National University of Singapore

ME5405 Machine Vision

Computing Project

Assignment – AY20/21 Semester 1

Authors:	*Matric No.*
Jiang Yuedong	A0225425L
Yong Yao Jun	A0195269M
Wang Yizhuo	A0225440R

November 18, 2020

Contents

1 Introduction

1.1 Backgrounds

In the past few decades, machine vision has played a significant role in the modern industries. It is the technology to take and extract information from an image, which can be widely used in automatic inspection and analysis. For instance, many production lines use machine vision to sort inferior products to control the quality. Robots can use it to provide location and orientation to grasp the product or guide it along a path. Machine vision can also be applied to fields like surveillance, medicine, agriculture, diagnostics, etc. In this module, we have learnt the basic image processing strategies and knowledges. And we will apply them to solve the questions in this project.

1.2 Task statement

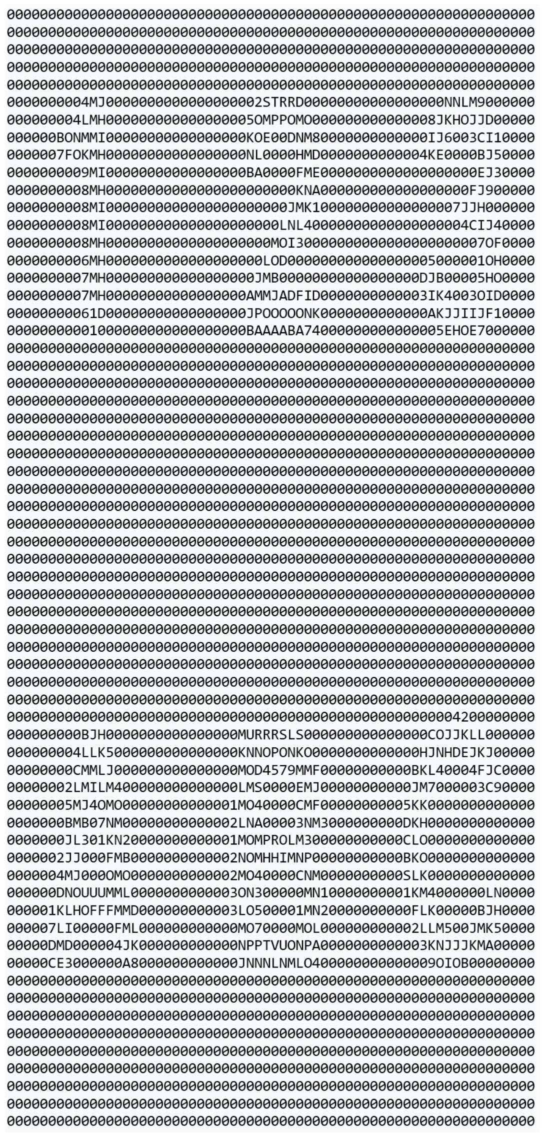
This project requires us to process two images. One has a resolution of 64x64 with 32 gray levels each represented by character 0-9 and A-V (`charact1.txt`), the other one is a JEPG format image of a label on a microchip, where the characters are inverted (texttttchar-act2.jpg). The raw characters/image is shown in Figure 1.

Tasks are required to solved sequentially for each image, and thereby it constructs the structure of this report. We will introduce the algorithms and methodology for each task in Chapter 2, then apply and compare them in our specified problems with MATLAB in Chapter 3. For each image, we are required to do the following operations.

1. Display the original image on screen.

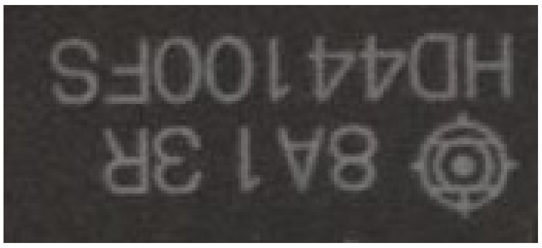
2. Create a binary image using thresholding.

3. Segment the image to separate and identify the different characters.



```
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000004MJ0000000000000000002STRRD00000000000000000NNLM9000000
000000004LMH000000000000000005OMPPOMO000000000000008JKHOJJD00000
000000BONMMI00000000000000000KOE00DNM80000000000000IJ6003CI10000
0000007FOKMH00000000000000000NL0000HMD0000000000004KE0000BJ50000
0000000009MI00000000000000000BA0000FME0000000000000000000EJ30000
0000000008MH00000000000000000000000KNA000000000000000000FJ900000
0000000008MI0000000000000000000000JMK100000000000000007JJH000000
0000000008MI000000000000000000000LNL40000000000000000004CIJ40000
0000000008MH00000000000000000000MOI30000000000000000000007OF0000
0000000006MH0000000000000000000LOD000000000000000005000001OH0000
0000000007MH000000000000000000JMB00000000000000000DJB00005HO0000
0000000007MH00000000000000000AMMJADFID0000000000003IK4003OID0000
00000000061D00000000000000000JPOOOONK00000000000000AKJJIIJF10000
00000000001000000000000000000BAAAABA74000000000000005EHOE7000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000420000000
000000000BJH0000000000000000MURRRSLS000000000000000COJJKLL000000
000000004LLK5000000000000000KNNOPONKO0000000000000HJNHDEJKJ00000
00000000CMMLJ000000000000000MOD4579MMF00000000000BKL40004FJC0000
00000002LMILM400000000000000LMS0000EMJ00000000000JM7000003C90000
00000005MJ4OMO00000000000001MO40000CMF00000000005KK0000000000000
0000000BMB07NM00000000000002LNA00003NM3000000000DKH0000000000000
0000000JL301KN20000000000001MOMPROLM300000000000CLO0000000000000
0000002JJ000FMB0000000000002NOMHHIMNP00000000000BKO0000000000000
0000004MJ000OMO0000000000002MO40000CNM0000000000SLK0000000000000
000000DNOUUUMML0000000000003ON300000MN10000000001KM4000000LN0000
000001KLHOFFFMMD000000000003LO500001MN20000000000FLK00000BJH0000
000007LI00000FML000000000000MO70000MOL000000000002LLM500JMK50000
00000DMD000004JK000000000000NPPTVUONPA0000000000003KNJJJKMA00000
00000CE3000000A8000000000000JNNNLNMLO4000000000000009OIOB00000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
```

(a) Raw text file `charact1.txt`



(b) Raw JPEG file `charact2.jpg`

Figure 1: Raw files.

4. Rotate the characters in the image about their own respective centroids by 90 degrees clockwise.
5. Rotate the characters in the image from Step 4 about their own respective centroids by 35 degrees counterclockwise.
6. Determine the outline(s) of characters of the image.
7. Determine a one-pixel thin image of the characters.
8. Arrange the characters in one line with the sequence: **A1B2C3** for Image 1 and **81344100ARHDFS** for image 2.

We will implement most of the image processing without MATLAB built-in functions and compare the performance of each methods.

2 Methodology and algorithms

2.1 Image display

Image can be stored in many formats. Generally, bitmap (or raster) and vector are two types of images stored and displayed by a computer. Bitmaps are composed of pixels each with one or several values, while vector images use lines, curves and its colors and positions to depict itself. Bitmaps are more commonly seen in our daily life for its convertibility and compatibility. The advantage of vector image is that it can be scaled freely without sacrificing the quality, but usually it needs corresponding software to decode and interpret the file.

Common bitmap format includes JPG/JPEG, PNG, BMP, GIF, etc. The compression technology of JEPG is very advanced and can compress images into a small storage space. But it is lossy at the same time, where excessive compression will lead to quality reduction. The JPEG format mainly cuts down high-frequency information and retains better color information, so it is particularly suitable for Internet, which can reduce image transmission and loading time. PNG is another common format, which supports alpha channel. In other words, PNG supports transparent background. BMP is the standard image format in Windows OS, which includes abundant image information. Normally it takes up a larger space because it barely compresses.

There are other format that can store the information of an image, as long as it tells you the way to decode them. As we can see in Figure 1(a), we need to map each character to a corresponding pixel. For image 2 shown in Figure 1(b), we need to do the rotation before our workflow. These are the preprocessing part of the image display.

2.2 Thresholding

Thresholding is a way to transfer a grey scale image to a binary one. The general thresholding method will replace each pixel in a graph with 0 (black) pixel if the intensity of this pixel $I_{i,j}$ is below some fixed value T_0, while other pixels whose intensity is larger than T_0 will be replaced by 1 (white). This is basically how a binary graph is generated.

The thresholding is under the assumption that the image is greyscale. The JPEG image 2 has three channels, namely R, G, B, representing the red, green and blue channels of an image. The intensity of the greyscale image E should be a weighted sum of these three separated images R, G and B, by eliminating the hue and saturation information while retaining the luminance. According to recommendation of ITU-R BT.601-7 standard, the colored image can convert to greyscale image by following the equation below.

$$E_{i,j} = 0.299 \times R_{i,j} + 0.587 \times G_{i,j} + 0.114 \times B_{i,j} \tag{1}$$

Figure 2 illustrates the intensity of each RGB channel and after the greyscale transformation. After converting image 2 to greyscale,

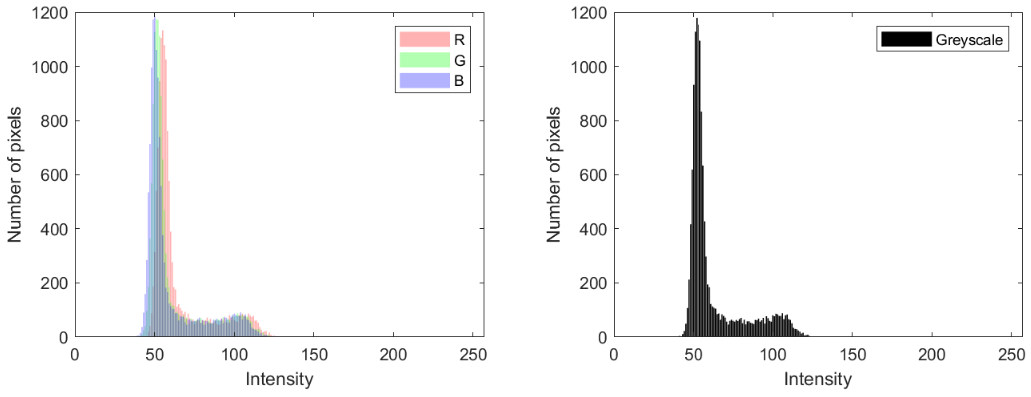


Figure 2: Intensity of RGB channel and the greyscale intensity after conversion of image 2.

we can start thresholding. The key issue of thresholding is to figure out the fixed constant T_0. Here are some common thresholding algorithms we have tried to calculate the threshold value.

- *Mean threshold.* Simply calculate the average pixel density of the greyscale image and use it as the threshold.

- *Bi-model threshold.* This is a method that we come up with, aiming to give a more robust and high-quality binary image. The workflow is demonstrated below.

  – Intercept a histogram interval $[a, b]$ with valid values.
  – Split the histogram into two parts from the middle pixel $m = round(\frac{1}{2}(a + b))$.
  – Find the intensity with the highest amount of pixel in the two intervals $T_1 = \arg\max_i[a, m]$ and $T_2 = \arg\max_j[m, b]$.
  – The threshold will be the average of the two selected intensities $T = \frac{1}{2}(T_1 + T_2)$.

- *Otsu's threshold.* Otsu's method divides the image into two parts - background and foreground, according to the gray level characteristics of the image. It aims at computing a thresholding value T such that the contract between background and foreground is maximized, or in other words, to minimizes the intra-class variance. To be more specific, here is the workflow.

- Calculate the histogram and intensity level probabilities.
- Initialize $w_i(0)$, $\mu_i(0)$
- Iterate over possible thresholds: $t = 0, ..., max\_intensity$.
  Update the values of w_i, μ_i, where w_i is a probability and μ_i is a mean of class i.
  Calculate the between-class variance value $\sigma_b^2(t)$.
- The final threshold is the maximum $\sigma_b^2(t)$ value.

Otsu's algorithm is simple in calculation and not easily affected by the brightness and contrast of images, so it has been widely used in digital image processing.

Finally, set $E_{i,j} = 1$ where the intensity is larger than the threshold $I_{i,j} > T$, and set $E_{i,j} = 0$ otherwise. Then we can get a binarized image.

2.3 Segmentation

Image segmentation refers to the image processing technology that divides an image into different regions with specific properties. There are many segmentation algorithms and can be roughly divided into three categories namely the segmentation based on thresholding, edge detection and region detection.

One of the most basic and important methods of binary image analysis is the connected domain marker, which is the basis of the binary image analysis. Therefore, our project uses this method to segment the image. It is an algorithm that assigns a specific label to each connected region in a binary image, which can be used to locate and count the objects in the image.

Generally, there are two definitions of connected region, which can be divided into 4 and 8 adjacencies. In the figure below, if we consider the 4 adjacencies, we will get three connected domains, and the 8 adjacencies is two connected domains.

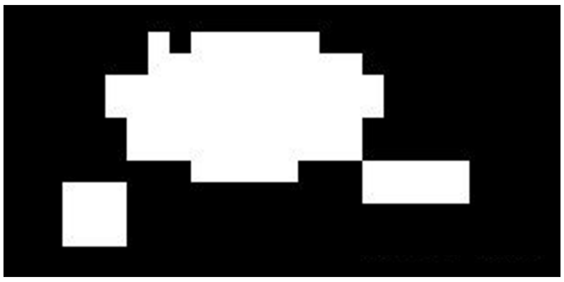


Figure 3: Connected region

There are two basic algorithms for connected region analysis: (1) two-pass scanning twice, (2) seed-filling method. Two-pass method means to find and mark all the connected domains existing in the image by scanning the image twice. The seed filling method comes from computer graphics and is often used to fill a graph. It is based on the region growth algorithm. We use the second method since it is more intuitive. The seed-filling method involves the following steps:

- **Step 1:** The coordinate value of the pixel is queued by mimicking the queue with tuples.
- **Step 2:** Traverse each pixel from left to right and top to bottom to determine the pixel value and if it has been marked. If it has not been marked and the pixel value is 1, queue the position of the pixel.
- **Step 3:** Find out if there are any unmarked pixels with pixel value 1 in the 8 neighborhoods of this pixel; if there is any, give them the same mark as this pixel, and queue this pixel.
- **Step 4:** Remember to add one to the queue header after judging each of the 8 neighborhoods of one pixel.

- **Step 5:** When the queue is empty, a connected region marker is completed. Instead, the queue is empty and the other pixels are judged.
- **Step 6:** Once the region segmentation is completed, record the corner coordinates of the region, for further operations.

The output of this step is important for subsequent rotations. If the area is divided too large, it will result in the same area containing more than one letter. On the contrast, if the partition is too small, a region will not be fully displayed. So, the previous binarization step had a huge impact on this step. And we also encountered the above problems in our actual practice. Currently, we adopt the operation of optimizing the binary image and process the segmented image twice to get a better crop. It is important to note that we ultimately need to obtain normalized coordinates for each segmented image. We can segment letters according to diagonal coordinates, which allows us to perform subsequent rotations and rearrangements.

2.4 Rotation

The result of the previous segmentation will be used in this step. Since we have the location information of each letter block, we can easily arrange them anywhere in the image. It is mentioned in the request: Rotate the characters in the image about their own respective centroids. Therefore, we need to take half the length of the border and calculate the coordinates of the center point.

To rotate any graph along its centroid, as the origin of coordinates, we need three steps:

- **Step 1:** Change coordinate system I into II. The transformation matrix is:

$$\begin{bmatrix} x\text{II} \\ y\text{II} \\ 1 \end{bmatrix} = \begin{bmatrix} x\text{I} \\ y\text{I} \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ -0.5w & 0.5h & 1 \end{bmatrix} = \begin{bmatrix} x\text{I} - 0.5w \\ -(y\text{I} - 0.5h) \\ 1 \end{bmatrix} \tag{2}$$

- **Step 2:** In the coordinate system II, rotate angle of θ. Then, the transformation matrix is:

$$\begin{bmatrix} x_1 & y_1 & 1 \end{bmatrix} = \begin{bmatrix} x_0 & y_0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

- **Step 3:** Change coordinate system II into I. The transformation matrix is:

$$\begin{bmatrix} x\text{I} \\ y\text{I} \\ 1 \end{bmatrix} = \begin{bmatrix} x\text{II} \\ y\text{II} \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0.5nW & 0.5nH & 1 \end{bmatrix} = \begin{bmatrix} x\text{II} + 0.5nW \\ -y\text{II} + 0.5nH \\ 1 \end{bmatrix} \tag{4}$$

We write our custom function `rotate()` which takes in three parameters. First one is the input image, second is the border information for the image, and the third one is degrees by which we want to rotate the image. However, after rotation, there will appear some serious distortion. So we use the nearest neighbor interpolation method to optimize the image. Specifically, after the point in the target image is corresponding to the original image, the pixel value of the nearest integer coordinate point is found and output as the pixel value of that point to fill in the "hole".

Nearest neighbour interpolation is the simplest approach to interpolation. Rather than calculate an average value by some weighting criteria or generate an intermediate value based on complicated rules, this method simply determines the "nearest" neighbouring pixel, and assumes the intensity value of it. To visualize nearest neighbour interpolation, consider Figure 4. The data points in the set X represent pixels from the original source image, while the data points in the set Y represent pixels in our target output image.

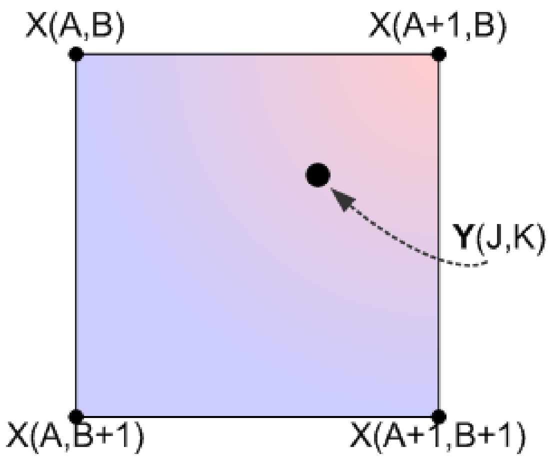


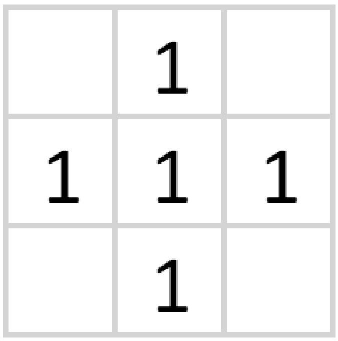
Figure 4: Nearest neighbour interpolation of a data point.

So, for each pixel in the output image Y, we must calculate the nearest neighbouring pixel in our source image X. Furthermore, we should only need to rely on 4 specific data points: $X(A, B)$, $X(A+1, B)$, $X(A, B+1)$ and $X(A+1, B+1)$. The major drawback to this algorithm is that, despite its speed and simplicity, it tends to generate images of poor quality. Although the checkerboard pattern was upsampled flawlessly, images such as photographs come out "blocky", though with little to no noticable loss of sharpness.

2.5 Outline

The outline of an image is to determine the boundary of the image by depicting the outer edge pixels and leaving the interior pixels out. Outlining an image very much depends on the image itself. For a binary image, outlining can be done by simply using a structuring element with 4 or 8 neighbours to erode the interior pixels away.

For determining the outline of the image, a 3×3 structuring element with 4 neighbours (shown in Figure 5 below) was used to erode the binary image.



	1	
1	1	1
	1	

Figure 5: 3×3 structuring element with 4 neighbours.

As long as the 4 neighbouring pixels are 1 (top, bottom, left and right) in the binary image, the centre pixel in the structuring element will be marked with a 1 in the outline image. This is done pixel-by-pixel, from $(i, j) = (2, 2)$ to $(i, j) = (63, 63)$ for char1 and $(i, j) = (2, 2)$ to $(i, j) = (76, 172)$ for char2. In this way, pixels that have 4 neighbours marked as 1 (black) will be marked with a 0 (white). In MATLAB, the function `imgOutline = extract_outline()` was written to find the outline or boundary of the binary image.

2.6 Thinning

Thinning of an image is thin down the image by erosion. For one-pixel thinning, also known as skeletonization, we erode the image in a way so as to reduce the binary image into 1-pixel wide representation.

There are many ways to thin an image, mainly the Stentiford algorithm, Hilditch algorithm, Zhang Suen algorithm, and the Lu Wang algorithm. In this report, the Zhang Suen algorithm will be used. In Zhang Suen thinning algorithm, a 3×3 structuring element is used, as shown in Figure 6 below.

Two steps will be applied on every pixel of the binary image marked with a 1. Note that $P1$ is the pixel to be tested, and $P2$ to $P9$ are the eight neighbouring pixels arranged in a clockwise manner starting from $P2$.

Let $A(P1)$ = number of transitions from 0 to 1 (white to black) in the

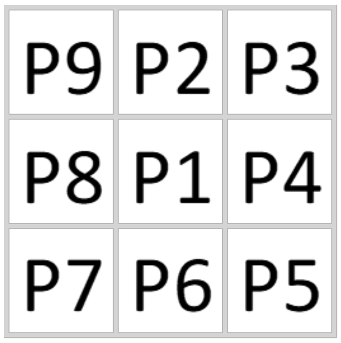


Figure 6: 3×3 thinning structuring element with 8 neighbours.

clockwise sequence of $P2, P3, P4, P5, P6, P7, P8, P9$ and $P2$. Note that $P2$ is repeated twice.

Let $B(P1)$ = number of black pixels in the neighbouring of $P1$ (sum of $P2$ to $P9$).

- **Step 1:** All pixels will be tested in Step 1, and pixels satisfying all 5 conditions below simultaneously will be noted at this stage.
  - **Condition 1.** The pixel is black and have eight neighbours. This stage will rule out white pixels and the corner pixels of the binary image.
  - **Condition 2.** $2 \leq B(P1) \leq 6$
  - **Condition 3.** $A(P1) = 1$
  - **Condition 4.** At least one of $P2$, $P4$ and $P6$ is marked as 0 (white).
  - **Condition 5.** At least one of $P4$, $P6$ and $P8$ is marked as 0 (white).

  After iterating all the pixels in the binary image, the noted pixels that satisfy all 5 conditions will be marked with 0 (white) in the thinned image.

- **Step 2:** All pixels will again be tested in Step 2, and pixels satisfying all 5 conditions below simultaneously will be noted at this stage.

- **Condition 1.** The pixel is black and have eight neighbours. This stage will rule out white pixels and the corner pixels of the binary image.
- **Condition 2.** $2 \le B(P1) \le 6$
- **Condition 3.** $A(P1) = 1$
- **Condition 4.** At least one of $P2$, $P4$ and $P8$ is marked as 0 (white).
- **Condition 5.** At least one of $P2$, $P6$ and $P8$ is marked as 0 (white).

Again, after iterating all the pixels in the binary image, the noted pixels that satisfy all 5 conditions will be marked with 0 (white) in the thinned image.

For any pixels that were marked with a 0 (white) in either stage 1 or stage 2, then the step will be repeated again until there are no more changes in the thinned image. In MATLAB, the function `imgThin = thinning()` is written for this purpose.

2.7 Rearrangement

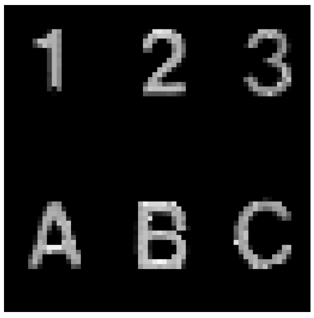
Rearrangement here means to put the segmented characters in order, while making sure that they are aligned and have basically the same size. We have extracted the characters in previous step after performing segmentation and labeling. Hence, we can detect the size of each segmentation, and concatenate them together as the given order. If the size of image varies a lot, we should use nearest neighbour interpolation mentioned in Chapter 2.4 to solve this problem. For the distinguishing of each character, one or a couple of vertical zero paddings can be added between the characters.

3 Implementation and analysis

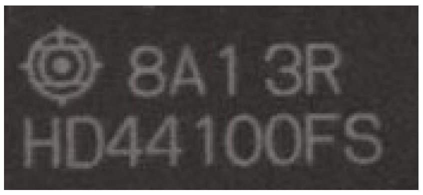
3.1 Image display

Image 1[1] is technically not a image file. It is a text file composed of characters of 0-9 and A-V, where A-V are mapped to number 10-31. The characters should be converted to numbers of 32 grey scale before display. One easy way to accomplish this is to form a dictionary-like table which maps each character to a number. We use `imshow()` function in MATLAB to display the image on the screen. Notice that for image 1, we should declare the grey level $L_{grey} \in [0, 31]$ when using the function, or it will regard the image as 256 grey levels by default.

The character in image 2 is flipped around, so we need to rotate the image by 180 degrees before display. It involves the rotation algorithm in Chapter 2.4. Figure 7 shows the images after preprocessing.



(a) Image 1.



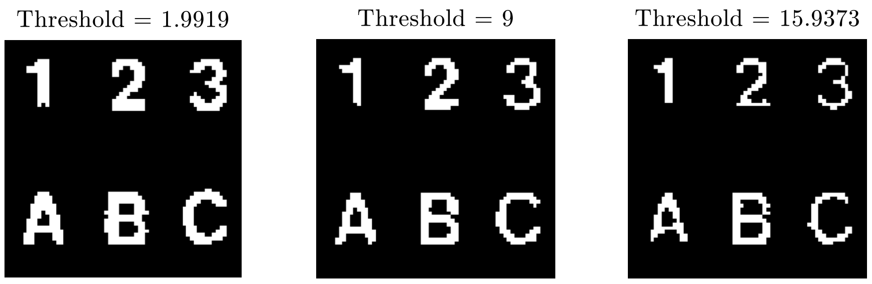
(b) Image 2.

Figure 7: Display image after basic preprocessing.

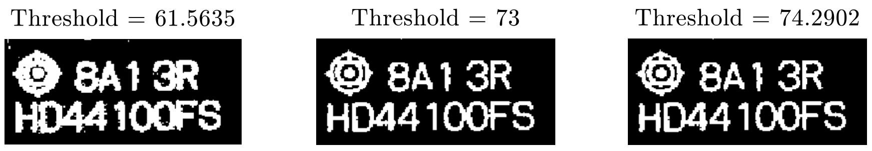
[1]In this project, image 1 refers to the text file, while image 2 refers to the JPEG one.

3.2 Thresholding

We have introduced three different thresholding methods to generate a binary image, and here are the implementations of each method on image 1 and 2 (see Figure 8).



(a) Thresholding on image 1.



(b) Thresholding on image 2.

Figure 8: Thresholding on two images with different algorithms, which are mean threshold, bi-model threshold and Otsu's threshold from left to right.

As we can see, the mean threshold method has the lowest threshold intensity compared to the others, which leads to a larger outline of the character. For instance, in Figure 8(b), many characters cling to each other and there exist much noise on the graph. Otsu's method is great, but the threshold is somewhat a little bit larger in image 1, which result in the loss of information. The character like '2' and 'A' are a little bit thin in edges. Our 'bi-model' method performs well in both image 1 and 2. The histogram shown in Figure 2 shows that there exist two peaks in image 2, which will be chosen as $T_1 and T_2$ in our method, and the threshold T_0 will be chosen to be the midpoint. It extracts the appropriate amount of pixels that have clear and smooth edges. Hence, we choose the method of 'bi-model' thresholding to process the image for further operation.

3.3 Segmentation

3.3.1 Segmentation on image 1

Now we have a binary image in which the pixel containing the letter and the letter is marked as 1, and the background is marked as 0. Then we perform seed-filling method and group pixels into components based on pixel connectivity. After this operation, the resulting tag image has seven unique tags. 0 corresponds to background, tags 1, 2, 3, 4, 5, 6 correspond to '1', '2', '3', 'A', 'B', and 'C', respectively (see Figure 11(a)).

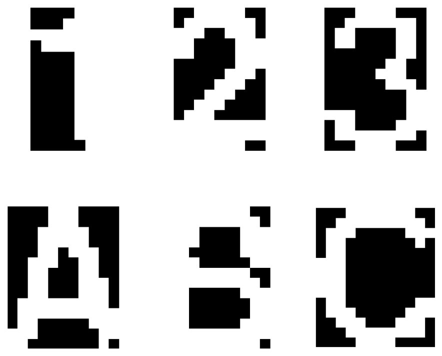


Figure 9: Segmentation of image 1.

From the tagged image, we can extract more features, such as center, bounding box, area and so on, for each label. A bounding box is the smallest rectangle, oriented along the rows and columns of the image, and it can fully contain the desired region of interest (ROI). In our case, ROI is each tag of the foreground object that we get from the tagged image. Once we have the bounding box, we have the center of it as the center of the object, the width of the box corresponds to the width of the object, and the height of the box corresponds to the height of the object. These are all in preparation for the subsequent rotation and rearrangement.

3.3.2 Segmentation on image 2

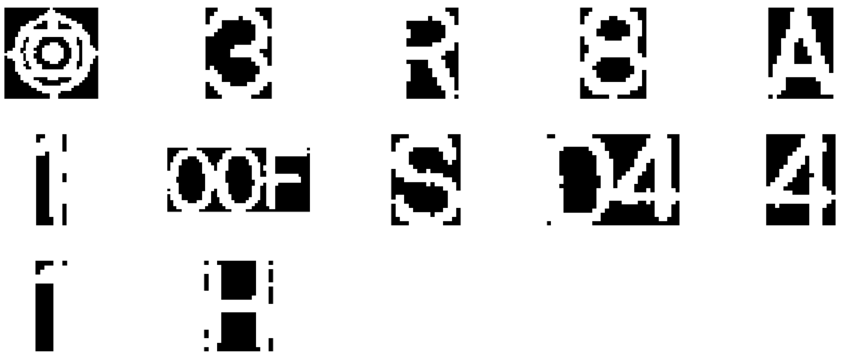
We ran into some problems with segmenting the second image in the same way. The first problem is that the center of the logo is also counted as a segmented image and being displayed. That is natural because the center of this logo is apart from the logo itself, and we display it according to the border. Hence, it will appear to us that the logo and its center circle will both be displayed. Therefore, we add a loop in our program to detect if the segmented area is unusually smaller than the others, if so, we will remove it from the border list.

Another problem that can be seen from Figure 10(a) is that some letters stick together. For instance, '00F'and 'D4' are connected together and not well separated. The reason is that in the binary image, there is no separation between these letters, which is due to the low resolution and image quality. Therefore, we reprocess the image whose length is larger than the width to judge the approximate numbers inside, and split them vertically according to the numbers. The output result of the second graph after segmentation is shown in Figure 10(b). Similarly, we also record the border information of the letter blocked for subsequent processing.

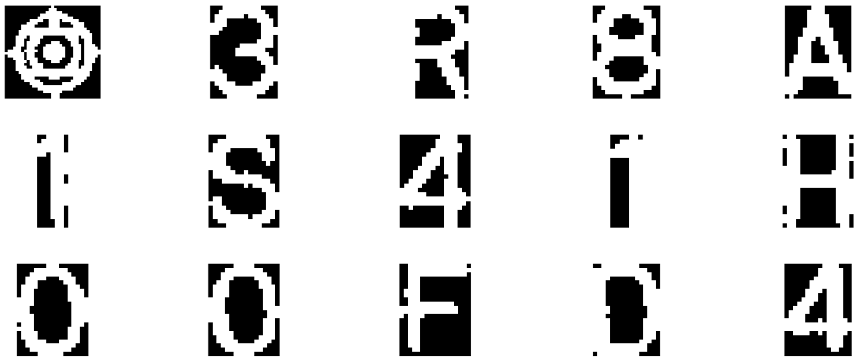
3.4 Rotation

3.4.1 Rotate characters $90°$ clockwise

Figure 11 below shows the letter rotated 90 degrees with the help of our custom function `rotate()`.



(a) Segmentation of image 2.

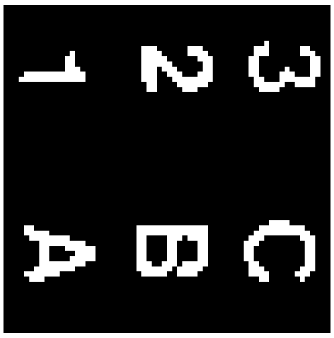


(b) Segmentation after splitting.

Figure 10: Segmentation before and after splitting of image 2.

3.5 Rotate characters 35° counter-clockwise based on last step

The Figure 12(a) and 12(c) illustrate the letter rotated 35 degrees from step 4 with the help of our custom function `rotate()`. However, we found that there are a lot of black spots in the rotated image, which makes the image not representable. In order to improve the effect, we use the nearest neighbor interpolation method to progress our graph, and the improved images are shown in Figure 12(b) and 12(d).


(a) Image 1.

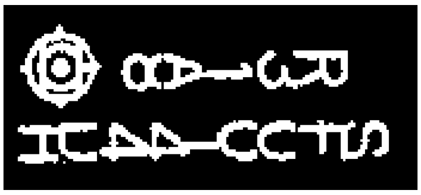

(b) Image 2.

Figure 11: Rotation about the respective centroids of characters by 90 degrees clockwise.

3.6 Outline

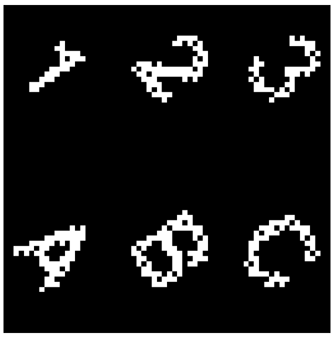
The results and analysis for outlining of char1 and char2 are described in this section.

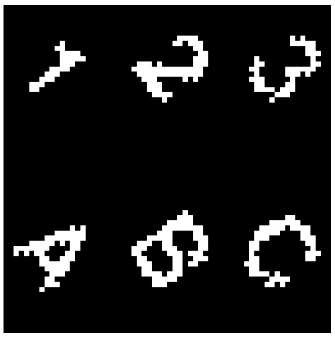
3.6.1 Outlining on image 1

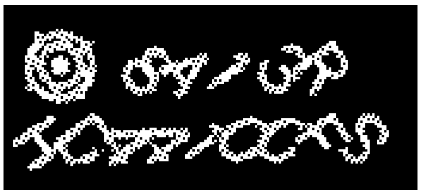
After the outlining process describe in Chapter 2.5 is done on the binary image 1, an outline binary image is created with all boundary pixel being marked as 1 shown in Figure 13 below.

As the binary image is thin at some areas, the outlined image looks fat because it is only 2 pixels thick. This is especially prominent for characters 1, 3 and B. A way to fix this is to first do a closing morphology on the binary image itself.

By using a closing morphological operation, we can thicken the binary image using dilation before we apply the erosion function on the image. In this way, the outlined image would not look fat. However, do note that as the characters are small, the characters A and B will likely look 'squeezed' and deformed.


(a) Rotation of image 1 before interpolation.


(b) Rotation of image 1 after interpolation.


(c) Rotation of image 2 before interpolation.

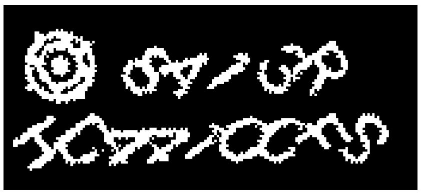

(d) Rotation of image 2 after interpolation.

Figure 12: Rotation about the respective centroids of characters by 35 degrees counter-clockwise based on last step, then interpolate the graph with the nearest point to fill in the interspace.

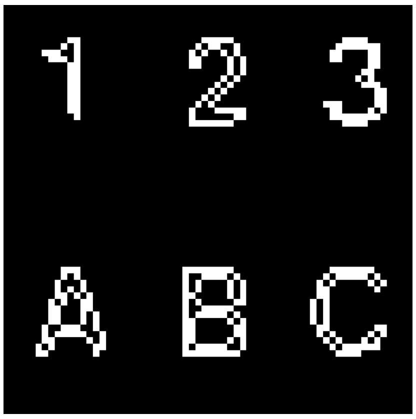


Figure 13: Outline image 1 after erosion using the 3 × 3 structuring element.

3.6.2 Outlining on image 2

Similarly, to find the boundary of image 2, an erosion morphological operation is applied on the binary image 2 with a similar structuring element as described in Figure 5. After the outlining process is applied, an outline image is created as in Figure 14 below. The edges of the characters will be marked as 1 and the interior marked as 0. For image 2, the outlined image does not have the issue as seen on

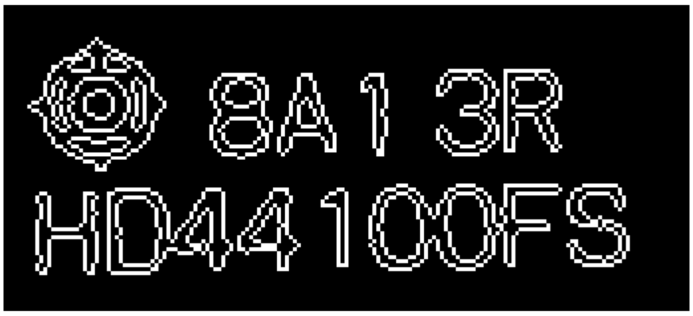


Figure 14: Outline image 2 after erosion using the 3 × 3 structuring element.

image 1. This is because the binary image 2 are much wider than that of image 1.

3.7 Thinning

The results and analysis for outlining of image 1 and 2 are described in this section.

3.7.1 Thinning on image 1

After applying the Zhang Suen algorithm on the binary image of char1, the one-pixel thinned image is shown below in Figure 15.

As described in Chapter 2.6, the algorithm will be applied on the binary image repeatedly until no changes are seen. All the characters are visible even after skeletonizing the image, which means that the Zhang Suen algorithm is working.

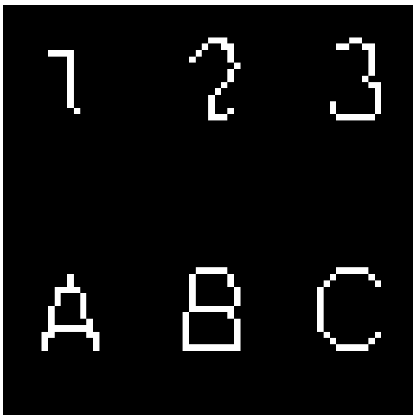


Figure 15: One-pixel thinned image 1 using Zhang Suen algorithm

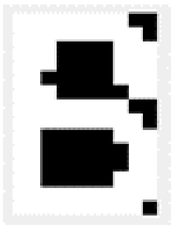


Figure 16: Binary image of character ‘B’

However, do note that the character B after thinning look very similar to the character 8. This is because the edges of B are not very well defined in the binary image as seen in Figure 16.

3.7.2 Thinning on image 2

Similarly, the Zhang Suen thinning algorithm are applied for the binary image of char2, and the result shown in Figure 17.

The thinned image is generally legible. However, the characters 4 does not have the tailed-end visible due to the fact that the binary image before thinning have very short tails. Also, it is noted that the characters D4 and 00F are joined. This is because the characters in the threshold binary image are joined together as well, and the

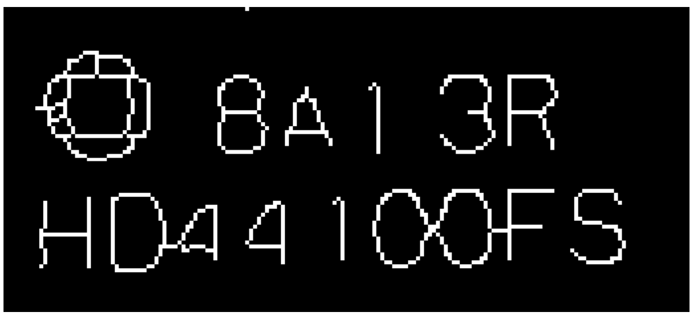


Figure 17: One-pixel thinned image 2 using Zhang Suen algorithm

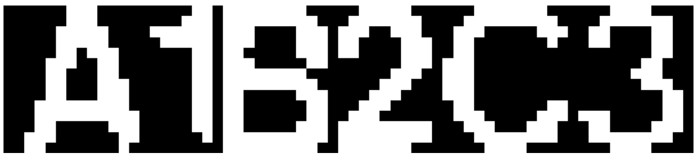
thinning process does not split the characters separately. This can be solved by using a better thresholding.

3.8 Rearrangement

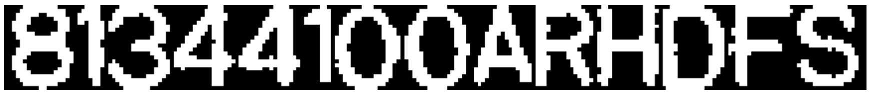
Till now, we have the border locations of all the characters, and their center location can be calculated by this. We find that for image 1, the height of all the six characters are exactly the same, and for image 2, the height of each character remains nearly equal, 5 of 14 characters are 22-pixel high, and the smallest in height is character '1', which has 20-pixel height, the rest of them have 21-pixel height. Hence we can see, they do not differ a lot in size, and there seems no need to scale them to the same size. Therefore, we find the maximum of the height and fill the character in it, and then concatenate them into a line in sequence as requested in the task. We write a MATLAB function `arr_char()` to input the image and border, then it will output the rearranged characters, which are shown in Figure 18.

4 Conclusion

This project includes the basic operations to process the image. Most progress are the same for the two images, however, we think image 2



(a) Arrange image 1 into A1B2C3.



(b) Arrange image 2 into 81344100ARHDFS.

Figure 18: Rearrange the characters into a given sequence.

is slightly harder to deal with compared to image 1. Because image 2 contains more characters and the shape is more vague. Although thresholding of two images are basically the same, but the result can influence the sequential segmentation operation a lot. A threshold too low will induce much noise and make characters cling to each other, making it difficult to segment. A threshold too high will lose information of the image and make it hard to distinguish.

For the two images, the segmentation effect makes a big difference. Because the first image is better after binarization and the image is relatively simple, there is no need for much processing after segmentation. In the second picture, due to the small spacing between letters and even the connection situation after binarization, the segmentation cannot be performed well. We need to carry on the follow-up processing to the segmented image to produce the better result.

For both images, the process of rotation is the same and the result is nearly the same.But in the second image, the spacing between the letters is smaller, so the rotated images overlap with each other.

During extracting the outline, we are able to outline all the characters in image 2 compared to image 1 as the binary image for image 2 is generally thicker and wider than in image 1. As described above, not all portion of image 1 is well outlined due to the thickness of the binary image itself.

We apply the same Zhang Suen thinning algorithm on both binary images. However for image 1, we are able to skeletonize each character of the binary image separately, as compared to image 2. The time taken to process image 2 is at least 2 times higher compared to image 1 because there are more pixels in image 2 than in image 1.

I believe this project would be a great starting for image processing and machine vision for us, and we as a team, have learnt a lot through the process. Thank you Professors!

5 Programming details

5.1 Usage of the program

Our program contains eleven MATLAB `.m` files (Octave is also supported) and two given raw character files that we need to process. Here are the usage explanation of each file. For better code readability and integration, the variable `mode` is equal to 1 for image 1, and 2 for image 2. This variable is incoming to make called functions distinguish the image type, since there exists some differences for image 1 and 2.

```
main1.m         % Main entrance, run this file to process
                % all the operations for image 1.
main2.m         % Main entrance, run this file to process
                % all the operations for image 2.
show_img.m      % Simply display the image.
img2binary.m    % Use three methods to do the thresholding
                % and display the value and image of them.
segment.m       % Segment the input binary image and
                % output the board locations of each corner.
rotate.m        % Rotate the image by a certain degree.
```

```
                  % Input binary image, boards and the
                  % rotation degree.
near_point_inter.m  % Nearest neighbour iteration.
extract_outline.m   % Extract outline edges of the image.
thinning.m        % Thin out the image to one-pixel edge.
arr_char.m        % Rearrange the characters to a given
                  % sequence. Input binary image and borders.
rgbhist.m         % Display the RGB and greyscale channels.
charact1.txt      % Given raw file 1.
charact2.jpg      % Given raw file 2.
```

5.2 Overall flowchart

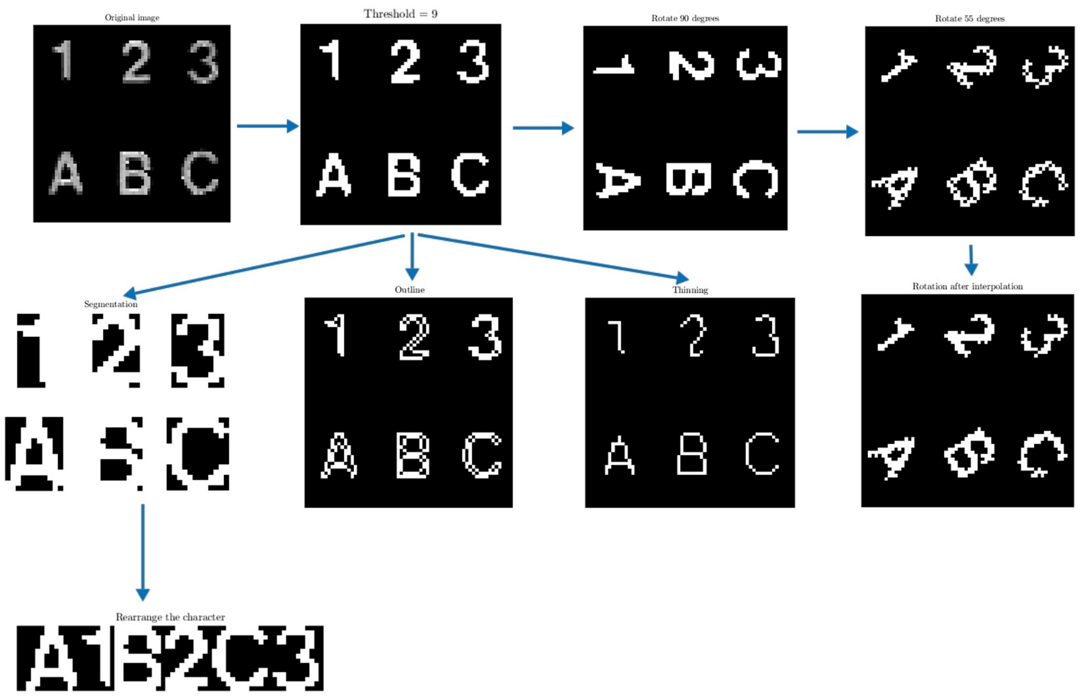


Figure 19: Image 1.

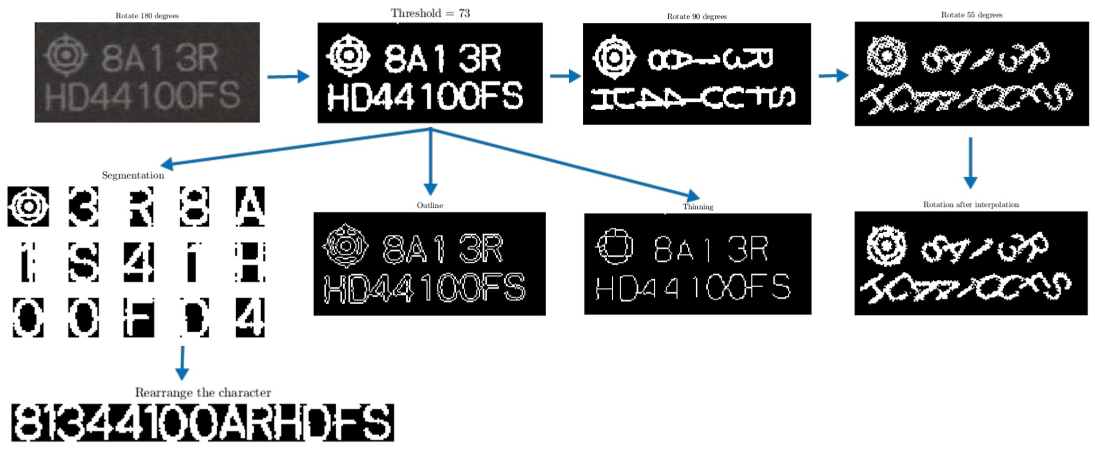


Figure 20: Image 2.

5.3 Algorithms flowchart

Bi-model thresholding:

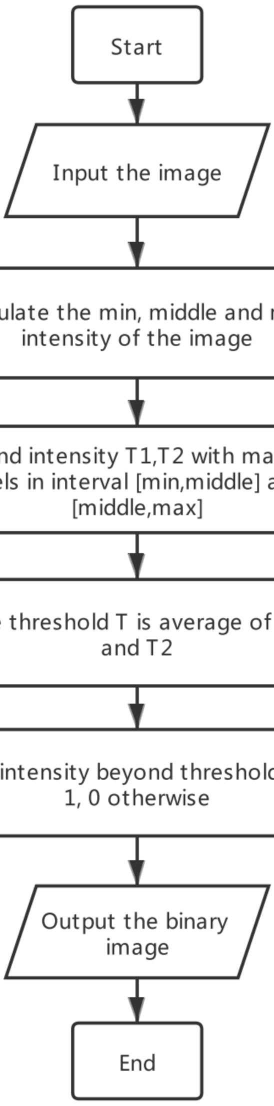

Start
Input the image
Calculate the min, middle and max intensity of the image
Find intensity T1,T2 with max pixels in interval [min,middle] and [middle,max]
The threshold T is average of T1 and T2
Set intensity beyond threshold to 1, 0 otherwise
Output the binary image
End

Figure 21: Bi-model thresholding flowchart.

Segmentation:

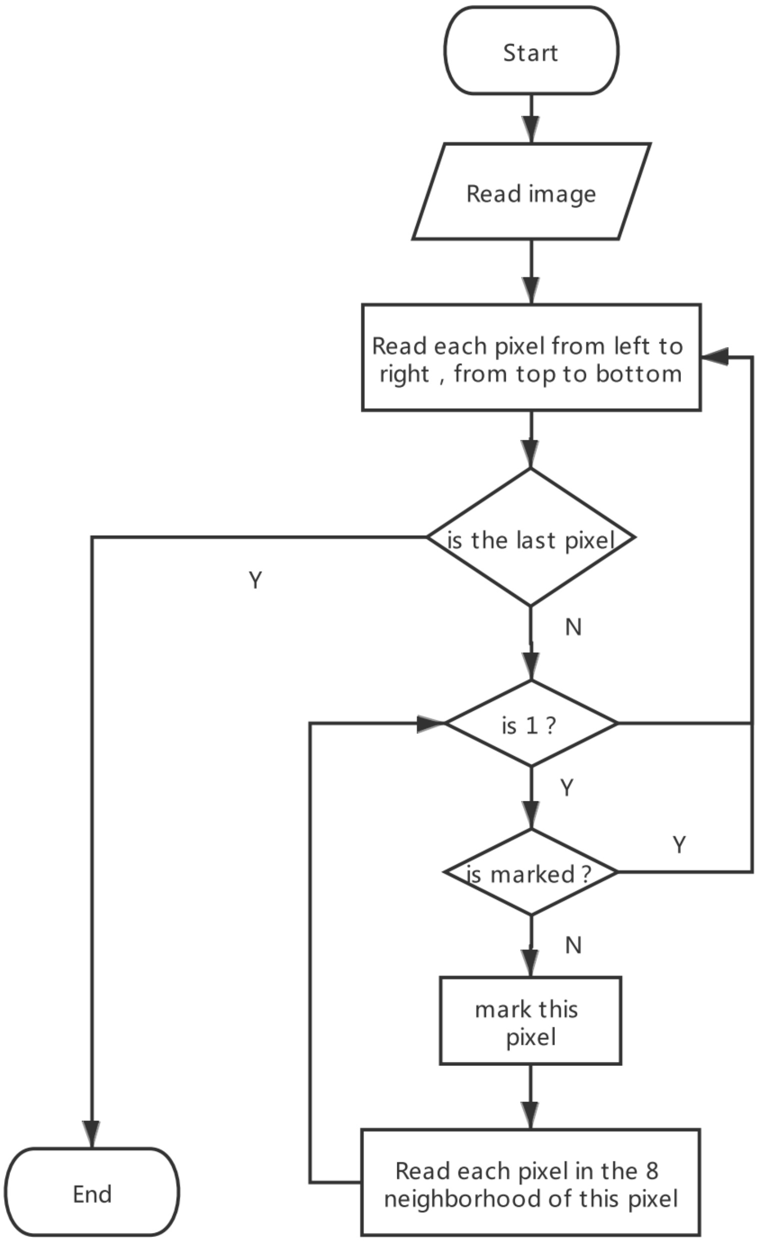
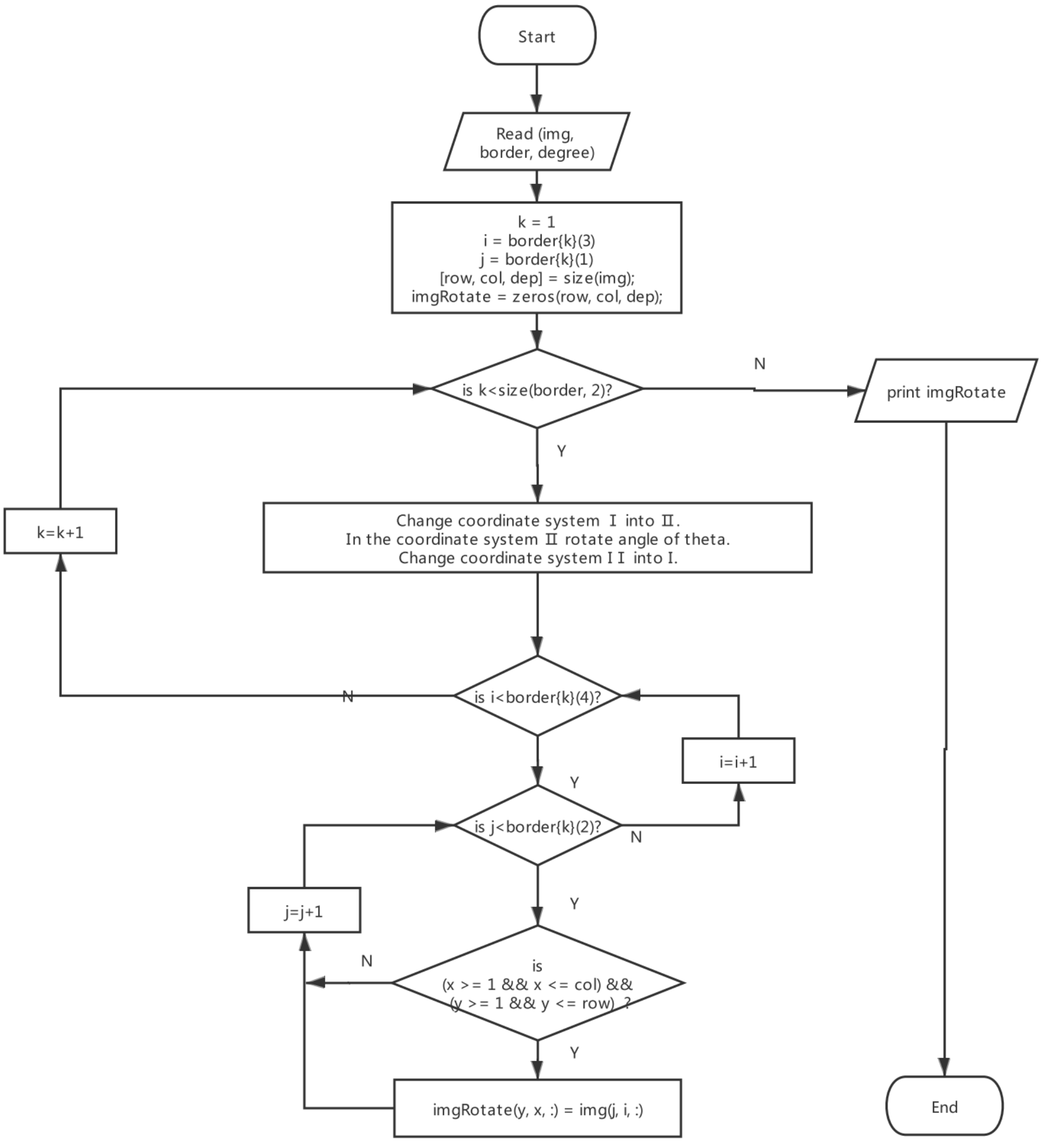

Start
Read image
Read each pixel from left to right , from top to bottom
is the last pixel
Y
N
is 1 ?
Y
is marked ?
Y
N
mark this pixel
Read each pixel in the 8 neighborhood of this pixel
End

Figure 22: Segmentation flowchart.

Rotation:



Start
Read (img, border, degree)
k = 1
i = border{k}(3)
j = border{k}(1)
[row, col, dep] = size(img);
imgRotate = zeros(row, col, dep);
is k<size(border, 2)?
N
print imgRotate
Y
k=k+1
Change coordinate system Ⅰ into Ⅱ.
In the coordinate system Ⅱ rotate angle of theta.
Change coordinate system ⅠⅠ into Ⅰ.
N
is i<border{k}(4)?
i=i+1
Y
is j<border{k}(2)?
N
j=j+1
Y
N
is
(x >= 1 && x <= col) &&
(y >= 1 && y <= row) ?
Y
imgRotate(y, x, :) = img(j, i, :)
End

Figure 23: Rotation flowchart.

References

Gonzalez, R. C., Woods, R. E., & Eddins, S. L. (2004). Digital image processing using MATLAB. Pearson Education India.

Otsu, N. (1979). A threshold selection method from gray-level histograms. IEEE transactions on systems, man, and cybernetics, 9(1), 62-66.

https://en.wikipedia.org/wiki/Rec._601

https://en.wikipedia.org/wiki/Otsu%27s_method