

MACHINE VISION PROJECT

(ME5405)

SUBMITTED

BY

LIN KANG (A0191818W)

REN SIJIE (A0191758N)

TANG SHENGNAN (A0169257R)

DATE

11ST NOVEMBER 2018

Contents

[1. Introduction 3](#_Toc23517)

[2. Methodology 4](#_Toc23255)

[2.1 Display the images 4](#_Toc14756)

[2.2 Pre-processing 4](#_Toc23169)

[2.3 Thresholding 6](#_Toc3789)

[2.4 Outline determination of the object 7](#_Toc302)

[2.5 Image Segmentation 8](#_Toc16497)

[2.6 Rotation 9](#_Toc13028)

[2.7 Thinning 9](#_Toc12419)

[2.8 Scaling and Translating 10](#_Toc18352)

[3. Results 11](#_Toc25914)

[3.1 Image 1 11](#_Toc27104)

[3.2 Image 2 12](#_Toc28883)

[4. Discussion 14](#_Toc22328)

[4.1 Origin image read 14](#_Toc17533)

[4.2 Pre-processing 14](#_Toc12128)

[4.3 Binary image 15](#_Toc25130)

[4.4 Outline determination 16](#_Toc4896)

[4.5 Image segmentation 16](#_Toc8369)

[4.6 Thinning 16](#_Toc24916)

[4.7 Spatial transformation 17](#_Toc17238)

[5. Conclusion 17](#_Toc20927)

[Annex A. Codes of Image 1 18](#_Toc32596)

[Annex B. Codes of Image 2 24](#_Toc1056)

1. **Introduction**

Digital Image Processing is a computer-based image enhancement, noise filtering, restoration, segmentation and feature extraction techniques. It is widely used in image sharpening and restoration, medical field, remote sensing, transmission and encoding, machine/robot vision, color processing, pattern recognition, video processing, microscopic imaging and so on.

In this project, we used digital image processing methods to deal with two different images separately, and programmed by MATLAB. For each image, the process procedures included:

1. Display the original image on screen.
2. Threshold the image and convert it into a binary image.
3. Determine the outline(s).
4. Segment the image to separate and identify the different characters.
5. Rotation of the characters about their own respective centroids by 90 degrees clockwise.
6. Rotation of the characters about their own respective centroids by 30 degrees counterclockwise.
7. Determine a one-pixel thin image of the characters from Step 4.
8. Scale and display the characters of Image 1 in one line with the sequence: A1B2C3.
9. Scale and display the characters of Image 2 in one line with the sequence: 7M2HD44780A00

The whole report includes 6 parts: chapter 1 introduces background and problem; chapter 2 illustrates the methodology and corresponding implementation steps of each image processing; chapter 3 shows the results of two images for each process; chapter 4 analyzes image processing results and discusses the difference of processing methods between two images; chapter 5 summarizes the whole project; and the Annex shows detailed MATLAB programs of each process.

1. **Methodology**
2. **Display the images**

Image can be considered as a 2D continuous light intensity function, f (x, y), where x and y are spatial coordinates, and f at (x, y) is related to the brightness or color of the image at that point. But an image to be processed by computer must be represented using discrete data structure like matrix or 2D array. Sampling and quantization are the usual way to digitalize the image. Sampling turns the continuous function f (x, y) into a matrix of N rows and M columns, each element of the matrix of samples is called pixel. Quantization assigns an integer value to each continuous sample, usually it refers to the representation of the brightness level (in grayscale image) or color value (in RGB format) at each sampled point (pixel).

In this project, we processed two different images. Images 1 is a 64x64, 32 level images. The image is shown a coded array that contains an alphanumeric character for each pixel in the image. The range of these characters is 0-9 and A-V, which corresponds to 32 gray levels. So to display this image in the screen, first we use 'fscanf'function with the limitation '%s' to read the data of image in form of character string. After that we use 'base2dec' function to transform character string to numbers. Finally, since the MATLAB 'imwrite' function could only save the gray image in form of 0-1 value, we need the matrix divided by 32 and then save it.

Image 2 is a BMP image of a label on a microchip. 'imread' function was used to read the image, and 'rgb2gray' function was applied to transform the color image into gray level image, finally 'imshow' function was used to display the image.

1. **Pre-processing**

The main purpose of image pre-preprocessing is to eliminate irrelevant information in the image and enhance the detectability of the relevant information.

Since the image 2 has a lot of noise, before further convert it to the binary image, removing noise to get better image quality is very necessary. There are many methods that can achieve noise reduction such as spatial domain filters and frequency domain filters.

In this project, Markov random field (MRF) formulation was applied to remove the noise from image 2. ( 'Introduction to Markov Random Fields', Andrew Blake and Pushmeet Kohli)

A Markov Random Field (MRF) is a graphical model of a joint probability distribution. It consists of an undirected graph IMG_256 in which the nodes IMG_257 represent random variables. An image is an assembly of many nodes. These nodes may represent the hidden variables (segmentations). A joint probabilistic model is built over the pixel values and the hidden variables.

The Hidden Markov models (HMM) are particularly useful as prior models for state variables Xi (labels) that are to be inferred from a corresponding set of measurements or observations z = () (pixel values). The observations z are themselves considered to be instantiations of a random variable Z representing the full space of observations that can arise. It leads naturally to an inference problem in which the posterior distribution for the possible states X, given the observations z, is computed via Bayes's formula as

P (X = x | Z = z) ∝ P (Z = z | X = x)P (X = x)

Here *P* (X = x) is the prior distribution over states, that is, what is known about states X in the absence of any observations.

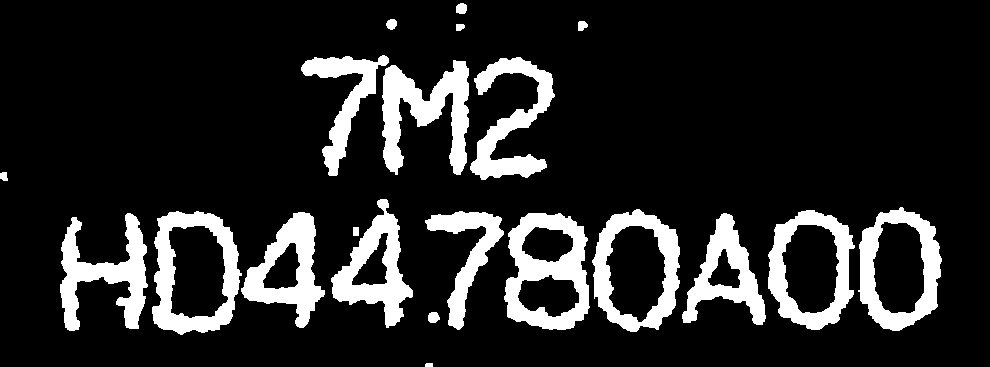
The omitted constant of proportionality would be fixed to ensure that . Often multiple models are considered simultaneously, and in that case this is denoted

The most common form of inference to solve this MRF formulation is to use Maximum A Posterior (MAP) estimation. MAP inference of x is done by computing , or equivalently by minimizing energy: .

To solve our problem of removing the noise from the graph. Two labels (x) are set for the graph: background (black) and objects (white). The aim is to label every pixel with the labels. The energy function formulated is consisted of unary term and the pairwise terms .

The unary term measures the photo consistencies of the original image and the output (the differences of the pixel intensities). The pairwise term measures the smoothness of the output image. It is assumed that the expected output is smooth. If a pixel is next to a pixel which belongs to the background, it is very likely the pixel also belongs to the background.

The paper 'Class Segmentation and Object Localization with Superpixel Neighborhoods' (B. Fulkerson, A. Vedaldi, and S. Soatto. *In Proceedings of the International Conference on Computer Vision,* 2009.) provides a graph-cut based energy minimization. A matlab implementation was developed by Brian Fulkerson and used here. Below is the output of the minimization.



1. **Thresholding**

In object recognition and detection by a computer vision system, the input image can be simplified by generating an output whose pixels: tend to have high values if they are part of an objects of interest; and low values if they are not. This is also called thresholding. The method is as follow:

* Determine a threshold value;
* Those pixels having gray value higher than the Threshold value are given the value 1;
* Those pixels having gray value lower than the Threshold value are given the value 0.

By this way, we can identify the extent of the objects. However, the main difficulty is to choose a good threshold value. Since most of time it may have distribution overlaps between the dark and bright pixels, which make it difficult to identify the threshold value. The existence of noise may also influence the threshold identification if the gray levels of the object and the background are fairly close.

For the image 1 of the project, since all the pixel values of background were already set to 0, and all the pixel values of object were higher than 0, so we need to scan the image row by row, and set these pixel values higher than 0 to be 1, then we can get a binary image.

For the image 2, in preprocess the 'graythresh' function was used to determine an optimal threshold for the image. Then it was used in the 'im2bw' function to convert the image to binary.

1. **Outline determination of the object**

We can determine the object outline by scanning the image of size N by N. A change in gray level from one band to the other denotes the presence of a boundaries. This is also called global thresholding. The procedures are as follows:

1. For each row in f(x, y), i.e. x = 0, 1, …., N-1; create a corresponding row in an intermediate image g1 using the following relation for y = 0, 1..., N-1:

where LE and LB are specified edge and background levels, respectively.

1. For each column in f(x, y), i.e., y = 0, 1, ….., N-1; create a corresponding row in an intermediate image using the following relation for y = 0, 1..., N-1:
2. Put them together:

For image 1, since we already had transformed the original images into binary images, all the pixel values of object were set to 1, and all the pixel values of background were set to 0. Then judge the vertical boundary and the horizontal boundary by subtract the neighbor pixel. The value of pixel which on the boundary (LE) is 1 or -1 and the value of background (LB) is 0. After that, put them together to determine the outline of the images.

For image 2, we used 'Erosion – Boundary Extraction': firstly, use '4-connected Structuring Element' to erode the object outline, then use the original binary image to subtract this result, by this way we can get the object outline. The methodology of erosion was illustrated in the section 2.7 in the thinning part, and detailed processing steps was described in section 3.3 when compare processing methods between two images.

1. **Image Segmentation**

Segmentation is a process that sub-divides an image into its constituent parts or objects. Segmentation algorithms for monochrome images generally are based on one of the two basic properties of gray-level values: discontinuity and similarity. Discontinuity is to partition an image based on abrupt changes in gray-level. The principal areas of interest within this category are the detection of isolated points and detection of lines and edges in an image. Similarity approach is based on thresholding, region growing, and region merging and splitting.

Detection of discontinuities include three types: points, lines and edges. The common way is to run a mask through the image. In edge detection, gradient operators are effective way to process images with sharp intensity transitions and relatively low noise. Edge linking is usually followed to link these edge pixels into meaningful boundaries.

In this project, it was much easier to segment the binary images than grey level images. So based on the binary images which have been achieved in section 2.2, we used 'bwconncomp' function and 8-connectivity way to find the connected regions in binary images. The output of 'bwconncomp' function is a struct which contains the connectivity method, imagesize, numobjects and the pixelidxlist. The coordinates of each componments' pixel are saved in pixelidxlist so that we can use random color to label each componments.

1. **Rotation**

Rotation is a form of pixel coordinate transformation, also known as spatial transformation. The key of process is to use a rotation matrix to map the pixel (x, y) to a new position . In 2D images, the rotation matrix can be written as

In this project, we need to rotate characters about their own respective centroids by 90 degrees clockwise and 30 degrees counterclockwise, then can be chosen as and separately. However, this rotation matrix is relative to the centroids of each characters, we can't use each pixel ordinate to multiply this matrix directly. So first we use 'regionprops (CC, 'Centroid')' function to return measurements for the set of centroid for each connected component, assume (x, y) is the centroid coordinates of one character, then amend rotation matrix by translation.

In this way, the derived new rotation matrix is relative to the system original ordinates, so for each pixel of one character, the rotated coordinates are

1. **Thinning**

Erosion is a good way to remove some parts of the image, such as removing lines and thinning lines. The erosion of A by B is defined by:

To determine a one-pixel thin image of the characters, erosion is an effective way. First create a structural element, then make the image be eroded by the structural element. For image 1, we divided it into four steps:

1. Based on the binary image in section 2.3, which the object was white and the background was black, we define the structure matrix by the function 'strel ('arbitrary', erosion\_7\_1)' and use 'imerode' function to erode the original binary image. The target of this step is to erode the image horizontally.
2. Created a structural element with the structure matrix , used 'imerode' function to finish image erosion vertically;
3. Since after two steps the image could not achieve goal that the image is one pixel thin, we need to delete the bottom pixel. In order to achieve this, we create a structural element with the structure matrix , and use 'imerode' function to find the extra pixel. Then subtract these pixel to get the final pixel.
4. Due to the ability limited, this method delete some pixels wrongly. We use the function 'bwmorph(A7, 'bridge') ' to connect some of the blanks.

For image 2, we tried to use 'bwmorph (image,'thin', Inf)' function in MATLAB. We set the function to operate infinite steps until the next result is the same as the previous one, then we get the one-pixel image.

1. **Scaling and Translating**

Scaling and translating are two forms of spatial transformation. For 2D images, the translation matrix is

The scaling matrix is

For image 1, first we need to create a new matrix to contain the series of components. Then put the ABC in the original positions with the pixelidxlist. After that translate the characters 123 and insert the blank between ABC to achieve the final sequence: A1B2C3, the translation matrix is

Then we use 'imresize' function to make image scale three times, which means scaling matrix is

For image 2, first we translate characters7M2 to the left, then move HD44780A00 to the right and up, in this way the characters will be displayed in one line with the sequence: 7M2HD44780A00

The translation matrix for characters 7M2 is

The translation matrix for characters HD44780A00 is

Then we use 'imresize' function to scale the image to three times, scaling matrix is

1. **Results**
2. **Image 1**

**  **

(a) Display image1 (b) Binary image (c) Determine outline  

(d) Segmented image (e) Rotation, 90CW (f) Rotation, 30CCW

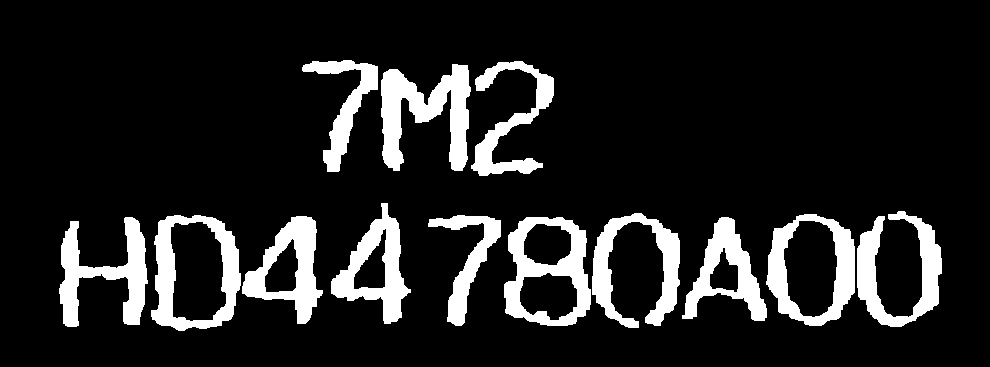
** **

(g) One-pixel thin image (h) Scale image and translated objects

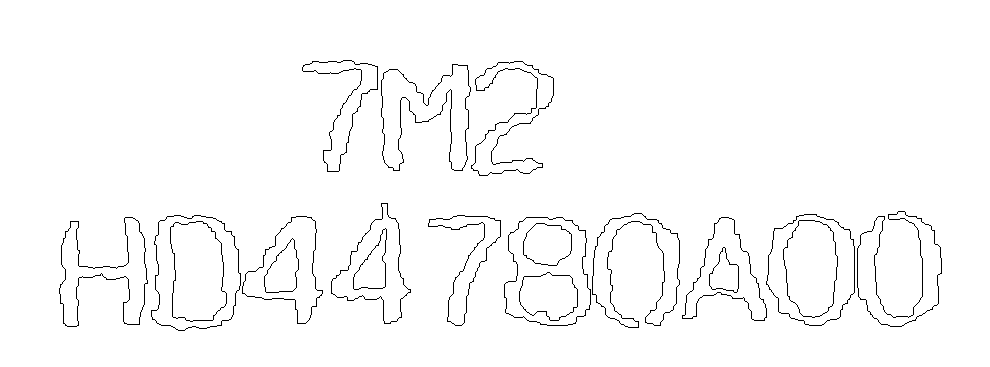
1. **Image 2**

****

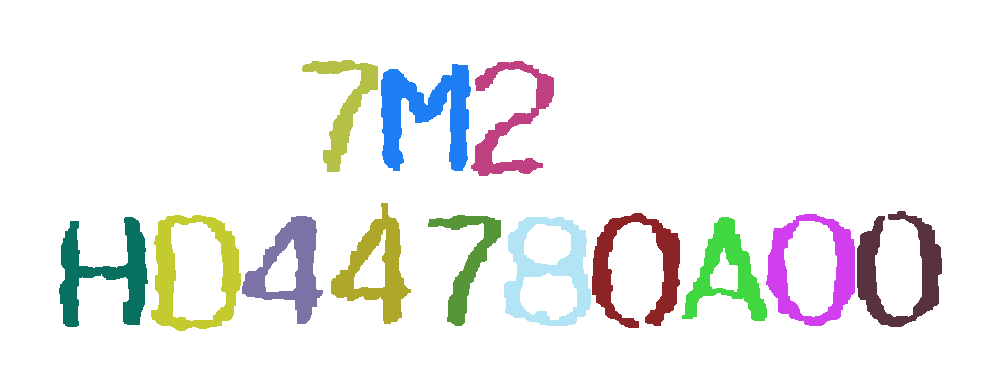
1. Display image2



1. Binary image after pre-processing



1. Determine outline



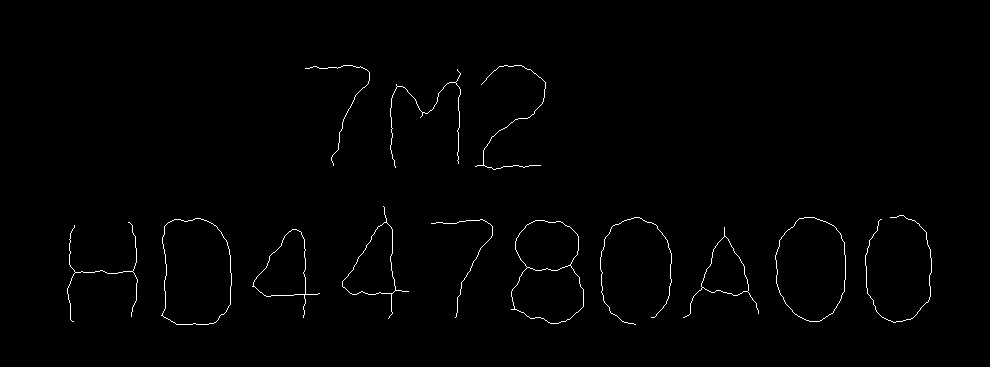
1. Segmented image



1. Rotation, 90CW



1. Rotation, 30CCW



1. One-pixel thin image



1. Scale image and translated objects
2. **Discussion**
3. **Origin image read**

Since the image 1 is a coded array that contains an alphanumeric character for each pixel in the image, we cannot use the 'imread' function to get the image matrix directly as the image 2. In order to achieve the image 1, we read the txt contents in form of character string to get the matrix of the whole image. However, this matrix should be converted since we cannot apply calculation on character string. Function 'base2dec' in MATLAB is used to achieve conversion. On the other hand, we can use function 'imread' to get the matrix of image directly.

1. **Pre-processing**

In fact, we have try some other ways to denoise. After converting the original picture to binary-image. We tried to use opening to denoise. For 4-connected structuring element, we operate it 4 times to eliminate all isolated noise(Fig 1.). For 8-connected structuring element, we operate it 3 times to eliminate all isolated noise(Fig 1.). But we can see from the two results, there are too much discontinuity respectively and some distortion compared to original shape.



Fig 1. Opening using 4-connected structuring element



Fig 2. Opening using 8-connected structuring element

Markov random field (MRF) formulation was applied to remove the noise. However, even though we have done noise reduction so that the characters of the image were discernable, there were also some isolated noise and some character outlines were still connected together. For example, the two characters '0' were connected together because of noise, made it difficult to segment. We choose some area in the image and make the value of the pixel to be 0, so that it is easy to be segmented. After this step, there are also some isolated noise in the image. So we still need to do something. We choose to erode and dilate the image to achieve that. From the result, however, there are also two discontinuity in two 'O' number. While we reduced the discontinuity, there are also some improvement we should do in our project.

1. **Binary image**

Hence, erosion and dilation operation was applied to split the connected characters. Any noise may be identified as an isolated object, therefore the noise removal of image 2 is very important.

Binary image is very useful to identify the object shape. Both two images applied thresholding method to get the binary images. For the image 1, it is easy to find the threshold value, while for image 2, the 'graythresh' function was used to determine an optimal threshold for the image before denoising.

1. **Outline determination**

Remove function

For image 1, global thresholding method was applied to determine the character outlines. However, in image 2, we tried to achieve it by erosion. Firstly, we used 'strel ('arbitrary', nhood)' ,function to create a structural element with the structure matrix , then applied 'imerode' function to make the image be eroded by the structural element, in this way, we could eliminate the outline of characters. Secondly, we let binary image which derived in section 2.2 to subtract this processed image, to get the object outline while remove the inside of the outline. Finally, complement operation was used to get the final results which shown in section 3.2. Both two methods could achieve outline determination successfully, global thresholding was much simpler, while erosion operation needed more effort to design structural element.

Additionally, the function 'bwmorph(A2,remove)' could also achieve the goal. And the effect of this method is quite similar to the former method. In order to accumulate the experience, we try to use knowledge learn from the lecture to solve the problem.

1. **Image segmentation**

The segmentation of image 1 was much easier since there were no characters that connected together, so we could use 'bwconncomp' function to segmentation based on connectivity. It is able to find and isolate connected objects.

Segmentation of image 2 needed more effors because any noise may be identified as an isolated object. So in pre-processing, we realized denoising and elimination of connections between character outlines, then we could use the same way as the image 1 to do further segmentation.

1. **Thinning**

For the image 1, it was done by erosion method. As you can see from the figure g in section 3.1, all of the characters achieved pixel thin successfully. However, some pixels were over processed resulting in some discontinuities. And in this method, we need to apply the erosion step by step to get one-pixel thin image.

For the image 2, we also want to apply erosion to the matrix at first. However, it's quite difficult to get the one-pixel thin image as the image 2 is more complicate than image 1 which makes it difficult to choose the appropriate matrix as the erosion matrix. Inappropriate matrix could make the final image discontinues or get two-pixel thin in some position. So we use new function 'bwmorph (image,'thin', Inf)' was applied to remove pixels on the boundaries of objects but does not allow objects to break apart, the pixels remaining make up the image skeleton. However, it also caused redundant pixels remaining on the image.

1. **Spatial transformation**

Since image rotation, translation and scaling are all belong to spatial transformation, so we would discuss them together. Both two images used transformation matrix to achieve ordinate transformation. By changing the variables in the matrix, we could realize rotation at any angle, translation in different directions and scaling of different multiples.

1. **Conclusion**

This project was aiming to get experience of using MATLAB to do the real image processing. Based on the knowledge learned from the lecture notes as well as extracurricular research, we finally finished the project. Different methods were applied during each processing stage. By analyzing the image results and trying different new methods, we had better understanding of the advantages and disadvantages of each image processing method and the scope of their application.



# Annex A. Codes of Image 1

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%%clear  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
clear all; close all;  
fid=fopen('charact1.txt'); %open the txt  
data=fscanf(fid,'%s'); %read the information from the txt in form of character string  
fclose(fid);  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%%Q1  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
A1=zeros(64); %create the white matrix  
for i=1:64  
 for j=1:64  
 A1(i,j)=base2dec(data((j-1)\*64+i),32); %convert the string number strn of the specified base into its decimal equivalent  
 end  
end  
A1=A1'/32; %transform the number to the form of 0-1 in order to imwrite gray image correctly  
imwrite(A1,'Q1-1.png');  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%%Q2  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
A2=zeros(64); %create the white matrix  
for i=1:64 %use thresholding to convert the gray image to black-white image  
 for j=1:64  
 if A1(i,j)>0  
 A2(i,j)=1;  
 else  
 A2(i,j)=0;  
 end  
 end  
end  
imwrite(A2,'Q1-2.png');  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%%Q3  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
b\_vb=zeros(64);%create vertical boundary matrix matrix  
b\_hb=zeros(64);%create horizontal boundary matrix  
A3=ones(64);%create the white matrix  
for i = 1:64  
 for j = 1:64  
 if j<64  
 b\_vb(i,j) = A2(i,j+1) - A2(i,j) ; %vertical boundary matrix  
 end  
 if i<64  
 b\_hb(i,j) = A2(i+1,j) - A2(i,j) ; %horizontal boundary matrix  
 end  
  
 if (b\_vb(i,j)==0 & b\_hb(i,j)==0)%judge the whole boundary  
 else  
 A3(i,j) = 0;  
 end  
 end  
end  
imwrite(A3,'Q1-3.png');  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%%Q4  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
Cal = bwconncomp(A2); %use the bwconncomp to segment the image  
A4=255\*ones(size(A2,1),size(A2,2),3,'uint8'); %create the basic matrix  
for i=1:Cal.NumObjects  
 Group=Cal.PixelIdxList(i);  
 r = randi([0 255],1,3); %random color  
 for j=1:size(Group{1,1},1)  
 a = Group{1,1}(j); %calculate the pixel coordinates  
 b = rem(a-1,size(A2,1))+1; %calculate the pixel coordinates  
 A4(b,(a-b)/size(A2,1)+1,:) = [ r(1),r(2),r(3)];%random color for each connected structure  
 end  
end  
imwrite(A4,'Q1-4.png');  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%%Q5  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
center=regionprops(Cal,'Centroid');%find the centroid of each componments as the rotation center  
A5 = 255\*ones(size(A2,1),size(A2,2),3,'uint8');  
rotation1 = pi/2;%rotate angle 90 degrees clockwise;  
for k=1:size(center,1)  
 t=rotation1;  
 xx = center(k).Centroid(1);%horizontal coordinates of centroid  
 yy = center(k).Centroid(2);%vertical coordinates of centroid  
 R = [cos(t) -sin(t) xx-cos(t)\*xx+sin(t)\*yy; sin(t) cos(t) yy-sin(t)\*xx-cos(t)\*yy; 0 0 1]; %compute rotation matrix  
 K = Cal.PixelIdxList(k);  
 for j=1:size(K{1,1},1)  
 a = K{1,1}(j);  
 b = rem(a,size(A2,2));  
 temp = [(a-b)/size(A2,2);b; 1]; %in homogeneous coordinates  
 temp = R\*temp; %apply rotation transformation  
 A5(floor(temp(2)),floor(temp(1)),:) = A4(b,(a-b)/size(A2,2),:); %assign the info of A4 to A5  
 end  
end  
imwrite(A5,'Q1-5.png');  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%%Q6  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
A6 = 255\*ones(size(A2,1),size(A2,2),3,'uint8');  
rotation2 = -pi/6; %%rotate angle 30 degrees counterclockwise;  
for k=1:size(center,1)  
 t=rotation2;  
 xx = center(k).Centroid(1);  
 yy = center(k).Centroid(2);  
 R = [cos(t) -sin(t) xx-cos(t)\*xx+sin(t)\*yy; sin(t) cos(t) yy-sin(t)\*xx-cos(t)\*yy; 0 0 1]; %compute rotation matrix  
 K = Cal.PixelIdxList(k);  
 for j=1:size(K{1,1},1)  
 a = K{1,1}(j);  
 b = rem(a,size(A2,2));  
 temp = [(a-b)/size(A2,2);b; 1]; %in homogeneous coordinates  
 temp = R\*temp; %apply affine transformation  
 A6(floor(temp(2)),floor(temp(1)),:) = A4(b,(a-b)/size(A2,2),:); %assign the info of A4 to A6  
 end  
end  
imwrite(A6,'Q1-6.png');  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%%Q7  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
erosion\_7\_1=[1 1 1];  
SE=strel('arbitrary',erosion\_7\_1);  
A7=imerode(A2,erosion\_7\_1);%erode the horizontal pixel  
erosion\_7\_2=[1;1];  
SE=strel('arbitrary',erosion\_7\_2);  
A7=imerode(A7,erosion\_7\_2);%erode the vertical pixel  
erosion\_7\_3=[1 1;1 1];  
SE=strel('arbitrary',erosion\_7\_3);  
temp=imerode(A7,erosion\_7\_3);  
A7=A7-temp; %erode the other pixel  
A7=bwmorph(A7,'bridge'); %connect one pixel blank  
imwrite(A7,'Q1-7.png');  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
%%Q8  
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  
vertical\_distance = 28;%define the distance between the first horizontal and the second line  
horizontal\_move = 10;%define the hroizontal move to stagger the componments  
A8 = 255\*ones(32,70,3,'uint8');  
for i=1:2:size(Cal.PixelIdxList,2)  
 K = Cal.PixelIdxList(i);  
 for j=1:size(K{1,1},1)  
 a=K{1,1}(j);  
 b=rem(a,size(A2,2));  
 A8(b,(a-b)/size(A2,2),:)=A4(b,(a-b)/size(A2,2),:);%define the new martix for new sequence  
 end  
end  
for i=2:2:size(Cal.PixelIdxList,2)  
 K=Cal.PixelIdxList(i);  
 for j=1:size(K{1,1},1)  
 a=K{1,1}(j);  
 b=rem(a,size(A2,2));  
 A8(b-vertical\_distance,(a-b)/size(A2,2)+horizontal\_move,:)=A4(b,(a-b)/size(A2,2),:); %move the 123 right and upward to insert into the blank between ABC  
 end  
end  
A8=imresize(A8,3);  
imwrite(A8,'Q1-8.png');

# Annex B. Codes of Image 2

Q1

close all;  
clear all;  
I = imread('charact2.bmp');  
I = rgb2gray(I);  
imshow(I);  
imwrite(I,'Q2-1.png')

Q2

clear all;  
close all;  
I2 = imread('./denoise/pre2.png');  
nhood = [1;1;1;1;1;1;1;1;1;1];  
SE = strel('arbitrary',nhood); %define Structuring element  
I2 = imerode(I2,SE); %erode vertically  
for i =1:2  
 nhood = [1;1];  
SE = strel('arbitrary',nhood); %define Structuring element  
I2 = imerode(I2,SE); %erode vertically  
end  
nhood = [1,1,1,1,1;1,1,1,1,1];  
SE = strel('arbitrary',nhood); %define Structuring element  
I2 = imdilate(I2,SE); %erode horizontally  
for i=1:2  
nhood = [1;1];  
SE = strel('arbitrary',nhood); %define Structuring element  
I2 = imdilate(I2,SE); %erode horizontally  
end  
imshow (I2);  
imwrite(I2,'Q2-2.png');

Q3

nhood = [0,1,0;1,1,1;0,1,0];  
SE = strel('arbitrary',nhood); %define Structuring element  
I3 = imerode(I2,SE); %erode  
I3 = I2-I3; %erosion-boundary extraction  
I3 = 1-I3;  
imshow (I3);  
imwrite(I3,'Q2-3.png');

Q4

CC = bwconncomp(I2,8); % compute connected structure  
I4 = 255\*ones(size(I2,1),size(I2,2),3,'uint8'); % initialise white img  
for(i=1:size(CC.PixelIdxList,2)) %operate on each connected component  
 K = CC.PixelIdxList(i);  
 r = randi([0 255],1,3); %random color for each connected structure  
 for(j=1:size(K{1,1},1))  
 a = K{1,1}(j); %calculate the pixel coordinates  
 b = rem(a-1,size(I2,1))+1; %calculate the pixel coordinates  
 I4(b,(a-b)/size(I2,1)+1,:) = [ r(1),r(2),r(3)];  
 end  
end  
imshow(I4);  
imwrite(I4,'Q2-4.png');

Q5

S = regionprops(CC,'Centroid');  
I5 = 255\*ones(size(I2,1),size(I2,2),3,'uint8'); % initialise white img  
  
t = pi/2; %set the angle of rotation  
for(k=1:size(S,1))  
 xx = S(k).Centroid(1);  
 yy = S(k).Centroid(2);  
 R = [cos(t) -sin(t) xx-cos(t)\*xx+sin(t)\*yy; sin(t) cos(t) yy-sin(t)\*xx-cos(t)\*yy; 0 0 1]; %compute rotation matrix  
 K = CC.PixelIdxList(k);  
 for(j=1:size(K{1,1},1))  
 a = K{1,1}(j); %calculate the pixel coordinates  
 b = rem(a-1,size(I2,1))+1; %calculate the pixel coordinates  
 temp = [ (a-b)/size(I2,1)+1;b; 1]; %in homogeneous coordinates  
 temp = R\*temp; % apply affine transformation  
 I5(floor(temp(2)),floor(temp(1)),:) = I4(b,(a-b)/size(I2,1)+1,:); %assign the info of I4 to I5  
 end  
end  
imshow(I5);  
imwrite(I5,'Q2-5.png');

Q6

S = regionprops(CC,'Centroid');  
I6 = 255\*ones(size(I2,1),size(I2,2),3,'uint8'); % initialise white img  
t = -pi/6; %set the angle of rotation  
for(k=1:size(S,1))  
 xx = S(k).Centroid(1);  
 yy = S(k).Centroid(2);  
 R = [cos(t) -sin(t) xx-cos(t)\*xx+sin(t)\*yy; sin(t) cos(t) yy-sin(t)\*xx-cos(t)\*yy; 0 0 1]; %compute rotation matrix  
 K = CC.PixelIdxList(k);  
 for(j=1:size(K{1,1},1))  
 a = K{1,1}(j); %calculate the pixel coordinates  
 b = rem(a-1,size(I2,1))+1; %calculate the pixel coordinates  
 temp = [ (a-b)/size(I2,1)+1;b; 1]; %in homogeneous coordinates  
 temp = R\*temp; % apply affine transformation  
 I6(floor(temp(2)),floor(temp(1)),:) = I4(b,(a-b)/size(I2,1)+1,:); %assign the info of I4 to I6  
 end  
end  
imshow(I6);  
imwrite(I6,'Q2-6.png');

Q7

I7 = bwmorph(I2,'thin',Inf);  
imshow(I7);  
imwrite(I7,'Q2-7.png');

Q9

move\_left = 300;  
move\_right = 200;  
move\_up = 155;  
I9 = 255\*ones(200,1200,3,'uint8'); % initialise white image  
for(i=1:size(CC.PixelIdxList,2))  
 K = CC.PixelIdxList(i);  
 if( i== 4 || i==6 || i==8)  
 for(j=1:size(K{1,1},1))  
 a = K{1,1}(j); %calculate the pixel coordinates  
 b = rem(a-1,size(I2,1))+1; %calculate the pixel coordinates  
 I9(b,(a-b)/size(I2,1)+1 -move\_left ,:) = I4(b,(a-b)/size(I2,1)+1,:); %translate characters7M2 to the left  
 end  
 else  
 for(j=1:size(K{1,1},1))  
 a = K{1,1}(j); %calculate the pixel coordinates  
 b = rem(a-1,size(I2,1))+1; %calculate the pixel coordinates  
 I9(b-move\_up,(a-b)/size(I2,1)+1+move\_right ,:) = I4(b,(a-b)/size(I2,1)+1,:);  
 end  
 end  
end  
I9 = imresize(I9,2); %scaling two times  
imshow(I9);  
imwrite(I9,'Q2-9.png');

**Pre-process:**

Step1:

close all;  
clear all;  
I = imread('charact2.bmp');  
level = graythresh(I);  
I = im2bw(I, level);  
I(70:90,377:383)=0;  
I(250:290,850:860)=0;  
I(285:306,590:594) = 0;  
  
I = 255 \* repmat(uint8(I), 1, 1, 3);  
source\_color = [255;255;255] ; %foreground  
sink\_color = [0;0;0] ; % background  
source\_color = double(source\_color);  
sink\_color = double(sink\_color);  
N = size (I,1) \*size(I,2);  
pairwise = sparse(N,N);  
unary = zeros(2,N);  
n=1;  
for ( n = 1:N)  
 j = rem(n-1,size (I,2))+1;  
 i = (n - j)./size (I,2)+1;  
 unary (1,n) = norm( double(reshape(I(i,j,:),[3,1])) - source\_color);  
 unary (2,n) = norm( double(reshape(I(i,j,:),[3,1])) - sink\_color);  
 if(unary (1,n)>unary (2,n))  
 segclass(n,1) = 1; % background  
 else  
 segclass(n,1) = 0; % foreground  
 end  
  
 if(i < size (I,1))  
 pairwise(n,n+size (I,2)) = 1;  
 pairwise(n+size (I,2),n) = 1;  
 end  
  
 if (j <size (I,2))  
 pairwise(n,n+1) = 1;  
 pairwise(n+1,n) = 1;  
 end  
 n = n+1;  
end  
save ('precomputed\_variables.mat')

Step2:

close all;  
clear all;  
load 'precomputed\_variables.mat';  
  
m\_lambda = 600;  
label\_cost = double([ 0,m\_lambda; m\_lambda,0]);  
[labels Eafter] = GCMex(segclass, single(unary), pairwise, single(label\_cost),0);  
  
I2 = I;  
for(n=1:N)  
 j = rem(n-1,size (I,2))+1;  
 i = (n - j)./size (I,2)+1;  
 if(labels(n,1) >0.5)  
 I2(i,j,:) = [0 0 0]; %yellow  
 else  
 I2(i,j,:) = [255 255 255] ; %blue  
 end  
end  
  
level = graythresh(I2);  
I2 = im2bw(I2, level);  
I2(70:90,377:383)=0;  
I2(245:297,854:859)=0;  
I2(285:306,590:594) = 0;  
  
imwrite(I2,'pre2.png');  
imshow(I2);