به نام خدا

CA2 - Signal & Systems

امیرعلی شهریاری (۸۱۰۱۰۰۱۷۳)

مقدمه : این پروژه شامل سه بخش است :

۱: پیاده سازی توابع درونی متلب به صورت دستی برای تشخیص پلاک انگلیسی.

۲: تشخیص پلاک فارسی به صورت تصویر صرفا پلاک.

۳: تشخیص پلاک از جلوبندی خودرو و سپس اتصال به بخش ۲

۱:

در بخش اول تصویر پلاک لود می شود و سپس به کمک تابع imresize ابعاد را به ۳۰۰*۵۰۰ پیکسل رسانده .

برای پیاده سازی دستی تابع rgb2gray تابع زیر را پیاده سازی میکنیم:

```
function gray_image = mygrayfun(color_image)
    red_channel = color_image(:,:,1);
    green_channel = color_image(:,:,2);
    blue_channel = color_image(:,:,3);

    gray_image = 0.299 * red_channel + 0.578 * green_channel + 0.114 * blue_channel;

    gray_image = uint8(gray_image);
end
```

در تابع فوق نخست با اکسترکت کردن و اعمال فرمول تصویر به صورت grayscale می شود:

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

در بخش بعدی میخواهیم تصویر خاکستری فوق را به سیاه و سفید صرف یعنی عملا باینری تبدیل کنیم پس برای این کار از یک threshold استفاده کرده و آستانه پیکسلی را برای آن به صورت دستی مشخص می کنیم.

در این تابع عملا مقدار threshold را اعمال می کنیم :

```
function binary_image = mybinaryfun(grayscale_image, threshold)
    binary_image = grayscale_image > threshold;
end
```

در اینجا عدد ۱۰۰ را با آزمون خطا و با مقایسه با بخش های پایین تر کد که در ادامه به توضیح آن می پردازیم ، بدست آوردیم.

از طرفی با mybinaryfun~ عملا سیاه و سفید های باینری را عوض میکنیم ، و یک و صفر جا به جا می گردند.

در بخش بعد میخواهیم نگاشت های نویزی را بهبود ببخشیم و بخش هایی ۰ و ۱ طبیعتا وجود دارند که شامل هیچ اطلاعاتی نیستند. به صورت عادی با استفاده از تابع bwareaopen میتوان این مسئله را حل کرد اما برای نوشتن دستی آن با تابع فوق :

```
function [cleaned_image, only_noisy_pic] = myremovecom(binary_image, n)

    [L, num] = bwlabel(binary_image);

    stats = regionprops(L, 'Area');

    small_components = find([stats.Area] < n);

    for i = 1:length(small_components)
        binary_image(L == small_components(i)) = 0;
    end

    cleaned_image = binary_image;
end
```

عملکرد تابع فوق به صورت در بخش نخست با یافتن کامپوننت های متصل سپس با regionprops اجزای متصل و شاخص های اجزای کوچک را دریافت می کنیم. و در حلقه فور اجزای کوچک را دانه دانه پاک می کنیم. و در نهایت خروجی تمیز شده را ارایه می دهد.

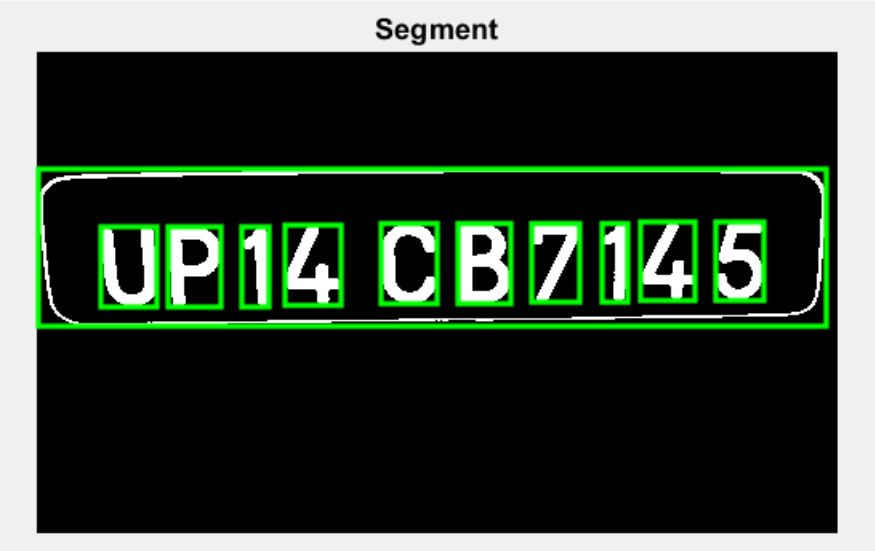
در بخش بعدی میخواهیم تابع را segmentبندی بکنیم که برای پیاده سازی دستی bwlabel به صورت زیر عمل می کنیم:

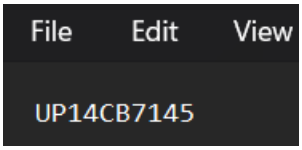
```
function [L, Ne] = mysegmentation(image)
    [rows, cols] = size(image);
    L = zeros(rows, cols);
    Ne = 0;

    neighbors = [0 1; 1 0; 0 -1; -1 0];

    for i = 1:rows
        for j = 1:cols
            if image(i, j) == 1 && L(i, j) == 0
                Ne = Ne + 1;
                stack = [i, j];
                top = 1;

                while top > 0
                    x = stack(top, 1);
                    y = stack(top, 2);
                    top = top - 1;

                    L(x, y) = Ne;

                    for k = 1:size(neighbors, 1)
                        x_n = x + neighbors(k, 1);
                        y_n = y + neighbors(k, 2);

                        if x_n >= 1 && x_n <= rows && y_n >= 1 && y_n <= cols && image(x_n, y_n) == 1 && L(x_n, y_n) == 0
                            top = top + 1;
                            stack(top, :) = [x_n, y_n];
                        end
```

در اینجا به صورت دستی اجزای متصل را با استفاده از الگوریتم جستجوی عمق (DFS) برچسب‌گذاری می‌کند. که با اعمال این تابع در تصویر باینری تمیز شده و در نهایت، مناطق تقسیم‌بندی شده را با کادرهای محدود نمایش می‌دهد. و عملا پیاده سازی (DFS) که البته پیچیدگی کد را بالا برده و مقداری کند می شود.

در نهایت باتوجه به دیتا ست از اعداد و حروف پلاک انگلیسی بخش `decision making` میتواند به سادگی عینا از کد تحلیل شده سر کلاس پیش برود و این سگمنت ها را که پیشتر جدا کردیم با آنها کورولیشن گیری کند و عملا با انجام پردازش تصویری روی آن مشخص میکند بیشتر شبیه کدام بوده و آنرا در نهایت در فایل تکست چاپ می کنیم.

تصاویر مراحل :





۲:

در بخش دوم میخواهیم وقتی که صرفا به ما تصویر پلاکی داده می شود آنرا با آنالیزی مشابه بخش قبل اما اینبار به پلاک زبان فارسی این کار را انجام دهیم . نکته مهم تفاوت آن در دیتا ستی است که در کورولیشن گیری استفاده می کنیم که در این بخش حروف و اعداد فارسی پلاک در فولدری به نام `Map Set` قرار داده شده است که عملا مشابه بخش قبلی انگلیسی است و به صورت زیر پیاده سازی می کنیم:

```
clc;
clear;
close all;


di=dir('Map Set');
st={di.name};
nam=st(3:end);
len=length(nam);


TRAIN=cell(2,len);
for i=1:len
   pic = imread(['Map Set','\',cell2mat(nam(i))]);
   %pic = im2gray(pic);
   threshold = graythresh(pic);
   pic = im2bw(pic,threshold);
   TRAIN(1,i)={pic};
   temp=cell2mat(nam(i));
   TRAIN(2,i)={temp(1)};
end

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

دیتا ست از حروف و اعداد که همه هم اندازه و به فرمت ۶۰*۵۰ از فولدر Map Set استخراج کرده و برای استفاده از آن در قسمت های بعدی پروژه باید آنها را در متلب لود بکنیم که در فایل training_loading.m قرار گرفته است.

نخست آنها را در استراکچر ریخته و در سلول st ذخیره می شوند دو سطر نخست بلا استفاده و سطر های سوم به بعد را در نام ریخته و در حلقه train تصاویر باینری را ذخیره سازی می کنیم و سلول train به صورت TRAININGSET.mat ذخیره می شود و در بخش های بعدی پروژه از آنها استفاده می کنیم.

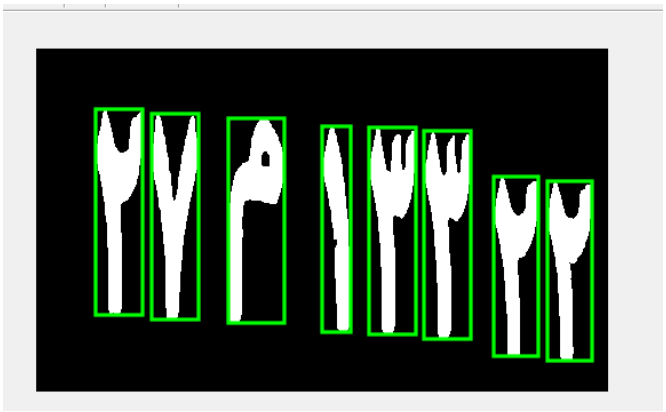
حال میخواهیم برنامه تشخیص پلاک را نوشته و عملیات بخش ۱ پروژه را روی آن نیز اعمال بکنیم . نخست ماتریس TRAININGSET که پیشتر تعبیه کردیم را لود کرده سپس تصویر پلاک دلخواه را به صورت ماتریس سه بعدی در متلب لود کرده و با تفاوت استفاده از توابع آماده در متلب عملیاتی مشابه انچه در بخش ۱ پروژه انجام دادیم را اینجا نیز انجام می دهیم و عملا توضیحات ساختاری بخش اول برای اینجا هم صادق است و بیشتر کتابخانه محور پیش رفتیم.

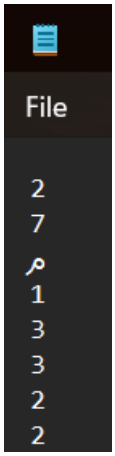
با استفاده از دستور آماده bwareaopen و ست کردن threshold روی ۵۰۰ اجزای کوچکتر را پاک کرده و از آن طرف روی ۸۰۰۰ و پاک کردن اجزای بزرگترو باقی ماندن صرفا پس زمینه و در ادامه دو تصویر باینری فوق را از یکدیگر کم کرده و تصویر صرفا اعداد و پلاک باقی می ماند.

اینبار با کمک تابع درونی متلب bwlabel بخش های متصل به یکدیگر و تعداد آنها را مشخص کرده و regionprops به ما در یافتن مختصات طرفین آن استراکت کمک می کند. این بخش های کار عملا مشابه پیاده سازی انگلیسی اما با توابع درونی متلب است که در کلاس پیاده شد. همچنین برای کاهش خطا شروطی برای ابعاد تکه ها قرار داده و شماره ردیف pripied تکه هایی که در این شرط ها صدق میکنند را در a نگهداری میکنیم.

در حلقه for و با استفاده از دستور rectangle دور پیکسل های ذخیره شده در a را خط کشیده و مستطیل های حائل آنها را ترسیم می کند.

و حالا که بخش ها و سگمنت ها جداسازی شده است میتوانیم با کورولیشن گیری از هرکدام از آنها و دیتا ستمان تشخیص دهیم که کامپوننت فوق چیست و چنانچه کورولیشن آن با هرکارکتری که بیشینه و بالاتر از treshold باشد برنامه به آن نسبت میدهد و خروجی در فایل تکست ذخیره می شود. این ترتیب نیز در راستای محور افقی بوده و برای جلوگیری از بهم ریختگی های احتمالی و بهم ریختن ترتیب آن این کار را انجام می دهیم. همچنین برای سهولت چاپ و رفع مشکل فارسی و انگلیسی بودن اعداد آن به صورت سطری در فایل چاپ می شوند.





۳:

در این فاز بایستی تصویر را از جلوبندی خودرو تشخیص داده و تصویر صرف پلاک را استخراج کنیم ، سپس با دادن ورودی عکس پلاک تفکیک شده به کد بخش قبل میتوانیم ارقام و حروف پلاک را بدست آورده و در فایل تکست بنویسیم.

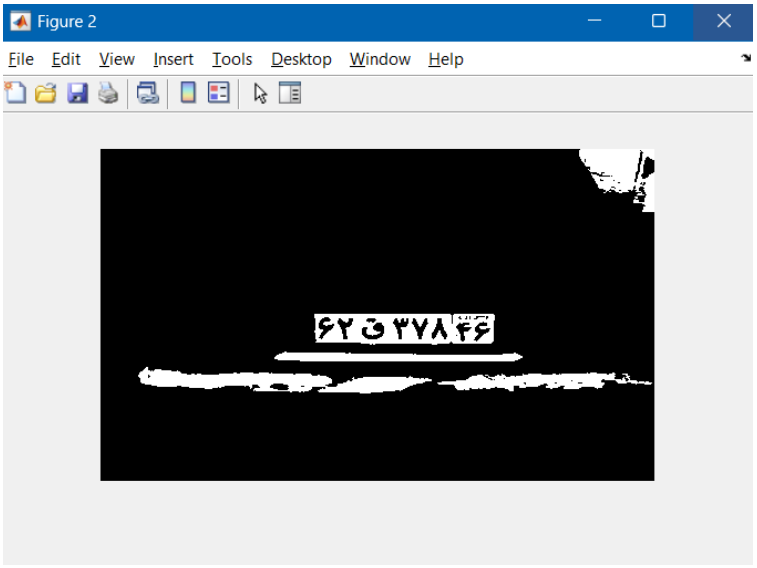
نخست مانند فاز های قبلی پروژه عکس را در متلب لود و پس از ریسایز کردن و مشابه بخش های قبلی و با کمک توابع درونی متلب آنرا به عکس باینری تبدیل می کنیم از آنجایی که عکس بخش های زاید بیشتری نسبت به فاز های قبلی دارد و برای

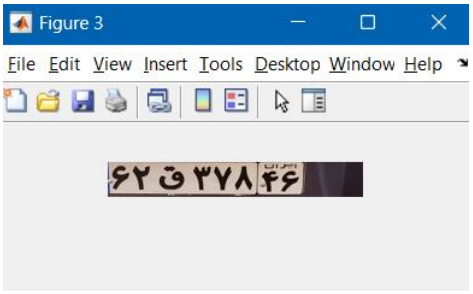
اطمینان از اینکه پیکسل های جلوبندی خودرو از پلاک تمیز یافته و به یکدیگر نچسبند ، threshold را بالاتر از حالت قبل برده.

در ادامه مجدد مشابه بخش قبلی با کمک تابع درونی متلب bwareaopen آستانه بالا و پایین را مشخص کرده و سپس دو تصویر را از یکدیگر کم کرده و در عکس نتیجه پلاک و جلو بندی ولی با پلان متفاوت خواهیم داشت.

در ادامه باز هم مانند فاز قبلی با دستور bwlabel اجزای چسبیده را جدا و مختصات های طرفین مشخص و در استراکچر ذخیره می کنیم.

و در تصاویر جلوبندی به طور معمول پلاک در وسط عکس است از طرفی میدانیم که تمامی پلاک های خودرو دارای طول و عرض و ابعاد یکسانی با یکدیگر هستند که با دانستن ابعاد خود تصویر و اینکه در تصویری که ما از جلوبندی داریم اغلب صرف خود پلاک دارای این مشخصات است ، این شرایط و حدود را باید در کد اعمال میکنیم . شماره ردیف مشخصات این جز در propied را در لیستی نگه داشته و با دستور نهایی کراپ ، بخش جدا شده را جدا می کنیم که پلاک خودرو بدست آمده و حاوی شرایط مطلوب برای ورودی دادن به فاز ۲ بخش قبل پروژه است تا پلاک خوانده شده و دیگر مراحل اجرایی گردد.





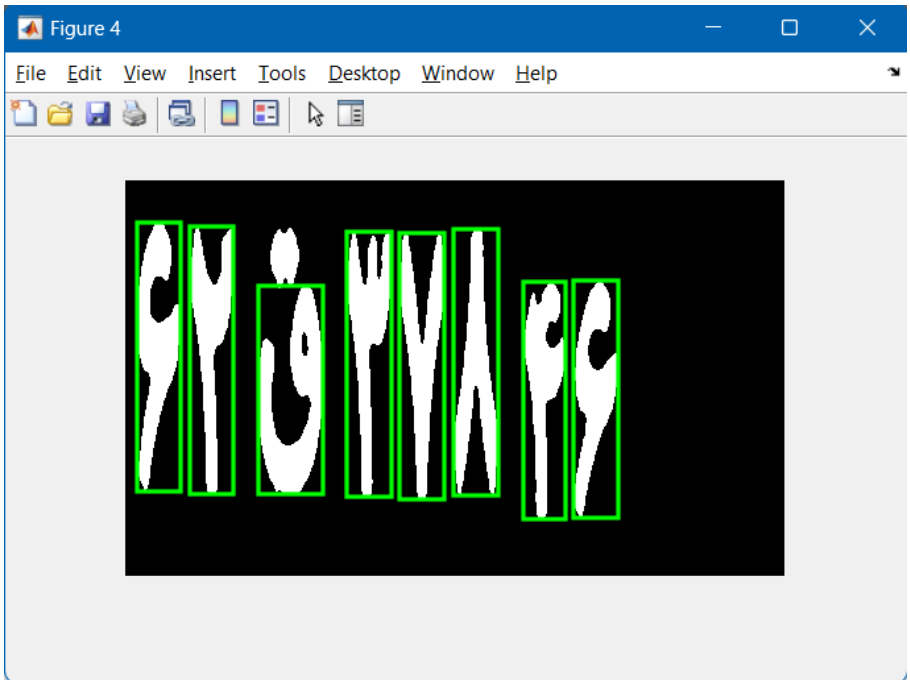
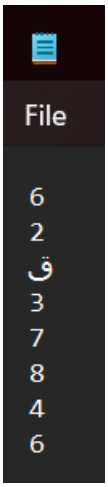


Figure 4
File Edit View Insert Tools Desktop Window Help


File
6
2
ق
3
7
8
4
6