2 Results of Image 1 rotated by 30 degrees in the app



method

(●) connectivity 8

( ) connectivity 4

Figure 3.33 Result of Image 2 labeled by connectivity 8 in the app

If the fundamental parameters or details of the function are not satisfied, users can change the parameters in the call-back function in the app design or the function we create for this project.

## Code Description

Our code can be divided into four parts as follows.

The first one is the main functions, "*Project_image1*" and "*Project_image2*", which can call other functions and return the final results we want, such as image after rotation, the image outline, one-pixel thin of the image …

The second one is the called functions, which can perform a defined task and return the result. For example, the "*CannyEdgeDetection*" function can grab the outline of the input grayscale images and return it. Most of the functions in our folder are called functions.

The third one is the dataset training function, "*knn_train*" and "*som*". Finally, the last one is the App file "*app1.mlapp*".

## Conclusion

In this project, we introduce the principle of five image-processing methods: thresholding, outline, one-pixel thinning, image segment and labeling, and rotation. And present the image-processing results of those methods for two images(Chromosomes Image and Characters Image).

After that, we use the training dataset to train the unsupervised classification method with self-ordered maps (SOM) and k-nearest neighbors (kNN) and introduce their principle. We also discuss the impact of hyperparameter tuning and pre-processing of the data.

Finally, we developed an APP, which has a simple interface and can perform the image processing results according to buttons.

## Reference

[1] File: ASCII-Table-wide.svg from Wikimedia Commons, the free media repository. The website is: https://commons.wikimedia.org/wiki/File:ASCII-Table-wide.svg

[2] ME5405 Machine Vision courseware PDF file “Chapter 2 - Binary Image Processing and Analysis”.

[3] Bradley, D., & Roth, G. (2007). Adaptive thresholding using the integral image. Journal of graphics tools, 12(2), 13-21.

[4] Digital Image Processing- Rafael Gonzalez - 9780133356724 - Computer Science - Computer Graphics (96) Edition 4 April 2017 ISBN13 9780133356724

[5] Guo, G., Wang, H., Bell, D., Bi, Y., & Greer, K. (2003, November). KNN model-based approach in classification. In OTM Confederated International Conferences” On the Move to Meaningful Internet Systems” (pp. 986-996). Springer, Berlin, Heidelberg.