Question (a)

(i) Cleaning and Smoothing
In my implementation, I used the 3 x 3 filter marks to clean and smooth the pattern, referring to the technique described in the tutorial and the book.[1]

The idea is to use following 3x3 masks. The '=' represent the considered adjacent pixels which can influent the center pixel to change the color, while '.' represent those ignored. If all considered adjacent pixels have the same color, the center pixel needs to be changed.

```
= = =     = = .     . . .     . = =
= T =     = T .     = T =     . T =
. . .     = = .     = = =     . = =
```

In one round smoothing/cleaning, for each pixel, all masks are tried to applied. If any pixel is changed, another round is proceeded until all pixels remain the same.

The border shall be also handled properly, so as to clean and smooth the pixels which are at the border. In order to do this, when a mask is applied to border pixel, the adjacent pixels out-of-range are treated as like the background color (white).

The sample effect of border case is as below:

```
---------     ---------     ---------     ---------
   1      =>                   11     =>     11
  111            111           111           111
```

Discussion: the 3x3 marsk is efficient if the original image has proper size, that the width of the lines in the character is just a few pixels, because the mask can filter single pixel only. However, if the original image resolution is too high, the algorithm does not work well.
Probably solutions: (1) resize the original image first. If the original image is already known to be a character, we can resize it to something just a little higher than the standard plain. (2) use larger masks, which is proportional to the image size.

(ii) & (iii) normalization and centering

Moment based normalization is applied, where both the gravity center and normalization can be done.[2]

Basic steps:
(1) Find the gravity center (xc, yc) of the original (smoothed) image, by using geometric moment:
xc = m10 / m00
yc = m01 / m00
(2) Find the adjusted width and height (W1, H1) of the image, using central moment u. The idea behind this is to use standard error of the pixel coordinates to estimate the character size.

1 Character Recognition System: A Guide for Students and Practioners, chapter 2.4.1
2 Ibid, chapter 2.4.4

W1 = 4 * sqrt (u20 / m00)
H1 = 4 * sqrt (u02 / m00)

(3) Decide target width and height (W2, H2), which ratio is 1 (W2=H2), fitting to L x L standard plain. The center of target image is (xpc, ypc). Consdering that the range of x,y is from (0,0) to (W2-1,H2-1), so the center is calculated as:

xpc = (W2-1) / 2
ypc = (H2-1) / 2

(4) Do linear backward transformation. The reason of backward transformation is that it will fill the whole standard target plain, without the need of interpolation.

For each pixel in target plain (xp, yp), calculate the corresponding coordinate in the original image.

Normalizing ratio:

alpha = W2 / W1
beta = H2 / H1

Original coordinate:

x = round ((xp - xpc) / alpha + xc)
y = round ((yp – ypc) / beta + yc)

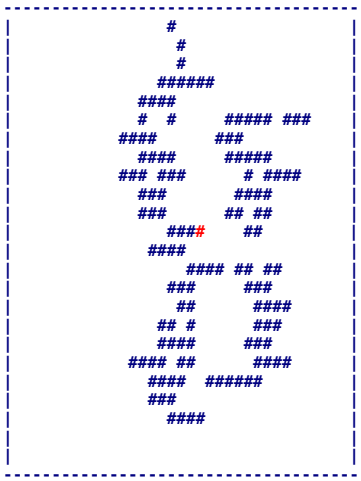
In the processing of the dataset given, I set the normalized standard plain to **16 x 16**, and the ratio of the normalized character **R2 = 1.** These parameters can be easily altered as they are function parameters.

Results:

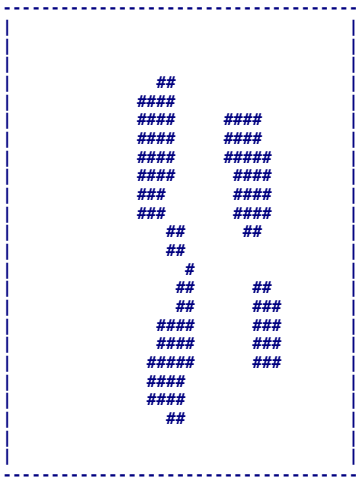
Note: the original center of the character is marked in RED.

Pattern 1:

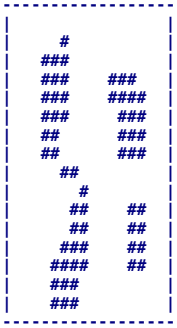
Original image



Cleaned and Smoothed

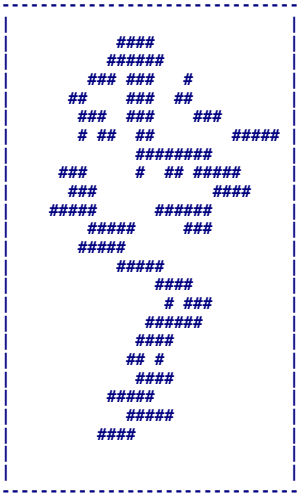


Normalized

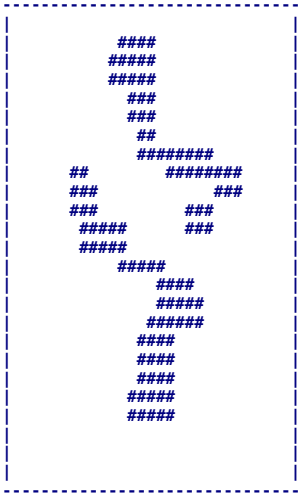


Pattern 2:

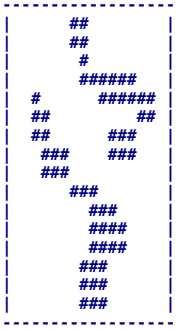
Original image



Cleaned and Smoothed



Normalized



Question (b)

The basic contour tracing algorithm is referred as the "square tracing" algorithm, which can be also found in the text book.[3] That algorithm is major focused on tracing one contour with a given start point. In order to trace the characters provided by an image, some other techniques need to be used.

I personally think about and practiced the below approach and implemented it. It can trace all the contour, and also the sub contour within the border of the already-found one. This feature is important because most characters are hidden in other contour.

The idea of the algorithm is to scan the image pixels from bottom to top, then from left to right and find out the start point of a contour. If a contour is successfully traced, recursively go into that, and use the same scan order to find out sub-contour, but restricted to the area of the outer contour.

3

Pattern Recognition, 2016 Full, exercise #2 **40012502 Guo Peng Liang**

Description of the algorithm:

The class **contour** contains a list of coordinates as the border of the contour

The class **Recursivecontour** contains:
color – the color of the contour
contour – the contour itself
subcontour - a collection of sub-contour (subcontour)

Function traceAllcontour (image, border) % where border is an object of contour
Returns Recursivecontour

```
Collection contours = []
For y = img.height – 1 to 0
    start = the most left point of the border at row y
    if start != null
        x = start.x + 1
        end = the most right point of the border at row y
        while x < end.x
            if (x,y) exists as border of any contours
                % skipped the inside area of found-contours of the same level
                inCantours = the existing contour which (x,y) belongs to
                x = the most right point of inCantours at row y
            else if ( pixel(x,y) != border.color )
                % we found a start point of sub-contour
                newcontour = tracecontour (image, x, y)
                recurcontour = new Recursivecontour
                                    .color = pixel(x,y)
                                    .contour = newControus
                recurcontour.subcontour = traceAllcontour (image, newcontour).subcontour
                contours.add recurcontour
            endif
            x = x + 1
```

It is found that the performance is poor in this kind of recursive tracing. So a max level of recursive tracing is also implemented.
Once all the contours are traced, feature extracting and classifying can be followed on each of the found contour, and based on that the real characters can be filtered.

Due to time limit, the filtering of character contour is not implemented. Following preliminary idea could lead to that achievement.
The real characters could be filtered based on the size and location of the contour. For characters, they shall have similar sizes. Furthermore, they shall be either in a line (ex. for stamps), or in the same circule (ex. for coins). So, if we get the histogram information of the contous, including location and size, and find out the most frequent similarity, then based on that, the characters can be filtered out.
Results of the contour tracing:

Original stamp image:

contour image (contour pixels marked with RED) – **Max Recursive Levels = 3**

Pattern Recognition, 2016 Full, exercise #2 **40012502 Guo Peng Liang**

The original coin image:

Image with contour highlighted **(Max Recursive Level = 2)**

Discussion: the effect of contour tracing for coin is poor. This is because of the poor quality image of coins, based on the fact that the color of characters are too closed to the background. To improvement the quality, I pre-processed the image with Edge Enhancement using image edit tools of GIMP image editor, and also applied the smoothing function done for Question (a). However even after that, the border of the characters are still not able to be differeciated, and there also left over a lot of noise.