|  |
| --- |
| ME 5405 Machine Vision |
| Assignment |
| Submitted by |
| Dai Weiheng 学号 |
| Luo Zhiyao学号 |
| Yang Wenjie学号 |

[1 Introduction 3](#_Toc21877381)

[2 Methodology 4](#_Toc21877382)

[2.1 Image display 4](#_Toc21877383)

[2.2 Thresholding **错误!未定义书签。**](#_Toc21877384)

[2.3 Pattern Fragmentation 7](#_Toc21877385)

[2.4 Character rotation 11](#_Toc21877386)

[2.5 Outline recognition 11](#_Toc21877387)

[2.6 Character Thinning 11](#_Toc21877388)

[2.7 Character Rescheduling 11](#_Toc21877389)

[3 Summary 12](#_Toc21877390)

[3.1 Overall flowchart and screen dumps 12](#_Toc21877391)

[3.2 Comparison of algorithm implemented in the two images 12](#_Toc21877392)

# Problem Introduction

The project requires us to process two images which are respectively composed of a 64 by 64 coded array that contains an alphanumeric character (i.e. image1) and a BMP colored image (i.e. image2). According to the instruction, we are expected to complete tasks including image display, binarization, character fragmentation, image rotation, outline depiction, picture skeletonization and character reordering using Matlab.

In chapter 2 we step by step introduce the algorithms implemented to fulfill the tasks and compare the peculiarity of using different methodologies.

# Methodology

## Image display

### Image Reading

Matlab provides a variety of image reading command supporting different formats. In terms of image2, we can simply read the file using the following command:

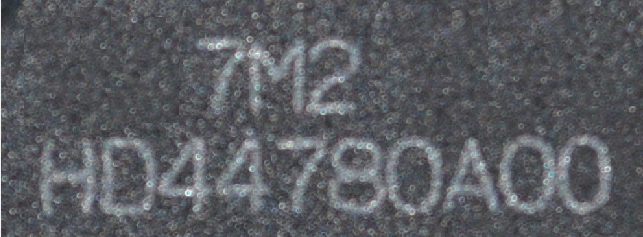
|  |  |  |
| --- | --- | --- |
|  | Im2\_rgb = imread('charact2.bmp'); |  |

where rgb is a variable assigned by the matrix containing color and intensity information of Image 2. More specifically, the variable rgb is a 3679903 uint8 array representing 367990 pixels in red, green and blue color channels respectively.

To show Image 2 on screen, we utilize command:

|  |  |  |
| --- | --- | --- |
|  | imshow(Im2\_rgb); |  |

The screen dump we obtain is shown in figure.



For Image1, it is not possible to derive the image array though ready-made function in Matlab. Firstly, the txt file is read into Matlab in the format of string matrix. By inspection the matrix is 14096 so we reshape it into a 2-dimensional 6464 string matrix. After that, every element is traversed in the matrix and judged by a conditional judgement statement to find out the string whose ASCII code is not less than the ASCII code of ‘A’. Then we reassign corresponding number value to those elements which satisfy the criteria of the conditional judgment so that the whole array will be transformed into a string array full of number strings. Finally we use function “double()” to convert the string matrix into a double-floating matrix. In this manner, Image 1 will then be able to displayed on screen using “imshow()”, as can be seen in figure. The Matlab code of showing Image1 is as follows:

|  |  |  |
| --- | --- | --- |
|  | fileID = fopen('charact1.txt');  formatSpec = '%s';  A = fscanf(fileID,formatSpec);  A = reshape(A, [64,64]);  image1 = zeros(size(A));  for i=1:size(A)  for j=1:size(A)  if A(i,j)-'A'<0  image1(i,j)= str2num(A(i,j));  else  image1(i,j)=A(i,j)-'A'+10;  end  end  end  image1 = double(image1); |  |

A much more succinct method of mapping A-Z to 10-35 is to apply table array. A ASCII match table shown as table is built in order to construct the corresponding one-to-one match.

|  |  |  |  |
| --- | --- | --- | --- |
| number | 0~9 | ASCII code between 10~A | A~Z |
| string | 0~9 | 0 | 10~35 |

Matlab code of the conversion is shown as follows:

|  |  |  |
| --- | --- | --- |
|  | table = [zeros(1,'0'-1) , 0:9 , zeros(1,'A'-'9'-1) , ('A':'Z')-'A' + 10];  image1 = table(Image'); |  |

The snapshot of displaying Image 2 is shown in figure.

### Image Monochrome

Matlab supports converting color image to image monochrome using function “rgb2gray()”. It converts RGB values to grayscale values by forming a weighted sum of the R, G, and B components:

Since floating-point operation is with low efficiency, the conversion of RGB image should involve floating-point operations as little as possible. Therefore, we can transform equation into integer division:

Then we can derive grayscale pictures of image1 and image2 as shown in fig

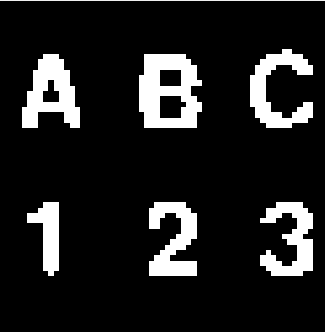
## Binarization

### Otsu’s threshold

Maximum interspecific variance method (as known as Otsu’s method, or Optimal Thresholding in this course) is an adaptive threshold method proposed by Japanese scholar Otsu in 1979. It divides the image into two parts, background and target, according to the gray level characteristics of the image. This method aims at computing a thresholding point T such that the errors in wrongly classifying is minimized.

原理没写

In matlab we can use the build-in functions “graythresh()” to obtain Otus’s optimal thresholding value, and use “imbinarize()” to get binary images, which is depicted in figure. It can be seen that Otus’s thresholding doesn’t work well on Image 2.



### Preprocessing of Image 2

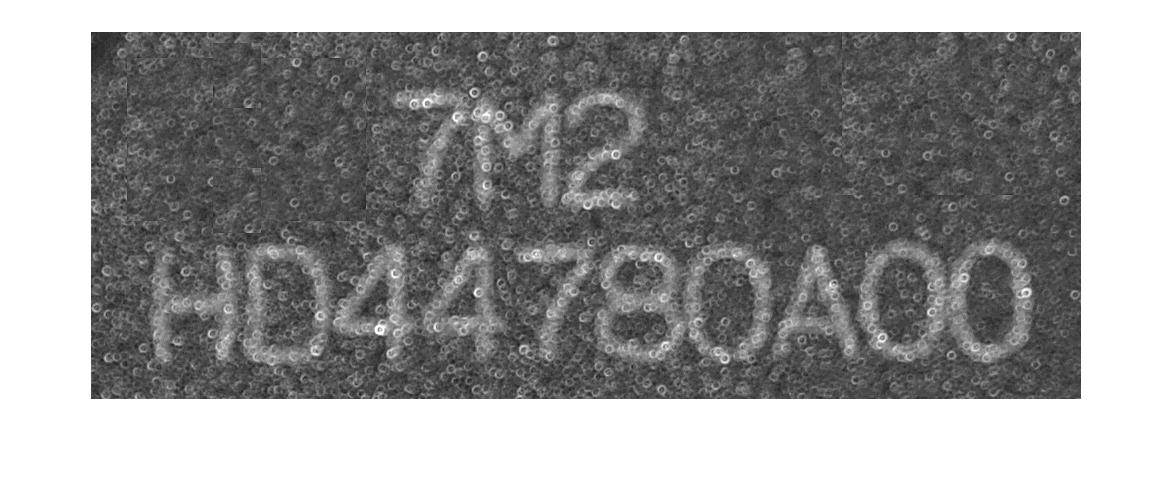
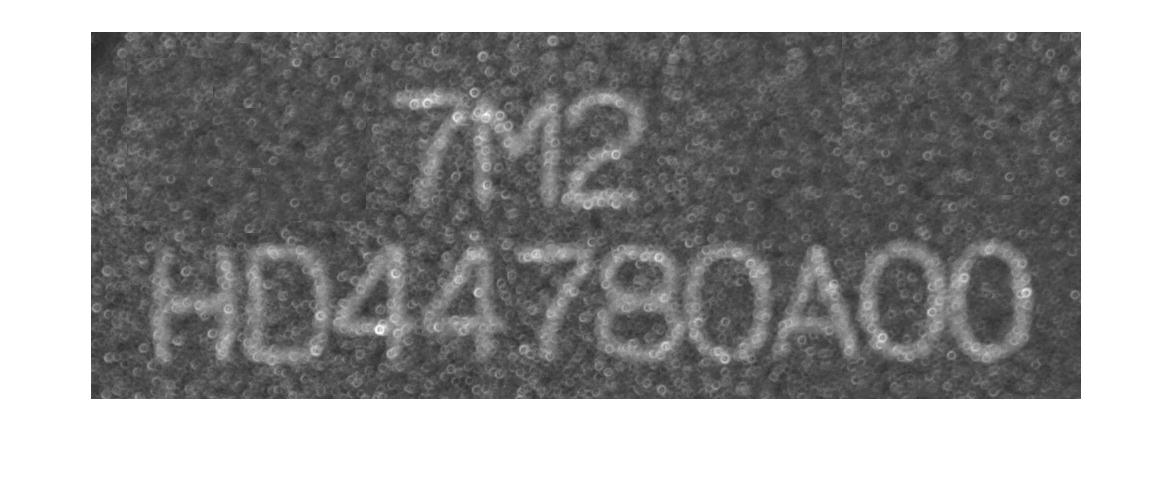


The reason why Otus’s thresholding cannot handle the spotted noise distributed in image 2 is thatnoise spots are with excessive brightness. A naïve way to attenuate brighten spots is to rise the thresholding value T, as a result of which, however, the outline of relevant characters will fade out as T inclines. An example is shown in figure. Therefore, preprocessing approaches are implemented to resolve this problem.

* Sharpening

  In order to improve the high frequency components and enhance the image contour, the original image is preprocessed by unsharp mask (equivalent to using low-pass filter). High-frequency-enhanced area is obtained by calculating the difference between original image and preprocessed image, discounted by a correction factor. And then the overall sharpened image is derived by summing the original image with the high-frequency-enhanced area. In Matlab, we can use imsharpen(image, ‘Radius’, r, ‘Amount’, a) to realize image sharpening, where 'Radius' denotes standard deviation of the Gaussian lowpass filter, and 'Amount' denotes strength of the sharpening effect, respectively.

The effect of image sharpening is shown in figure



* Gaussian Filtering

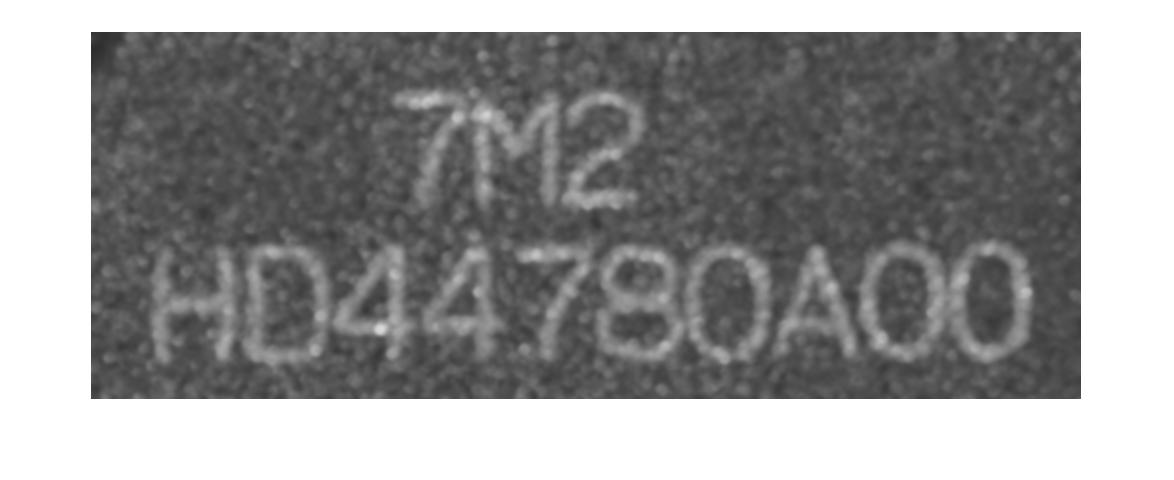
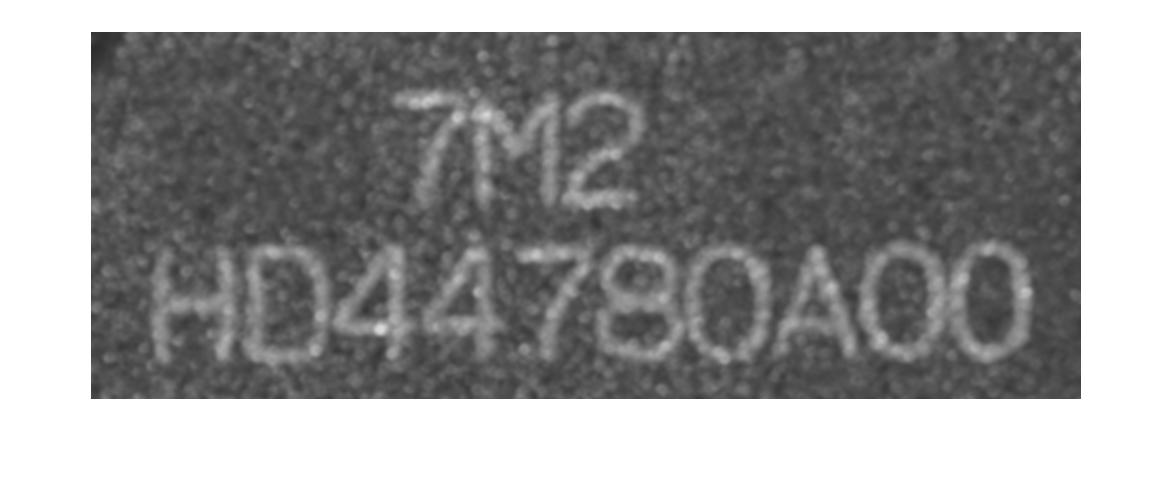
高斯滤波是一种线性平滑滤波，适用于消除高斯噪声，广泛应用于图像处理的减噪过程。通俗的讲，高斯滤波就是对整幅图像进行加权平均的过程，每一个像素点的值，都由其本身和邻域内的其他像素值经过加权平均后得到。

高斯滤波的具体操作是：用一个模板(或称卷积、掩模)扫描图像中的每一个像素，用模板确定的邻域内像素的加权平均灰度值去替代模板中心像素点的值。

高斯滤波后图像被平滑的程度取决于标准差。它的输出是领域像素的加权平均，同时离中心越近的像素权重越高。因此，相对于均值滤波（mean filter）它的平滑效果更柔和，而且边缘保留的也更好。

* Bilateral filtering with Gaussian kernels

网址<https://blog.csdn.net/Jfuck/article/details/8932978>

Gausianbalitary

Binarized image after preprocessing

* Open Operation (Image Geometry)

|  |  |  |
| --- | --- | --- |
|  | se = strel('disk', 3);  Io = imopen(image2, se);  % Ioc = imclose(Io, se);  imshow(Io);  imshow(imbinarize(Io,0.42)); |  |



* Connected components elimination

|  |  |  |
| --- | --- | --- |
|  | CC=bwconncomp(binary\_image);  numPixels = cellfun(@numel,CC.PixelIdxList);%the pixels in each region  [big\_arrange,idx]=sort(numPixels);  meanPixels= mean(big\_arrange); |  |

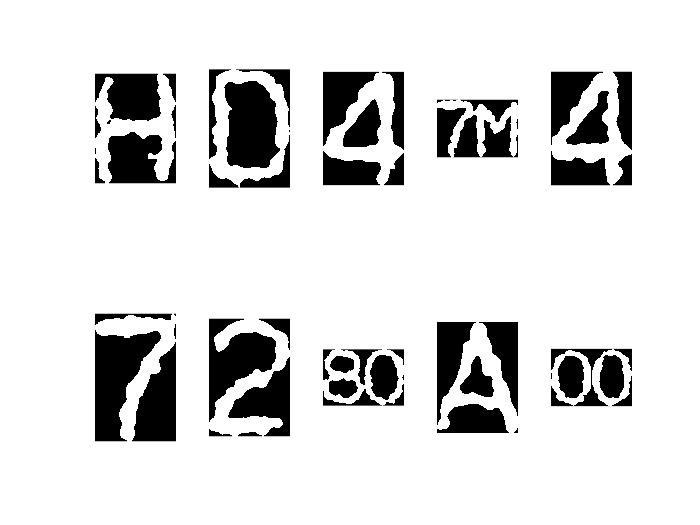


### Summary of binarization

## Pattern Fragmentation

|  |
| --- |
| region=regionprops(binary\_image,'Image','BoundingBox'); |

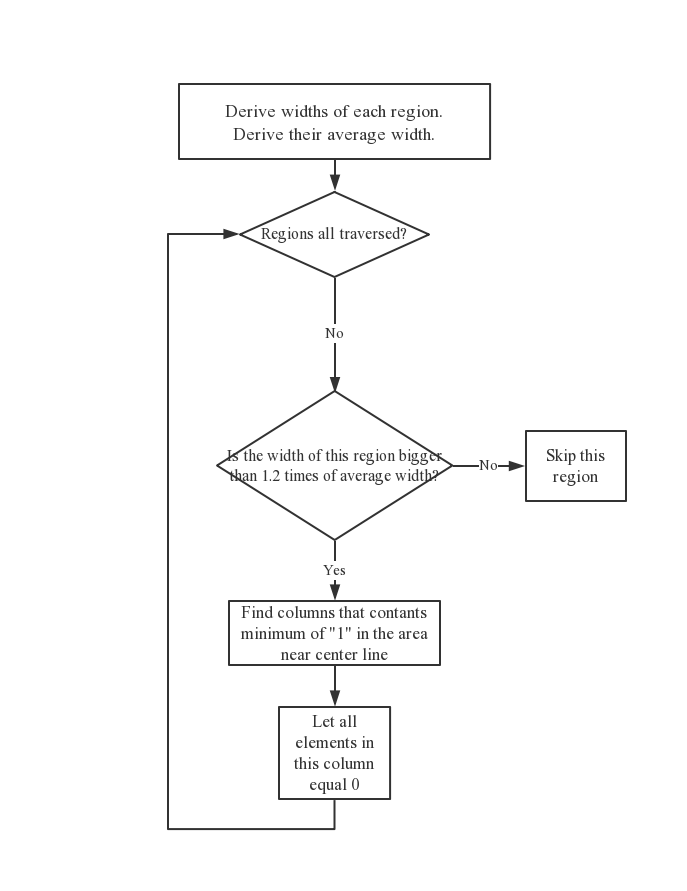
[stats](https://www.mathworks.com/help/releases/R2019a/images/ref/regionprops.html?searchHighlight=regionprops&s_tid=doc_srchtitle#buoixjn-1-stats) = regionprops(binary\_image,'Image','BoundingBox')  returns a struct array containing a struct for each 8-connected component (object) in the binary image. Therefore, we can obtain characters fragments as shown in figure.



By inspection some characters are connected with each other. To separate them, we develop an algorithm to detect and isolate characters.

|  |  |  |
| --- | --- | --- |
|  | [width,length]=arrayfun(@(x) size(x.Image),region);  length\_mean=mean(length);  width\_mean=mean(width);  Rank=zeros(11,1);  for i=1:size(region,1)  if(size(region(i).Image,2)>1.2\*width\_mean)  for k=round(length(i)/2)-5:round(length(i)/2)+5  Rank(k-round(length(i)/2)+6)=size(find(region(i).Image(:,k)==1),1);  end  [a,b]=min(Rank);  b=b+round(length(i)/2)-6;  region(i).Image(:,b)=0;  end  end  segment\_image=zeros(size(binary\_image,1),size(binary\_image,2));  for i=1:size(region,1)  for n=round(region(i).BoundingBox(1)):round(region(i).BoundingBox(1))+region(i).BoundingBox(3)-1  for m=round(region(i).BoundingBox(2)):round(region(i).BoundingBox(2))+region(i).BoundingBox(4)-1  segment\_image(m,n)=region(i).Image(m-round(region(i).BoundingBox(2))+1,n-round(region(i).BoundingBox(1))+1);  %match the centroid of the original image and the rotated image  end  end  end  segment\_image=logical(segment\_image);  subplot(3,4,7);  imshow(segment\_image); |  |

Flow chart of above algorithm can be seen in figure.



## Character rotation

<https://blog.csdn.net/lkj345/article/details/50555870>

## Outline recognition

<https://blog.csdn.net/wqvbjhc/article/details/6065484>

<https://blog.csdn.net/GoodShot/article/details/10051309>

## Character Thinning

<https://blog.csdn.net/superdont/article/details/4621820>

## Character Rescheduling

# Summary

## Overall flowchart and screen dumps

## Comparison of algorithm implemented in the two images

conclusion