Xi'an Jiaotong-Liverpool University
西交利物浦大学

EEE330 - Image Processing
Assessment 5

Student Name: Xiaoyang Wang
Student ID: 1405888

May 30, 2018

Contents

1 Counting the Coins **1**
1.1 Erosion and Dilation 1
1.2 Coins Detection 2
1.2.1 Image Binarization 2
1.2.2 Coin Outline Refinement 3
1.2.3 Erosion and Counting 3
1.3 Results of Each Step 4

2 Car Plate Recognition **5**
2.1 Template Segmentation 5
2.2 Recognition with Erosion Only 7
2.3 Recognition with Hit-and-miss 10

Appendices **13**

A Function: alpha_process **13**

B Function: detect_car_license_plate_v1 **13**

C Function: hit_miss_generation **15**

D Function: detect_car_license_plate_v2 **15**

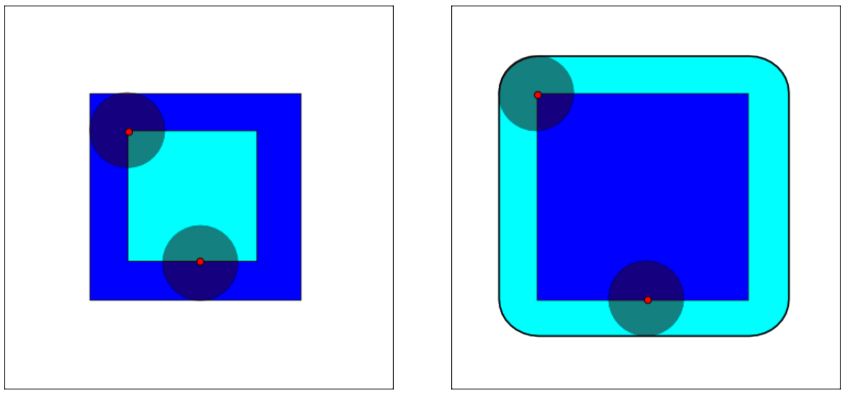
1 Counting the Coins

1.1 Erosion and Dilation

In the first part, the aim is to design an morphological-based algorithm to count the number of coins shown in Figure.2(a). Two important operations, erosion and dilation, are the core for this task. Therefore, it is necessary to investigate the relevant concepts. In binary morphology, an image to be processed contains foreground (represented by 1s) and background (0s). Then, the erosion and dilation rely on an structuring element (also known as morphological operator) of certain shape to do block-to-pixel mapping on the foreground of original image. Each structuring element (SE) contains an origin as the center of mapping.

Erosion can be expressed as an AND operation between SE and image foreground. Only when the SE is fully covered in the foreground a single mapping will happen on the SE center, which means the edge will be partly removed till the SE fully moves into the foreground. As a result, an eroded image is produced and this is the reason to name the operation as *erosion*. An example can be seen in Figure.1(a) that the original binary square image (shown as dark blue area) is eroded by a SE of disk. The SE radius is the erosion boundary and red point is the location for mapping pixel. The light blue square is smaller than the original image as a result of erosion.

Dilation can be regarded as an OR operation between SE and image foreground. It means a block-to-pixel mapping will be performed once the SE touches the foreground with even one pixel overlapping. Then, the original edge will be appended by the pixel on SE center till SE is completely separated from original foreground. The result seems to be a dilated version of original image, which can be observed in Figure.1(b) where the dark blue part are expanded to light blue area by dilation.



(a) Erosion by SE (b) Dilation by SE

Figure 1: Result illustration of binary erosion and dilation on the dark blue square, resulting a smaller light blue square from erosion and a larger light blue area with round conner by dilation

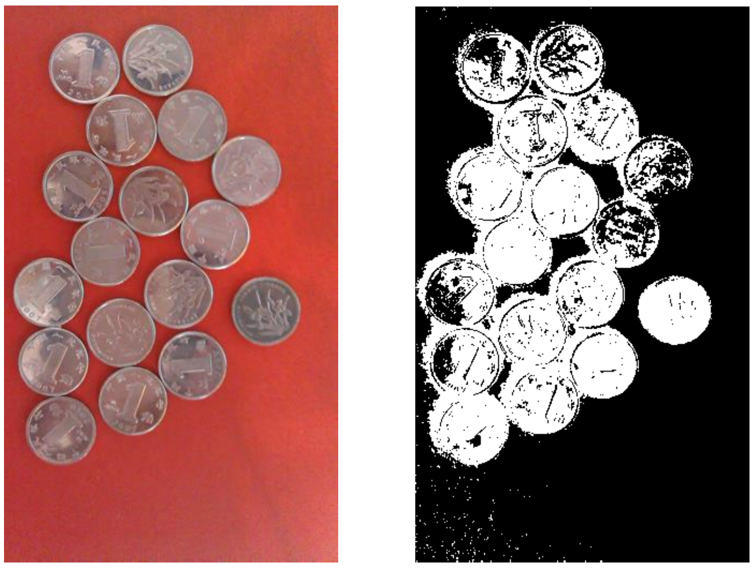
1.2 Coins Detection

The erosion operation can be applied to do object detection. To detect a shape indicated as A in an image B, the general idea is to create a SE with the same shape with A (better shrink a little bit) and use it to do erosion on B. Areas with shape A will be eroded by SE, resulting a foreground pixel or small area indicating the general location of the center of shape A. The coin counting is based on this technique. There are three main steps in this task:

1. Binarize image containing images
2. Preprocess the binary image to get clear coin outlines
3. Apply erosion on coin outlines and shrink them to count

1.2.1 Image Binarization

In the first step, the loaded RGB image is firstly resized to speed up calculation, which has no effect on detection. Then, in the binarization process, it is noticed that there is no appropriate threshold values in RGB channel to separate coins and its background. Different illumination conditions on coins make it hard to do correct segmentation, resulting a diffused coin boundary as in Fig.2(b). However, in the saturation channel of HSV (hue, saturation and value) color model as in Fig.3(a), the problem can be perfectly solved by setting a threshold of 0.5. The binarization result can be seen in Fig.3(b). The whole process is done by script in list 1 with comment.



(a) Original image (b) Binarization on red channel

Figure 2: Original Image and direct binarization

Listing 1: Image Binarization

```
%% Load coin image and binarize it.
im = imread('Coins.jpg');
im_coin = imresize(im,0.2);
im_hsv = rgb2hsv(im_coin);
im_sat = im_hsv(:,:,2);
im_bw = ~imbinarize(im_sat,0.5);
```

1.2.2 Coin Outline Refinement

The next step is to get clear coin outlines of coins. As shown in Fig.3(b), the binarized coins contains background points. This will prevent the coin-shape SE to erode the outline for detection. The refine process is done by list 2 with comment. To remove all the noise points, first apply a erosion by a disk of radius 1 indicated as $se1$ to remove white points in the right bottom conner. Then use a larger SE (se2) to do dilation to clear dark areas inside coins. Both results are shown in Fig.3(c) and (d).

Listing 2: Coin Outlines Creation

```
% Perform erosion and dilation to get clear coin outline
se1 = strel('disk',1);
se2 = strel('disk',2);
im_erode = imerode(im_bw,se1);
im_dilate = imdilate(im_erode,se2);
```

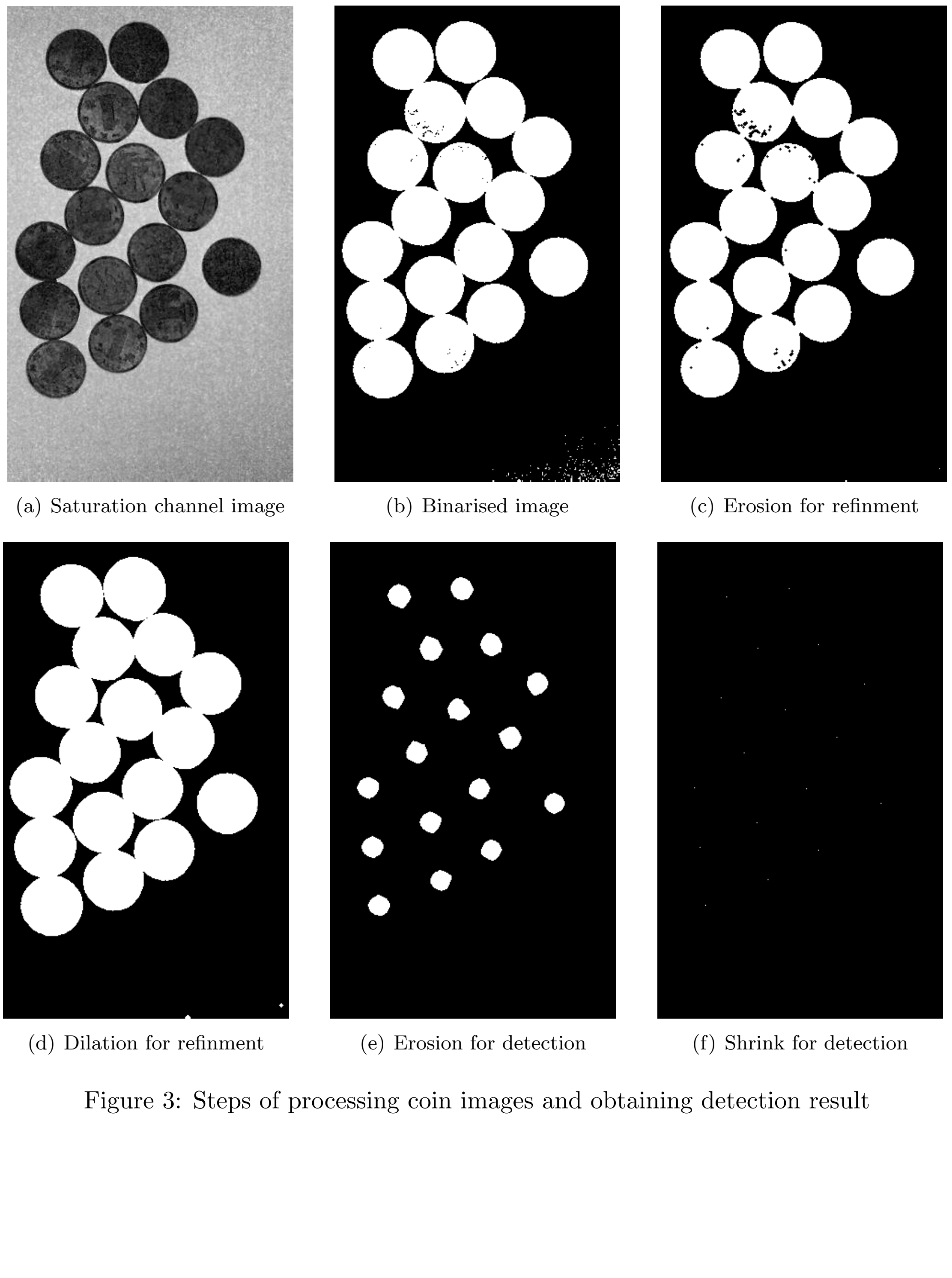
1.2.3 Erosion and Counting

After getting clear coins outlines, it is time to do erosion with coin-shaped SE for final detection. The choice of SE is crucial part in this step and the radius of SE disk is finally determined to be 21 pixels after several trials. This size can guarantee that SE is fully covered and has room to move around for larger plotting. Then we get result shown in Fig.3(e), which clearly point out all central locations of coins. The final step is to shrink all the foreground areas to pixels for counting as shown in Fig.3(f). This process is done by list 3 and the output is 17, which is correct.

Listing 3: Coin Erosion and Counting

```
% Perform erosion and shrink for counting
se_count = strel('disk',21);
im_pre_count = imerode(im_dilate,se_count);
% Shrink to pixles for counting
im_count = bwmorph(im_pre_count,'shrink',Inf);
coin_count = length(find(im_count==1));
fprintf('There are %d coins!', coin_count);
```

1.3 Results of Each Step


(a) Saturation channel image (b) Binarised image (c) Erosion for refinment

(d) Dilation for refinment (e) Erosion for detection (f) Shrink for detection

Figure 3: Steps of processing coin images and obtaining detection result

2 Car Plate Recognition

This section is to develop an algorithm based on erosion and dilation to recognize a car license plate containing a series characters. The general idea is to create a character library from provided alphanumeric chart template and then apply morphological operation for detection. The theory is the same as mentioned in previous part that we use created character template as SE to erode the car plate for detection. The car plate is also cropped to get separate character to be eroded across all template to guarantee final prediction order. In the first stage, only erosion is applied in the algorithm, leading to some unexpected problem. In the second stage, the algorithm uses hit-and-miss method to solve the problem. Details are illustrated in following parts.

2.1 Template Segmentation

To create a character library, the template should be binarized first. This can be done only by an inverse operation as shown in line 3 of listing 4 because the template are all black (0) characters with white (0) background. Then apply Matlab built-in function *regionprops* to get the target locations and then crop them accordingly. The region proposal result is shown in Fig.4 where all targets are marked by green bounding boxes. Some sample characters are listed in Fig.5. Then, a relevant index table is established in line 11 of listing 4. This is for the convenience of getting right index. For example, if the plate character is eroded by SE *alpha*{1} and get a positive mapping area, we can know the character of *alpha*{1} is detected. Then we index back to the table and know that index 1 is ”A”.

Listing 4: Template Binarization and Crop

```
 1  % Template binarization
 2  im_alpha = imread('alphanumeric_templates.png');
 3  im_bw_alpha = (im_alpha~=0);
 4  bbox = struct2cell(regionprops(im_bw_alpha,'BoundingBox'));
 5  alpha = cell(1,36);
 6  % Crop the character according to bounding box
 7  for i = 1:1:36
 8  alpha{i} = imcrop(im_bw_alpha,bbox{i+1});
 9  end
10  % Alpha table generation
11  alpha_table = ['A','0','5','K','U','6',...
12                 '1','B','V','L','W','2',...
13                 '7','M','C','3','8','X',...
14                 'D','N','4','9','Y','O',...
15                 'E','Z','F','P','G','Q',...
16                 'H','R','S','I','J','T'];
```

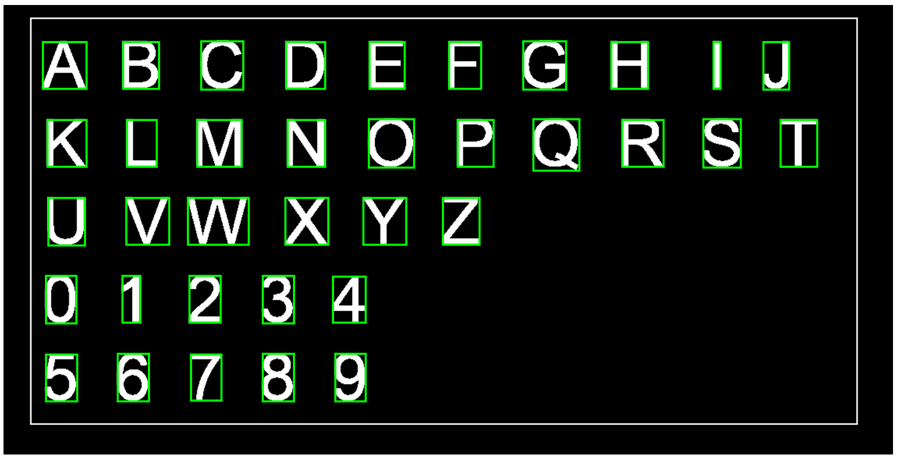


Figure 4: Region proposal result on template

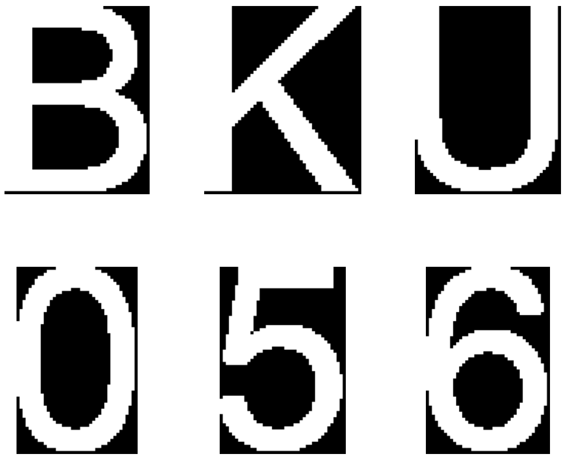


Figure 5: Sample cropped characters

2.2 Recognition with Erosion Only

Then it is time to process car plate for detection. The steps are similar to those in handling template, which are binarization and crop but with an additional padding operation. Listing 5 illustrates the whole process. Line 3 shows the first crop region, which aims to get rid of Chinese character because it is not under consideration. Then in line 5, the cropped plate is binarized by a threshold of 8 (determined by trials). After that, do region proposal to crop each character on car plate (default order is from left to right). Then do zero padding to expand the background of each character to 100×100. This is to normalize each block because it is expected to obtain erosion results of the same size in order to apply the same checking rule. Binarized car plate results with bounding box is shown in Fig.6 and the crop result is in Fig.7.

Listing 5: Car Plate Pre-process

```
 1  %% Car palte binarizatio and crop
 2  im_plate = imread('car_license_plate.png');
 3  rect = [146 1 480 176]; % Avoid Chinese character
 4  im_plate_crop = imcrop(im_plate,rect);
 5  im_bw_plate = (im_plate_crop>=8); % Binarization
 6  bbox = struct2cell(regionprops(im_bw_plate,'BoundingBox'));
 7  region = [100 100]; % expected character size
 8  lic_alpha=cell(1,8);
 9  %% Create character block for detection
10  for i = 1:1:8
11      lic_alpha{i} = imcrop(im_bw_plate,bbox{i});
12      template_size = size(lic_alpha{i});
13      lic_alpha{i} = padarray(lic_alpha{i},...
14                     floor((region-template_size)/2));
15  end
```

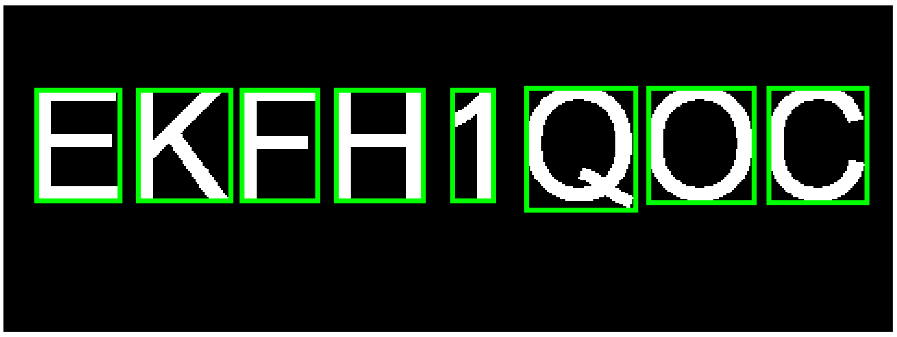


Figure 6: Binarized car plate with bounding boxes

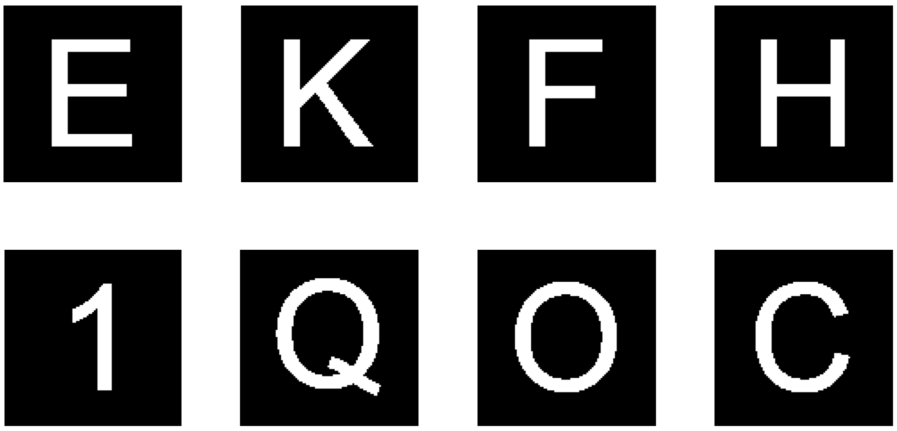


Figure 7: Cropped characters from car plate

The following steps are shown in Listing 6. Before we start detection, it is necessary to erode the templates as shown in line 9 to avoid mismatch in the following steps. Unexpected errors can happen (probably caused by the edge on boundary) and zero padding in line 8 is used to guarantee a correct result. Then start the detection-aimed erosion by comparing all templates for a single plate character. The flag *cmp* in line 12 is set to indicate certain positive result. If any foreground area is detected in the erosion result, the flag will be set to 1. Once the flag is set, the template index will be recorded.

Listing 6: Character Detection

```
 1  % Start comparision
 2  template_num = length(alpha);
 3  lic_num = length(lic_alpha);
 4  se = strel('square',3);
 5
 6  for i=1:1:lic_num
 7      for j = 1:1:template_num
 8          temp1 = padarray(alpha{j},[10,10]);
 9          temp2 = imerode(temp1,se);
10          result = imerode(lic_alpha{i},temp2);
11          % Focus on central region
12          cmp = (sum(sum(result(25:75,25:75)))>1);
13          if cmp
14              alpha_idx(i)= j;
15              break; % Stop at first match
16          end
17      end
18  end
```

For the first run, a *break* is set in the loop which only return the first match for each plate character. The output of the first run is LKFH1OOC. Some correct detection results can be seen in Fig.8. Then, we comment the *break* in line 15 for the second run and get a result of IIII1QOC. This is to overwrite the result by the latest SE. The problem is clear that all SEs with subset shape of a plate character will have positive detection result. For example, E contains the shape of L or I so it can be detected by SE of L and I, which are shown in both output. This type of mismatch is illustrated in Fig.9. Another exception occurs in erosion result of E in Fig.9. A white region appears on the left bottom conner which is unexpected. Therefore, a trick is applied in line 12 of listing 6 that we only focus on the possible result area which is the central region of 50×50 in the result. Then the problem can be solved.

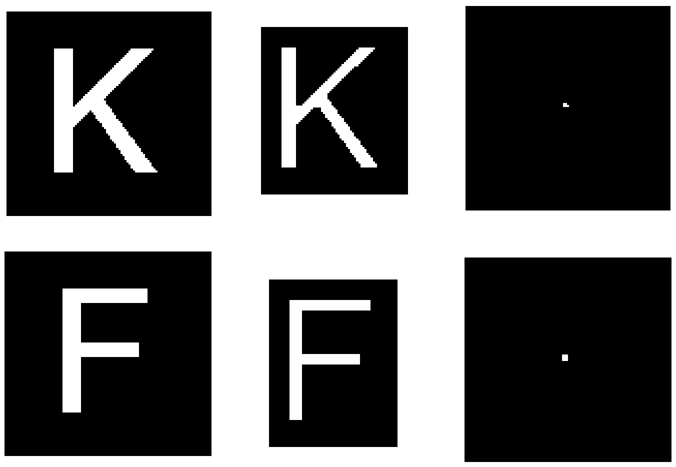


Figure 8: Correct detection result. (Left column: Plate characters. Middle column: SE. Right column: Erosion results)

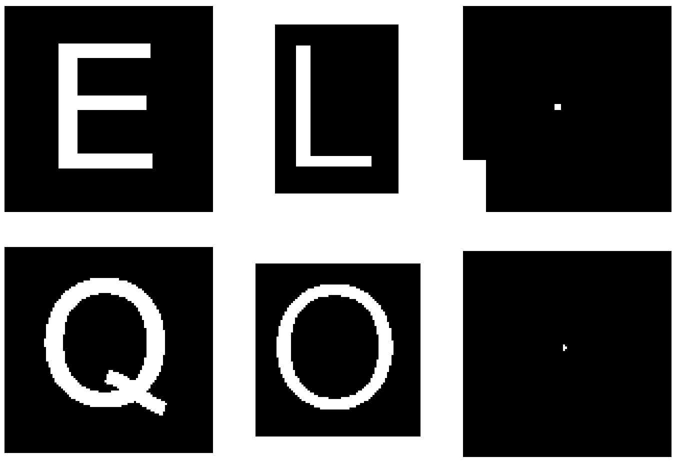


Figure 9: Mismatch results. (Left column: Plate characters. Middle column: SE. Right column: Erosion results)

2.3 Recognition with Hit-and-miss

This section is to debug the erosion-only algorithm. Hit-and-miss method can be used to solve the problem in previous section. The general idea of it can be expressed as equation.1. SE_1 and SE_2 are the two structuring elements to perform hit-and-miss operation and $SE_1 \cap SE_2 = 0$. Then, the result is obtained by matching A with SE_1 and the complement of A with SE_2. The *hit* operation is done by SE_1 to match the foreground information while *miss* is performed by SE_2 to check the background. Only when both hit and miss results are positive the final detection result is positive. This can be regarded as a double check both on foreground and background for certain shape, which can solve the mismatch pattern in last section.

$$HMS = (A \ominus SE_1) \cap (A^c \ominus SE_2) \tag{1}$$

The first step is to create SE for both hit and miss operations. The process is shown by function in listing 7. The *hit* SE is the same as that in previous algorithm, which is done by a erosion on character. The SE for *miss* is done by dilation followed by an inversion. The dilation operation is to expand background shape in *miss* SE to make sure the shape can match. The consideration is similar to doing erosion on hit SE. Then the created SE are stored in variable *alpha_hit* and *alpha_miss*.

Listing 7: Hit and miss SE generation

```
function [alpha_hit,alpha_miss] = hit_miss_generation(alpha)
% Initialization
se = strel('square',3);
alpha_num = length(alpha);
region = [100,100];
alpha_hit = cell(1,alpha_num);
alpha_dilate = cell(1,alpha_num);
alpha_miss = cell(1,alpha_num);

for i = 1:1:alpha_num
    temp = padarray(alpha{i},[10,10]);
    alpha_hit{i} = imerode(temp,se);
    alpha_dilate{i} = imdilate(alpha{i},se);
    alpha_size = size(alpha_dilate{i});
    alpha_miss{i} = padarray(alpha_dilate{i},floor((region-
       alpha_size)/2));
    alpha_miss{i} = ~alpha_miss{i};
end
end
```

The detection stage is illustrated in listing 8. In line 7 and 8, erosion results are obtained both for hit operation and miss operation. Then, in line 9, we get the intersection of those two results. The flag is set according to this intersection. This time the algorithm is able to handle subset shape for correct detection. In Fig.11, complement shape of Q can be only detected by relevant SE and the subset shape O cannot activate the detection. This fixes the wrong result in previous part where Q are recognized as O. The final detection result is EKFH1QOC where no mismatch happens.

Listing 8: Character Detection

```
 1  %% Detection stage
 2  template_num = length(alpha_hit);
 3  lic_num = length(lic_alpha);
 4  for i=1:1:lic_num
 5      for j = 1:1:template_num
 6          % Perform hit and miss
 7          hit_result = imerode(lic_alpha{i},alpha_hit{j});
 8          miss_result = imerode(lic_bg{i},alpha_miss{j});
 9          final_result = hit_result.*miss_result;
10          % Focus on central region
11          cmp = (sum(sum(final_result(25:75,25:75)))>1);
12          if cmp
13              alpha_idx(i)= j;
14              break;
15          end
16      end
17  end
```

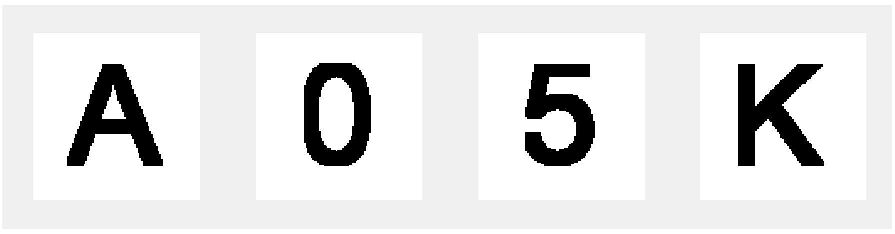


Figure 10: Sample SE for *miss* operation

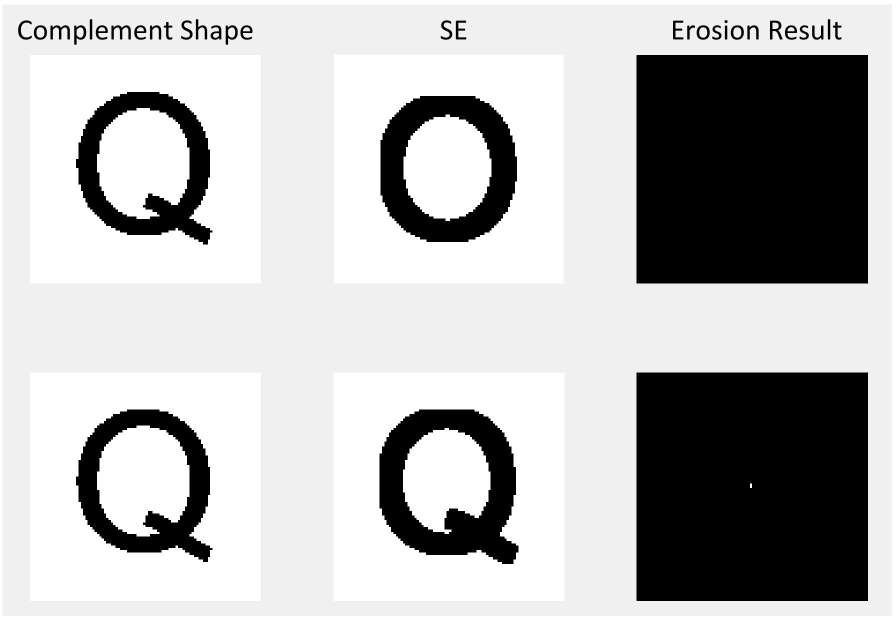


Figure 11: Mismatch is fixed by *miss* operation

Appendices

A Function: alpha_process

```
function [alpha,alpha_table] = alpha_process(im_alpha)
% This function is to crop alphabet template and create
% a character library to store all characters
% Args:
%   im_alpha: input template image
% Returns:
%   alpha: cells containing separate characters
%   alpha_table: an array containing index in alpha
%---------------------------------------------------

% Alphbet segmentation
im_bw_alpha = (im_alpha~=0);
bbox = struct2cell(regionprops(im_bw_alpha,'BoundingBox'));
alpha = cell(1,36);
for i = 1:1:36
    alpha{i} = imcrop(im_bw_alpha,bbox{i+1});
end
% Alpha table generation
alpha_table = ['A','O','5','K','U','6','1','B','V',...
               'L','W','2','7','M','C','3','8','X',...
               'D','N','4','9','Y','0','E','Z','F',...
               'P','G','Q','H','R','S','I','J','T'];
end
```

B Function: detect_car_license_plate_v1

```
function [lic_preds] = detect_car_license_plate_v1(im_plate,
   im_alpha)
% This function is do car plate recognition by erosion
% Args:
%    im_plate: car plate image
%    im_alpha: alphanumeric template image
% Returns:
%    lic_preds: string containing recognize result
%--------------------------------------------------------
[alpha,alpha_table] = alpha_process(im_alpha);
rect = [146 1 480 176];
```

```
im_plate_crop = imcrop(im_plate,rect);
im_bw_plate = (im_plate_crop>=8);
bbox = struct2cell(regionprops(im_bw_plate,'BoundingBox'));

region = [100 100];
lic_alpha=cell(1,8);
alpha_idx = zeros([1,8]);

% Create licence block for predition
for i = 1:1:8
    lic_alpha{i} = imcrop(im_bw_plate,bbox{i});
    template_size = size(lic_alpha{i});
    lic_alpha{i} = padarray(lic_alpha{i},...
                   floor((region-template_size)/2));
end

% Start comparision
template_num = length(alpha);
lic_num = length(lic_alpha);
se = strel('square',3);

for i=1:1:lic_num
    for j = 1:1:template_num
        temp1 = padarray(alpha{j},[10,10]);
        temp2 = imerode(temp1,se);
        result = imerode(lic_alpha{i},temp2);
        % Focus on central region
        cmp = (sum(sum(result(25:75,25:75)))>1);
        if cmp
            alpha_idx(i)= j;
            %break;
        end
    end
end

% Map prediction from alphabet table
lic_preds = zeros([1,8]);
for i = 1:1:8
    lic_preds(i) = alpha_table(alpha_idx(i));
end
    lic_preds = char(lic_preds);
end
```

C Function: hit_miss_generation

```matlab
function [alpha_hit,alpha_miss] = hit_miss_generation(alpha)
% This function is to generate SE for hit and miss operation
% Args:
%     alpha: the character library.
% Returns:
%     alpha_hit: SE for foreground detection
%     alpha_miss: SE for background detection
%------------------------------------------------------------
% Initialization
se = strel('square',3);
alpha_num = length(alpha);
region = [100,100];
alpha_hit = cell(1,alpha_num);
alpha_dilate = cell(1,alpha_num);
alpha_miss = cell(1,alpha_num);

for i = 1:1:alpha_num
    temp = padarray(alpha{i},[10,10]);
    alpha_hit{i} = imerode(temp,se);
    alpha_dilate{i} = imdilate(alpha{i},se);
    alpha_size = size(alpha_dilate{i});
    alpha_miss{i} = padarray(alpha_dilate{i},...
                    floor((region-alpha_size)/2));
    alpha_miss{i} = ~alpha_miss{i};
end
end
```

D Function: detect_car_license_plate_v2

```matlab
function [lic_preds] = detect_car_license_plate_v2(im_plate,
   im_alpha)
% This function is to do plate recognition with hit and miss
% Args:
%     im_plate: car plate image
%     im_alpha: alphanumeric template image
% Returns:
%     lic_preds: string containing recognize result
%---------------------------------------------------------

```

```matlab
[alpha,alpha_table] = alpha_process(im_alpha);
[alpha_hit,alpha_miss] = hit_miss_generation(alpha);
rect = [146 1 480 176];
im_plate_crop = imcrop(im_plate,rect);
im_bw_plate = (im_plate_crop>=8);
bbox = struct2cell(regionprops(im_bw_plate,'BoundingBox'));

region = [100 100];
lic_alpha=cell(1,8);
lic_bg = cell(1,8);
alpha_idx = zeros([1,8]);

% Create licence character and its background
for i = 1:1:8
    lic_alpha{i} = imcrop(im_bw_plate,bbox{i});
    char_size = size(lic_alpha{i});
    lic_alpha{i} = padarray(lic_alpha{i},...
                   floor((region-char_size)/2));
    lic_bg{i} = ~lic_alpha{i};
end

% Start comparision
template_num = length(alpha_hit);
lic_num = length(lic_alpha);

for i=1:1:lic_num
    for j = 1:1:template_num
        hit_result = imerode(lic_alpha{i},alpha_hit{j});
        miss_result = imerode(lic_bg{i},alpha_miss{j});
        final_result = hit_result.*miss_result;
        % Focus on central region
        cmp = (sum(sum(final_result(25:75,25:75)))>1);
        if cmp
            alpha_idx(i)= j;
            %break;
        end
    end
end

% Map prediction from alphabet table
lic_preds = zeros([1,8]);
for i = 1:1:8
    lic_preds(i) = alpha_table(alpha_idx(i));
```

```
end
lic_preds = char(lic_preds);
end
```