# گزارش پروژه دوم درس سیگنال و سیستم

**کوثر شیری جعفرزاده 810101456**

**پریا پاسه‌ورز 810101393**

## بخش اول

در هر قسمت، خروجی متناظر با هر سه testcase را نمایش می‌دهیم:

1) انتخاب تصویر مورد نظر و لود آن در قالب یک ماتریس:

از دستور uigetfile برای این منظور استفاده کردیم:

کد:

```
% 1.SELECTING THE TEST DATA
[file,path]=uigetfile({'*.jpg;*.bmp;*.png;*.tif'},'Choose an image');
s=[path,file];
picture=imread(s);
figure
imshow(picture)
```

خروجی:

تست‌کیس اول:


تست‌کیس دوم:

تست‌کیس سوم:

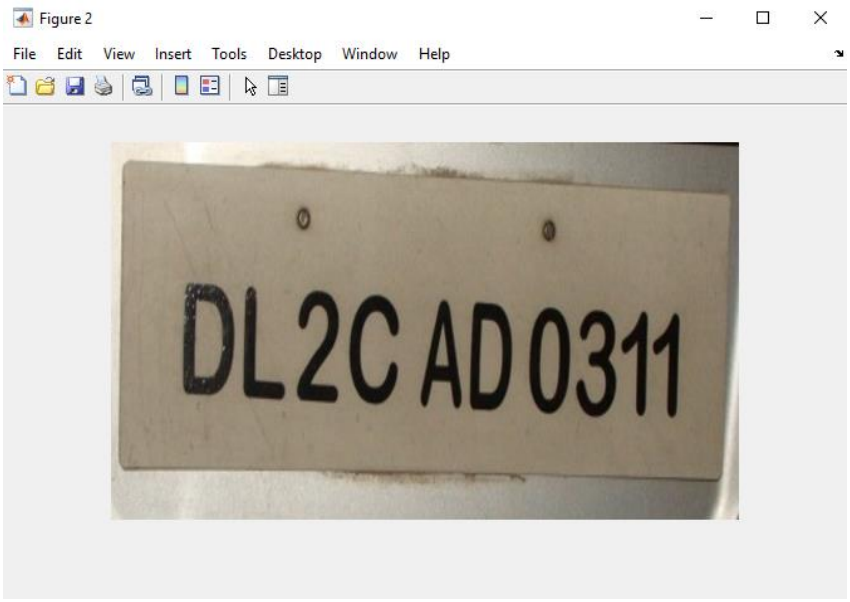

2) تغییر سایز عکس با استفاده از تابع آماده immresize:

کد:

```
% 2.RESIZE THE IMAGE
picture=imresize(picture,[300 500]);
figure
imshow(picture)
```

خروجی تست‌کیس اول:

خروجی تست‌کیس دوم:




خروجی تست‌کیس سوم:

3) سیاه سفید کردن عکس با کمک فرمول زیر:

$$Graychannel = 0.299 \times Redchannel + 0.578 \times Greenchannel + 0.114 \times Bluechannel$$

کد تابع mygrayfun:

```
function gray_scaled_image = mygrayfun(original_image)
Red_channel = original_image(:, :, 1);
Green_channel = original_image(:, :, 2);
Blue_channel = original_image(:, :, 3);

% GRAYSCALE CONVERSION FORMULA
Gray_channel = 0.299 * Red_channel + 0.578 * Green_channel + 0.114 * Blue_channel;

% CONVERT TO UNIT8 TO MATCH IMAGE FORMAT
gray_scaled_image = uint8(Gray_channel);
```

این تابع عکس را به عنوان ورودی می‌گیرد و در مرحله اول کانالهای سبز و قرمز و آبی آن را جدا می‌کند. سپس با کمک رابطه داده شده gray channel محاسبه شده است.

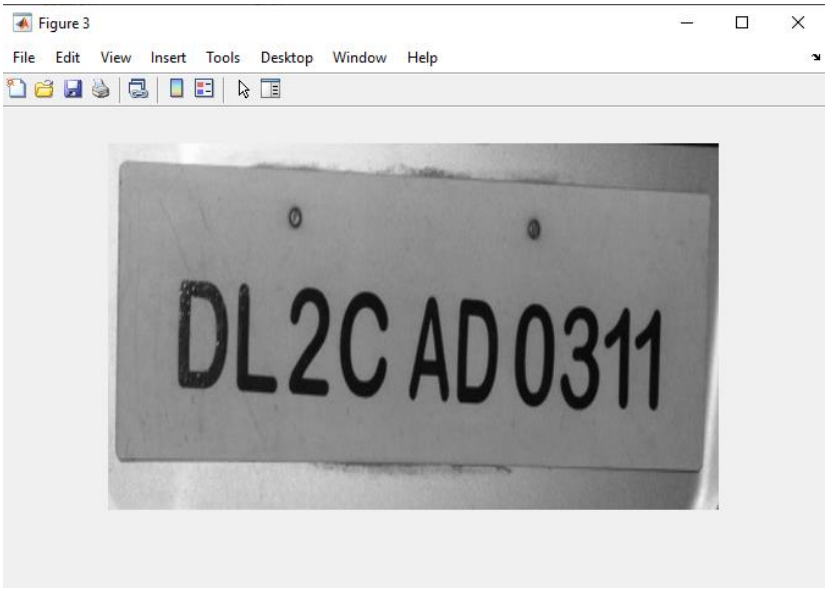
دقت کنید که مقدار هر پیکسل عددی بین 0 تا 255 است، پس برای اینکه ممکن شویم اعداد متناظر با هر پیکسل در gray chnannel در این بازه قرار دارند، ازتابع unit8 استفاده می‌کنیم.

کد:

```
% 3.CREATE BLACK & WHITE IMAGE
gray_scaled_image=mygrayfun(picture);
figure
imshow(gray_scaled_image)
```

خروجی تست کیس اول:

خروجی تست کیس دوم:




خروجی تست کیس سوم:

4) سیاه سفید کردن کامل عکس (باینری کردن عکس):

کد تابع mybinaryfun:

```
function binary_image = mybinaryfun(original_image)

% SET THRESHOLD VALUE
threshold = graythresh(original_image);

% MAKE IMAGE IN BINARY FORMAT
binary_image = imbinarize(original_image,threshold);

% MAKE COLOURS SIMILAR TO MAPSET
binary_image = ~binary_image;
```

این تابع عکس را به عنوان ورودی می‌گیرد و با کمک تابع graythresh، یک threshold برای سیاه سفید کردن عکس تعیین کردیم. سپس با کمک تابع imbinarize، تصویر باینری را تولید کردیم.

برای انطباق تصویر تولید شده با تصاویر map set موجود، عکس را complement کردیم تا اعداد و کادر سفید باشند و بقیه سیاه باشد.

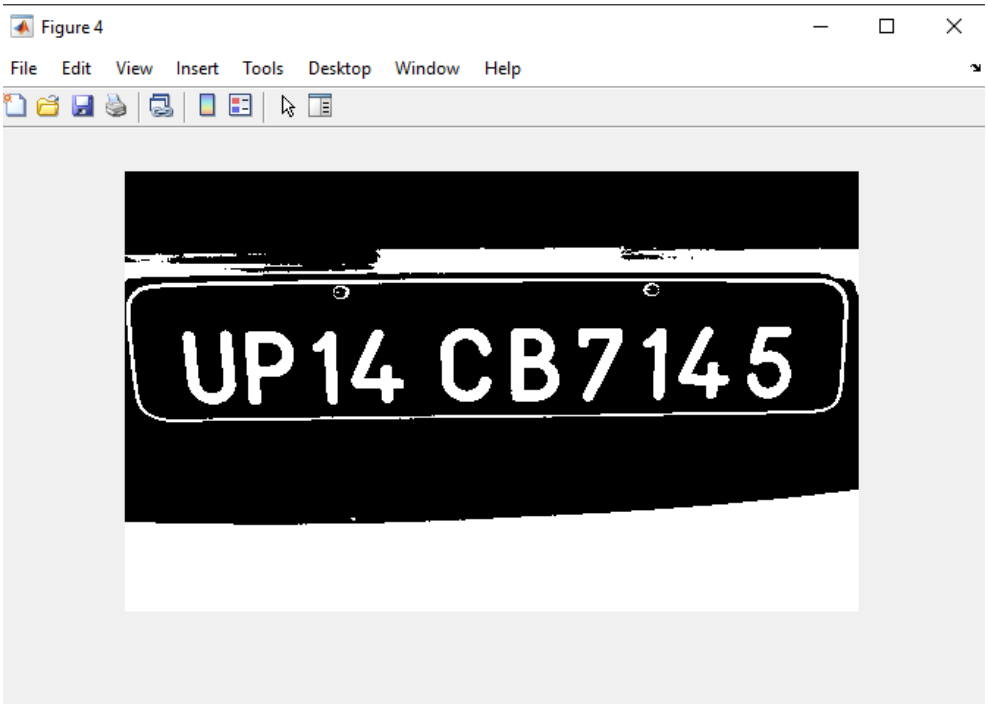
کد:

```
% 4.CREATE BINARY IMAGE
binary_image=mybinaryfun(gray_scaled_image);
figure
imshow(binary_image)
```

خروجی تست کیس اول:

خروجی تست کیس دوم:


خروجی تست کیس سوم:



5.1) حذف نویزهای کوچک:

کد تابع myremovecom:

```
function filtered_picture = myremovecom(binary_picture, threshold)
    % Get the coordinates of the '1's in the binary picture
    [row, col] = find(binary_picture == 1);
    POINTS = [row'; col'];
    POINTS_NUM = size(POINTS, 2);
```

در بخش اول، بخشهایی که در عکس یک بودند، که شامل اعداد و حروف و نویز می‌شود جدا کردیم و سطر و ستون و تعدادشان را مشخص کردیم.

```
% Initialize
if POINTS_NUM == 0
    filtered_picture = binary_picture; % If no points, return the original picture
    return;
end

initpoint = POINTS(:, 1);
POINTS(:, 1) = [];
POINTS_NUM = POINTS_NUM - 1;
CurrectObject = [initpoint];
t = 1;
FINALOBJECT = {};  % Store connected objects
```

اگر چنین چیزی در در عکس وجود نداشت، return می‌کنیم. در غیر این صورت برای پیدا کردن لیست objectهای متصل به هم، یک لیست از current object می‌سازیم و آن را با نقطه اول مقدار دهی اولیه می‌کنیم.

```
% Find all connected objects
while POINTS_NUM > 0
    [POINTS, newPoints] = close_points(initpoint, POINTS);
    newPoints_len = size(newPoints, 2);
    CurrectObject = [CurrectObject newPoints];

    while newPoints_len > 0
        initpoint = newPoints(:, 1);
        newPoints(:, 1) = [];
        [POINTS, newPoints2] = close_points(initpoint, POINTS);
        CurrectObject = [CurrectObject newPoints2];
        newPoints = [newPoints newPoints2];
        newPoints_len = size(newPoints, 2);
    end

    % Only add to FINALOBJECT if it has more than 'threshold' pixels
    if size(CurrectObject, 2) >= threshold
        FINALOBJECT{t} = CurrectObject;
        t = t + 1;
    end

    POINTS_NUM = size(POINTS, 2);
    if POINTS_NUM > 0
        initpoint = POINTS(:, 1);
        CurrectObject = initpoint;
    end
end
```

در این بخش می‌خواهیم Objectهای متصل به هم را پیدا کنیم. ابتدا با کمک تابع close point که آن را در ادامه معرفی می‌کنیم، یک لیست new points از نقاط نزدیک به Init point برمی‌گردانیم. علاوه بر آن، این نقاط را نیز از لیست اولیه نقاط حذف می‌کنیم( لیست POINTS).

نقاط جدید را که به نقطه اولیه‌مان نزدیک بودند، به لیست current object اضافه می‌کنیم.

مجددا همین کار را برای نقاط جدید اضافه شده تکرار می‌کنیم. یعنی سعی می‌کنیم نقاط نزدیک به آنها را نبز پیدا کنیم. آنقدر این کار را می‌کنیم تا تمام نقاظ نزدیک بررسی شوند.

دقت کنید این تابع اجزایی با سایزی بزرگ‌تر از یک threshold خاص را به عنوان نویز در نظر می‌گیرد. پس اگر سایز current object از این threshold بیشتر بود، آن را به لیست objectهای نهایی‌مان اضافه می‌کنیم.

در انتهای هر مرحله و پس از بررسی کامل یک Initipoint، باید مقدار آن را با point سر لیست POINTS آپدیت کرده و به تبع آن current object را نیز آپدیت کنیم.

```
    % Create the output image with only the filtered objects
    filtered_picture = zeros(size(binary_picture));  % Initialize the output image
    for i = 1:length(FINALOBJECT)
        object_points = FINALOBJECT{i};
        for j = 1:size(object_points, 2)
            filtered_picture(object_points(1, j), object_points(2, j)) = 1;
        end
    end
end
```

در نهایت نیز با کمک لیست FINALOBJECT ای که تشکیل داده‌ایم، filtered picture را می‌سازیم. برای این کار در عکس تمام نقاط متناظر با هر عضو final object را یک می‌کنیم.

کد تابع close_points:

```
function [POINTS,newPoints]=close_points(initpoint,POINTS)

POINTS_NUM=size(POINTS,2);
DIF=repmat(initpoint,1,POINTS_NUM)-POINTS;
DIF=abs(DIF);
ind=find(DIF(1,:)<=1 & DIF(2,:)<=1);
newPoints=POINTS(:,ind);
POINTS(:,ind)=[];
end
```

در این تابع ابتدا تعداد نقاط موجود در POINTS را پیدا می‌کنیم. سپس با کمک تابع repmat، به تعداد نقاط موجود نمونه از Initipoint می‌سازیم و و جفت جفت اختلافشان را محاسبه می‌کنیم.

دقت کنید در اینجا فاصله مهم است و علامت آن مهم نیست، پس قدرمطلقش را در نظر می‌گیریم.

اینجا تعریف ما از نقاط نزدیک نقاطی است که حداکثر در یک سطر و ستون اختلاف داشته باشند. با در نظر گرفتن این تعریف، تمام Indexهایی که این ویژگی را دارند جدا می‌کنیم. نقاط با این ویژگی نقاط نزدیک ما خواهند بود.(newpoints)

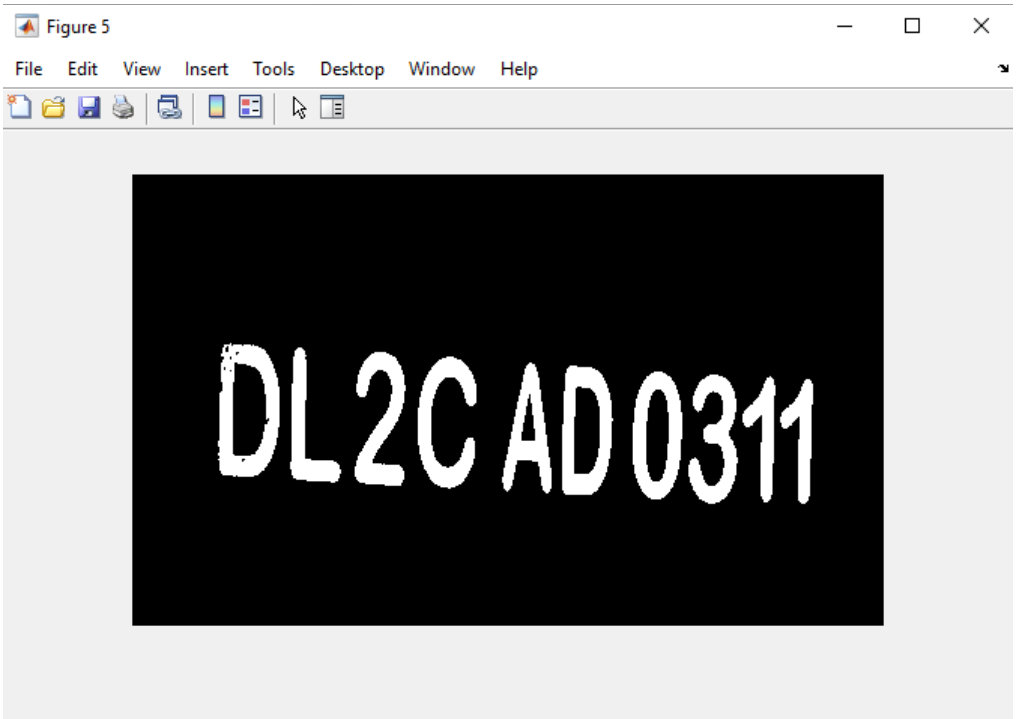
چون این نقاط بررسی شده‌اند، آنها را از لیست اولیه نقاط حذف می‌کنیم و بر می‌گردانیم.

کد:

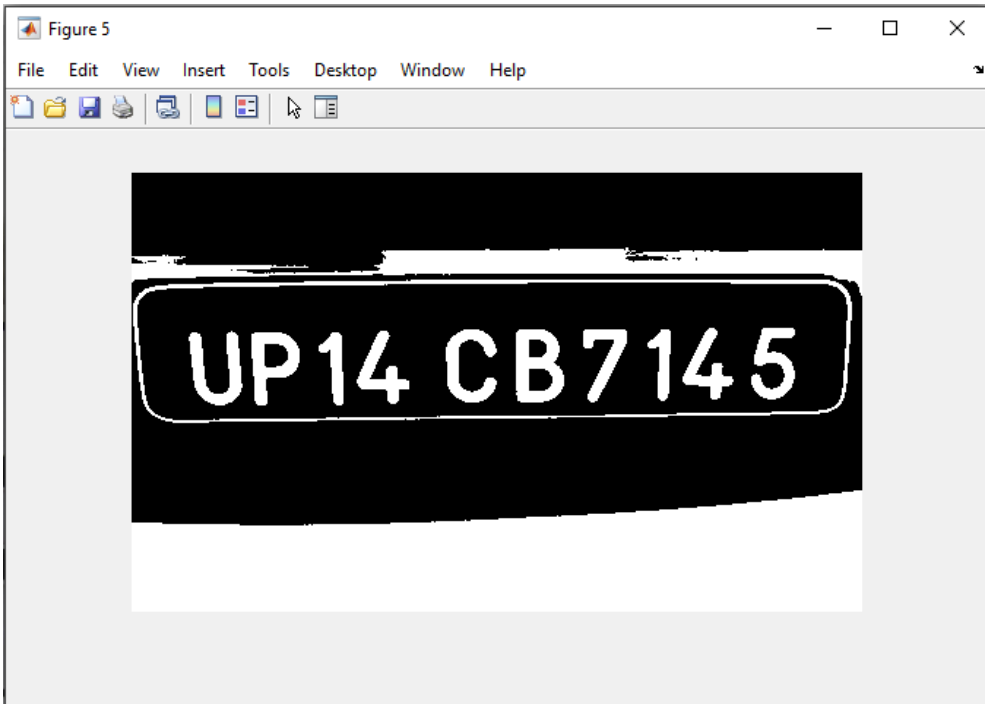
```
% 5.1 CLEAR SMALL NOISES
new_binary_image=myremovecom(binary_image, 400);
figure
imshow(new_binary_image)
```

خروجی تست کیس اول:


خروجی تست کیس دوم:



خروجی تست کیس سوم:



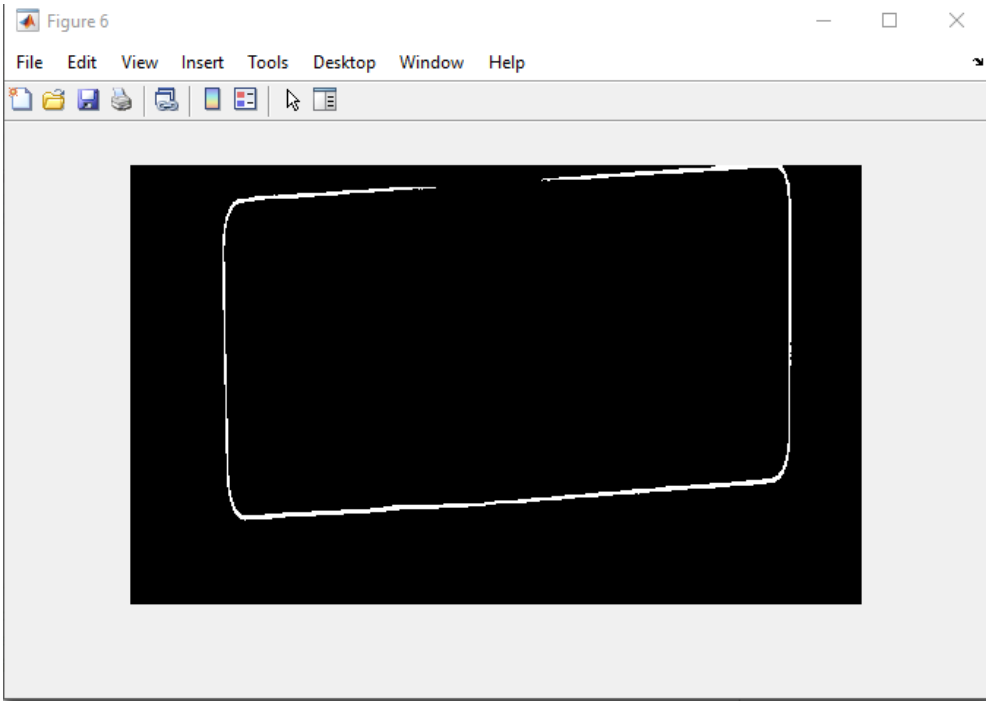

5.2) حذف کادر دور پلاکها:

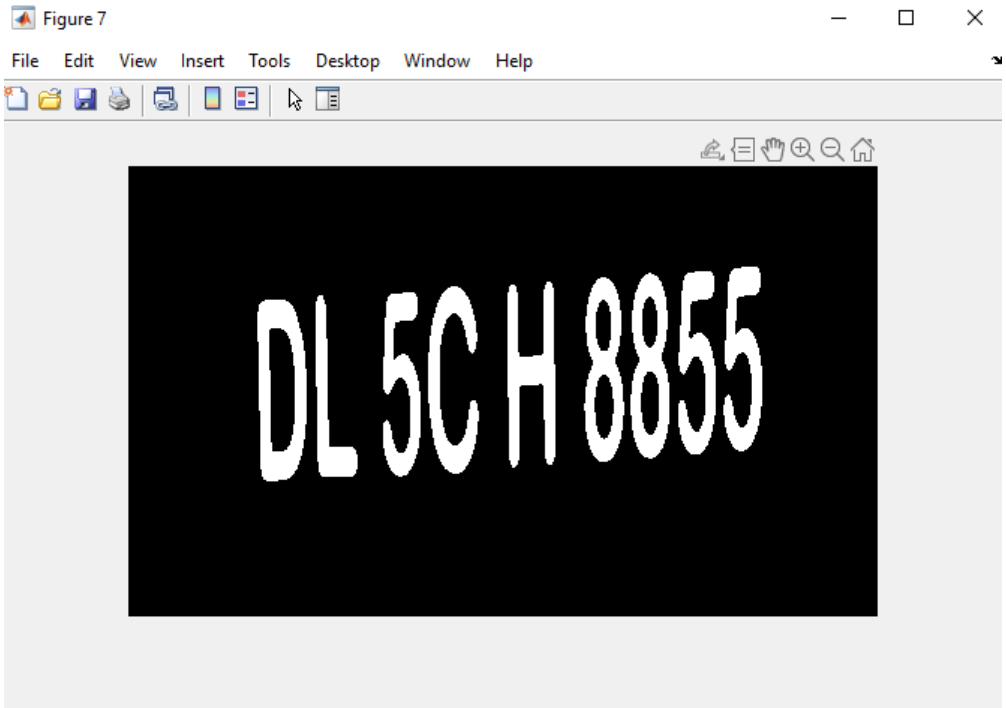
کد:

```
% 5.2 REMOVE BACKGROUND
back_ground = myremovecom(new_binary_image, 2500);
figure
imshow(back_ground)
clear_picture = new_binary_image-back_ground;
figure
imshow(clear_picture)
```

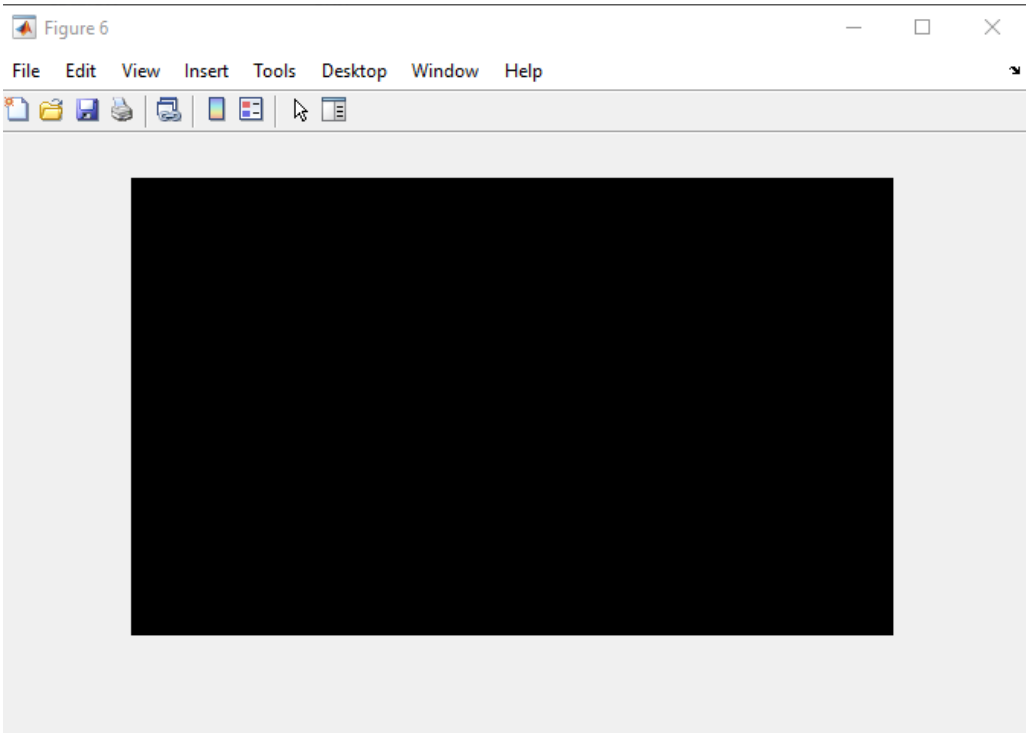
برای حذف کادر دور عکسها، چون این کادر تعداد زیادی پیکسل دارد، با استفاده از تابع myremovecom ابتدا کادر را تشخیص داده، سپس آن را از عکس اصلی حذف کردیم.

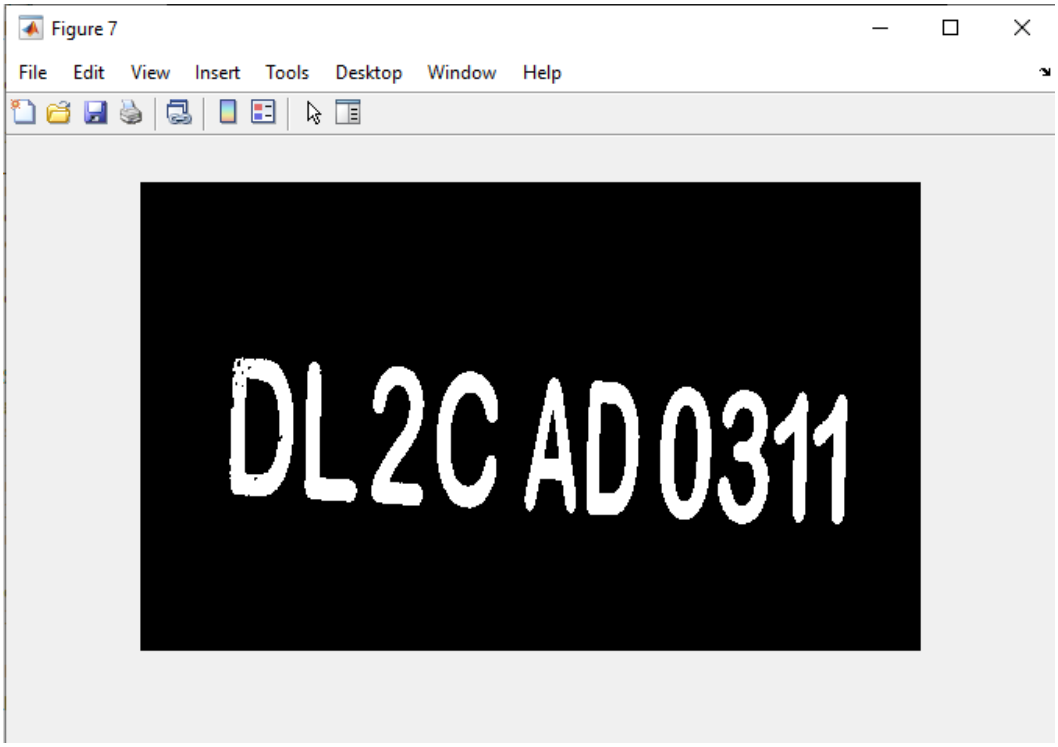
خروجی تست کیس اول:



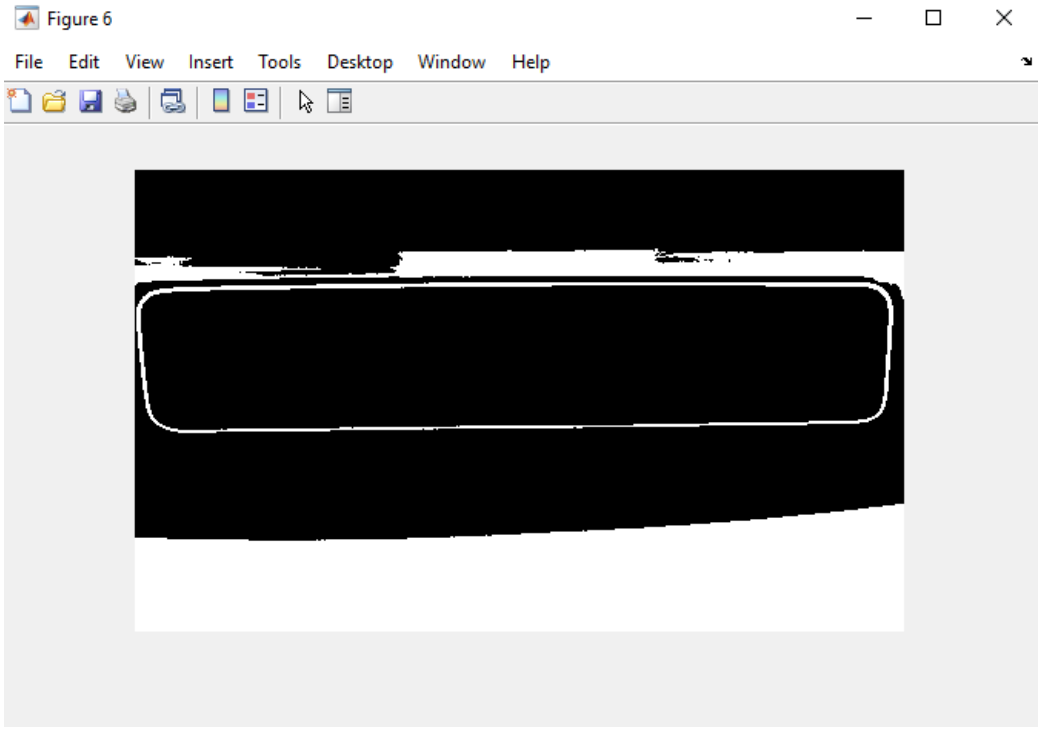



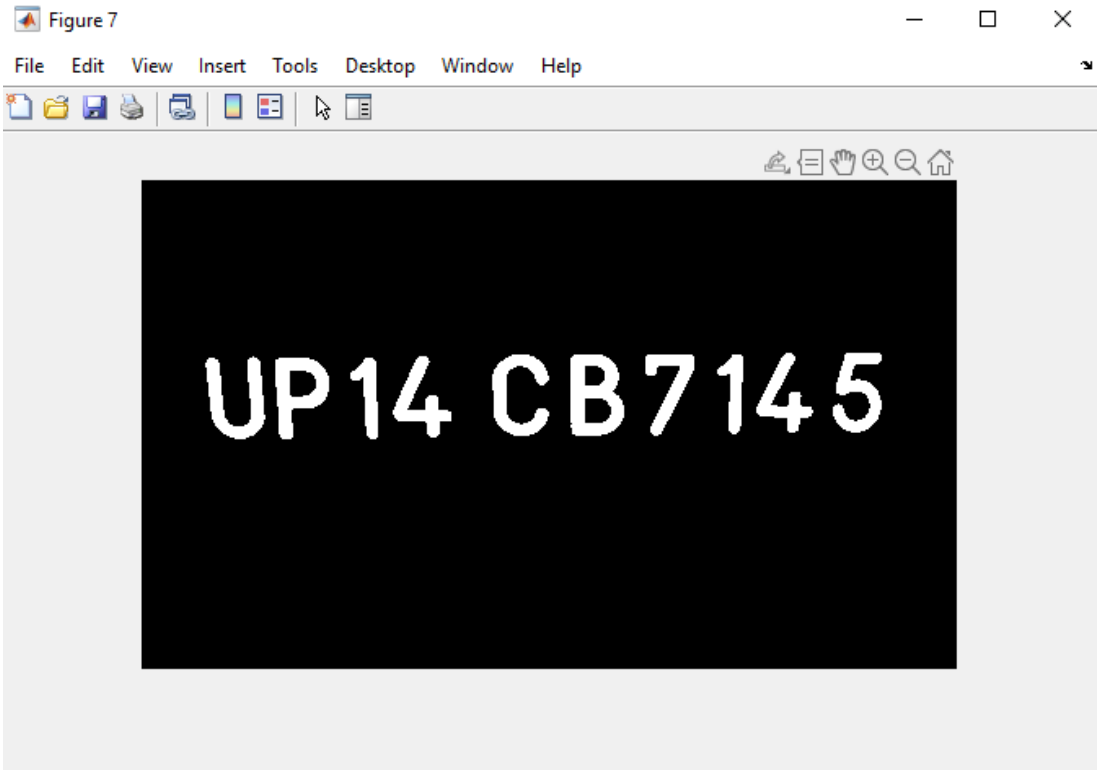
خروجی تست کیس دوم:



خروجی تست کیس سوم:



6. segment بندی عکس:

تابع mysegmentation :

```
function [L, num] = mysegmentation(BW)
    % Input: BW is a binary image (2D matrix) with 0's and 1's.
    % Output: L is the labeled image, num is the number of connected components.

    [rows, cols] = size(BW);       % Get the size of the input image
    L = zeros(rows, cols);         % Initialize the label matrix with zeros
    label = 0;                     % Initial label (no components yet)

    % Define the 4-connected neighborhood (up, down, left, right)
    neighbors = [-1, 0; 1, 0; 0, -1; 0, 1];
```

این تابع یک عکس باینری دریافت می‌کند و یک لیست متناظر با لیبلها و تعدا segmentهای detect را برمی‌گرداند.

اول لیست لیبلها را با صفر مقداردهی اولیه می‌کند و تعداد سطر و ستون شکل اولیه را مشخص می‌کند.

یک لیست از مختصات نسبی همسایه‌ها نیز تعریف می‌شود که از آنها در ادامه استفاده می‌کنیم.

```
    % Main loop to scan through the binary image
    for r = 1:rows
        for c = 1:cols
            % If the pixel is a foreground pixel (1) and hasn't been labeled
            if BW(r, c) == 1 && L(r, c) == 0
                % Increment the label count (new connected component)
                label = label + 1;
                % Perform DFS to label the entire connected component
                DFS(r, c, label);
            end
        end
    end
```

حال روی کل عکس می‌چرخیم و هر قطعه را عددگذاری می‌کنیم. پس در هر خانه عکس چک می‌کنیم آیا مقدار آن در عکس اولیه یک است؟(یعنی عدد یا حرف است؟) و آیا در لیست نهایی عدد گذاری شده است؟ اگر شرط اول درست و شرط دوم غلط بود، یکی به تعداد اجزای یافت شده اضافه می‌کنیم و با کمک تابع DFS که در ادامه آن را معرفی می‌کنیم، تمام نقاط وصل در یک component را پیدا می‌کنیم و مقدار label را به آنها assign می‌کنیم.

```
% Function to perform depth-first search (DFS) to label a connected component
function DFS(r, c, current_label)
    stack = [r, c];  % Initialize a stack for DFS
    while ~isempty(stack)
        % Pop the last element from the stack
        [row, col] = deal(stack(end, 1), stack(end, 2));
        stack(end, :) = [];  % Remove from stack

        % Check if the current pixel is within the bounds and has not been labeled
        if row >= 1 && row <= rows && col >= 1 && col <= cols && BW(row, col) == 1 && L(row, col) == 0
            % Label the current pixel
            L(row, col) = current_label;

            % Add the neighbors (up, down, left, right) to the stack
            for i = 1:size(neighbors, 1)
                newRow = row + neighbors(i, 1);
                newCol = col + neighbors(i, 2);
                stack = [stack; newRow, newCol];  % Push onto the stack
            end
        end
    end
end
```

این تابع دقیقا مشابه الگوریتم DFS که با آن آشنا هستیم عمل می‌کند. یعنی یک stack می‌سازیم که با سطر و ستون نقطه اولیه مقدار دهی شده است و مادامی که خالی نباشد، یکی از سرش برمی‌داریم، اگر مقدارش در عکس اصلی یک بود و در محدوده مورد نظرما قرار داشت، مقدارش را برابر label قرار می‌دهیم و سطر و ستون متناظر با همسایه‌های آن را به عنوان یک جفت جدید به stack اضافه می‌کنیم.

```
    % Output the number of connected components
    num = label;
end
```

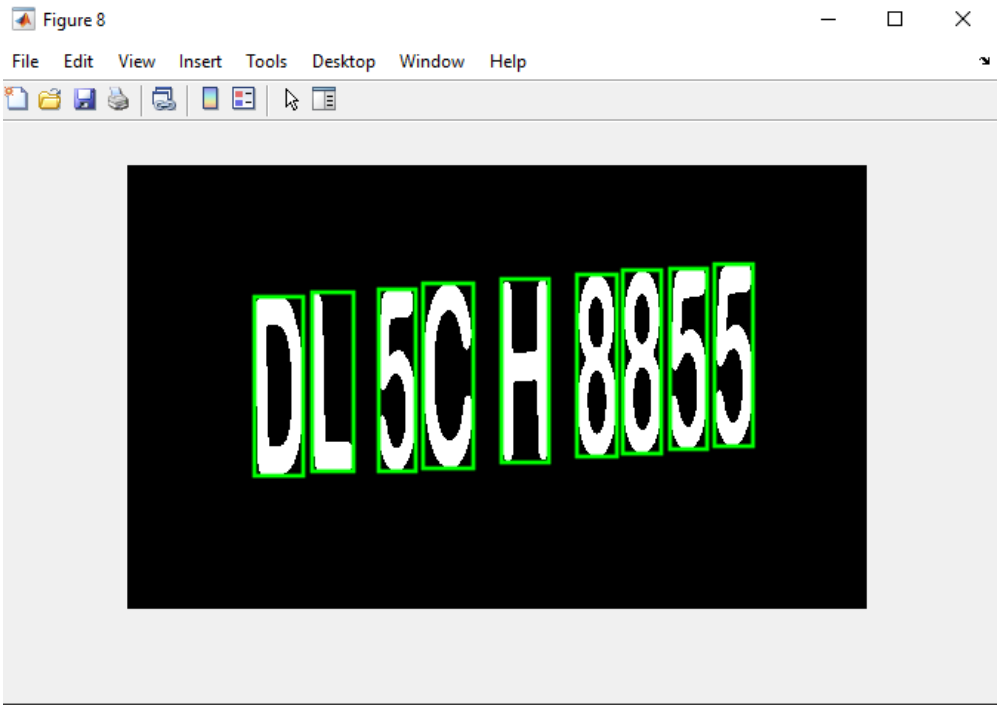
در انتها تعداد segmentهای مشخص شده را ست می‌کنیم.

کد:

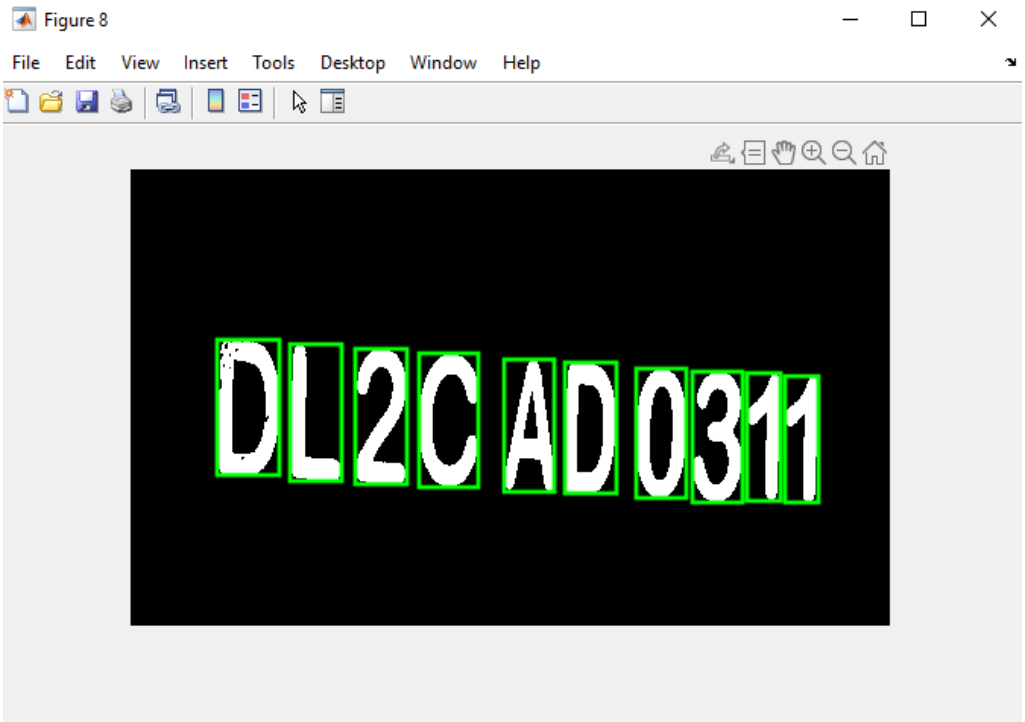
```
% 6. DO SEGMENTATION
figure
imshow(clear_picture)
[L,Nseg]=mysegmentation(clear_picture);
Cordinate=regionprops(L,'BoundingBox');
hold on
for n=1:Nseg
    rectangle('Position',Cordinate(n).BoundingBox,'EdgeColor','g','LineWidth',2)
end
hold off
```

نهاینا با کمک لیست label شده عکس و تابع regiogroups، مختصات هر segment را به دست آورده و مستطیل‌های سبز دور هر segment رسم می‌کنیم.

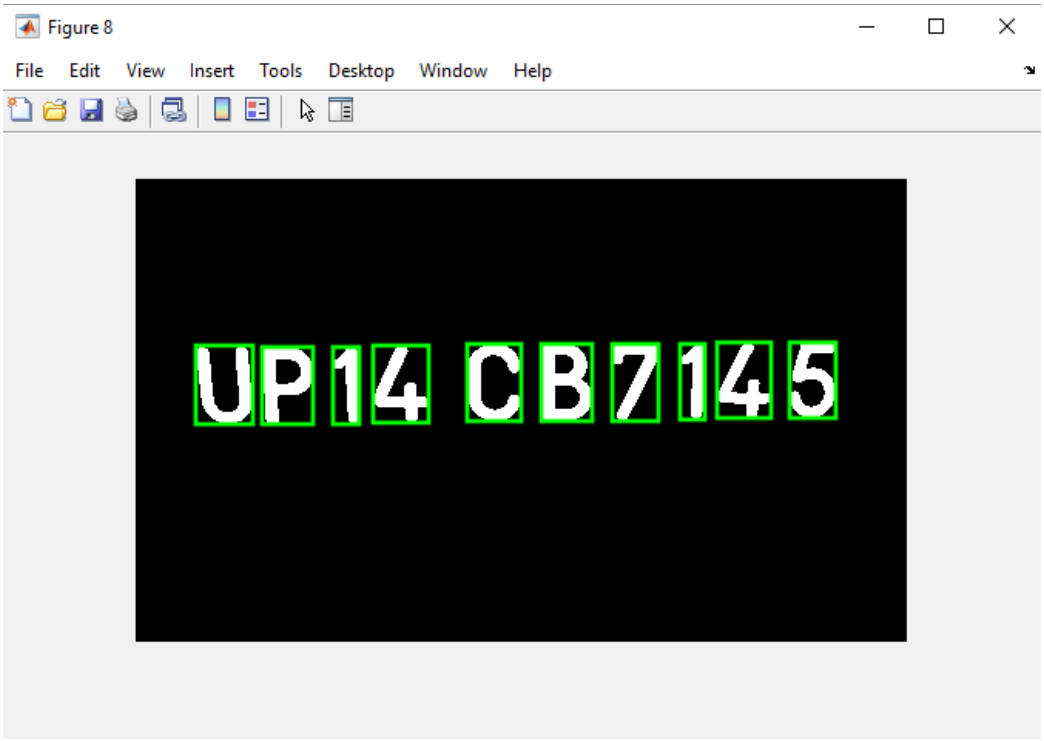
خروجی تست کیس اول:




خروجی تست کیس دوم:



خروجی تست کیس سوم:




مشکلی که در این روش وجود دارد این است که segment با label یک، لزوما اولین عدد موجود در پلاک ماشین نیست و این ترتیب ممکن است به هم خورده باشد. برای رفع این مشکل، x position ها را sort می‌کنیم و به بعدا به همین ترتیب، decision making را انجام می‌دهیم.

```
% Extract x-coordinates (BoundingBox(1)) for sorting
x_positions = arrayfun(@(x) x.BoundingBox(1), Cordinate);

% Sort based on the x-coordinate (left-to-right order)
[~, sortIdx] = sort(x_positions);
```

7. لودکردن mapset:

کد تابع mapset_loading:

```
clc;
clear;
close all;

files=dir('MapSet');
len=length(files)-2;
TRAIN=cell(2,len);

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

save TRAININGSET TRAIN;
```

با کمک این تابع یک cell دو بعدی می‌سازیم و در هر ستون، سطر اول عکس و سطر دوم اسم آن را نمایش می‌دهد.

کد:

```
% LOAD MAPSET
load TRAININGSET;
totalLetters=size(TRAIN,2);
```

8. decision making:

```
%7. DECISION MAKING
figure
final_output=[];
t=[];

for n=1:Nseg
    idx = sortIdx(n);  % Get sorted index
    [r,c]=find(L==idx);
    Y=clear_picture(min(r):max(r),min(c):max(c));
    imshow(Y)
    Y=imresize(Y,[42,24]);
    imshow(Y)
    pause(0.5)

    ro=zeros(1,totalLetters);

    for k=1:totalLetters
        ro(k)=corr2(TRAIN{1,k},Y);
    end

    [MAXRO,pos]=max(ro);
    if MAXRO> 0.45
        out=TRAIN{2,pos};
        final_output=[final_output out];
    end
end
```

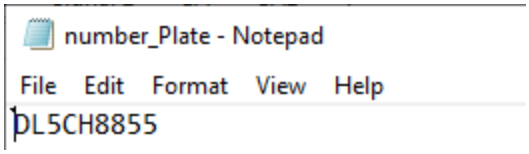
حال در این مرحله برای هر segment، سطر و ستون آن را از لیست labelها پیدا می‌کنیم و هر segment یافت شده را رسم می‌کنیم. سپس correlation را با عکسهای موجود در mapset محاسبه می‌کنیم. عکسی که بیشترین مقدار correlation را داشته باشد یعنی بیتشرین شباهت را دارد و عکس مورد نظر ماست. اسم آن را در یک لیست ذخیره می‌کنیم و سراغ segment بعدی می‌رویم.

9. نوشتن در فایل:

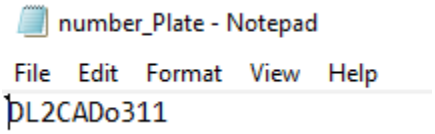
کد:

```
% 8. WRITE DETECTED PLATE NUMBER IN A FILE
file = fopen('number_Plate.txt', 'wt');
fprintf(file,'%s\n',final_output);
fclose(file);
winopen('number_Plate.txt')
```

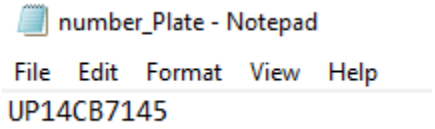
خروجی تست کیس اول:



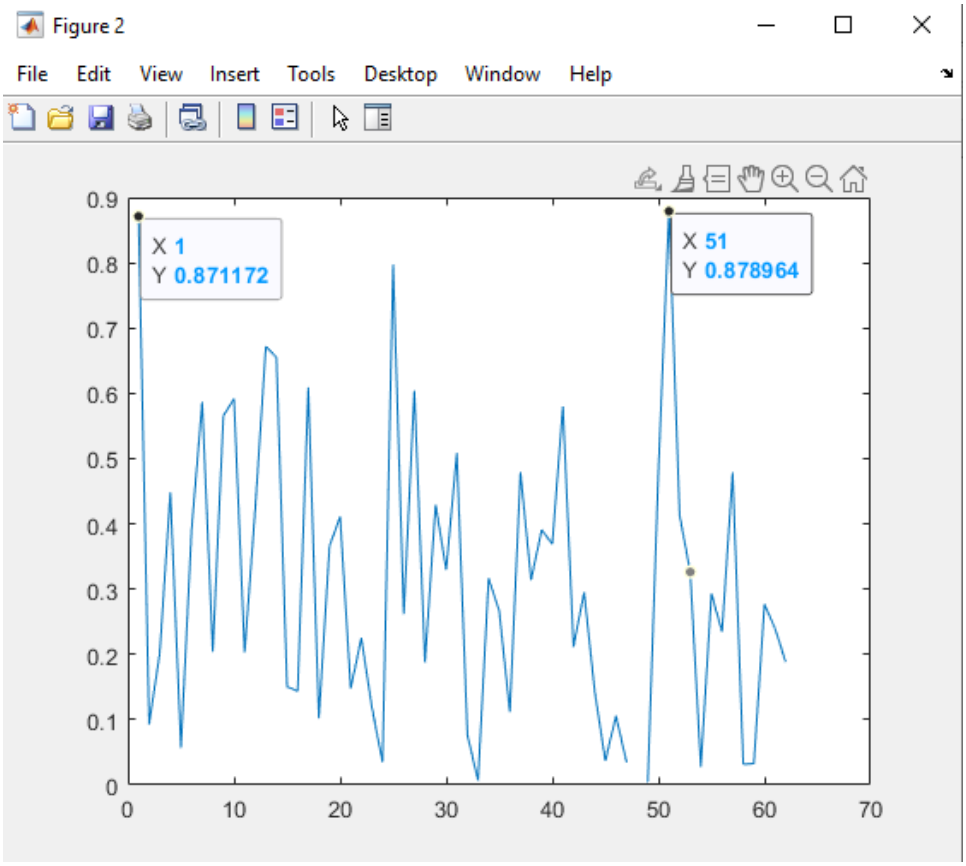

خروجی تست کیس دوم:




خروجی تست کیس سوم:




دقت کنید که در تست کیس شماره دو، 0 اشتباها o detect می‌شود. حال نمودار correlation بین 0 و o را رسم می‌کنیم تا تفاوت را مشاهده کنیم:




همانطور که می‌بینیم، تفاوت بین peakها بسیار اندک است و در اردر 0.07 است، به همین دلیل تشخیص بین 0 و o امکان پذیر نیست.

## بخش دوم

در ابتدا اسکریپت p2.m را شرح می دهیم:

قسمت 1) در ابتدا توسط دستور uigetfile فایل تصویر را از کاربر دریافت می کنیم و سپس تغییرات سایز را برروی آن اعمال می کنیم. ( در این بخش جهت نشان دادن پلاک اولیه آن را نمایش می دهیم).

قسمت 2)در این قسمت با استفاده از دستور rgb2gray شکل مورد نظر را از 3 بعدی (Red,Green,Blue) به Gray scale تغییر می دهیم که یعنی هر پیکسل تصویر عددی بین 0-255 را می گیرد.

حال در قدم بعدی از تابع graythresh استفاده می کنیم این کار به این دلیل است که مرز بین مشکی و سفید را تعیین کنیم ( ممکن بود بدون استفاده از این تابع مرز را عدد 127 تعیین کنیم که هرچه کمتر از این مقدار باشد به سیاه و هرچه بیشتر از آن باشد به سفید تغییر پیدا کند این تابع با در نظر گرفتن مقادیر موجود در پیکسل ها بهترین مقدار را به عنوان threshold تعیین می کند.)

در قدم بعدی از imbinarize استفاده می کنیم و هر مقداری از پیکس های تصویر که کمتر از عدد thresholdداده شده است را به سیاه و هرچه بیشتر از این مقدار است را به سفید مقدار دهی می کنیم) بعد از اینکار مقادیر 0و1 را قرینه می کنیم.

در اینجا باید تصویر را به ساده ترین حالت ممکن تبدیل کنیم تا بتوان از آن پلاک را استخراج کرد بدین منظور تمامی بخش هایی که تعداد پیکسل آن ها کمتر از 500 می باشد را حذف می کنیم. همچنیم قاب پلاک با این حرکت از بین نمی رود پس نیاز است در قدم بعدی تمامی پلاک را حذف کرده تا قاب پلاک به دست بیاید و سپس با کم کردن این قاب از عکس تنها پلاک ماشین را به دست آورد  بنابراین تصویر picture2 تصویر پلاک ماشین است که می توانیم از این پس از آن استفاده کنیم.

قسمت 3)حال در این قسمت باید pixel هایی را که با هم تشکیل یک component می دهند شناسایی کنیم از تابع bwlabel برای این کار استفاده می کنیم این تابع تعداد object های متصل را برمی گرداند و همچنین مشخصات این object هارا نیز به ما می دهد حالا باید مشخصات این بخش هارا به تابع regionprops بدهیم تا براساس این اطلاعات برای ما تصویر را section بندی کند.

این تابع با پارامتر مشخص شده 'BoundingBox' کوچک ترین مستطیلی که object مشخص شده را پوشش دهد انتخاب می کند و ما این مستطیل را در متغییر Cordinate ذخیره می کنیم که با چاپ کردن این مستطیل ها برروی تصویر اصلی نحوه section بندی را تعیین می کند که برای صحت از درستی چاپ می شود ( در این مرحله تمامی سگمنت های مشخص شده بروری تصویر نشان داده می شود )

قسمت 4) در این مرحله نیاز است که دیتاست را لود کنیم ( توضیحات مربوط به این تابع در انتهای توضیح این بخش ارائه می شود )

قسمت 5)در اینجا نیاز است تا در یک حلقه تمامی سگمنت های داده شده را بررسی کنیم تا عدد یا حرف مدنظر را پیدا کنیم.

در قدم اول مختصات سگمنت را یافته ( شامل سطر و ستون های مربوطه) و آن را از عکس ادیت شده بر میداریم  و آن را تحت نام current_segment ذخیره می کنیم حالا نیاز است تا سگمنت را تغییر سایز دهیم این موضوع به این خاطر است که در هنگام correlation گیری سایز تصاویر یکسان باشند.

باید در این قسمت ماتریسی از 0 تشکیل دهیم تا بتوانیم در مرحله ی بعدی نتیجه correlation را در آن ذخیره کنیم ( باتوجه به اینکه مقدار correlation مربوط به هر عنصر دیتاست را در آن یادداشت می کنیم پس باید طول این ماتریس برابر تعداد عناصر دیتاست باشد)

حالا در این بخش به سراغ دیتاست می رویم وcorrelation  سگمنت را با هر عنصر دیتاست حساب می کنیم و نتیجه را در ro میریزیم در نهایت باید مقدار ماکسیمم را از این نتایج انتخاب کنیم ( دلیل انتخاب ماکسیمم این است که باید بیشترین تطبیق را سگمنت مورد نظر داشته باشد) نکته این است که برای نهایی کردن کار باید مطمئن شویم که از یک مقدار threshold این مقدار ماکسیمم بیشتر باشد این به این دلیل است که ممکن است یک بخش correlation بالایی را کسب کند اما یک نوشته بی معنا یا حتی جزو قاب باشد پس باید حتما مقدار correlation از یک مقدار مشخص بیشتر باشد تا آن را یک match در نظر بگیریم در صورت قبول شدن در این بخش نتیجه یعنی حرف یا عدد یافته شده را در آرایه نهایی ذخیره می کنیم.

لازم به ذکر است فرایند یاد شده در بالا به ازای تمامی سگمنت های یافت شده انجام می شود تا پلاک اصلی یافت شود.

قسمت 6) در این قسمت دیگر نتیجه آماده است پس یک فایل تکست باز می کنیم و نتیجه را درون آن می نویسیم پس از بستن این فایل برای نمایش آن را باز می کنیم .

توضیح تابع create_mapset :

این تابع فایل mapset را باز کرده و تصویر آن ها و اسامی آن ها را درون یک cell با دو بخش ذخیره می کند بدین ترتیب علاوه بر داشتن عکس مورد نظر، نام آن جهت نوشتن در تکست نهایی نیز در دسترس می باشد.

تغییرات انجام شده برروی mapset:

اگر دقت کنیم در هنگام سگمنت بندی نقاط حروف از خود آن ها جدا می شود که به شکل زیر نمایش داده شده است:

این موضوع ممکن است به دو طریق برروی نتیجه اثر بگذارد:

1- ممکن است با کاهش مقدار correlation از حد threshold بالاتر نرویم و همین باعث شود حرف مدنظر به اشتباه دور ریخته شود
2- ممکن است به دلیل نزدیکی حرف به حروف دیگر مقدار correlation در حرف دیگر بیشتر شده و نتیجه اشتباهی را حاصل کند.

برای حل این مشکل می توانیم نقاط این حروف را حذف کنیم ( لازم به ذکر است با توجه به اینکه حروف پ ت ث را نداریم حذف نقطه ب مشکلی را ایجاد نمی کند ) با حذف این نقاط مقدار correlation افزایش می باید که در دقت کد تاثیرگذار است

برای نمایش میزان تغییرات پس از انجام این حرکت حروف با نقطه نیز در دیتاست موجود هستند که می توان با رسم مقدار correlation در for ذکر شده میزان تغییرات و درصد بهبود عملکرد را مشاهده کرد.

نکته: دلیل وجود تابع mapset_edit این است که عکس داده شده برای دیتاست را دریافت کند که می تواند به فرم rgb باشد و سپس آن را تغییز سایز داده به رنج 0-255 تبدیل کرده و در نهایت آن را باینری کند همچنین آن را به فرمت logical نیز ذخیره می کنیم.

تست :

تمامی تست کیس های انجام شده در فایل تست کیس موجود هستند.

## بخش سوم

ایده ی انجام این کار:

در ابتدا شروع به correlation گیری می کنیم بین تصویر آرم آبی پلاک و کل محیط موجود، انتظار داریم که در نقطه ای که آرم آبی واقعا وجود دارد مقدار این correlation بیشینه شود و در اینصورت توانستیم محدوده پلاک را پیدا کنیم با استفاده از اندازه تصویر و نسبت آن به آرم آبی رنگ محدوده وافعی پلاک یافت می شود. حال در ادامه توضیح این موارد را به صورت کد بیان می کنیم.

در این بخش نیاز است تا از جلوبندی ماشین پلاک آن را استخراج کنیم.

قسمت 1) در این بخش همانند گذشته از کاربر می خواهیم تا ورودی خود را وارد کند برای این کار از دستور uigetfile استفاده می کنیم بعد از این دستور با استفاده از دستور imread تصویر انتخاب شده را می خوانیم و در متغی انتخاب شده می ریزیم.

قسمت 2) در این بخش تمام bluestrip هایی که به عنوان دیتاست این بخش آماده کرده ایم را می خوانیم اینکار به کمک تابع load_bluestrip انجام می شود که به همان ترتیبی که الفبا و اعداد را خواندیم آرم های مدنظر مارا می خواند و در یک ماتریس ذخیره می کند

قسمت 3) در این بخش نیاز داریم تا بدانیم محدوده پلاک در کجای تصویر واقع شده است به همین دلیل تابع find_plate را صدا می زنیم تا محدوده پلاک را برای ما پیدا کند توضیح عملکرد این تابع به شرح زیر است:

واضح است که خروجی این تابع در واقع حدود پلاک می باشد و براساس عکس ورودی باید این حدود را پیدا کند و برگرداند.

برای شروع فرآیند در ابتدا عکس پایه ای که از آرم آبی خودرو داریم را لود می کنیم ( آدرس این عکس را خودمان تعیین کرده ایم ) لازم به ذکر است باید ماتریسی که از آرم های آبی ساخته ایم را نیز لود کنیم تا در ادامه بتوانیم از آن ها استفاده کنیم.

در قدم بعدی نیاز است تا تصویر وارد شده را تغییر سایز بدهیم به آن شکل که بتوان در ادامه برای محاسبات از آن استفاده کرد.

در این قسمت مقدار correlation مربوط به آرم آبی و تصویر resize شده را حساب می کنیم که مقادیر برگشتی این تابع شامل تصویر میکس شده براساس این correlation، بیشترین مقدار یافت شده برای این correlation و همچنین محدوده پلاک می باشد. ( این تابع به صورت جزئ تر در ادامه توضیح داده می شود)

حال در این بخش ما یک مقدار correlation را داریم و باید مقادیر دیگر را به ازای تصاویر دیگر داخل دیتاست حساب کنیم عکسی که بیشترین مقدار correlation را بگیرد می تواند به درستی پلاک را به ما نشان دهد.( نحوه یافتن بیشترین correlation به این صورت است که هربار درصورت بزرگ ترین بودن مقدار محاسبه شده این مقدار جدید را به عنوان خروجی معرفی میکنیم.)

بعد از این بخش دیگر محدوه ی آرم آبی رنگ را داریم پس تمام کاری که باید انجام دهیم این است که محدوده ی پیدا شده را برروی تصویر تغییر دهیم bound_box یافت شده پارامترهای زیر را به ما نشان می دهد

- طول بالاترین نقطه سمت چپ
- عرض بالاترین نقطه سمت چپ
- طول کل پلاک
- عرض کل پلاک

حال با توجه به اینکه ما از correlation استفاده می کنیم به منظور در نظر گرفتن خطاها باید عددی را به عنوان ارور از مختصات کنونی کم کنیم تا در جای نسبی خود قرار بگیرد.( نکته ی حائز اهمیت این است که در هنگام کم کردن ارور از طول و عرض کل پلاک باید دوبرابر ارور در نظر گرفته شده را کم کنیم تا به مختصات نصبی برسیم)

در حالتی که ارور را اعمال نکرده ایم پلاک به صورت زیر تشخیص داده می شود:


همانطور که دیده می شود محدوده ی یافت شده توسط bound_box بسیار بزرگ تر از محدوده ی اصلی است که نشان می دهد ما در حال محاسبات با خطا هستیم حال اگر ارور را به این معادله اضافه کنیم برابر زیر می شود:


همانطور که مشخص است مقدار خطا کاهش پیدا کرده است که کمک می کند در مرحله های بعدی با دقت بیشتری عملیات detect را انجام دهیم.

عدد ratio نیز برای scale کردن به عکس اصلی استفاده می شود ( همانطور که قبلا ذکر شد ما تصویر را تغییر سایز دادیم که باید در نتیجه اعمال شود)

در نهایت نیاز است تا علاوه بر ارتفاع عکس، طول عکس را نیز با توجه به ratio آپدیت کنیم

در صورتی که یک بریک پوینت در خط مربوط به رسم متغیر bound_box_pre بگذاریم و رسم کنیم متوجه می شویم که تنها ارتفاع این bound_box درست مشخص شده است اما طول آن همچنان آپدیت نشده است که همین موضوع اهمیت اعمال ratio بر طول عکس را نشان می دهد

قبل از اعمال تغییرات برروی طول عکس:


بعد از اعمال این تغییرات بر روی عکس :


همانطور که مشخص است قسمت سبز پلاک را به درستی نشان می دهد.

در نهایت باید از threshold استفاده کنیم چرا که ممکن است تصاویر نامرتبط دیگری نیز باعث رسیدن به یک correlation بالا شوند اما تصویر مدنظر ما نباشند پس با گذاشتن threshold برابر 0.45 مطمئن میشویم که حتما بخش یافت شده مربوط به آرم آبی می باشد.

در نهایت این محدوده یافت شده را برمی گردانیم.

تابع rgb_correlation:

در ابتدا لازم است که ذکر کنیم در این بخش مجبور هستیم که correlation را در حیطه RGB حساب کنیم این به دلیل آن است که اگر تصویر را به صورت خاکستری و طیف 0-255 دربیاوریم ممکن است که نتوانیم به درستی بخش مربوط به آرم آبی رنگ را پیدا کنیم.

در ابتدا channel های مربوط به رنگ های مختلف را از هم جدا می کنیم و normalized cross correlation را برای هر یک از چنل ها محاسبه می کنیم ( همانطور که دیده می شود در ابتدا از تصویر بعد یک و از آرم آبی نیز بعد یک را جدا میکنیم واین correlation را حساب می کنیم و همین کار را در ادامه برای بقیه چنل ها نیز انجام می دهیم)

دلیل استفاده از این نوع correlation :

Comparing color channels independently

Highlight structural similarity

با توجه به اینکه ما تنها نیاز به یک عدد برای correlation داریم پس از اعداد به دست آمده میانگین می گیریم.

در نهایت بخشی را که دارای ماکسیمم correlation است را پیدا می کنیم و مقدار و ایندکس آن را بر میگردانیم. برای تبدیل index به مختصات (x,y) از دستور ind2sub استفاده می کنیم تا نقطه ی مورد نظر را بیابیم .

در نهایت نیاز است تا bounding box را بسازیم و آن را برگردانیم برای این کار به شکل زیر عمل می کنیم:

در ابتدا باید نقطه سمت چپ بالا را پیدا کنیم برای این کار نیاز است تا از مقدار peak x یافت شده مقدار template را کم کنیم به این خاطر که عدد یافت شده به عنوان ماکسیمم در حقیقت وسط این template را به دست آورده است که باید به جای درست منتقل شود

در قدم بعدی نیز برای به دست آوردن عرض نقطه سمت چپ بالا نیز همان کار را تکرار می کنیم

حالا در نهایت تمامی مختصات های لازم برای شکل دادن bounding box را داریم که شامل طول و عرض نقطه چپ بالا می شود و علاوه بر آن طول و عرض template را نیز داریم که می توانیم آن را استفاده کنیم ( از جمله دلایلی که در قسمت بعدی مجبور به resize هستیم همین است که در اینجا از طول و عرض عکس template استفاده کرده ایم)

حال دوباره باید به اسکریپت p3 باز گردیم با توجه به اینکه حدود پلاک را یافته ایم آن را از عکس اصلی پاک می کنیم تا تنها خود پلاک را به دست بیاوریم.

پلاک یافت شده را به تابع find_plate می دهیم که همان تابع بخش قبلی است با این تفاوت که دیگر ورودی از سمت کاربر وارد نمی شود و ما آن را تعیین می کنیم که همان عکس پلاک یافت شده در طی پروسه بالاست. در طی این تابع تمامی اعمال ذکر شده در بخش 2 اعمال شده و در نهایت نتیجه در فایل نوشته می شود.

تست:

تمامی تست های انجام شده در فایل test case موجود می باشند.

## بخش چهارم

در قدم اول، باید video را لود کنیم:

```
% Load video
video = VideoReader('car_video.mp4');
```

حال یک لیست خالی برای ذخیره‌سازی centroidهای موجود نگه می‌داریم و framerate ویدیو را استخراج می‌کنیم.

برای تشخیص objectهایی که درون ویدیو وجود دارد، از vision.ForegroundDetector استفاده کردیم. در واقع این تابع objectهایی که در ویدیو در حال حرکت هستند را از Objectهای ثابت در background متمایز می‌سازد.

برای تشخیص از 50 فریم از ویدیو استفاده کردیم و برای متوجه نشدن تغییرات، از سه مدل گاوسی بهره بردیم.

```
% Initialize variables
centroids = [];
frameRate = video.FrameRate;
detector = vision.ForegroundDetector('NumGaussians', 3, 'NumTrainingFrames', 50);
```

حال مادامی که فریمی در ویدیو وجود دارد، این عملیات را روی هر فریم انجام می‌دهیم:

- یک فریم می‌خوانیم.
- فریم را سیاه سفید می‌کنیم.
- از detectorای که تعریف کردیم استفاده می‌کنیم تا با کمک تابع bwareopen، نویزها و اجسامی که کوچکتر از یک سایز مشخص هستند و قطعا خودرو مورد نظر ما نیستند، حذف کنیم.
- در اینجا فرض شده که بزرگ‌ترین object موجود در عکس همان خودرو است، پس مساحت اجسام detect شده را محاسبه می‌کنیم و در areas ذخیره می‌کنیم.
- stats، Centroid اجسام را در خود دارد، پس با کمک آن centroid بزرگ‌ترین جسم را محاسبه می‌کنیم و به لیست centroidها اضافه می‌کنیم.
- در هر فریم، نقطه centroid محاسبه شده را مشخص می‌کنیم و دور objectای که به عنوان خودرو شناخته شده نیز یک مستطیل سبز می‌کشیم.
- نهایتا این frame را نمایش می‌دهیم.

```
while hasFrame(video)
    frame = readFrame(video);
    grayFrame = rgb2gray(frame);

    % Use background subtraction to detect moving objects
    mask = detector(grayFrame);
    mask = bwareaopen(mask, 50); % Remove small objects

    % Get the properties of the detected regions
    stats = regionprops(mask, 'Centroid', 'BoundingBox');

    if ~isempty(stats)
        % Assuming the largest blob is the car
        areas = arrayfun(@(s) s.BoundingBox(3) * s.BoundingBox(4), stats);
        [~, maxIdx] = max(areas);
        centroid = stats(maxIdx).Centroid;
        centroids = [centroids; centroid];

        % Display centroid and bounding box on frame
        frame = insertMarker(frame, centroid, 'o', 'Color', 'red', 'Size', 5);
        bbox = stats(maxIdx).BoundingBox;
        frame = insertShape(frame, 'Rectangle', bbox, 'Color', 'green');
    end

    % Display the video with detected centroids and bounding boxes
    imshow(frame);
    pause(1/frameRate); % Pause to match the frame rate of the video
end
```

دقت کنید در این روش ممکن است در ابتدا خودرو از ما دور باشد و centroid اشتباها تشخیص داده شود. به همین دلیل outlierها را حذف می‌کنیم.

کد تابع:

```
% Function to remove outliers from centroids
function filtered_centroids = remove_centroid_outliers(centroids)
    mean_centroid = mean(centroids);
    std_centroid = std(centroids);
    z_scores = abs((centroids - mean_centroid) ./ std_centroid);
    threshold = 3;
    valid_indices = all(z_scores < threshold, 2);
    filtered_centroids = centroids(valid_indices, :);
end
```

از روش z score برای حذف داده‌های پرت استفاده کردیم.

```
% Remove outliers from centroids
centroids = remove_centroid_outliers(centroids);
```

حال که centroidها را داریم، به روش اقلیدسی فاصله بین هر جفت متوالی را در نظر می‌گیریم و نهایتا جمع فاصله ها را به زمان و framerate تقسیم می‌کنیم تا به یک سرعت میانگین ر مقیاس پیکسل بر ثانیه برسیم.

مجددا به دلیل وجود داده‌های پرت در distances، از یک تابع برای حذف آنها استفاده کردیم:

کد تابع:

```
% Function to remove outliers from distances
function filtered_distances = remove_distance_outliers(distances)
    mean_distance = mean(distances);
    std_distance = std(distances);
    z_scores = abs((distances - mean_distance) / std_distance);
    threshold = 3;
    filtered_distances = distances(z_scores < threshold);
end
```

```
% Calculate speed if there are enough valid centroids
if size(centroids, 1) > 1
    distances = sqrt(sum(diff(centroids).^2, 2));
    filtered_distances = remove_distance_outliers(distances);
    total_distance = sum(filtered_distances);
    total_time = (length(filtered_distances) / frameRate);
    average_speed = total_distance / total_time; % Average speed in pixels per second
    disp('Average speed of the car in pixels per second:');
    disp(average_speed);
end
```

برای اینکه از محاسباتمان اطمینان حاصل کنیم، با در نظر گرفتن pixels_per_meter = 1400، سرعت را بر حسب متر بر ثانیه نیز محاسبه کردیم.

```
pixels_per_meter = 1400; % Conversion factor (pixels in one meter)

% Convert speed from pixels per second to meters per second
speed_meters_per_second = average_speed / pixels_per_meter;

disp('Speed of the car in meters per second:');
disp(speed_meters_per_second);
```

خروجی:

```
Average speed of the car in pixels per second:
  735.5143

Speed of the car in meters per second:
    0.5254
```