

# Computer Assignment 2

مصطفی کرمانی نیا 810101575

امیر نداف فهمیده 810101540

بخش اول) پیش از هر کار دیگری، در ابتدا مپست را لود می‌کنیم:

```
%% Part 0: Loading the mapset

% Get a list of all files in the "Map Set" directory
files = dir('Map Set');

% Calculate the number of valid files (not '.' and '..')
% The first two elements ('.' and '..') are not image files, so delete them.
len = length(files) - 2;

% Initialize a cell array to store training data.
% The first row will store the images, and the second row will store the corresponding labels (characters).
TRAIN = cell(2, len);

% Loop through each file in the directory, starting from the third file (since the first two are '.' and '..')
for i = 1:len
    % Read the image from the file and store it in the first row of the TRAIN cell array
    TRAIN{1, i} = imread([files(i + 2).folder, '\\", files(i + 2).name]);

    % Extract the first character of the file name (its the label)
    % and store it in the second row of the TRAIN cell array
    TRAIN{2, i} = files(i + 2).name(1);
end

% Save the TRAIN cell array to a file named "TRAININGSET.mat"
% This will store the images and corresponding labels for future use.
save TRAININGSET TRAIN;
```

- برای اینکار ابتدا از فولدری که عکس‌های اعداد و حروف در آن است، عکس‌ها را با `imread` خوانده (البته دو عضو اول را همانطور که در کامنت کد نوشته ایم، نمی‌خوانیم چون تصویر نبوده و صرفاً علامت (.) و (..) هستند). و در یک `cell` بنام `TRAIN` می‌ریزیم. که ردیف اول آن عکس‌هایی با پیکسل‌های باينری و به اندازه‌ی  $24 \times 24$  بوده و ردیف دوم لیبل آن عکس‌ها (همان حرف اول نام فایل هر عکس) است.

Variables - TRAIN

	1	2	3	4	5	6	7	8
1	42x24 logical	42x24 logic...	42x24 lc					
2	'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'
3								

- سپس این **cell** را در فایلی انکود شده بنام **TRAININGSET.mat** می‌ریزیم تا بعدا آنرا لود کرده و استفاده کنیم. (در بخش 7)

**1**- در این بخش صرفا عکس پلاک ورودی را از کاربر می‌گیریم:

```
%> part 1 : input picture
[file,path]=uigetfile({'*.jpg;*.png;*.jpeg'}, 'Select your image');

% build full file path and read the image from the selected file path
img = imread(fullfile(path, file));
figure, imshow(img);
```

- ابتدا با **uigetfile** یک پنجره جهت انتخاب تصویر به کاربر نمایش می‌دهیم.

- سپس توسط **fullfile** آدرس دایرکتوری عکس را به اسم خود فایل عکس وصل کرده (میتوانستیم این کار را همانند بخش قبلی بصورت دستی و با بک اسلش گذاشتن و این ها انجام دهیم اما این تابع را هم میخواستیم امتحان کنیم). و توسط **imread** آن تصویر را خوانده و با تولید یک **figure** و استفاده از **imshow** آنرا نمایش می‌دهیم که از درستی کارمان مطمئن شویم.

مثال:



-2

در این بخش هم با استفاده از تابع `imresize` ابعاد عکس را به ابعاد خواسته شده می‌رسانیم و باز هم جهت اطمینان از درستی عکس حاصل، آنرا نمایش می‌دهیم.

```
%% part 2 : resize
img = imresize(img, [300, 500]);
figure, imshow(img);
```

مثلا:



**3**- در این بخش بدلیل کاهش حجم اطلاعات، تصویر را با رابطه‌ی داده شده، سیاه و سفید می‌کنیم.

```
%% part 3 : build grayscale image
function grayImg = mygrayfun(colorImg)
    % Convert the input color image to grayscale using the weighted sum of color channels
    % grayImg = 0.299 * Red + 0.578 * Green + 0.114 * Blue

    % Extract color channels from the input image
    redChannel = colorImg(:,:,1);
    greenChannel = colorImg(:,:,2);
    blueChannel = colorImg(:,:,3);

    % Apply the grayscale formula
    grayImg = 0.299 * redChannel + 0.578 * greenChannel + 0.114 * blueChannel;
end
```

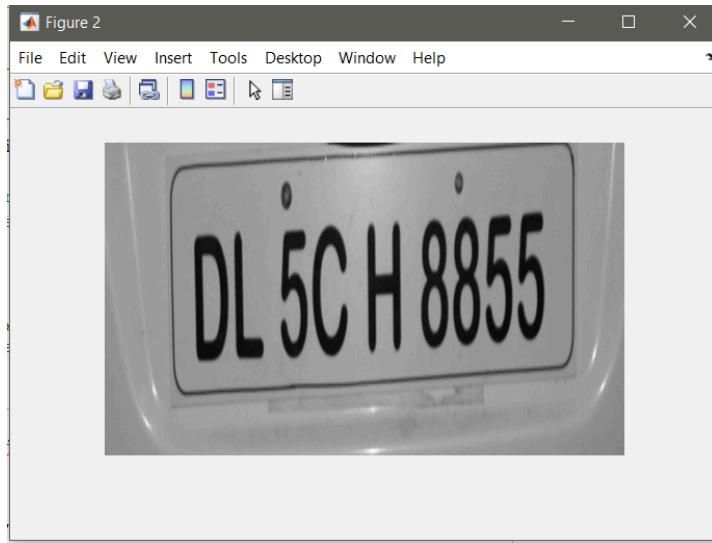
```
%% Convert the color image to grayscale using mygrayfun function
grayImg = mygrayfun(img);
figure, imshow(grayImg);
```

- ساختن تابع `mygrayscale` که عکس رنگی را گرفته و عکس سیاه و سفید برمی‌گرداند را بوسیله‌ی شکستن تصویر اولیه‌ی `RGB` بود به سه ماتریس دو بعدی نمایانگر مقدار رنگ قرمز، آبی و سبز شروع می‌کنیم.

- حالا ضرایب خواسته شده توسط سوال را در ماتریس هر رنگ ضرب کرده و سپس هر سه را با هم جمع می‌کنیم تا طبق فرمولی صورت سوال، به عکس نهایی برسیم.

- جهت استفاده از این تابع هم عکس رنگی را به آن میدهیم و خروجی که عکس سیاه و سفید است در یک ماتریس جدید ریخته و در صورت نیاز، آنرا باز نمایش می‌دهیم.

: مثلاً



4- در این بخش عکس را باينری ميكنيم:

```
%% part 4: Convert the grayscale image to binary using a threshold
function binaryImage = mybinaryfun(grayImage, threshold)
    binaryImage = grayImage > threshold; % Convert to binary using the threshold
    binaryImage = ~binaryImage;
end

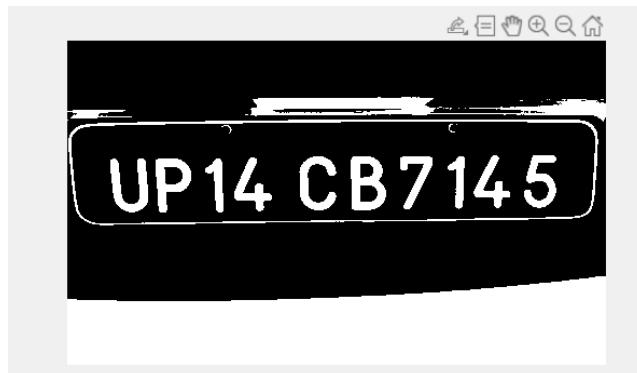
%% Convert the grayImg to binary image using mybinaryfun function
threshold = 100; % Selected threshold value
binaryImage = mybinaryfun(grayImg, threshold);
figure, imshow(binaryImage);
```

- برای زدن این تابع از یک ایده و قابلیت های متلب استفاده کرده و برای اعمال `threshold` صرفا از عبارت شرطی `grayImage > threshold` استفاده کردیم، که برای تک تک پیکسل های آن عکس اعمال شده و یعنی اگر مقدار آن پیکسل از `threshold` بیشتر بود، این عبارت شرطی عدد 1 را بمعنی `true` برمیگرداند و ما هم آن را به آن پیکسل منسوب میکنیم که بمعنای رنگ سفید است و بلعكس، اگر آن پیکسل مقدارش از عدد `threshold` کمتر بود که یعنی از یک حدی سیاهتر بود، این عبارت شرطی مقدار 0 یا `false` برمیگرداند که بمعنی رنگ سیاه در یک عکس باينری است.

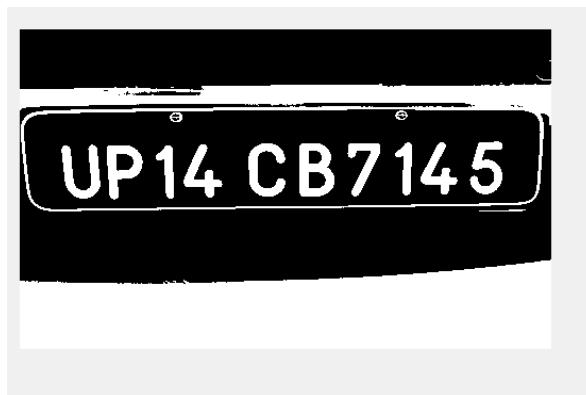
- در نهایت تمام مقادیر را معکوس میکنیم، چون در حالت عادی اعداد روی پلاک، به رنگ مشکی هستند که باعث گرفتن مقدار 1 در این تبدیل میشود، اما اعداد موجود در دیتابیس ما، سفید هستند، پس اگر بیت ها را عکس را همینجا معکوس کنیم تا اعداد روی پلاک به رنگ سفید و با مقدار 1 نمایش داده شوند، بعدا در correlation کارمان راحتتر و سر راست میشود.

- چون تصویر نوشته های پلاک معمولا خیلی سیاه است پس اگر threshold را کمتر از 128 بگیریم هم مشکلی رخ نمیدهد پس اینکار را کردیم تا بخشی از نویز ها در همین مرحله رسمآ حذف شوند:

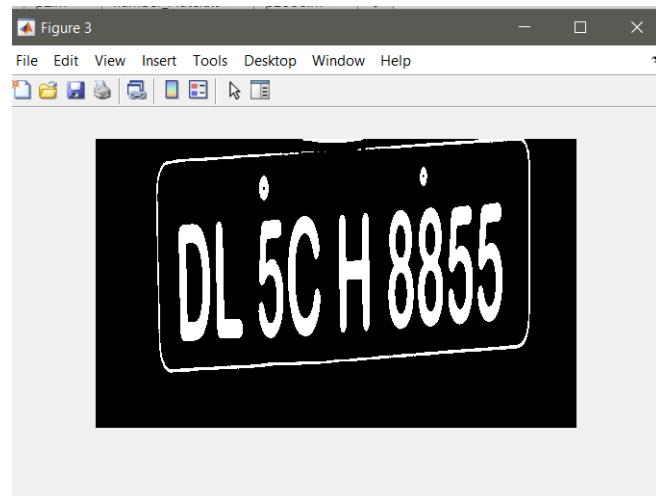
:threshold 100 با



:threshold 128 با •



مثال:



---

5- در این بخش نویزهای تصویر (کامپوننت های بسیار کوچک) را حذف میکنیم. چون تابع طولانی است، آنرا تقسیم کرده ام و توضیح میدهم.

```
%> part 5: Remove small connected components (noise) without using bwareaopen
function cleanImage = myremovecom(binaryImage, n_minSize)
    % Step 1: Find all points where the picture is 1 (white pixels in the binary image)
    [row, col] = find(binaryImage == 1); % Get row and column indices of white pixels (value 1)
    POINTS = [row'; col']; % Store points as a 2*N matrix (row in first row, col in second row)

    if isempty(POINTS)
        cleanImage = binaryImage; % If no points are found, return the original image
        return;
    end

    cleanImage = zeros(size(binaryImage)); % Initialize the clean image (binary)

    % Step 2: Process all points and group them into connected components
    while ~isempty(POINTS)
        % Initialize the current object with the first point
        initpoint = POINTS(:, 1);
        POINTS(:, 1) = []; % Remove the used point

        % Use a stack to explore the connected component
        points_to_explore = initpoint;
        currentObject = initpoint;
```

- این تابع در ابتدا لوکیشن بخش هایی از تصویر را که مقدار 1 دارند(سفیدند) پیدا میکند و دو وکتور به نام row و col می‌ریزد. سپس ما این دو وکتور را به هم چسبانده و یک ماتریس با دو ردیف تشکیل میدهیم که هر ستون آن نمایانگر شماره‌ی سطر و ستون یکی از نقاط سفید رنگ است.

- سپس برای اطمینان فقط بررسی میکنیم که آیا اصلا عکس ما نقطه‌ی سفیدی داشته است یا نه، اگر نداشته بود کلا همان تصویر را بر می‌گردانیم. مگرنه، در وهله‌ی اول، عکس نهایی را بصورت یک ماتریس از 0 ها و به سایز عکس اولیه تعریف میکنیم.(یعنی یک تصویر کاملا سیاه داریم)

- حالا در یک حلقه‌ی while قرار است به روش هایی که گفته می‌شوند، دانه به دانه کامپوننت های عکس را یافته و اگر آن کامپوننت از یک حدی بزرگتر بود، آنرا مورد تایید قرار داده و به عکس نهایی اضافه کنیم، و هر نقطه‌ای که به یک کامپوننت اضافه میکنیم، آنرا از لیست نقاط اولیه حذف میکنیم و این حلقه‌ی بیرونی، تا وقتی تمام نقاط به کامپوننت های مربوط به خودشان اضافه شوند، ادامه خواهد داشت.

- حلقه‌ی بیرونی به این شکل آغاز می‌شود که نقطه‌ی اول از لیست نقاط سفید را برداشته و آنرا از لیست نقاط سفید حذف کرده و به لیست نقاط کامپوننت اول اضافه می‌کنیم. یک استک هم می‌سازیم بنام تمام نقاط سفید DFS که قرار است بوسیله‌ی آن و الگوریتم points\_to\_explore را بازدید کنیم.

```
while ~isempty(points_to_explore)
    % Take the first point from the stack
    current_point = points_to_explore(:, 1);
    points_to_explore(:, 1) = []; % Remove the used point

    % Find new points connected to the current_point
    [POINTS, newPoints] = close_points(current_point, POINTS);

    % Add the new points to the current object and exploration stack
    currentObject = [currentObject newPoints];
    points_to_explore = [points_to_explore newPoints];
end

% If the connected component's size is large enough, add it to the clean image
if size(currentObject, 2) >= n_minsize
    for j = 1:size(currentObject, 2)
        cleanImage(currentObject(1, j), currentObject(2, j)) = 1;
    end
end
end
```

- در حلقه‌ی دوم، تا وقتی استک ما خالی نشده است هر بار عضو اول استک نقاط بازدید نشده را برداشته و همسایه‌های سفید و مجاور آن را بوسیله‌ی تابع close\_points (که در انتهای شرح میدهم آنرا) پیدا کرده و آنها را به کامپوننت فعلی اضافه کرده و در استک بازدید نشده‌ها قرار می‌دهیم تا در iteration‌های بعدی، همسایه‌های آنها را هم به لیست اضافه کنیم.

- پس نهایتاً استک نقاط بازدید نشده خالی می‌شود که یعنی دیگر نقطه‌ی سفیدی نیست که بتوان از طریق همسایگی، به کامپوننت فعلی افزود، پس از حلقه‌ی داخلی بیرون آمده و بررسی می‌کنیم که اگر تعداد نقاط این کامپوننت که تکمیل شده است، از مقدار مینیمم تعیین شده بیشتر بود، آنرا به عکس نهایی اضافه کنیم و اگر کمتر بود آنرا دور میریزیم.

- نهایتاً دوباره به حلقه‌ی بیرونی رفته و بین نقاط سفید باقی مانده دوباره کامپوننت پیدا میکنیم و همینطور میرویم تا تمام نقاط وارد کامپوننت هایی شده و یا وارد عکس نهایی شوند و یا دور ریخته شوند.

```
%% Helper function to find points close to a given point
function [remainingPoints, neighbor_Points] = close_points(initpoint, POINTS)
    % This function returns points in POINTS that are close to
    % the initpoint and remove them from POINTS

    neighbor_Points = [];
    remainingPoints = POINTS;
    for i = size(POINTS, 2):-1:1
        % Check the distance between the initpoint and the current point
        if abs(POINTS(1, i) - initpoint(1)) <= 1 && abs(POINTS(2, i) - initpoint(2)) <= 1
            % Add the point to newPoints and remove it from remainingPoints
            neighbor_Points = [neighbor_Points POINTS(:, i)];
            remainingPoints(:, i) = [];
        end
    end
end
```

- عملگرد تابع `close_opoints` هم به این گونه است که نقطه‌ی مدنظر و لیست تمام نقاط سفید را به آن میدهیم و آن از بین ۸ همسایه‌ی نزدیک به آن نقطه جستجو کرده و می‌بیند کدام نقاط سفیدند. سپس همسایه‌های سفید را داخل لیست `neighbor_points` ریخته و آنها را از لیست نقاط سفید اولیه حذف میکند.

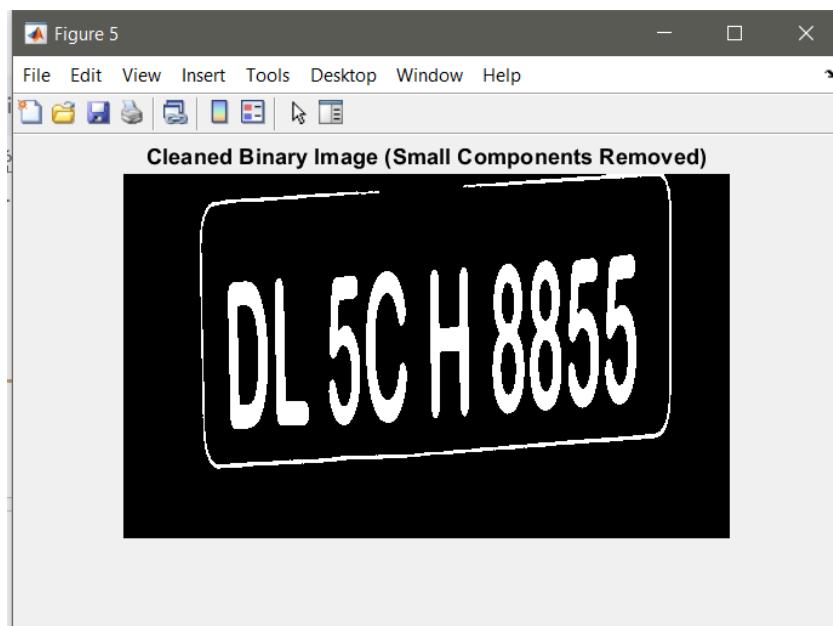
- دلیل استفاده از حلقه‌ی معکوس هم این است که چون در طی این حلقه اگر همسایه‌ای یافت شود، از لیست `points` که در ابتدا مساوی با همان `remaining_points` بوده است، حذف میشود و حلقه هم در حال لوپ زدن تا رسیدن به آخرین ایندکس `points` است پس اگر حلقه‌ی مستقیم بزنیم ممکن است به مشکلات ایندکسی بخوریم.

- نهایتاً بعنوان خروجی لیستی از همسایگان سفید نقطه‌ی ورودی، به همراه نقاط سفید باقی مانده بر میگردد.

```
%% the cleaned binary image
minSize = 300; % Minimum size of objects to keep
cleanImage = myremovecom(binaryImage, minSize);
figure, imshow(cleanImage);
title('Cleaned Binary Image (Small Components Removed)');
```

- برای استفاده ازینتابع هم کافیست تصویر باینری شده و مینیمم سایز معتبر برای یک کامپوننت را به آن بدھیم و تصویر بدون نویز را دریافت کنیم.

:مثال



6- در این بخش تصویر را segment بندی میکنیم.

```
%% part 6: segmenting the picture
function [labeledImage, numObjects] = mysegmentation(cleanImage)
    % Step 1: Find all points where the picture is 1 (white pixels in the clean binary image)
    [row, col] = find(cleanImage == 1); % Get row and column indices of white pixels (value 1)
    POINTS = [row'; col']; % Store points as a 2*N matrix (row in first row, col in second row)

    if isempty(POINTS)
        labeledImage = zeros(size(cleanImage)); % If no points are found, return an empty labeled image
        numObjects = 0; % No objects detected
        return;
    end

    labeledImage = zeros(size(cleanImage)); % Initialize the labeled image
    currentLabel = 0; % Start with label 0

    % Step 2: Process all points and group them into connected components
    while ~isempty(POINTS)
        % Increment the label for the new object
        currentLabel = currentLabel + 1;

        % Initialize the current object with the first point
        initpoint = POINTS(:, 1);
        POINTS(:, 1) = []; % Remove the used point

        % Use a stack to explore the connected component
        points_to_explore = initpoint;
        currentObject = initpoint;
```

- کد این بخش از خیلی لحاظ شبیه بخش قبل است. چون رسما بار باید کامپوننت ها را با DFS یا الگوریتم های دیگر تشخیص اما اینبار به هر کدام لبیل جداگانه میزنیم.

- پس مجددا با یافتن نقاط سفید شروع کرده و تصویر اولیه را یک تصویر کاملا سیاه در نظر گرفته و حلقه‌ی بیرونی که تا تمام شدن تمام نقاط سفید ادامه دارد را ساخته و حلقه‌ی درونی که تا خالی شدن لیست نقاط بازدید نشده در استک ادامه دارد را ساخته و در حلقه‌ی درونی هر بار همسایه‌های نقطه‌ی مدنظر (که بوسیله‌ی تابع close\_points که قبلا توضیح دادیم، پیدا میکنیم) را به استک نقاط بازدید نشده اضافه کرده و هر بار که همسایه‌های یک نقطه را میبینیم خود آن نقطه را از استک حذف میکنیم و نهایتا طبق الگوریتم مشابه DFS تمام نقاطی که میتوانند توسط همسایگی به یک کامپوننت متصل شوند، به آن متصل میشوند.

```

while ~isempty(points_to_explore)
    % Take the first point from the stack
    current_point = points_to_explore(:, 1);
    points_to_explore(:, 1) = []; % Remove the used point

    % Find new points connected to the current_point
    [POINTS, newPoints] = close_points(current_point, POINTS);

    % Add the new points to the current object and exploration stack
    currentObject = [currentObject newPoints];
    points_to_explore = [points_to_explore newPoints];
end

% Step 3: Label the connected component in labeledImage
for j = 1:size(currentObject, 2)
    labeledImage(currentObject(1, j), currentObject(2, j)) = currentLabel;
end

% Return the labeled image and the number of objects
numObjects = currentLabel; % Total number of labeled objects
end

```

- تفاوت کوچک این بخش با بخش قبلی وقتی شروع می شود که ما یک کامپوننت را بطور کامل یافتیم. پس از حلقه ی درونی خارج شده و اینجا لبیلی که در حلقه ی بیرونی با اتمام هر کامپوننت یک واحد به آن اضافه می کنیم را به لوکیشن مربوط به تک تک نقاط آن کامپوننت نسبت می دهیم.

- نهایتا هم یکی از خروجی های تابع، عکس لبیل زده شده است، که در آن تمام اعضای هر کامپوننت لبیل یکسان داشته ولی هر کامپوننت با دیگری لبیلش فرق میکند. و تعداد نهایی کامپوننت ها یا بعبارتی لبیل ها هم خروجی داده میشود.

```

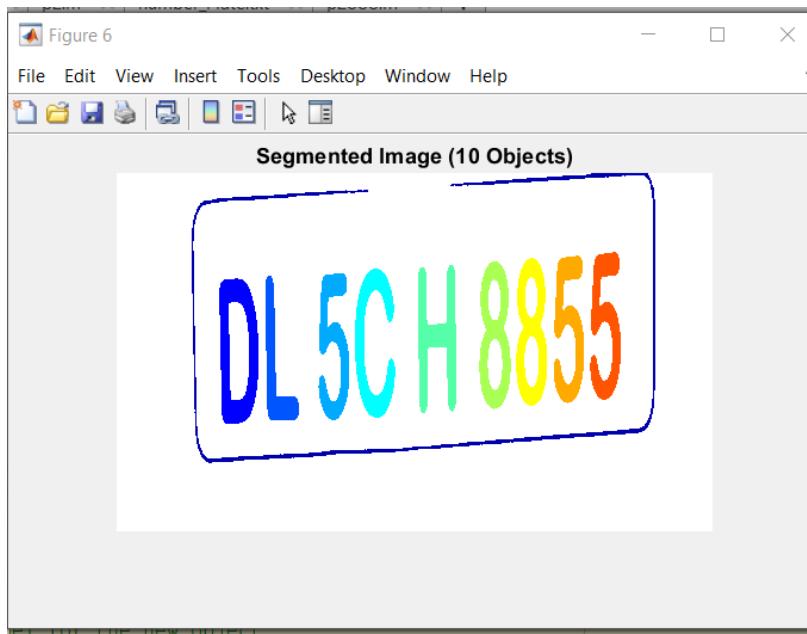
%% the segmented (labeled) image
[labeledImage, numObjects] = mysegmentation(cleanImage);
figure, imshow(label2rgb(labeledImage)); % Convert labels to colors for visualization
title(['Segmented Image (' , num2str(numObjects) , ' Objects')']);

```

- نکته اینجاست که تصویر نهایی، یک تصویر بازیری نیست و فقط یکسری پیکسل ها دارد که لبیل خورده اند. پس بطور مستقیم قابل رسم نیست، پس برای دیده شدن نتیجه ی کار این تابع، از تابعی کمکی بنام

استفاده کردیم که هر لیل را به یک رنگ `rgb` تبدیل کرده و بدین ترتیب میتوان عملیات لیل زدن را قابل مشاهده کرد.

:مثل



## 7- تصمیم گیری نهایی

```
%& Part 7: Compact Decision-Making Using Correlation (corr2)

% Load the saved training set
load TRAININGSET;
numTemplates = size(TRAIN, 2);

recognizedText = ''; % Store recognized characters
for segmentIndex = 1:numObjects
    [rowIndices, colIndices] = find(labeledImage == segmentIndex); % Find row and column indices of the current object

    % Extract the sub-image (the current segment) from the binary cleanImage.
    % The segment is defined by the bounding box surrounding the object (min/max row and column indices).
    % Then, resize the extracted segment to 42x24 pixels to match the size of the templates in the training set.
    currentSegment = imresize(cleanImage(min(rowIndices):max(rowIndices), min(colIndices):max(colIndices))), [42, 24]);

    % Compute correlation scores for all templates
    ro = zeros(1, numTemplates); % Initialize correlation scores array
    for templateIndex = 1:numTemplates
        ro(templateIndex) = corr2(TRAIN{1, templateIndex}, currentSegment); % Correlation for each template
    end
end
```

- در ابتدا دیتاستی که در مرحله‌ی اول سیو کرده بودیم را لود میکنیم. سپس یک رشته‌ی خالی برای ثبت نتیجه‌ی نهایی پلاک خوانی تولید میکنیم.

- حالا در یک حلقه `for` تک تک `segment` های یافته شده را از روی لیبل آنها در تصویر لیبل دار پیدا کرده ( به این صورت که ستون‌ها و ردیف‌های آنها را در قالب دو وکتور دریافت میکنیم با تابع `find`) سپس از کمترین سطrix که آن لیبل را دارد تا بزرگترین سطrix که آن لیبل را دارد و از کوچکترین تا بزرگترین ستونی که آن لیbel را دارند را از تصویر بدون نویز انتخاب کرده و رسما انگار یک مربع دور هر `segment` میکشیم) و سپس `resize` میکنیم تا هم اندازه با عکس‌های موجود در دیتاست شوند.

- حالا یک وکتور به اندازه‌ی تعداد اعضای دیتاست درنظر میگیریم تا با یک حلقه‌ی `for` درونی مقدار کورلیشن تک تک اعضای دیتا است را با آن `segment` های که مدنظر است، بیابیم. و عملیات کورلیشن را با تابع `corr2` میکنیم.

```
% Create a new figure for each segment with 2 subplots: one for the image, one for the plot
fig = figure; % Create a new figure for every segment

% Subplot 1: Show the image of the segment
subplot(1, 2, 1); % 1 row, 2 columns, 1st subplot
imshow(currentSegment); % Display the segment image
title(['Segment ', num2str(segmentIndex)]);

% Subplot 2: Show the correlation scores (ro)
subplot(1, 2, 2); % 1 row, 2 columns, 2nd subplot
bar(ro); % Bar plot of correlation scores
title(['Correlation Scores for Segment ', num2str(segmentIndex)]);
xlabel('Template Label');
ylabel('Correlation Score');

% Customize the x-axis to display the template labels instead of indices
xticks(1:numTemplates); % Set x-ticks at positions corresponding to each template
xticklabels(TRAIN(2, :)); % Set x-tick labels to the corresponding template labels (characters)

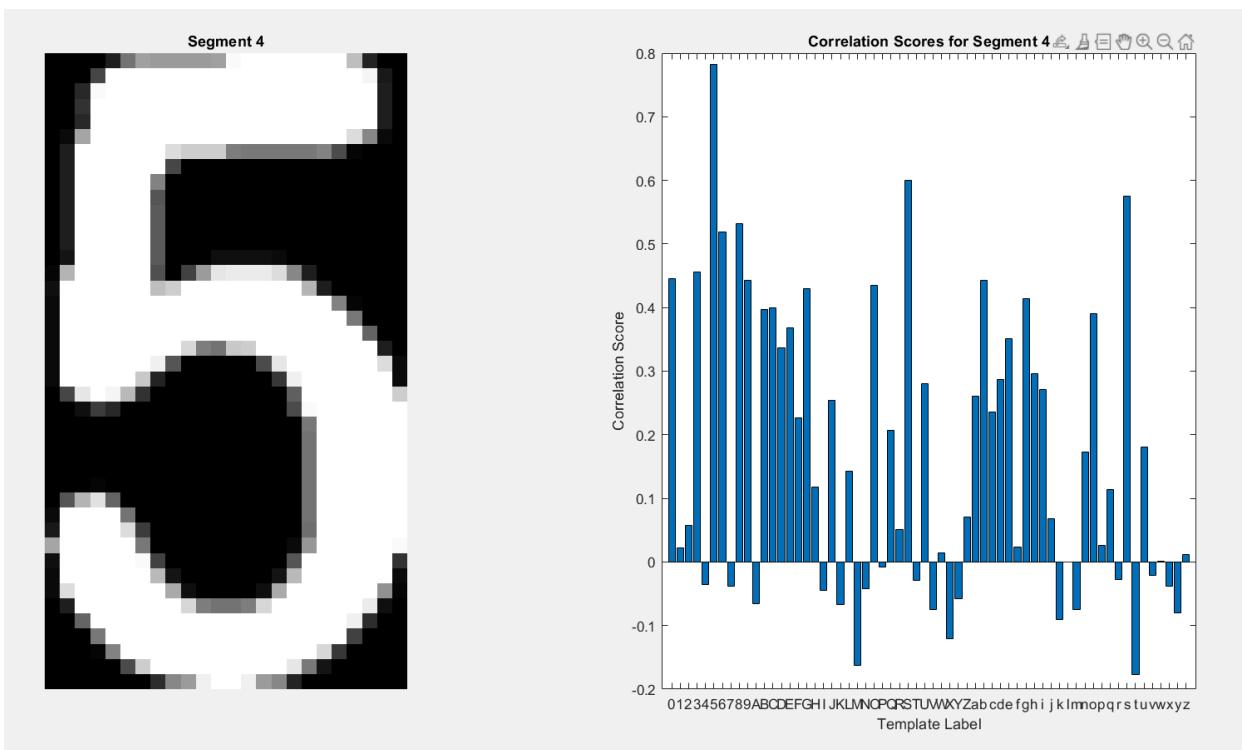
drawnow; % Ensure that the figure is rendered before the next iteration

% Find the best match and apply a threshold
[bestScore, bestMatchIndex] = max(ro); % Get the best match score and index
if bestScore > 0.48 % Threshold for accepting a match
    recognizedText = [recognizedText, TRAIN(2, bestMatchIndex)]; % Append detected character
end
end
```

- حالا برای هر **segment** یک **figure** میسازیم تا ضمن نشان دادن تصویر خود آن کامپوننت، نموداری شامل مقدار کورلیشن آن کامپوننت با تک تک اعضای مپ است به ما نمایش بدهد و البته محور افقی را طوری تنظیم میکنیم که خود حرف یا عددی که عکس آن در مپ است را برایمان بنویسد تا نمودار قابل فهم تر باشد.

- سپس عکسی که بالاترین مقدار کورلیشن را داشت انتخاب میکنیم و بعنوان حدس خود از حرف یا عدد آن کامپوننت اعلام میکنیم و به رشتہ‌ی مربوط به حدس نهایی اضافه میکنیم و به سراغ کامپوننت بعدی می‌رویم. در این میان چون ممکن است عکسمان در مرحله‌ی **denoise** بطور کامل هم نشده باشد. یک **threshold** میگذاریم که اگر مقدار کورلیشن هیچ کدام از اعضای مپ است با یک کامپوننت خاص، از مقدار آن بیشتر نشد، کلا آن را بعنوان نویز یا کامپوننت ناخوانا دور بریزد.

مثلا **figure** مربوط به یکی از بخش‌های پلاکی که داشتیم:



- همانطور که مشخص است، مقدار کورلیشن برای عدد 5 از بقیه اعداد و حروف مپ ست بیشتر بوده و در نتیجه این segment به درستی تشخیص داده میشود

- دلیل منفی بودن مقدار کورلیشن در برخی مقادیر هم این است که تابع corr2 مقدار میانگین را هم از حاصل کرلیشن کم کرده تا آنرا تا حدی متعادل کند.

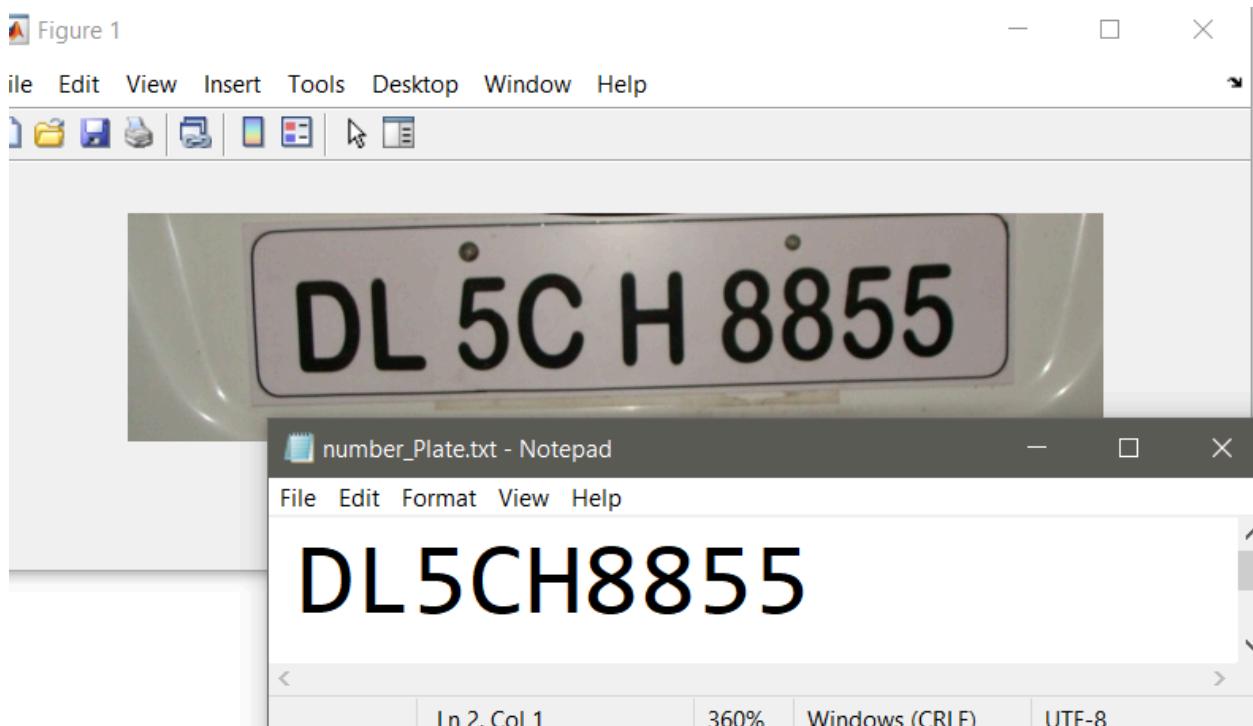
## 8- نتیجه ای نتیجه

```
%% Part 8: Save the Detected Characters to a File
% Display the result
disp(['Detected pelaak: ', recognizedText]);

% Save the detected characters to a file
fileID = fopen('number_Plate.txt', 'wt');
fprintf(fileID, '%s\n', recognizedText);
fclose(fileID);
winopen('number_Plate.txt');
```

- نهایتا نتیجه را چاپ کرده و در یک فایل txt نوشته و فایل را می بندیم و یکبار هم جهت بررسی نهایی، آن فایل را open میکنیم.

نتیجه ای نهایی در این مثال:



- همانطور که مشخص است، عدد و حروف ها به درستی تشخیص داده شده اند.

## بخش دوم)

1. چالش اول این بخش، ساختن دیتاست فارسی است. ما با جستجوی بسیار در اینترنت (و حتی پرداخت 4 هزار تومان پول!) نهایتاً متوجه شدیم که شبیه ترین فونت فارسی به فونت به کار رفته در پلاک ها، فونت Titr B است. پس به کمک مقداری کدهای پایتون که در نوت بوک persian\_DS\_generator سیو شده است، یک دیتاست بر اساس فونت Titr B ساختیم.

```
[ ] !pip install pillow
→ Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (10.4.0)

[ ] from google.colab import files
uploaded = files.upload() # آپلود کردن فایل فونت، با نام B_Titr.ttf
→ Show hidden output

▶ from PIL import Image, ImageDraw, ImageFont
import shutil

# مسیر فونت آپلود شده
font_path = "/content/B_Titr.ttf"

[ ] # سایز و رنگ فونت
font_size = 100
font_color = (0, 0, 0) # مشکی
background_color = (255, 255, 255) # سفید

# ایجاد فونت
font = ImageFont.truetype(font_path, font_size)

# لیست اعداد فارسی و حروف مورد نظر
characters = [str(num) for num in range(10)] + ['ا', 'ب', 'ج', 'ه', 'و', 'ن', 'م', 'ل', 'ط', 'ا', 'ا']

# ساخت تصویر برای هر کاراکتر (عدد یا حرف)
for char in characters:
    # اندازه تصویر
    img_size = (150, 150)

    # ایجاد تصویر جدید
    img = Image.new("RGB", img_size, background_color)
    draw = ImageDraw.Draw(img)

    # اندازه متن
    text_bbox = draw.textbbox((0, 0), char, font=font)
    text_width = text_bbox[2] - text_bbox[0]
    text_height = text_bbox[3] - text_bbox[1]

    # مرکز کردن متن در تصویر
    text_position = ((img_size[0] - text_width) // 2, (img_size[1] - text_height) // 2)

    # کشیدن کاراکتر روی تصویر
    draw.text(text_position, char, font=font, fill=font_color)

    # ذخیره تصویر به نام کاراکتر (عدد یا حرف)
    img.save(f"{char}.png")

[ ] # شامل همه تصاویر با استفاده از دستور ZIP ایجاد فایل
!zip -r characters_images.zip /content/*.png

# دانلود فایل
files.download("characters_images.zip")
→ Show hidden output
```

- نحوه‌ی کار کد هم اینجوری است که ابتدا فونت مدنظر را آپلود کرده و سایز و رنگ فونت و پس زمینه را انتخاب میکنیم، که من مثلاً متى مشکی روی زمینه‌ی سفید میخواهم. سپس بکمک کتابخانه‌ی `pillow` یک تصویر به سایز دلخواه مثلاً  $150*150$  ایجاد کرده و کاراکتر‌های مورد نظر را با فونتی که اپلود

کرده ایم فایل tff آنرا، می نویسیم به ترتیب. حالا توسط تابع boundary box موقعیت مکانی گوشه های چپ و راست و بالا و پایین آن حرف یا عدد موجود در عکس را یافته و سپس طول و عرض آن را بوسیله ای همین مقادیر یافته و آنرا به مرکز تصویر می آوریم. حالا تمام تصاویر را زیپ کرده و دانلود میکنیم.

- حالا وقت لود کردن دیتاست به داخل متلب است:

```
%% Part 0: Loading the mapset

% Get a list of all files in the "Map Set" directory
files = dir('persian_dataset');

% Calculate the number of valid files (not '.' and '..')
% The first two elements ('.' and '..') are not image files, so delete them.
len = length(files) - 2;

% Initialize a cell array to store training data.
% The first row will store the images, and the second row will store the corresponding labels (characters).
TRAIN = cell(2, len);

% Loop through each file in the directory, starting from the third file (since the first two are '.' and '..')
for i = 1:len
    % Read the image from the file and store it in the first row of the TRAIN cell array
    TRAIN{1, i} = imread([files(i + 2).folder, '\\", files(i + 2).name]);

    % Extract the first character of the file name (its the label)
    % and store it in the second row of the TRAIN cell array
    TRAIN{2, i} = files(i + 2).name(1);
end
```

- چون این بخش از کد دقیقا مثل سوال اول است، توضیحی نمیدهیم چون صرفا تک تک عکس های مپست، لود شده اند در یک cell بنام train. اما مشکل اینجاست که دیتاست ساخته شده توسط پایتون دو مشکل دارد، اولی اینکه تصاویر آن سیاه و سفیدند اما در فرمت RGB هستند و هم 3 بعدی هستند و هم مقادیر پیکسل هایشان بازیزی نیست، از طرف دیگر ما تصویر حروف و اعداد را در پایتون صرفا به مرکز آوردهیم، اما برای کورلیشن بهتر، لازم است دقیقا هر حرف و عدد به لبه های تصویر بچسبد و در حد توان، بزرگ باشد.

Variables - TRAIN															
2x25 cell															
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
[150x150x3 ...]	[150x150x3 ...]	[150x150x3 ...]	[150x150x3 ...]	[150x150x3 ...]	[150x150x3 ...]	[150x150x3 ...]	[150x150x3 ...]	[150x150x3 ...]	[150x150x3 ...]	[113x150x3 ...]	[150x150x3 ...]	[150x150x3 ...]	[150x150x3 ...]	[150x150x3 ...]	
'0'	'1'	'2'	'3'	'4'	'5'	'6'	'7'	'8'	'9'	'ا'	'ب'	'ج'	'س'	'ص'	
3															

```
%% part 0.1 : convert RGB mapset to Binary images and save them again
% Define the threshold value for binarization
threshold = 128; % Adjust the threshold as needed

% Loop through each image in the TRAIN dataset
for i = 1:size(TRAIN, 2)
    % Extract the RGB image from the training set
    rgbImage = TRAIN{1, i};

    % Convert the RGB image to grayscale using your custom function
    grayImage = mygrayfun(rgbImage);

    % Convert the grayscale image to binary (logical) using your custom function
    binaryImage = mybinaryfun(grayImage, threshold);

    % Replace the RGB image in TRAIN with the binary image
    TRAIN{1, i} = binaryImage;
end
```

- پس اولا به کمک توابعی که خودمان ساختیم در بخش اول، تصویر های مپ ست را ابتدا سیاه و سفید و سپس با یک `threshold` خوب، باینری کردیم.

```

%% part 0.2 : Code to Detect the Bounding Box, Crop, and Resize

% Loop through each image in the TRAIN dataset
for i = 1:size(TRAIN, 2)
    % Extract the binary image from the training set
    binaryImage = TRAIN{1, i};

    % Find the rows and columns that contain the value 1
    [rows, cols] = find(binaryImage == 1);

    % If no '1' pixels are found, keep the image as is
    if isempty(rows) || isempty(cols)
        croppedImage = binaryImage; % No cropping needed, no '1' found
    else
        % Detect the smallest bounding box that contains all the '1' pixels
        minRow = min(rows);
        maxRow = max(rows);
        minCol = min(cols);
        maxCol = max(cols);

        % Crop the image to that bounding box
        croppedImage = binaryImage(minRow:maxRow, minCol:maxCol);
    end

    % Resize the cropped image back to 150x150
    resizedImage = imresize(croppedImage, [150 150]);

    % Overwrite the original image in the TRAIN cell array with the resized image
    TRAIN{1, i} = resizedImage;
end

% Save the modified TRAIN cell array back to the same TRAININGSET.mat file
save TRAININGSET TRAIN;

```

- ثانیا قرار است آنها را تا حد ممکن برش بدھیم تا حروف و اعداد دقیقاً تابه های تصویر ببینند. پس بکمک `find` ابتدا جاھایی که مقدار 1 دارند(سفید هستند) را شناسایی کرده و اگر کلا هیچ بخش سفیدی نبود که همان تصویر را برミگردانیم چون تصویر کاملا سیاه است. اما اگر بخش های سفید داشت، تصویر را بکمک `imcrop` کوچکترین و بزرگترین ستون و سطر هایی که سفیدند، برش میزنیم و نهایتاً هم `imresize` کرده تا به سایز عکس های مپ سمت برسند. نهایتاً هم دیتا سمت بدست آمده را سیو میکنیم در فایل `.mat`:

- برای مراحل بعدی کار، رسما از همان کدهای بخش اول استفاده کردیم، اما تفاوت‌هایی بود که ذکر میکنیم:

- اولین تفاوت این بود که اکنون، دیتاستی که داشتیم  $150 \times 150$  بود بجای  $42 \times 24$ .
- دومین تفاوت این بود که برخی حروف فارسی، نقطه دارند، و چون نقطه ها جدا از بدنھی حرف هستند، نقطه و حرف بعنوان کامپوننت های جداگانه شناخته میشند و چون نقطه ها شبیه عدد 0 فارسی هستند، پس تشخیص های اشتباه رخ میداد. برای حل این مشکل با توجه به اینکه این حروف

تعداد کمی داشتند و فارغ از نقطه‌ای که داشتند، فرم خاصی هم داشتند، متوجه شدیم میتوان بدون در نظر گرفتن نقطه هم آنها را تشخیص داد.

- پس در وهله‌ی اول، در دیتاست اولیه، علاوه بر حروف نقطه‌دار، معادل بدون نقطه‌ی آنها را هم با همان لیبل قرار دادیم

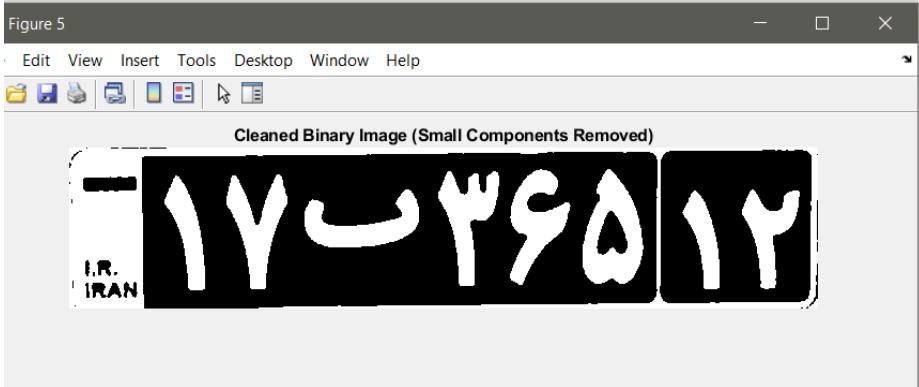
مثال:



این کار مشکلی پیش نمی‌آورد چون در بین 12 حرف گفته شده برای تشخیص، هیچ دو حرفی نبودند که فقط در نقطه اختلاف داشته باشند.

- خب تا اینجا مشکل تشخیص حروف نقطه‌دار، حل شد، اما با مشکل تشخیص نقطه‌هایی با عنوان عدد 0 چه کنیم؟ برای اینکار صرفاً لازم بود مقدار threshold مدنظر برای denoise را آنقدر بالا ببریم که هم تک نقطه‌هایی مثل نقطه‌ی (ب) و هم دونقطه‌های چسبان مثل نقطه‌های (ق) عنوان نویز تشخیص داده شده و حذف بشوند.

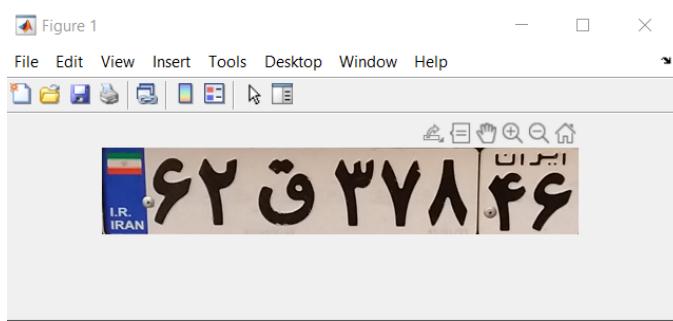
مثال:



- نکته‌ی قابل توجه بعدی، فرمی بود که حرف (ه) نوشته می‌شد در پلاک‌ها که برخلاف فرم معمولی، به صورت چسبان نوشته می‌شد که این را هم در کد پایتون اولیه اعمال کردیم و مشکل حل شد.

پس نهایتاً مثلاً برای یک پلاک خاص، داریم:

عکس اصلی:



بعد از resize برای شبیه شدن به طول و عرض عدد ها و حروف در عکس های دیتابست:



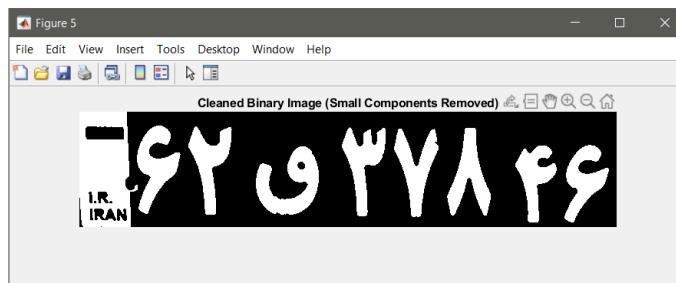
سیاه و سفید کردن:



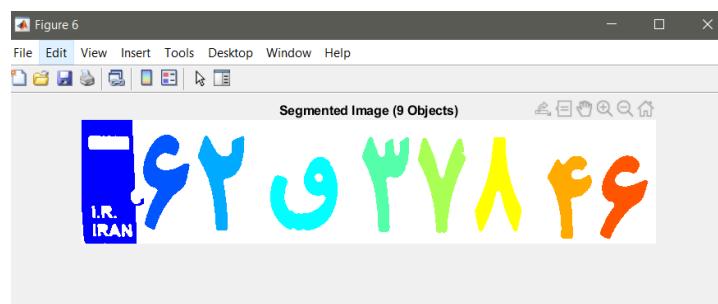
باینری کردن:



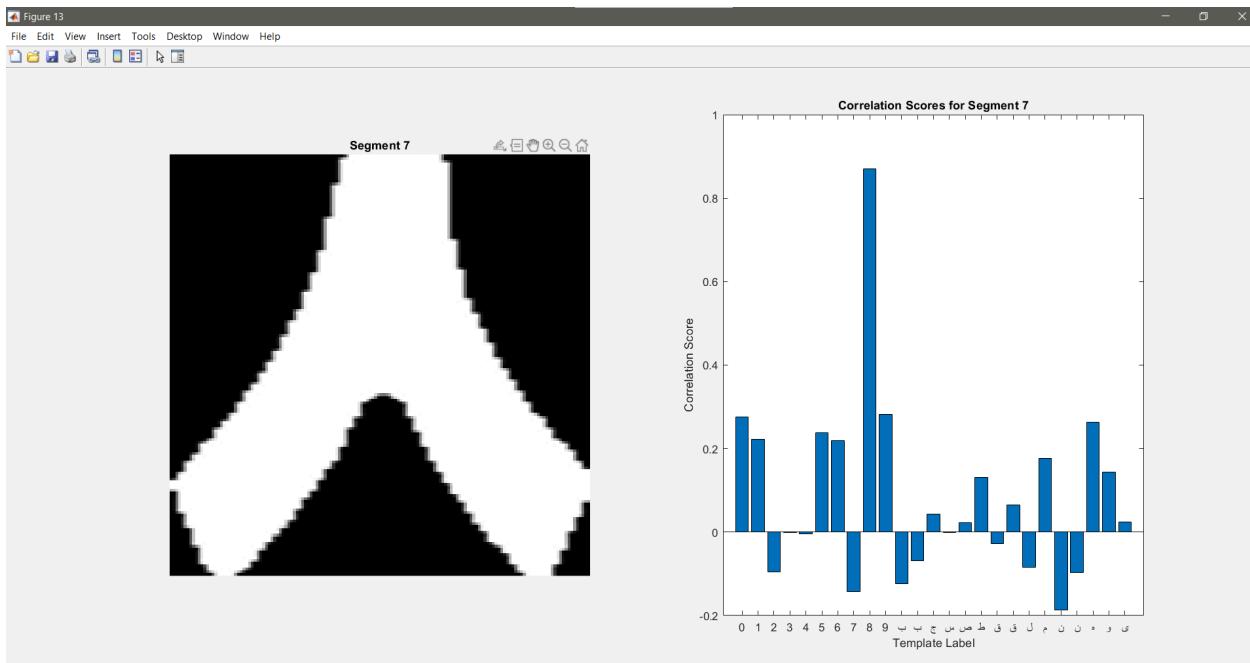
از بین بردن نویز ها و نقطه های حروف:



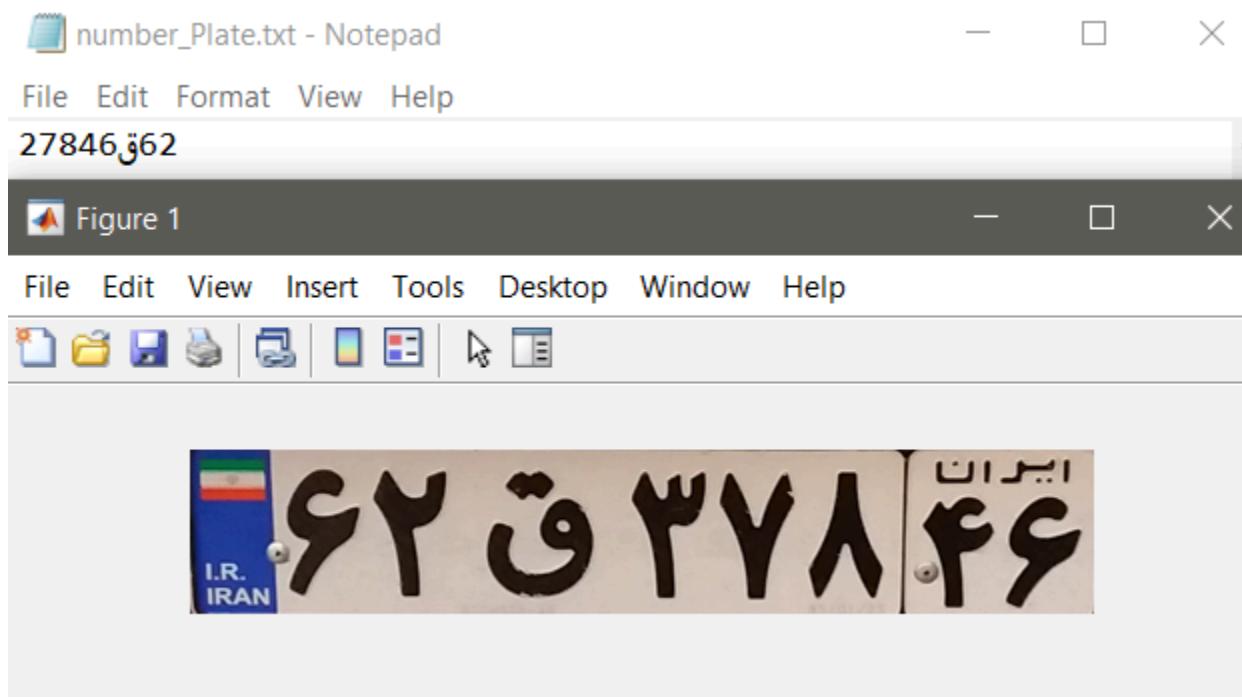
لیل زدن به segment ها:



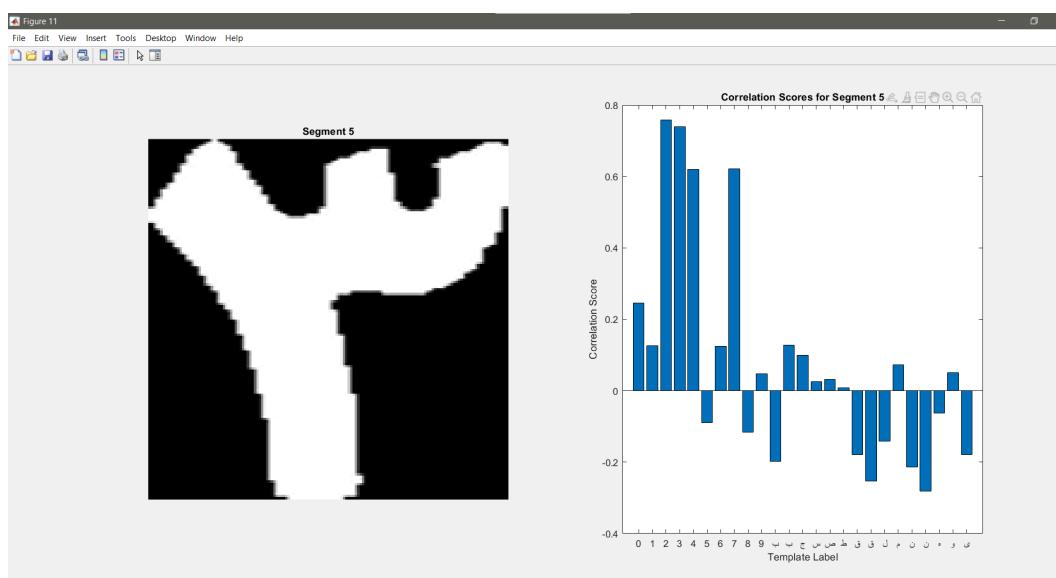
مثالی از تشخیص درست یکی از اعداد:



نتیجهٔ نهایی:



نکته‌ی مهم اینکه چون حروف فارسی در note pad جایه‌جا می‌شوند اینگونه نمایش داده می‌شود، مگرنه از بین 62 و قاف و 378 و 46 فقط عدد 3 در 378 اشتباه‌ها 2 تشخیص داده شده که آن هم با فاصله‌ی کمی در کورلیشن بوده که میتوان بکمک دقیق‌تر کردن بررسی‌ها و فیلتر‌ها، مشکلش را حل کرد:



### بخش سوم)

---

برای این بخش ما از الگوریتم های زیر استفاده کردیم:

- گرفتن تفاضل بین عکس، و عکسی که یک پیکسل به چپ یا راست رفته برای یافتن لبه های عمودی. و همین کار برای یافتن لبه های افقی
- یافتن بخش آبی کنار پلاک و یافتن کل پلاک توسط تناسب طول و عرض پلاک
- گرفتن تفاضل بین عکس و نمونه های شیفت شده با راست یا چپ به مقدار کم، برای فهمیدن منطقه ای با بیشترین تغییرات بین رنگ های خالص سیاه و سفید یا عبارتی مناطقی با ماکسیمم مشتق رنگی
- از بین روش های بالا، یافتن بخش آبی کنار پلاک و یافتن کل پلاک توسط تناسب طول و عرض پلاک، روش بهتری بود اما باز هم دقت کافی را نداشت زیرا تشخیص دادن رنگ آبی در بین رنگ های مختلف سخت بود. پس از مقادیری جستجو و کمک گرفتن از هوش مصنوعی، به فضای HSV رسیدیم:

در فضای رنگی HSV، هر رنگ به سه مؤلفه تقسیم می شود: S (Saturation)، H (Hue) و V (Value).

Hue زاویه‌ای در یک دایره رنگی است که نوع رنگ را مشخص می‌کند. به عنوان مثال، ۰ درجه برای رنگ قرمز، ۱۲۰ درجه برای سبز و ۲۴۰ درجه برای آبی است. Saturation یا اشباع رنگ، میزان خلوص رنگ را نشان می‌دهد که بین ۰ (حاکستری) تا ۱ (رنگ خالص) متغیر است. در نهایت، Value به معنای روشنایی رنگ است که از ۰ (کاملاً سیاه) تا ۱ (کاملاً سفید) تغییر می‌کند.

در فضای HSV، تشخیص رنگ آبی بسیار راحت‌تر از فضای RGB است. در فضای RGB، برای تشخیص هر رنگ باید ترکیب سه رنگ اصلی (قرمز، سبز و آبی) به صورت همزمان بررسی شود که سخت است اما در فضای HSV همان Hue به تهابی به ما می‌گوید که هر پیکسل متعلق به چه رنگی است. به عنوان مثال، برای پیدا کردن رنگ آبی تنها کافی است بازه‌ای از مقادیر Hue را بررسی کنیم که این رنگ را شامل می‌شود؛ که با اندکی جستجو مشخص شد بین ۲۴۰ تا ۳۰۰ درجه است.

- حالا همانطور که در تشخیص اعداد و حروف پلاک ها، اینکه تصویر را باینری کنیم به ما کمک می‌کرد، الان هم در تشخیص رنگ آبی میتوان از همان مفهوم باینری کردن استفاده کرد، یعنی برای هر پیکسل

---

بررسی میکنیم که آیا در محدوده‌ی منسوب به رنگ آبی قرار دارد یا خیر. اگر پیکسل در محدوده رنگ آبی باشد، به آن مقدار 1 (سفید) داده می‌شود و اگر نباشد، مقدار 0 (سیاه) دریافت می‌کند. نتیجه این کار یک تصویر باینری است که فقط بخش‌های آبی تصویر به صورت سفید (با مقدار 1) و بقیه تصویر به صورت سیاه (با مقدار 0) نمایش داده می‌شوند.

- اما استفاده از محدوده‌ی رنگ آبی باز هم آنقدر که باید، دقیق نداشت، نهایتاً نکته‌ای که به ذهنمان رسید این است که در پلاک‌ها، همانطور که اعداد و حروف دارای رنگ مشکی خیلی خالصی هستند و زمینه‌هم رنگ سفید نسبتاً خالصی دارد، رنگ آبی کنار پلاک هم معمولاً هم خلوص بالایی دارد و هم نسبت به نور فلاش و دیگر نور‌ها، دارای بازتاب نسبتاً زیاد و روشنایی زیادی است، پس برای مقدار‌های saturation و value نیز یک مینیمم تعریف کردیم که با حس و آزمایش به مقادیر نهایی آنها رسیدیم و از آنها نیز در فرایند باینری کردن عکس بهره گرفتیم.

- در این میان تلاش کردیم این نکته که این بخش آبی کنار پلاک، دارای شکل مستطیلی است را هم در نظر بگیریم که در طول کدها به آن اشاره میکنیم.

- در نهایت پس از شناسایی ناحیه آبی، با توجه به اینکه طول این ناحیه تقریباً یک یازدهم کل پلاک است، ابعاد کل پلاک را تخمین می‌زنیم. و در نهایت، پلاک را از تصویر اصلی برش داده، نمایش داده و ذخیره می‌کنیم.

---

حالا که روند یافت الگوریتم را فهمیدیم، مرحله به مرحله روی کد جلو می رویم:

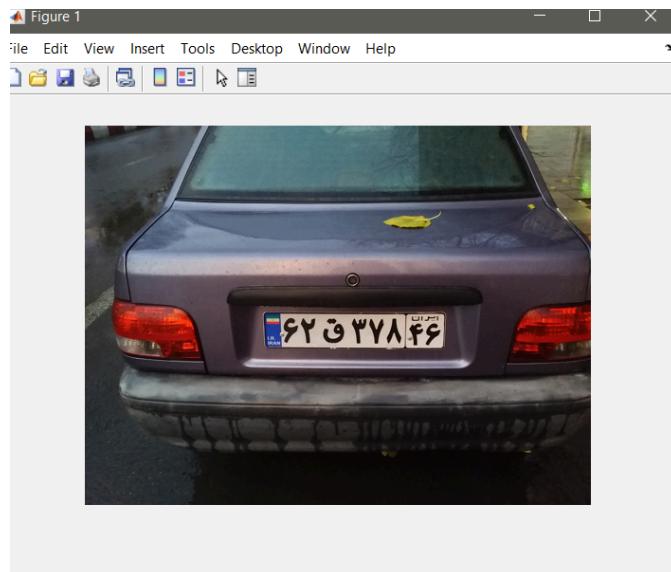
1. در مرحله ای اول همانند بخش های قبلی ابتدا تصویر را از کاربر گرفته و سپس آنرا به یک سایز نرمال رسانده تا اگر اندازه ای عکس خیلی بزرگ یا خیلی کوچک بود، قابل رویت و بررسی شود.

```
clc
close all;
clear;

%% Load the image
[file, path] = uigetfile({'*.jpg;*.png;*.jpeg'}, 'Select an image');
imagePath = fullfile(path, file);
img = imread(imagePath);

%% Resize the image
img = imresize(img, [400, NaN]);
figure, imshow(img);
```

مثل:

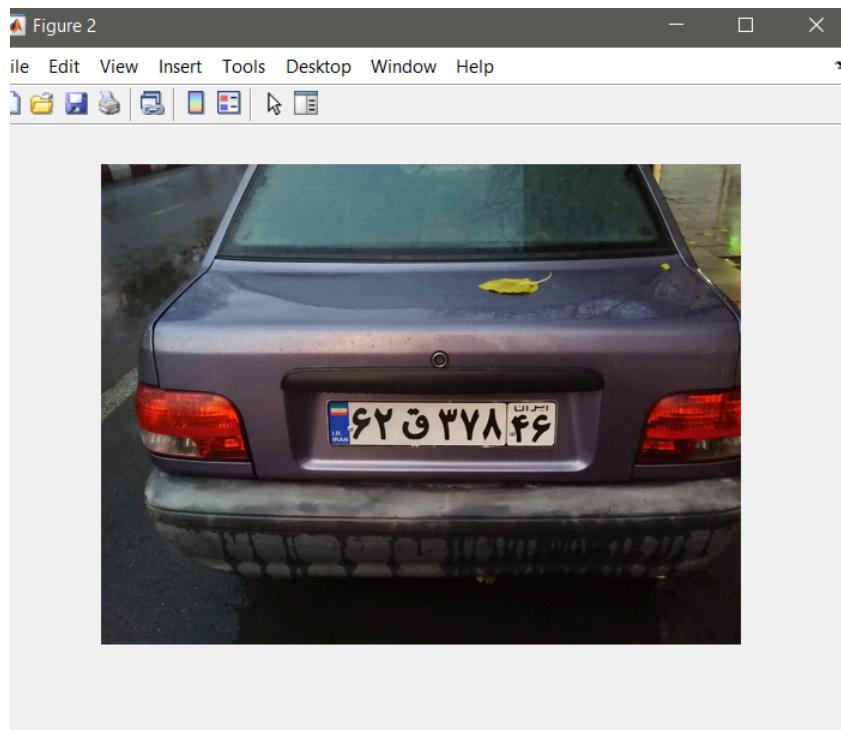


2. در این مرحله بنابر دلایلی که ذکر شد از جمله تشخیص راحتتر یک رنگ خاص مثل آبی روی یک جنس خاص مثل پلاک فلزی که رنگ هایش خلوص و روشنایی بالایی دارند، تصویر را بوسیله ای

تابع `rgb2hsv` به سیستم HSV می برم و آنرا نمایش میدهیم(تفاوتی در نمایش آن بوجود نمی اید که محسوس باشد)

```
%% Convert the image to HSV color space (to detect blue more easily)
hsvImage = rgb2hsv(img);
figure, imshow(img);
```

مثال:



3. حالا یکسری `threshold` که بنابر حدس و آزمایش روی پلاک ها و عکس های مختلف به آن ها رسیدیم را تعریف میکنیم برای `Hue` و `Value` و `Saturation` و سپس با توجه به آنها، به هر پیکسل لیبل 0 (به معنای نداشتن شرایط مورد نیاز و رنگ مشکی) و یا 1 (معنای داشتن شرایط مورد نیاز و رنگ سفید) میزنیم.

```

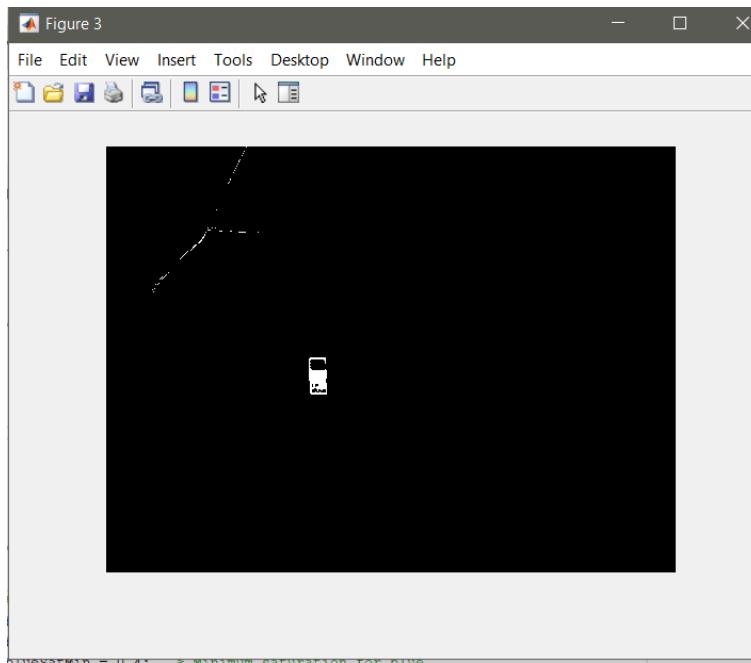
%% Define thresholds for the blue color in HSV
blueHueMin = 0.58; % Minimum hue for blue
blueHueMax = 0.75; % Maximum hue for blue
blueSatMin = 0.4; % Minimum saturation for blue
blueValMin = 0.2; % Minimum value for blue (brightness)

%% Create a binary image
binaryImg = (hsvImage(:,:,:,1) >= blueHueMin) & (hsvImage(:,:,:,1) <= blueHueMax) & ...
            (hsvImage(:,:,:,2) >= blueSatMin) & ...
            (hsvImage(:,:,:,3) >= blueValMin);

figure, imshow(binaryImg);

```

مثلا:

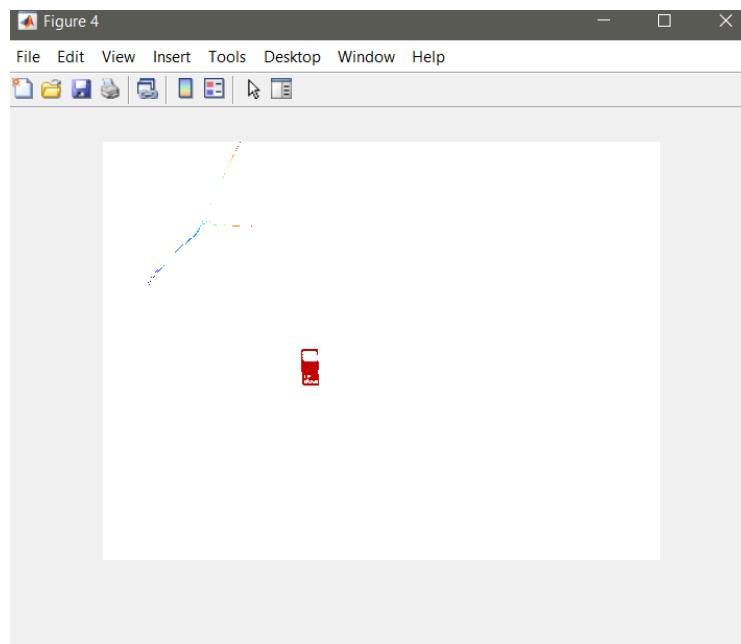


همانطور که حس میزدیم، گوشه‌ی پلاک که رنگ آبی دارد، جزو بزرگترین محدوده هایی است که تمام ویژگی‌های مدنظر را داراست.

4. حالا جهت یافتن بزرگترین component ای که شرایط انتخاب شدن بعنوان گوشه‌ی پلاک را دارد، از تابع `bwlabel` استفاده کرده و کامپوننت‌های مختلف را لیبل میزنیم. روش نمایش اینگونه عکس‌های لیبل دار را هم در بخش‌های قبلی گفتیم.

```
%% Label connected components
[LabeledImg, num] = bwlabel(binaryImg);
figure, imshow(label2rgb(LabeledImg)); % Convert labels to colors for visualization
```

مثال:



5. حالا برای یافتن بزرگترین کامپوننت کافیست یک حلقه `for` زده و هر بار پیکسل های مربوط به یکی از کامپوننت ها را از روی لیبل اش یافته و بوسیله `i` کمترین و بزرگترین سطر و ستون هایی که دارای نقاطی مربوط به آن کامپوننت هستند، جهات طرف آن کامپوننت را بدست می اوریم و در نتیجه می توانیم طول و عرض مستطیل در بردارنده `i` آن کامپوننت را هم بدست بیاوریم که بعدا در یافت ابعاد کل پلاک بدردمان میخورد

سپس با شمردن تعداد اعضای وکتور شماره سطر های نقاطی `i`، در اصل به تعداد پیکسل های آن کامپوننت یا بعارتی مساحت آن کامپوننت میرسیم و میتوانیم به این صورت تا اتمام حلقه `i` ، بزرگترین کامپوننت آبی را بیابیم.

---

```

%% find the largest blue component
maxArea = 0;
blueRegionIdx = 0;
for k = 1:num
    % Find the pixel coordinates of the current component
    [rows, cols] = find(LabeledImg == k);

    % Calculate the bounding box (Bounding Box)
    minRow = min(rows);
    maxRow = max(rows);
    minCol = min(cols);
    maxCol = max(cols);

    % Compute the width and height of the bounding box
    blueWidth = maxCol - minCol + 1;
    blueHeight = maxRow - minRow + 1;

    % Calculate the area of the region (Area)
    area = length(rows); % Number of pixels corresponds to the area

    % Find the largest region (the blue part of the plate)
    if area > maxArea
        maxArea = area;
        blueRegionIdx = k;
        blueEdges = [minCol, minRow, blueWidth, blueHeight];
    end
end

```

6. در این بخش نهایتاً پلاک را می‌یابیم: با توجه به عرض بخش آبی که قبلاً یافته‌ایم و این تقریب که طول پلاک حدوداً 11 برابر عرض آن بخش آبی است، به طول پلاک میرسیم. از طرف دیگر مختصات ضلع‌های چپ، بالا و ارتفاع بخش آبی کنار پلاک، تقریباً همانند کل پلاک است و گوشه‌ی راست را هم که از جمع مختصات گوشه‌ی چپ بعلاوه‌ی طول پلاک که محاسبه کردیم قابل یافتن است.

پس رسمًا تمام اطلاعات لازم برای دانستن محل پلاک را داریم، پس برآحتی تصویر اصلی را برش زده و به پلاک میرسیم

---

```

%% Extract the edges of the plate
% Estimate the right boundary based on the fact that blue part is 1/11 of the plate
blueWidth = bluestatus(3); % Width of the blue region
plateWidth = blueWidth * 11; % The whole plate is approximately 11 times the blue region width

% Define the plate bounding box using the top, bottom, left, and calculated right boundary
plateX = round(bluestatus(1)); % The left boundary is the same as the blue region's left boundary
plateY = round(bluestatus(2)); % The top boundary is the same as the blue region's top boundary
plateHeight = round(bluestatus(4)); % Use the same height as the blue region

% Calculate the right boundary by adding the plate width to the left boundary
plateRightX = plateX + round(plateWidth);

% Manually crop the plate using array indexing (rows and columns)
licensePlate = img(plateY:(plateY + plateHeight - 1), plateX:plateRightX, :); % Crop the region

% Step 10: Display the license plate
figure, imshow(licensePlate);
title('Detected Plate');

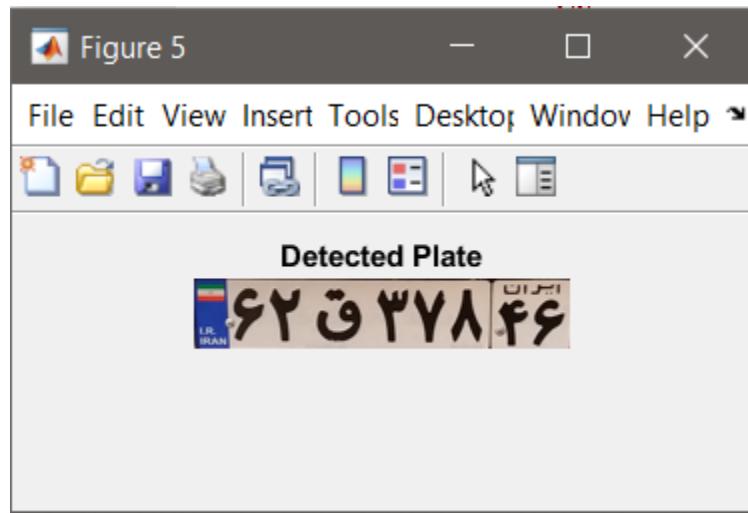
% Save the extracted license plate
imwrite(licensePlate, 'detected_plate.png');

disp(' plate detected and saved successfully.');

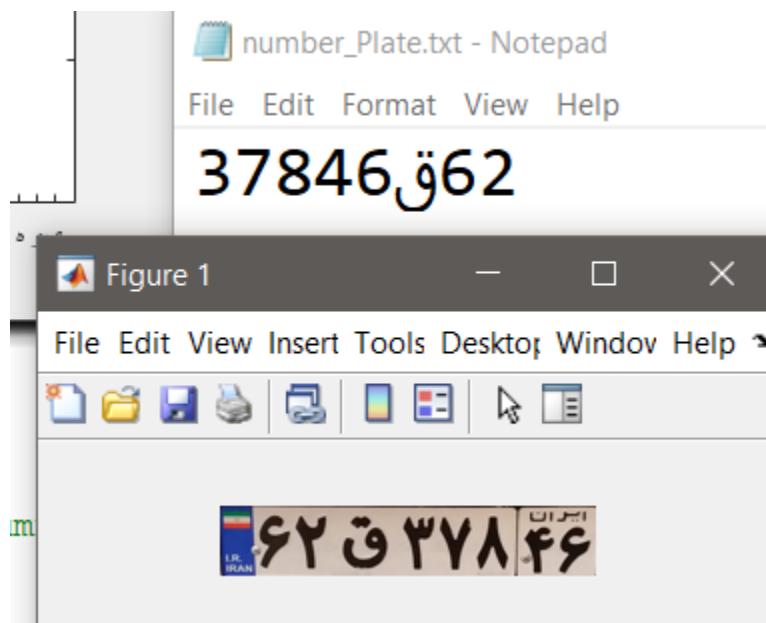
```

---

مثلا:



که دقیقاً شبیه عکس هایی است که میتوان به فایل سوال دوم داده و متن را از آن استخراج کرد. اگر این کار را هم بکنیم داریم:

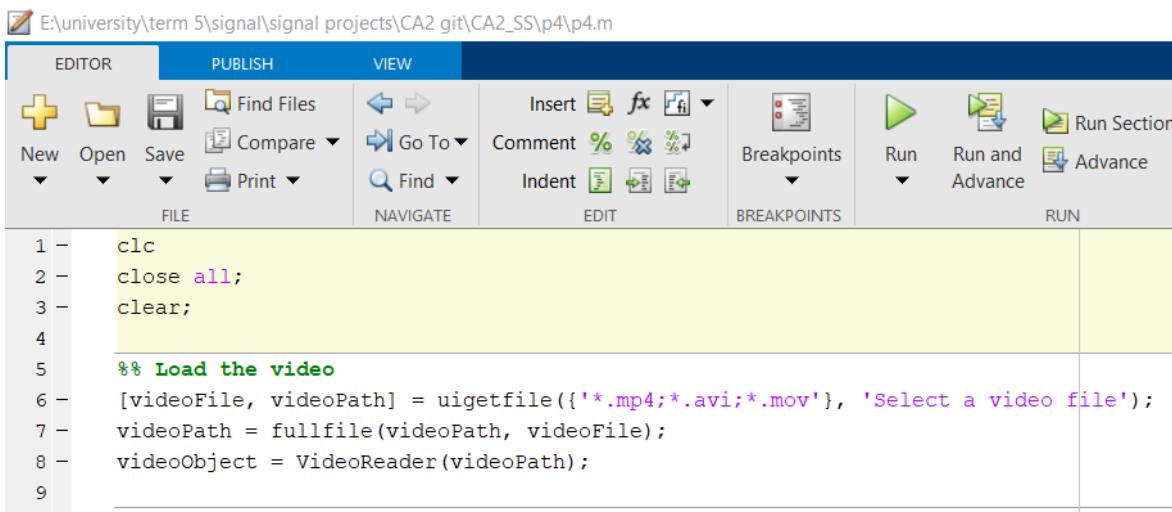


\* همانطور که مشاهده میشود، با درنظر گرفتن جایه جایی کلمات فارسی در متن، دقیقا به جواب درست رسیده و با استفاده از عکس جلوبندی، به اعداد و حروف پلاک رسیدیم.

## بخش چهارم)

در این بخش برای پیدا کردن سرعت خودرو نیاز به یک نقطه مشخص در خودرو داریم. چون در بخش قبل توانستیم پلاک را از یک عکس استخراج کنیم ما وسط پلاک را به عنوان نقطه انتخاب شده روی خودرو انتخاب کردیم.

ایده این است که با توجه به این که دوربین ثابت است، در دو فریم مختلف وسط پلاک را تشخیص دهیم و فاصله آن را بر حسب پیکسل در بیاوریم و در آخر با توجه به مدت زمان بین دو فریم، سرعت ماشین را بر حسب فریم بر ثانیه به دست آوریم.



The screenshot shows the MATLAB Editor window with the following code:

```
E:\university\term 5\signal\signal projects\CA2 git\CA2_SS\p4\p4.m
```

```
1 - clc
2 - close all;
3 - clear;
4 -
5 - %% Load the video
6 - [videoFile, videoPath] = uigetfile({'*.mp4;*.avi;*.mov'}, 'Select a video file');
7 - videoPath = fullfile(videoPath, videoFile);
8 - videoObject = VideoReader(videoPath);
9 -
```

ابتدا ویدیوی مورد نظر را لود می‌کنیم.

```

10 %% Select two different frames randomly
11 -
12 frameNumber1 = 50;
13 -
14 -
15 %% Read both frames as image
16 -
17 frame1 = read(videoObject, frameNumber1);
18 frame2 = read(videoObject, frameNumber2);

```

سپس دو فریم دلخواه که اینجا فریم های 50 و 120 انتخاب شده اند را جدا می کنیم.

```

19 %% Detect the license plate and find the center for both frames using p3 code
20 -
21 plateCenter1 = detectPlateAndCenter(frame1);
22 plateCenter2 = detectPlateAndCenter(frame2);

```

سپس به وسیله تابع `detectPlateAndCenter`، ابتدا پلاک را مانند بخش p3 استخراج کرده و سپس مختصات وسط آن را در می آوریم.

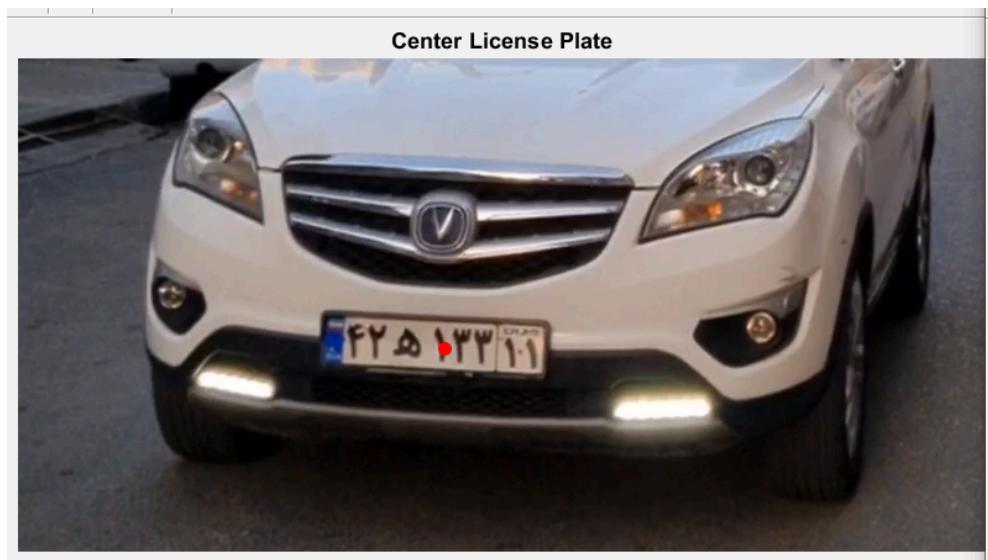
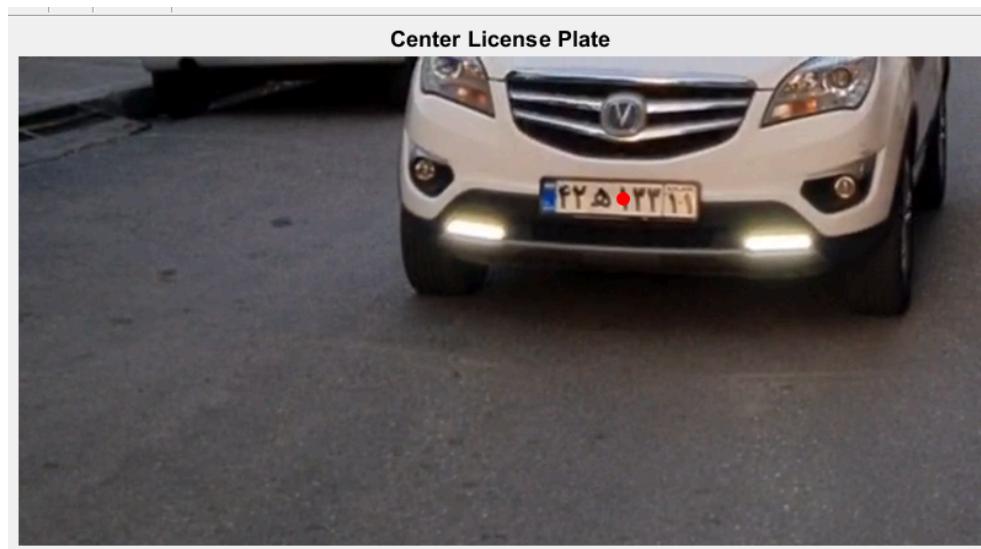
```

95
96 %% Extract the center of the plate
97 if maxArea > 0
98     % Estimate the right boundary based on the fact that blue part is 1/11 of the plate
99     blueWidth = bluestatus(3); % Width of the blue region
100    plateWidth = blueWidth * 11; % The whole plate is approximately 11 times the blue region width
101
102    % Define the plate bounding box using the top, bottom, left, and calculated right boundary
103    plateX = round(bluestatus(1)); % The left boundary is the same as the blue region's left boundary
104    plateY = round(bluestatus(2)); % The top boundary is the same as the blue region's top boundary
105    plateHeight = round(bluestatus(4)); % Use the same height as the blue region
106
107    % Calculate the center pixel
108    plateCenterX = plateX + round(plateWidth) / 2;
109    plateCenterY = plateY + plateHeight / 2;
110    plateCenter = [plateCenterX, plateCenterY];
111
112    % Show the selected pixel in the frame
113    figure, imshow(img);
114    hold on;
115    radius = 5; % Set the radius of the dot that shows the center of plate
116    rectangle('Position', [plateCenterX - radius, plateCenterY - radius, 2*radius, 2*radius], ...
117              'Curvature', [1, 1], 'EdgeColor', 'r', 'FaceColor', 'r');
118    title('Center License Plate');
119    hold off;
120 else
121     plateCenter = []; % If no plate is detected, return empty
122     disp('No plate detected in the frame.');
123 end

```

در واقع بعد از درآوردن مختصات بخش آبی کد به صورت بالا تغییر می‌کند. به این صورت که مختصات plateCenterX و plateCenterY با اضافه کردن به ترتیب نصف طول پلاک و نصف عرض پلاک به plateX و plateY به دست می‌آیند. برای اینکه ببینیم درست تشخیص داده شده، تصویر را دوباره نشان می‌دهیم اما یک نقطه با شعاع 5 در مرکز پلاک محاسبه شده توسط ما قرار می‌دهیم. چون برای دو فریم این را رسم می‌کنیم می‌توانیم مطمئن شویم که کد تا حدودی درست کار می‌کند.

تصویر های نمایش داده شده به صورت زیر می‌باشد.



همانطور که دیده می‌شود مرکز پلاک تقریباً به درستی پیدا شده و در دو فریم روی عدد یک افتاده است و تقریباً در یک نقطه قرار دارند. حال فقط کافی است مختصات به دست آمده را استفاده کنیم تا جا به جایی و سپس سرعت را بیابیم.

```

-- 
24 %% Report the detected plate centers, calculated the distance between them and the speed (pixel/sec)
25 if ~isempty(plateCenter1) && ~isempty(plateCenter2)
26     % Print the centers coordinates
27     fprintf('Center of plate in Frame 1: (%.2f, %.2f)\n', plateCenter1(1), plateCenter1(2));
28     fprintf('Center of plate in Frame 2: (%.2f, %.2f)\n', plateCenter2(1), plateCenter2(2));
29
30     % Calculate the movement of the plate center between the two frames
31     displacementX = plateCenter2(1) - plateCenter1(1); % Horizontal movement
32     displacementY = plateCenter2(2) - plateCenter1(2); % Vertical movement
33
34     % Print the Horizontal and Vertical movement
35     fprintf('Horizontal movement: %.2f pixels\n', displacementX);
36     fprintf('Vertical movement: %.2f pixels\n', displacementY);
37
38     % Calculate speed (pixel/sec)
39     frameRate = videoObject.FrameRate;
40     frameDifference = abs(frameNumber2 - frameNumber1); % Difference in frame numbers
41     timeDifference = frameDifference / frameRate; % Time difference in seconds
42     speed = sqrt(displacementX ^ 2 + displacementY ^ 2) / timeDifference;
43     % Print the calculated speed
44     fprintf('The average speed of the car between frame 50 and 120 is %.2f pixel per second\n', speed);
45 else
46     disp('Failed to detect plate in one or both frames.');
47 end
-- 

```

با استفاده از این بخش از کد، ابتدا مختصات مرکز را گزارش می‌کنیم. سپس فاصله‌های عمودی و افقی بین دو مختصات مرکز را به دست می‌آوریم و گزارش می‌کنیم. در انتها نیز فاصله بین دو فریم را بر حسب ثانیه به دست می‌آوریم و فاصله کلی را بر آن تقسیم می‌کنیم تا سرعت میانگین در بین این دو فریم را محاسبه کرده باشیم. خروجی به صورت زیر خواهد بود. دقت شود که واحد سرعت پیکسل بر ثانیه است.

```

Center of plate in Frame 1: (492.00, 117.00)
Center of plate in Frame 2: (346.00, 236.00)
Horizontal movement: -146.00 pixels
Vertical movement: 119.00 pixels
The average speed of the car between frame 50 and 120 is 161.45 pixel per second

```

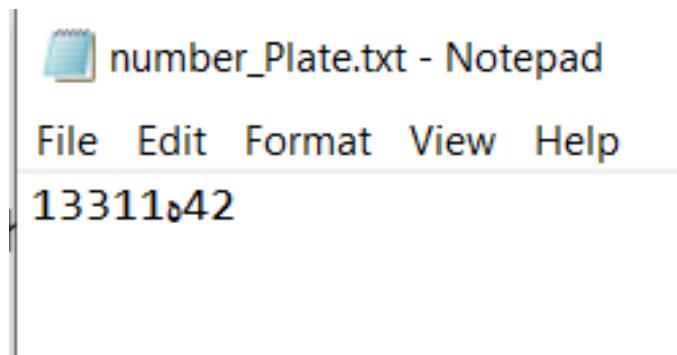
همچنین در آخر کل کد فایل p3 را در فانکشن plate\_reader قرار دادم و با استفاده از فریم دو پلاک را استخراج کردم.

```
%% Find the plate number
```

```
plate_reader(frame2);
```



همانطور که می‌بینید، پلاک به درستی استخراج شده و آبجکت‌های آن نیز به درستی تشخیص داده شده‌اند.



همچنین نتیجه نهایی که در فایل تکست نوشته شده است، درست است و پلاک توسط برنامه به درستی خوانده شده است.