University of Tehran

Computer Assignment 2

March 2024

Amirreza Nadi Chaghadari | 810101538

Signals and Systems

Contents

Part 1

Task 1

"uigetfile" function is pre-built function that opens a file explorer-like visual user interface and asks for a file to open. It returns 2 values: the file's path and its name (file extension included). the syntax looks like this:

```
[file, path] = uigetfile({filter}, title);
```

Where filter is the semicolon-separated list of acceptable file extensions and title is the title of the window that opens. Chosen file's name will be saved in "file" variable and its path will be saved in "path" variable. This project is all about processing the picture of a car plate, meaning that only image formats are acceptable. Therefore, I used this function as follows:

```
[file,path] = uigetfile({'*.jpg;*.bmp;*.png;*.tif'},'Select An Image File Containing A Car Plate');
```

Task 2

"imresize" is another pre-built function of MATLAB that takes the data of a matrix and resizes it by given height and width. Here's the syntax:

```
imresize(image_data_matrix, [height, width]);
```

This task requires the image to become a 300×500 image. Therefore' we pass 300 and 500 as height and width arguments to the function. Here's how I used the function:

```
image = imresize(image, [300 500]);
```

Task 3

When an image Is loaded in MATLAB using “imread” function, a 3-demential matrix is created. Since its second dimension has a length of 3, we could interpret the matrix like a 3-layar matrix. Each entry of this matrix contains the data of one single pixel from the image in RGB[1] system -three 8-bit unsigned integer numbers describing how much the pixel is red, green, and blue-. In order to make things easier, we apply a “Grayscale” filter on the image, creating a 2-dimentional matrix. Grayscale filter turns a colored image into a gray one. The amount of darkness or light that every pixel has is determined by its RGB values and is stored in only one 8-bit unsigned integer number -which in terms of decimal coding is number between 0 to $2^8 - 1 = 256 - 1 = 255$-. In this new image, 0 means total darkness (black) and 255 means total brightness (white). Anything else comes between these two numbers. Look at the example:

Figure 1. Original

Figure 2. grayscale

[1] Red – Green - Blue

There are several ways to apply grayscale filter on an image, but they mostly use one simple algorithm:

$$Gray = C_R \times Red + C_G \times Green + C_B \times Blue$$

Where C_R, C_G, and C_B are positive fractional numbers that sum up to 1. Different algorithms choose different values for the coefficients. Some algorithms favor red[2], some favor green, and some favor blue. There is also an algorithm that favors none, meaning that the coefficients are all equal to $\frac{1}{3}$. In this task, we are going to use these coefficients:

$$\begin{cases} C_R = 0.299 \\ C_G = 0.578 \\ C_B = 0.114 \end{cases}$$

Here's how I implemented this:

```
1  function grayscaled = mygrayfun(image)
2      r_c = 0.299;
3      g_c = 0.578;
4      b_c = 0.114;
5      grayscaled = r_c * image(:,:,1) + g_c * image(:,:,2) + b_c * image(:,:,3);
6  end
```

The overall flow is just as explained. Lines 2 to 4 generate coefficients, and line 5 multiplies each layer of the given image by its coefficient and sums up the results.

[2] In these algorithms, C_R is larger than the rest of the coefficients

Task 4

The final result of Task 3 is a matrix with unsigned 8-bit integer entries, describing the "brightness' or "darkness" of each pixel in a scale of 0 to 255. The brighter the pixel, the larger the entry. Therefore, we can turn our image into a black-and-white image by comparing its entries to a specific value, which may be referred to as "threshold". As we know, plate characters are written in black, which is the darkest color and therefore, smallest number in grayscale format. So, I decided to assign 1 to pixels with values less than the threshold. Here's how its done:

```
1  function binary_im = mybinaryfun(image)
2      binary_im = image <= graythresh(image)*256;
3  end
```

In this function, entries of the "image" matrix are compared to a threshold that is generated using a MATLAB pre-built function called "graythresh". This function takes an image and returns a value between 0 to 1, describing what the proper threshold is. By multiplying this functions output to 255 -the highest pixel value in grayscale format-, the proper threshold is generated. Entries containing values smaller than or equal to that threshold will become **Logical 0** and entries with values larger than it will become **Logical 1**. So now we have a **Logical Matrix**, which is a matrix with entries containing only 0 or 1. MATLAB's "imshow" function will interpret these matrixes as black-and-white images[3]. The results of Task 1 to Task 4 are shown in the following page.

[3] Obviously, 0 means black and 1 means white

Results up until Task 4

Figure 3. Original

Figure 4. Loaded And Resized

Figure 5. Grayscaled

Figure 6. Binarized

Codes up until Task 4

```
% Loading The Image
clear;clc;close;
[file,path] = uigetfile({'*.jpg;*.bmp;*.png;*.tif'},'Select An Image File Containing A Car Plate');
if file == 0
    return;
end

image = imread([path file]);


image = imresize(image, [300 500]);
subplot(2,2,1)
imshow(image)
title('Original')
% Generating Gray Scale And Black-And-White
subplot(2,2,2)
image = mygrayfun(image);
imshow(image)
title('Gray Scaled')

subplot(2,2,3)
image = mybinaryfun(image);
imshow(image)
title('Binary')

subplot(2,2,4)
image = myremovecom(image,300);
im2 = myremovecom(image,2500);
image = image - im2;
imshow(image)
title('Ready For Segmentation And Correlation')
```

Task 5

There are several algorithms to do this part. The one I chose has these steps:

1. Clone the image matrix
2. Find adjacent ones in the image and group them together
3. Assign a number to each group
4. In the clone matrix, replace ones with their group's assigned number
5. Find out how many times each group number is repeated
6. Remove the group numbers that are less than a given threshold by making them 0 in the image matrix and its clone

The first 4 steps group adjacent ones and the rest of them remove small groups. So, I decided to put the first 4 steps in a different function called "labelImageArea" and used a specific algorithm to implement that function. The rest was just a simple matrix processing. The algorithm I used for "labelImageArea" function is explained in the following page. After that, I implemented a function that takes the "labeled" matrix and an integer number and determines how many times that integer is repeated in the given matrix. The function is called "mycounterarea", In the end, I coded a function that removes a labeled area from the image matrix. this function is named "myremovearea". The advantage of defining all these functions is easier implementation of the desired functions.

labelImageArea Algorithm

Starting from the **First** row and going all the way to the **Last**, I grouped all adjacent ones by assigning the same numbers to them in a clone matrix. The clone matrix is named “labeled”. After that, starting from the **Second Last** row of the “labeled” matrix and going all the way to the **Top**, detected the entries that had the same values in each row, which are grouped together as explained. After that, I detected the entries of the row bellow it which were adjacent to the detected group vertically and diagonally and saved their assigned values in a temporary matrix, along with the value assigned to the detected group. After that, using “unique” function of MATLAB, removed the repeated values of the temporary matrix and using “min” function, extracted its minimum value. In the end, I searched the entire “labeled” matrix for values equal to the temporary matrix’s entries and replaced the values with the minimum I found. The implementation is explained part by part in the following pages along with the picture of the entire code.

Horizontal Grouping

```
3       labeled = zeros(size(image));
4       maxIndex = 1;
5       for i = 1:size(image,1)
6           flag = boolean(0);
7           for j = 1:size(image,2)
8               if image(i,j) == 1
9                   labeled(i,j) = maxIndex;
10                  flag = boolean(1);
11              else
12                  if flag
13                      flag = boolean(0);
14                      maxIndex = maxIndex + 1;
15                  end
16              end
17          end
18      end
```

Line 3 generates the clone matrix “labeled”. We need a variable for storing values to assign to groups. I named it “maxIndex”. After that, the matrix is swept row by row in a “for” loop with i as its iterating variable. For each row, I swept every entry. If its value is 1, the “labeled” matrix gets “maxIndex” at the same position which is being swept. If the value is 0 and the previously received value was 1 -which will be determined by a Boolean variable called “flag”-, “maxIndex” gets incremented. Otherwise, nothing happens. Once the function sweeps the last row, all adjacent ones of each row are grouped together, meaning that their corresponding entries in “labeled” matrix has the same value.

Merging Horizontal Groups

```
20        for i = size(image,1)-1:-1:1
21           j = 1;
22           while j <= size(image,2)
23               if image(i,j) == 1
24                   [r1,c1] = find(labeled(i,:) == labeled(i,j));
25                   r1(end+1) = i;
26                   r1(end+1) = i;
27                   c1(end+1) = max(min(c1,[],'all')-1,1);
28                   c1(end+1) = min(max(c1,[],'all')+1,size(image,2));
29                   values = labeled(i+1,c1);
30                   values(end+1) = labeled(i,j);
31                   values = values(values>0);
32                   values = unique(values);
33                   val = min(values,[],'all');
34                   for p = values
35                       labeled(labeled == p) = val;
36                   end
37                   j = j + length(c1);
38               else
39                   j = j + 1;
40               end
41           end
42       end
43   end
```

This time, sweeping starts from the second last row, “size(image,1)-1”. If the entry of “image” matrix located at the position we’re sweeping contains 1, it means that “labeled” matrix has a group number on the corresponding entry. So, It searches for that entry in the entire matrix and finds the columns of the entries that have the same group number. The positions are saved in 2 matrixes: r1 and c1. Then the diagonal adjacencies are added to the list of positions, which are located in the same row. The values of the detected adjacent entries are saved in a matrix called “values”. Zeros are removed from it and then its minimum value is extracted and saved it “val” variable. In the end, a for loop sweeps “values” and replaces all equal entries in “labeled” matrix with “val”. As for the iteration, there are 2 possible scenarios:

1. The current entry of “image” is 1
2. The current entry of “image” is 0

If the first scenario has occurred, j goes one column forward. Otherwise, it goes forward by the number of ones the detected horizontal group has.

labelImageArea Full Code

```
function [labeled,maxIndex] = labelImageArea(image)

    labeled = zeros(size(image));
    maxIndex = 1;
    for i = 1:size(image,1)
        flag = boolean(0);
        for j = 1:size(image,2)
            if image(i,j) == 1
                labeled(i,j) = maxIndex;
                flag = boolean(1);
            else
                if flag
                    flag = boolean(0);
                    maxIndex = maxIndex + 1;
                end
            end
        end
    end

     for i = size(image,1)-1:-1:1
        j = 1;
        while j <= size(image,2)
            if image(i,j) == 1
                [r1,c1] = find(labeled(i,:) == labeled(i,j));
                r1(end+1) = i;
                r1(end+1) = i;
                c1(end+1) = max(min(c1,[],'all')-1,1);
                c1(end+1) = min(max(c1,[],'all')+1,size(image,2));
                values = labeled(i+1,c1);
                values(end+1) = labeled(i,j);
                values = values(values>0);
                values = unique(values);
                val = min(values,[],'all');
                for p = values
                    labeled(labeled == p) = val;
                end
                j = j + length(c1);
            else
                j = j + 1;
            end
        end
    end
end
```

“mycounterarea” and “myremovearea”

This is the code for “mycounterarea”:

```
function result = mycountarea(labeled_map, n)
    result = sum(labeled_map == n,'all');
end
```

The code’s logic is simple. It compares “labeled_map” to n and sums up all the entries of the result. Since the comparison result contains only 0 and 1, the summation of its entries is equal to the number of ones it has, which is the same the numbers of entries equal to 1 in the matrix.

As for the “myremovearea” function, here’s the body:

```
function result = myremovearea(image, labeled_map, n)
    image(labeled_map == n) = 0;
    result = image;
end
```

This one is also a simple function. It searches for entries equal to n in “labeled_image” matrix and assigns 0 to the corresponding entries in “image” matrix. in the end, “image” is returned.

“myremovecom” function

Now that all necessary tools are ready, the implementation of a function that removes small white areas from an image becomes easy. Here’s the body of the function:

```
1  function result = myremovecom(image, threshold)
2      [labeled,maxIndex] = labelImageArea(image);
3      for i = 1:maxIndex
4          if mycountarea(labeled,i) < threshold
5              image = myremovearea(image,labeled,i);
6          end
7      end
8      result = image;
9  end
```

Line 4 labels the image. “maxIndex” saves the largest group number in the labeled matrix. after that, a for loop iterates all possible group numbers and removes the corresponding area if its smaller than the given threshold.

Task 6

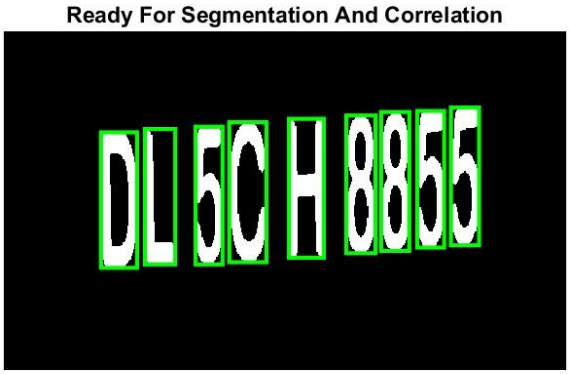
This task can be done easily using the functions implemented for the previous part. All there is to do is use “labelImageArea” function to separate the image into segments and re-label the segments to make them easier to use. Here’s the function:

```
1  function [segmentation,lasSegment] = mysegmentation(image)
2     [labeled, maxIndex] = labelImageArea(image);
3      lasSegment = 1;
4      for i = 1:maxIndex
5          if sum(labeled==i,'all') ~= 0
6              labeled(labeled == i) = lasSegment;
7              lasSegment = lasSegment + 1;
8          end
9      end
10     segmentation = labeled;
11 end
```

Line 2 separates the image as explained. Then, a for loop is used to sweep all possible segments and relabel them. In the end, the results are returned. Now all there is to do is to use this function! Here’s how I used it:

```
33    [labeled,maxSegment] = mysegmentation(image);
34    hold on
35    for i = 1:maxSegment-1
36
37        [r,c] = find(labeled == i);
38        mc = min(c,[],'all');
39        mr = min(r,[],'all');
40        Mc = max(c,[],'all');
41        Mr = max(r,[],'all');
42        rectangle('Position',[mc,mr,Mc-mc,Mr-mr],'EdgeColor','g','LineWidth',2)
43    end
```

Line 33 does the segmentation. The rest of the code draws green boxes around the founded areas. The result looks like this:



Task 7

To do this part, we need to create a dataset first. I decided to do it in a separate separated file so that I could use it for the rest of this project's parts as well. Here's how the map set loader's code looks like:

```
files=dir('English Map Set');
len=length(files)-2;

TRAIN=cell(2,len);

for i=1:len
   TRAIN{1,i}=imread(['English Map Set','\',files(i+2).name]);
   TRAIN{2,i}=files(i+2).name(1);
end

save('TRAININGSET.mat','TRAIN');
```

It reads all the files in "English Map Set" folder and saves their data in a cell array called TRAIN, along with the name of the files, which are purposefully named the same as the character they are. This makes it easier to generate the final result. Then we can use this map set for decision making. We correlate each separated segment with every single member of the map set and compare the results to find their maximum value. If it's larger than 0.45, we save it somewhere as the character representing the current segment. Otherwise, the segment is considered trash. Here's the code for all of it:

```
result = [];
for i = 1:maxSegment-1
    [r,c] = find(labeled == label_positions(1,i));
    character = labeled(min(r):max(r), min(c):max(c));
    subplot(2,colcount,i)
    imshow(character)
    decision = cell(2,1);
    decision{1,1} = 0;
    decision{2,1} = '';
    for j = 1:length(TRAIN)
        temp = corr2(TRAIN{1,j},imresize(character,size(TRAIN{1,j})));
        if temp >= 0.45 && temp >= decision{1,1}
            decision{1,1} = temp;
            decision{2,1} = TRAIN{2,j};
        end
    end
    result = [result decision{2,1}];
end
```

Lines 57 and 58 separate a rectangle around the segment and save it in “character” matrix. Afterwards, a 2-by-1 cell is created, holding the current maximum correlation result and its corresponding character. The inner for loop sweeps the entire map set and correlates every member with the given segment. In the end, the decided character is concatenated with the rest at line 71.

Task 8

The code for this task looks like this:

```
74    disp(result)
75    file = fopen('result.txt', 'wt');
76    fprintf(file,'%s\n',result);
77    fclose(file);
78    winopen('result.txt')
```

This displays the result and saves it in a text file, then opens the file. The results for 3 plates are in the following page.

Test Results


```
Command Window
DL5CH8855
fx >>
```


```
Command Window
UP14CB7145
fx >>
```


```
Command Window
UP14CB7145
fx >>
```

p1 Script Screenshot

```
% Loading The Image
clear;clc;close;
[file,path] = uigetfile({'*.jpg;*.bmp;*.png;*.tif'},['Select An Image' ...
    ' File Containing A Car Plate']);
if file == 0
    return;
end
image = imread([path file]);
image = imresize(image, [300 500]);
subplot(2,2,1)
imshow(image)
title('Original')
% Generating Gray Scale And Black-And-White
subplot(2,2,2)
image = mygrayfun(image);
imshow(image)
title('Gray Scaled')

subplot(2,2,3)
image = mybinaryfun(image);
imshow(image)
title('Binary')

subplot(2,2,4)
image = myremovecom(image,300);
im2 = myremovecom(image,2500);
image = image - im2;
imshow(image)
```

```
imshow(image)
title('Ready For Segmentation And Correlation')
% Segmentation And Correlation
[labeled,maxSegment] = mysegmentation(image);
hold on
label_positions = zeros(2,maxSegment-1);
for i = 1:maxSegment-1
    [r,c] = find(labeled == i);
    mc = min(c,[],'all');
    mr = min(r,[],'all');
    Mc = max(c,[],'all');
    Mr = max(r,[],'all');
    label_positions(1,i) = i;
    label_positions(2,i) = mc;
    rectangle('Position',[mc,mr,Mc-mc,Mr-mr],'EdgeColor','g','LineWidth',2)
end

[temp,indexs] = sort(label_positions(2,:));
label_positions(1,:) = label_positions(1,indexs);
label_positions(2,:) = label_positions(2,indexs);

english_training_loading;
load TRAININGSET.mat
figure
colcount = ceil(maxSegment/2);
result = [];
```

```
for i = 1:maxSegment-1
    [r,c] = find(labeled == label_positions(1,i));
    character = image(min(r):max(r), min(c):max(c));
    subplot(2,colcount,i)
    imshow(character)
    decision = cell(2,1);
    decision{1,1} = 0;
    decision{2,1} = '';
    for j = 1:length(TRAIN)
        temp = corr2(TRAIN{1,j},imresize(character,size(TRAIN{1,j})));
        if temp >= 0.45 && temp >= decision{1,1}
            decision{1,1} = temp;
            decision{2,1} = TRAIN{2,j};
        end
    end
    result = [result decision{2,1}];
end

disp(result)
file = fopen('result.txt', 'wt');
fprintf(file,'%s\n',result);
fclose(file);
winopen('result.txt')
```

Part 2

All this part needs are a new dataset and a few adjustments in threshold values and addresses[4]. The new training loading script looks like this:

```
files=dir('Persian Map Set');
len=length(files)-2;

TRAIN=cell(2,len);

for i=1:len
   TRAIN{1,i}=imread(['Persian Map Set','\',files(i+2).name]);
   TRAIN{2,i}=files(i+2).name(1);
end

save('TRAININGSET.mat','TRAIN');
```

All that changed are the name of the script and the folder from which the map set is loaded. As for the main script -p2.mat-, the only thing that changes is the value of 2 threshold:

1. Noise Margin Threshold
2. Background Removing Threshold

The noise margin threshold changes from 300 to 500, so that it could remove "ایران" from the plate as well. Background removing threshold changes from 2500 to 4000, because some of Persian characters such as "ی" contain too many pixels and are considered background if the threshold is 2500. The final script code and the test result for 2 plates are shown in the following pages.

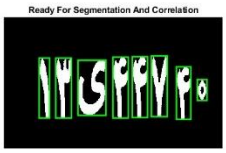
[4] For Persian_training_loading

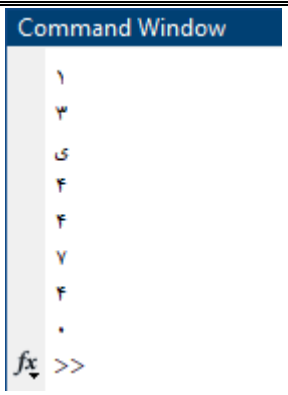
p2 Script Screenshots

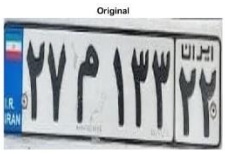
```
% Loading The Image
clear;clc;close;
[file,path] = uigetfile({'*.jpg;*.bmp;*.png;*.tif'},['Select An Image' ...
    ' File Containing A Car Plate']);
if file == 0
    return;
end
image = imread([path file]);
image = imresize(image, [300 500]);
subplot(2,2,1)
imshow(image)
title('Original')
% Generating Gray Scale And Black-And-White
subplot(2,2,2)
image = mygrayfun(image);
imshow(image)
title('Gray Scaled')

subplot(2,2,3)
image = mybinaryfun(image);
imshow(image)
title('Binary')

subplot(2,2,4)
image = myremovecom(image,500);
im2 = myremovecom(image,4000);
image = image - im2;
imshow(image)
title('Ready For Segmentation And Correlation')
% Segmentation And Correlation
[labeled,maxSegment] = mysegmentation(image);
hold on
label_positions = zeros(2,maxSegment-1);
for i = 1:maxSegment-1
    [r,c] = find(labeled == i);
    mc = min(c,[],'all');
    mr = min(r,[],'all');
    Mc = max(c,[],'all');
    Mr = max(r,[],'all');
    label_positions(1,i) = i;
    label_positions(2,i) = mc;
    rectangle('Position',[mc,mr,Mc-mc,Mr-mr],'EdgeColor','g','LineWidth',2)
end

[temp,indexs] = sort(label_positions(2,:));
label_positions(1,:) = label_positions(1,indexs);
label_positions(2,:) = label_positions(2,indexs);
```

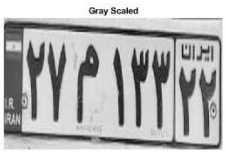
```
48
49      persian_training_loading;
50      load TRAININGSET.mat
51      figure
52      colcount = ceil(maxSegment/2);
53      result = [];
54      for i = 1:maxSegment-1
55          [r,c] = find(labeled == label_positions(1,i));
56          character = image(min(r):max(r), min(c):max(c));
57          subplot(2,colcount,i)
58          imshow(character)
59          decision = cell(2,1);
60          decision{1,1} = 0;
61          decision{2,1} = '';
62          for j = 1:length(TRAIN)
63              temp = corr2(TRAIN{1,j},imresize(character,size(TRAIN{1,j})));
64              if temp >= 0.45 && temp >= decision{1,1}
65                  decision{1,1} = temp;
66                  decision{2,1} = TRAIN{2,j};
67              end
68          end
69          result = [result decision{2,1}];
70      end
71
72      disp(result')
73      file = fopen('result.txt', 'wt');
74      fprintf(file,'%s\n',result);
75      fclose(file);
76      winopen('result.txt')
```

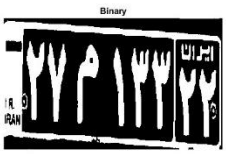
Test Results

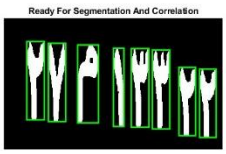


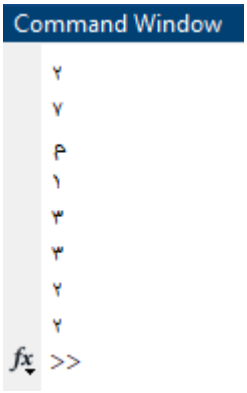












Part 3

This part is a bit tricky, since the image contains a large amount of unneeded data. There are several ways to take out the plate from the car. Considering it's a Persian plate, I chose this algorithm:

1. Find blue and white areas of the image
2. Remove noises of the blue area using previously coded "myremovecom" function
3. Discard blue areas that are not in the center (20% margin is chosen for the image)
4. Find the remaining blue area that are next to a white area
5. Crop the image from the spotted area with proper size

To do the first step, I coded a function that extract areas with RGB values in a given range. Here's how it looks like:

```
1  function result = RGBRange(image, red, green, blue)
2  result = boolean(zeros(size(image(:,:,1))));
3  r = image(:,:,1);
4  g = image(:,:,2);
5  b = image(:,:,3);
6  result(r>=red(1)) = 1;
7  result(r>=red(2)) = 0;
8  result(g<=green(1)) = 0;
9  result(g>=green(2)) = 0;
10 result(b<=blue(1)) = 0;
11 result(b>=blue(2)) = 0;
12 result = boolean(result);
13 end
```

Lines 3 to 5 separate the image's 3 layers, which are meant to store its RGB values. The rest of the code assigns 1 to the entries of the output that correspond to areas within desired color range in "image" matrix.

In the main script[5], I used this function to extract blue and white parts of the image, by specifying the RGB range for blue and white. Both white part and blue part along with the image itself is resized afterwards. The code for this part is in the following page.

[5] p3.mat

```
10    blue_part = RGBRange(image,[0,80],[0,120],[100,255]);
11    white_part = RGBRange(image,[100,255],[100,255],[100,255]);
12
13    blue_part = imresize(blue_part,[300,500]);
14    white_part = imresize(white_part,[300,500]);
15    image = imresize(image,[300,500]);
```

The separated blue part and white part of the image are saved in "blue_part" and "white_part" matrixes. After that, the noises of the blue part are removed, using "myremovecom" function[6] and the results of the processes so far are plotted. Then "blue_part" matrix is separated into segments and a for loop sweeps every segment looking for the one that belongs to the plate. This segment has 2 properties that can be used for detecting it:

1. The area to its right is black and white, so the corresponding entries in "white_part" matrix can't be all 0
2. Its almost located in the center, so it can't be in the first or last 20% of the image, horizontally or vertically.

So, all there is to do is to search for a blue area with theses properties. Here's how I did it:

```
28    for i = 1:maxSegment-1
29        [r,c] = find(blue_labeled==i);
30        mr = min(r);
31        Mr = max(r);
32        Mc = max(c);
33        mc = min(c);
34        deltar = Mr-mr;
35        if mc >= 0.8*500 || mc <= 0.2*500 || mr <= 0.2*300 || mr >= 0.8*300
36            continue
37        end
38        rectangle('Position',[mc,mr,Mc-mc,deltar],'EdgeColor','g','LineWidth',2)
39        if any(white_part(mr:Mr,Mc+2)==1)
40            image = imcrop(image,[mc-20,mr-20,7*deltar+40,deltar+40]);
41            break;
42        end
43    end
```

"blue_labeled" is the segmentation result of "blue_part". For every labeled area within it, the area borders are calculated in lines 29 to 34. Lines 35 to 37 make sure the areas in the marginal 20% of the image are discarded. Line 38 draws rectangles on all central detected blue areas. Lines 39 to 42 check if the currently spotted area has a white area to its right. If so, the image will be cropped from there,

[6] Developed and explained in **Part 1**

by the width and length specified using the size of spotted blue area. Once the image is cropped, all there is left to do is to copy the code of part 2 and paste it in the script[7]! The final code is in the following pages, along with the results of the code for 3 tests.

[7] A few changes in threshold values might be needed.

p3 Script Screenshots

```
% Loading The Image
clear;clc;close;
[file,path] = uigetfile({'*.jpg;*.bmp;*.png;*.tif'},['Select An Image' ...
    ' File Containing A Car Plate']);
if file == 0
    return;
end
image = imread([path '\' file]);

blue_part = RGBRange(image,[0,80],[0,120],[100,255]);
white_part = RGBRange(image,[100,255],[100,255],[100,255]);
blue_part = imresize(blue_part,[300,500]);
white_part = imresize(white_part,[300,500]);
image = imresize(image,[300,500]);
blue_part = myremovecom(double(blue_part),30);

subplot(2,2,1)
imshow(blue_part)
title('Separated Blue Part')
subplot(2,2,2)
imshow(white_part)
title('Separated White Part')

[blue_labeled,maxSegment] = mysegmentation(blue_part);
subplot(2,2,3)
imshow(image)
title('Spotted Central Blue Areas')
hold on
for i = 1:maxSegment-1
    [r,c] = find(blue_labeled==i);
    mr = min(r);
    Mr = max(r);
    Mc = max(c);
    mc = min(c);
    deltar = Mr-mr;
    if mc >= 0.8*500 || mc <= 0.2*500 || mr <= 0.2*300 || mr >= 0.8*300
        continue
    end
    rectangle('Position',[mc,mr,Mc-mc,Mr-mr],'EdgeColor','g','LineWidth',2)
    if any(white_part(mr:Mr,Mc+2)==1)
        image = imcrop(image,[mc-20,mr-20,7*deltar+40,deltar+40]);
        break;
    end
end
hold off
```

```
46
47      subplot(2,2,4)
48      imshow(image)
49      title("Ready For Plate Reading")
50      figure
51      %================================================================================
52      image = imresize(image,[300,500]);
53      subplot(2,2,1)
54      imshow(image)
55      title('Original')
56      % Generating Gray Scale And Black-And-White
57      subplot(2,2,2)
58      image = mygrayfun(image);
59      imshow(image)
60      title('Gray Scaled')
61
62      subplot(2,2,3)
63      image = mybinaryfun(image);
64      imshow(image)
65      title('Binary')
66
67      subplot(2,2,4)
68      image = myremovecom(image,400);
69      im2 = myremovecom(image,5000);
70      image = image - im2;
71      imshow(image)
72      title('Ready For Segmentation And Correlation')
73      % Segmentation And Correlation
74      [labeled,maxSegment] = mysegmentation(image);
75      hold on
76      label_positions = zeros(2,maxSegment-1);
77      for i = 1:maxSegment-1
78          [r,c] = find(labeled == i);
79          mc = min(c,[],'all');
80          mr = min(r,[],'all');
81          Mc = max(c,[],'all');
82          Mr = max(r,[],'all');
83          label_positions(1,i) = i;
84          label_positions(2,i) = mc;
85          rectangle('Position',[mc,mr,Mc-mc,Mr-mr],'EdgeColor','g','LineWidth',2)
86      end
87
88      [temp,indexs] = sort(label_positions(2,:));
89      label_positions(1,:) = label_positions(1,indexs);
90      label_positions(2,:) = label_positions(2,indexs);
91
92      persian_training_loading;
93      load TRAININGSET.mat
94      figure
95      colcount = ceil(maxSegment/2);
96      result = [];
```

```
for i = 1:maxSegment-1
    [r,c] = find(labeled == label_positions(1,i));
    character = image(min(r):max(r), min(c):max(c));
    subplot(2,colcount,i)
    imshow(character)
    decision = cell(2,1);
    decision{1,1} = 0;
    decision{2,1} = '';
    for j = 1:length(TRAIN)
        temp = corr2(TRAIN{1,j},imresize(character,size(TRAIN{1,j})));
        if temp >= 0.45 && temp >= decision{1,1}
            decision{1,1} = temp;
            decision{2,1} = TRAIN{2,j};
        end
    end
    result = [result decision{2,1}];
end
result = result(1:min(8,end));
disp(result')
file = fopen('result.txt', 'wt');
fprintf(file,'%s\n',result);
fclose(file);
winopen('result.txt')
```

Test Results

Original Image

Command Window

```
ه
۹
ی
۸
۴
۴
۱
fx >>
```

Ready For Plate Reading

Original Image

Command Window

```
۲
۷
۴
۱
۳
۳
۲
۲
fx >>
```

Ready For Plate Reading

Original Image

Command Window

۶
۲
ق
۳
۷
۸
۴
۶
fx >>

Ready For Plate Reading

THE END