**RENSSELAER POLYTECHNIC INSTITUTE**

**License Plate Recognition**

**João Paulo Aguilera Borges**

**Introduction to Image Processing**

**ECSE 4540**

**Spring 2013**

1. **INTRODUCTION**

An automatic plate number recognition (APNR) is a system that automatically detects the license plate based in a photo of the cars plate. It is used for controlling the number of cars in a park, catching car thieves by tracing suspect plates back to forged documents and in an electronic toll-collection system. For each application, the complications and problems characteristics should be analyzed. For example the angle of the cameras and the quality of the images change from one application to another, some images can be blurry (especially motion blurry), or an object obscuring, such as dirt or fog, can be on plate. This project aims to simulate an automatic plate number recognition system and will evaluate different techniques of image enhancement to improve the character recognition. Finally the edge histogram, followed by connected component techniques were chosen and a Matlab application was developed to demonstrate.

1. **TECHNICAL APPROACH**

Many techniques have been proposed for plate detection, each having its own advantages and disadvantages. They are all made of some assumption for example the license plate have the same color [5], the plates represent a certain portion of the image and have more luminosity than the other areas [2], and the most used one is that plates area have a high concentration of edges [6][7]. In this project a system that simulates the functionality of an electronic toll-collection system such as EZ pass was created using Matlab. The system received an image such as the one below and prompt to the user select which technique to use, using a friendly user interface and finally returned a text with the license plate number.



Figure 1: Example of input image

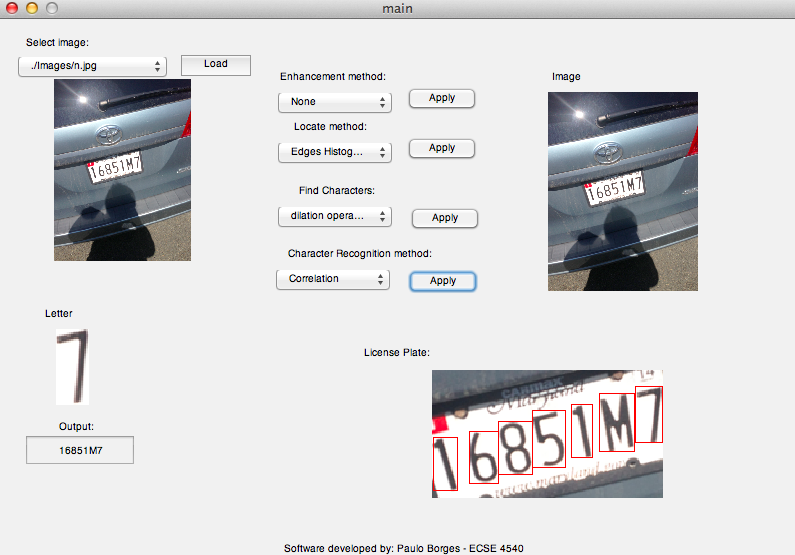


Figure 2: Software interface

To accomplish those tasks the system was divided in five steps:

1. Image acquisition

The imagery source is a variety of photos of license plates taken in different locations around campus with different angles and positions. Those images have different noise ratio, luminosity and directions but fixed size 800x600 and resolution.

1. Image enhancement:

The plate localization algorithm uses two assumptions: 1. The plate like area has more intensity than the other areas, and 2. The plates have more edges than the rest of the images. Therefore this step aims to guaranty that those assumptions were right by increasing image contrast in plate-like regions and intensifying the edges. The following methods were tested:

1. Do a sobel operator in the vertical axis, [1 0 -1;2 0 -2; 1 0 -1] and add the result back to the original image creating an crisper result. However this approach is not very good because the noise, high frequency, was added back.



Figure 3: Image after a Sobel operation

1. Apply a Gaussian filter, different values of sigma, to the image after the sobel operator. The result was even worse and the characters in the plate were illegible. The image bellow is using sigma = 2

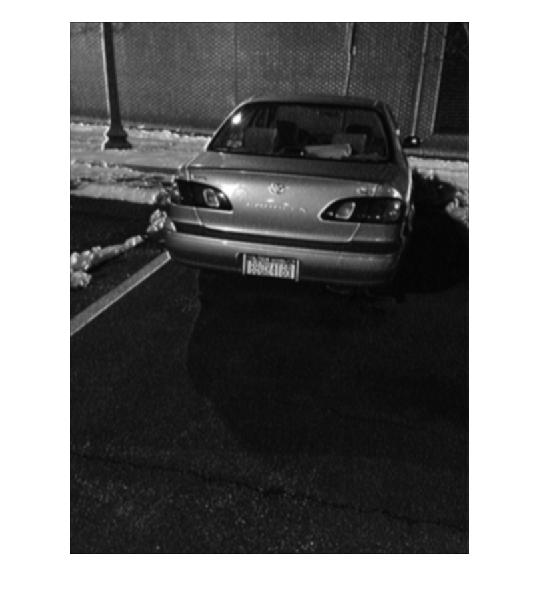


Figure 4: Using only Gaussian to enhance the image

1. Apply a LOG filter, this filter would reduce the noise trough a Gaussian and then enhance the edges using a Laplacian. It was used sigma = .2 and a 5x5 window. The result was the best if the edge histogram technique was used for locating the plate.



Figure 5: Using Laplacian of Gaussian

1. Apply a Gaussian using sigma = 10 times the graythreshold (around 2). Then use a sobel operator to enhance the vertical axes. This approach was proposed in [3], and indeed had shown the best result when the location technique was smearing, however it presented a poor result if the location technique used was edge histogram.

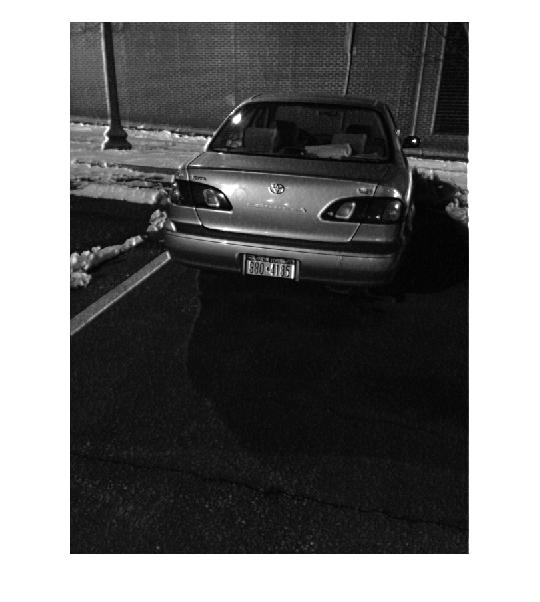


Figure 6: Using Gaussian filter followed by a Sobel Operation

In all the cases the central area of the image, supposedly where the license plate is, was increased.

1. Plate location:

This was the most challenging step. The localization is based on a series of assumptions and constrains any change in those constrains result in a wrong result. Most of the images in the database follow the constraint of having the license plate in the central region and taken from the same angle. However there are images that were taken with random distance and angle. In the United States license plates do not follow a single pattern, each state has its own regulation and even inside the state of New York there are two kind of plate, becoming the implementation even harder.

To locate the plate 2 different techniques were initially proposed: boundary line-based method combining the Hough transform and Contour algorithm such as proposed in [1] and one that is much simpler called smearing as proposed in [2]. However the edge histogram technique [6][7] was used instead of the Hough Transform method because of implementation issues. In the literature one can find other techniques[5] and there is no “state of the art “ technique that works for every country or application.

* 1. Image binarization followed by smearing:

Smearing is a method for the extraction of text areas on a mixed image. With the smearing algorithm, the image is processed along vertical and horizontal runs (scan-lines). If the number of white pixels is less than a desired threshold or greater than any other desired threshold, white pixels are converted to black [2]. After smearing, a morphological operation, dilation, is applied to the image for specifying the plate location.

In order to find the best solution for the set of images various combinations of parameters were tested. For the image binarization the threshold was define in a try and error approach as 0.5 of the average intensity. [APPENDIX 1]

However applying the algorithm as described in [2] didn’t result in a very good solution, this is justified because the intensity of the light in the images in the dataset varies a lot (photos were taken in different time of the day).

1. Using only the binarization
2. Using the binarization followed by a horizontal smearing
3. Using the binarization followed by a vertical smearing
4. Using the binarization followed by a horizontal and vertical smearing

Figure 7: Image after applying horizontal smearing Figure 8: Image after vertical smearing

The plate localization step generates a mask, same size of the image, that ideally has 1 in the area of the plate and 0 elsewhere. However none of the localization algorithm resulted in a perfect mask for a good amount of images. Therefore after the smearing a cut operation had to be performed to separate the plate regions and return a collection with a reduced size matrix with only plate candidates.

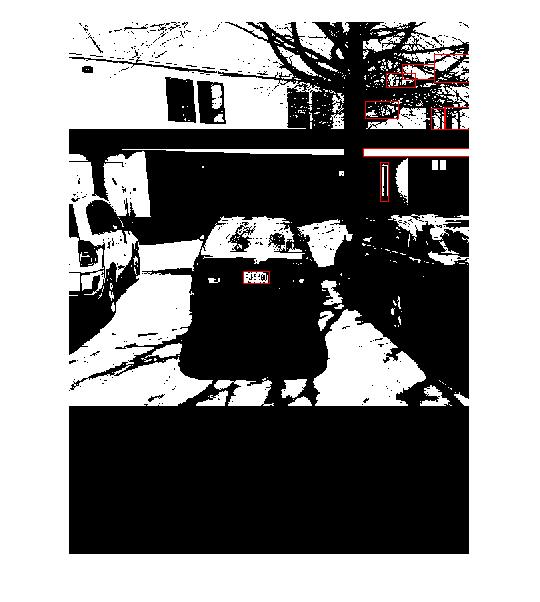


Figure 9: Connected component applied to the mask

The algorithm that was used to cut the area of interest is a recursive implementation of connected component. [APPENDIX 2]

Find the start point and mark.

Go to the neighbors of this point. If its value is 1, mark this point and go to its neighbor. If its value is 0 or this point has no neighbor return.

The marked points are the one in the region.

* 1. Edge Histogram

Histogram is a graph representing the values of a variable quantity over a given range. In this Number Plate Detection algorithm, the writer has used horizontal and vertical histogram, which represents the column-wise and row-wise histogram respectively. These histograms represent the sum of differences of gray values between neighboring pixels of an image, column-wise and row-wise[ 6]. In order to get the differences of gray values between neighboring, i.e. edges, the Roberts filter was used as suggested by [7].

Using the edge histogram the detection of the correct region followed the procedure: if the number of white pixels (one edge is white) per column is above a calculated threshold than this region is assumed to indicate the horizontal position of the license plate. In case there are several possible regions selected, we will continue to work with the widest.

For vertical edge processing the same procedure was used. If the number of vertical edges, in the pre selected horizontal area, is above a certain threshold than this area represents a license plate. The whole process can be seen as passing a low-pass digital filter into the histogram, this filter is applied to remove unwanted areas from an image. In this case, the unwanted areas are the rows and columns with low histogram values. A low histogram value indicates that the part of image contains very little variations among neighboring pixels. Since a region with a license plate contains a plain background with alphanumeric characters in it, the difference in the neighboring pixels, especially at the edges of characters and number plate, will be very high.

In this algorithm, the dynamic threshold is equal to the average value of a histogram. The output of this process is a histogram showing regions having high probability of containing a number plate. The result can be seen in the figure below and the code used is available in appendix 3.

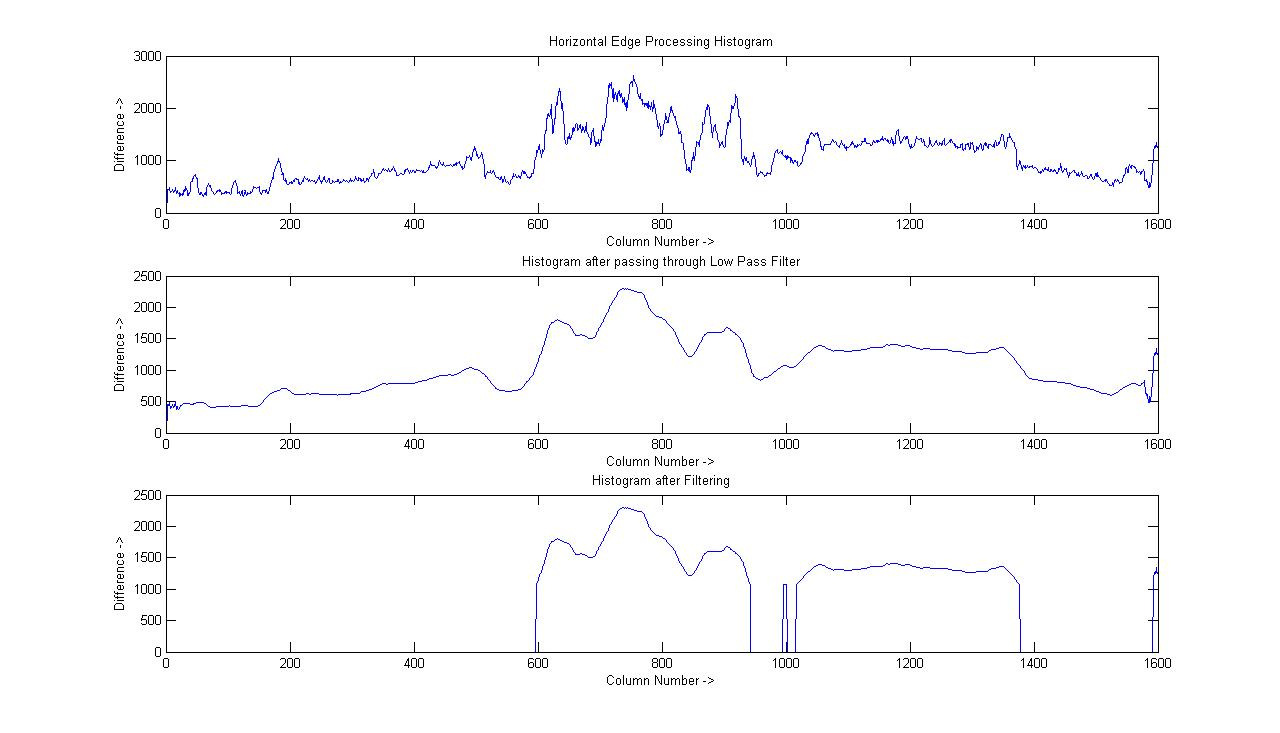


Figure 10: Histogram of the license plate. image from[6]

1. Character segmentation

This step aim to separate each character in the plate so it can be later recognized. The algorithm used is as follow [APPENDIX 4].

1. Create a binary image with enhanced edges
2. Separate connected characters using a erosion operation
3. Use a connect component approach to create regions candidates
4. Filter the candidates based on size and position

Most of the papers implement similar techniques to do the one described above [2][3][4], however there are more sophisticated methods but with higher computational cost.

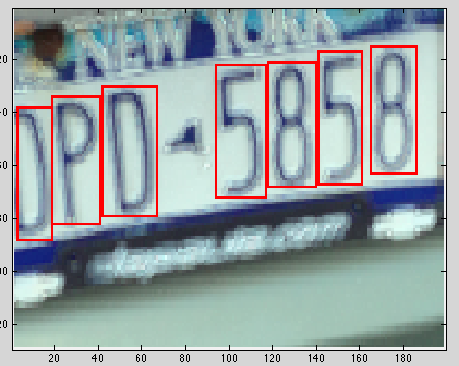


Figure 11: Result of the letter segmentation

1. Optical Character Recognition

Initially two different techniques had been proposed for the character recognition Template Matching [2] and Support Vector Machine [4]. However, due the time constraint only the Template Matching was implemented.

To measure the similarity and find the best match the template matching approach uses correlation. This is an effective technique for image recognition which was developed by Horowitz. This method measures the correlation coefficient between a number of known images with the same size unknown images or parts of an image with the highest correlation coefficient between the images producing the best match. The following steps were executed to recognize the character[APPENDIX 5].

1. Create a template database containing all the possible characters in a license plate.
2. Improve the quality of the received character using a dilatation operation to fill the holes.
3. Resize the image so it has the same size of the template
4. Perform a 2-D correlation between the image and all the images in the template.
5. Return the character with the highest correlation.



Figure 12: Example of letters in the template

**III TEST AND EVALUATION**

The system was evaluated empirically; a table showing the different results for the combination of the different algorithms is presented bellow. One hundred images were tested to generate the table. The background and the image condition modified the final result. The higher frequency regions in the background generated a false positive in the plate localization as expected. The enhancement used to generate this table was the Laplacian of Gaussian for the edge histogram algorithm and the Gaussian followed by a Sobel for the smearing technique because those presented the best results and was suggested in [2]. The average number of candidates generated for each image is also shown in the following table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Result using a fixed set | Result using a random set | Result using a fixed set with previous enhance | Result using a random set with previous enhance |
| Mask A | 80% - 20 | 60 % - 10 | 80 % - 8 | 60 % - 10 |
| Mask B | 70% - 15 | 50% - 8 | 70 % - 7 | 50 % - 12 |
| Mask C | 60% - 12 | 60% - 12 | 50 % - 10 | 50 % - 3 |
| Mask D | 90% - 4 | 40% - 6 | 90% - 5 | 40% - 7 |
| Direct Image | 60% - 19 | 30% - 20 | 80% - 18 | 40% - 20 |
| Edge Histogram | 80% | 20% | 95% | 35% |

Table 1: Results of the location step

The remaining steps were tested in a set made of 20 images. Those images were generated using the edge histogram method.

The character segmentation method were able to detect all the letters in the plate however in 5 cases. i.e. 25% it also detect areas that were not letters.



Figure 13: Example of false positive

The correlation had presented a perfect result because most of the images used to create the template were from the plates. Even if the image was a rotate or not centralized the algorithm was able to detect because more than one image was used as template for each character.

**IV CONCLUSION AND FUTURE WORK**

This project consisted of an application designed for the recognition of car’s license plate. The main goal of the system was testing different techniques available in the literature and applied the knowledge learned in the introduction to image processing class. The software firstly enhance the input then extracted the plate location, separated the plate characters individually by segmentation and finally applied template matching with the use of correlation for recognition of plate character. For future work, other techniques should be included in the project. Hough Transform is a method for locating plates that shows great results and is discussed in many paper and should be included in this application. In the template matching step a future improve is consider the angle of the image therefore performing an image rotation to have better results.

**V REFERENCE**

[1]http://users.utcluj.ro/~rdanescu/proiecte/01-licenseplate.pdf

[2]http://pdf.aminer.org/000/349/486/gray\_scale\_character\_recognition\_by\_gabor\_jets\_projection.pdf

[3] <http://ac.els-cdn.com/S0167865505001406/1-s2.0-S0167865505001406-main.pdf?_tid=acdf9518-8985-11e2-adb3-00000aacb35f&acdnat=1362921979_902d579f2e482404e4439330e451c25b>

[4] <http://www.itfrindia.org/ICCIC/Vol2/9024ICCIC.pdf>

[5] http://ieeexplore.ieee.org/xpls/abs\_all.jsp?arnumber=413580

[6] <http://vortex.cs.wayne.edu/papers/ijns1997.pdf>

[7] <http://www.win.tue.nl/aime/Files/apr2002_license.pdf>

**VI APPENDIX**

1. **Code for the smearing**

function out = PlateLocalization(input,method)

%This function aims to isolate the plate region in the image.

%It can receive as input a cell containing a set of images or only one

%image. It is based on smearing. This is to be used if the plate is not the

%determinant of the image. Obs: Not working as expected

t = .5;

if (nargin < 2)

method = 4;

end

if (~iscell(input))

img = cell(1,1);

img{1} = input;

else

img = input;

end

out = cell(1,length(img));

for k=1:length(img)

%smearing

mask = im2bw(img{k},t\*graythresh(img{k}));

if (method ==1 || method == 3) % horizontal smearing

hori\_thresh = sum(mask,2);

h\_finx = find(hori\_thresh>=500);

mask(h\_finx,:) = 0;

end

if (method == 2 || method == 3) % vertical smearing

ver\_thresh = sum(mask,1);

v\_finx = find(ver\_thresh<30);

mask(:,v\_finx) = 0;

end

if (length(size(img{k})) == 3) %color img

masked = zeros(size(img{k}));

masked(:,:,1) = img{k}(:,:,1) .\* uint8(mask);

masked(:,:,2) = img{k}(:,:,1) .\* uint8(mask);

masked(:,:,3) = img{k}(:,:,1) .\* uint8(mask);

else %grayscale

masked = img{k} .\* uint8(mask);

end

figure, imshow(masked)

out{k} = mask;

end

end

1. **Code for the connected component algorithm**

function [ out ] = cut( Img,i,j )

%Img is a binary image

if (nargin < 3)

%first iteraction

%Img = [0 0 0 1 1 1 0 0;0 0 0 1 1 1 0 0; 0 0 0 0 0 0 1 1; 0 0 0 0 0 0 1 1];

[a b] = find(Img,1);

%start the recursive part

out = cut(Img,a,b);

else

%recursive part

[h w] = size(Img);

out = NaN;

if ((i <= h) && (j <= w) && (i >= 1) && (j >= 1) && (Img(i,j) == 1))

out = [i,j];

Img(i,j) = -1; %visited

q = cut(Img,i-1,j);

if (~isnan(q))

out = [out;q];

%set all the values in out as visited

for k=1:length(out)

Img(out(k)) = -1; %visited

end

end

w = cut(Img,i+1,j);

if (~isnan(w))

out = [out;w];

%set all the values in out as visited

for k=1:length(out)

Img(out(k)) = -1; %visited

end

end

r = cut(Img,i,j-1);

if (~isnan(r))

out = [out;r];

%set all the values in out as visited

for k=1:length(out)

Img(out(k)) = -1; %visited

end

end

t = cut(Img,i,j+1);

if (~isnan(t))

out = [out;t];

%set all the values in out as visited

for k=1:length(out)

Img(out(k)) = -1; %visited

end

end

out = unique(out,'rows');

end

end

end

1. **Code for the edge histogram isolation method**

function [ plate ] = PlateIsolation( images, mask )

%If this function receives two arguments, image and mask it tries to

%isolate the ROI in the image based on that mask. If it only receive the

%image it assumes that the license plate represents the majority of the image and

%tries to isolate the ROI based on the number of edges in the images. Both

%image and mask can be cell

if (~iscell(images))

img = cell(1,1);

img{1} = images;

else

img = images;

end

plate = cell(1,length(img));

if (nargin == 2)

for k=1:length(img)

p = cut(mask);

a = min(p);

b = max(p);

plate{k} = img{k}(a(1):b(1),a(2):b(2));

%disp(mask);

end

else

for k=1:length(img)

%%%%%%isolate plate

Plate = 0;

A=img{k};

[h,w,~]=size(A);

%Convert the image to grayscale and do edge detection

A = rgb2gray(A);

A = im2bw(A,graythresh(A)\*1.3); %The factor 1.3 was decided in a try and error approach

A=edge(A,'roberts');

horHist = sum(A);

%define treshold for horizontal histogram

gem=max(horHist)/2.3; %2.3 was according to the paper

hstart=0;

hcounter=0;

arc=0;

hcoor=zeros(1,2);

%find the start and end of the license plate based on the

%horizontal histogram

for i=1:w

if horHist(i)>gem(1)

if(hstart==0)

hstart=i;

end

hcounter=0;

else

if hstart>0

if hcounter>(w\*0.07)

heinde=i-hcounter;

width=heinde-hstart;

if(width>(w\*0.1))

arc=arc+1;

hcoor(arc,1)=hstart;

hcoor(arc,2)=width;

end

hstart=0;

hcounter=0;

end

hcounter=hcounter+1;

end

end

end

[ww,~]=size(hcoor);

hstart=0;

hwidth=0;

% If there are many horizontal locations, just pick out the widest.

for i=1:ww

if(hcoor(i,2)>hwidth)

hwidth=hcoor(i,2);

hstart=hcoor(i,1);

end

end

if ((hstart > 0) && (hwidth > 0)) %if some error happen this if will avoid the propagation

A=A(:,hstart:(hstart+hwidth),:); %isolate the horizontal part

verHist=zeros(h);

%vertical edge counting, based on neighbor

for j=1:h

tot=0;

for i=2:hwidth

if (A(j,i-1)==1 && A(j,i)==0) || (A(j,i-1)==0 && A(j,i)==1)

tot=tot+1;

end

end

verHist(j)=tot;

end

verh=zeros(1);

count=1;

for i=1:h

if(verHist(i)>0)

verh(count)=verHist(i);

count=count+1;

end

end

gem=mean(verh);

vstart=0;

vcounter=0;

arc=0;

vcoor=zeros(1,2);

h\*0.07;

%define the vertical region

for i=1:h

if verHist(i)>gem(1)

if(vstart==0)

vstart=i;

end

vcounter=0;

else

if vstart>0

if vcounter>(h\*0.03)

veinde=i-vcounter;

height=veinde-vstart;

if(height>(h\*0.05))

arc=arc+1;

vcoor(arc,1)=vstart;

vcoor(arc,2)=height;

end

vstart=0;

vcounter=0;

end

vcounter=vcounter+1;

end

end

end

[l,~]=size(vcoor);

%Crop the plate

if ((vcoor(l,1) > 0) && (vcoor(l,2) > 0))

Plate=img{k}(vcoor(l,1):vcoor(l,1)+vcoor(l,2),hstart:(hstart+hwidth),:);

end

end

if (Plate ~= 0)

plate{k} = imresize(Plate, [100 NaN]);

else

%An error happened

disp('Error on : ');

disp(k);

end

end

plate = plate(~cellfun('isempty',plate));%eliminate empty cells. I.E cells with errors.

%plate = ~cellfun(@isempty,plate); %eliminate empty cells. I.E cells with errors.

end

end

1. **Code for the character segmentation**

function [ letters ] = extractLetters( plates )

%extractLetters receives a set of license plate and return a set with

%possible letters in these plates

if (~iscell(plates))

img = cell(1,1);

img{1} = plates;

else

img = plates;

end

letters = cell(1,length(img));

for k=1:length(img)

[s1 s2 ~] = size(img{k});

i2 = edge(rgb2gray(img{k}),'canny',0.3); %increase the edges

%separate one letter from another

se = strel('square',2);

i3 = imdilate(i2,se);

i4 = imfill(i3,'holes');

%Connect components

[Ilabel num] = bwlabel(i4,4);

Iprops = regionprops(Ilabel);

Ibox = [Iprops.BoundingBox];

Ibox = reshape(Ibox,[4 num]);

%%figure;

%%imshow(Plate);

%%hold on;

out = cell(1,num); %worse case it will have num images as letter candidates

for cnt = 1:num

%the size and area of the image has to make sense...

%the image was resized to [100 NaN]

if (Iprops(cnt).Area > 100 && Iprops(cnt).Area < 1000)

aux = Ibox(:,cnt);

if ((aux(3) < s2/4) && (aux(4) > s1/3) )

%w = waitforbuttonpress;

%%rectangle('position',Ibox(:,cnt),'edgecolor','r');

out{cnt} = imresize(imcrop(img{k}, aux),[45 20]);

%%figure; imshow(subImage);

end

end

end

letters{k} = out(~cellfun(@isempty,out)); %eliminate empty cells

%%hold off;

end

letters = letters(~cellfun(@isempty,letters)); %eliminate empty cells

end

1. **Code for the character recognition**

function letter=im2char(img,Template)

%Receives a img and return a Char with the best correlation

if (nargin < 2)

load('Template.mat') % Loads the templates of characters in the memory.

end

img=imresize(img,[45 20]); % Resize the input image so it can be compared with the template's images.

%improve the image based on dilatation and invert the binary, so its easy

%to correlate

img = imdilate(~im2bw(img,0.6),strel('line',3,90));

try

probabily= zeros(1,length(Template));

for i=1:length(Template)

probabily(i) = corr2(Template{i},img); % Correlation the input image with every image in the template for best matching.

end

%%%disp(probabily);

if (max(probabily) < 0.01)

letter = '-'; %did not recognize

else

vd=find(probabily==max(probabily)); % Find the index of the most probable character

%Series of ifs based on the created template

if (vd==1 || vd==2)

letter='A';

elseif vd==3 || vd==4

letter='B';

elseif vd==5 || vd==6

letter='C';

elseif vd==7 || vd==8

letter='D';

elseif vd==9 || vd==10

letter='E';

elseif vd==11 || vd==12

letter='F';

elseif vd==13 || vd==14

letter='G';

elseif vd==15

letter='H';

elseif vd==16

letter='I';

elseif vd==17 || vd==18

letter='J';

elseif vd==19

letter='K';

elseif vd==20 || vd==21

letter='L';

elseif vd==22 || vd==23

letter='M';

elseif vd==24|| vd==25

letter='N';

elseif vd==26 || vd==27

letter='O';

elseif vd==28 || vd==29

letter='P';

elseif vd==30 || vd==31

letter='Q';

elseif vd==32 || vd==33

letter='R';

elseif vd==34 || vd==35

letter='S';

elseif vd==36

letter='T';

elseif vd==37

letter='U';

elseif vd==38

letter='V';

elseif vd==39

letter='W';

elseif vd==40

letter='X';

elseif vd==41 || vd==42 || vd==10

letter='Y';

elseif vd==43

letter='Z';

elseif vd==44

letter='0';

elseif vd==45 || vd==46

letter='1';

elseif vd==47|| vd==48

letter='2';

elseif vd==49 || vd==50

letter='3';

elseif vd==51 || vd==52

letter='4';

elseif vd==53 || vd==54

letter='5';

elseif vd==55 || vd==56

letter='6';

elseif vd==57|| vd==58

letter='7';

elseif vd==59 || vd==60

letter='8';

elseif vd==61 || vd==62

letter='9';

else

letter='-';

end

end

catch

letter = '-'; %error

end

end

1. **Code for the main and GUI**

function varargout = main(varargin)

% MAIN MATLAB code for main.fig

% MAIN, by itself, creates a new MAIN or raises the existing

% singleton\*.

%

% H = MAIN returns the handle to a new MAIN or the handle to

% the existing singleton\*.

%

% MAIN('CALLBACK',hObject,eventData,handles,...) calls the local

% function named CALLBACK in MAIN.M with the given input arguments.

%

% MAIN('Property','Value',...) creates a new MAIN or raises the

% existing singleton\*. Starting from the left, property value pairs are

% applied to the GUI before main\_OpeningFcn gets called. An

% unrecognized property name or invalid value makes property application

% stop. All inputs are passed to main\_OpeningFcn via varargin.

%

% \*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one

% instance to run (singleton)".

%

% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help main

% Last Modified by GUIDE v2.5 01-May-2013 12:00:27

% Begin initialization code - DO NOT EDIT

gui\_Singleton = 1;

gui\_State = struct('gui\_Name', mfilename, ...

'gui\_Singleton', gui\_Singleton, ...

'gui\_OpeningFcn', @main\_OpeningFcn, ...

'gui\_OutputFcn', @main\_OutputFcn, ...

'gui\_LayoutFcn', [] , ...

'gui\_Callback', []);

if nargin && ischar(varargin{1})

gui\_State.gui\_Callback = str2func(varargin{1});

end

if nargout

[varargout{1:nargout}] = gui\_mainfcn(gui\_State, varargin{:});

else

gui\_mainfcn(gui\_State, varargin{:});

end

% End initialization code - DO NOT EDIT

% --- Executes just before main is made visible.

function main\_OpeningFcn(hObject, eventdata, handles, varargin)

% This function has no output args, see OutputFcn.

% hObject handle to figure

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% varargin command line arguments to main (see VARARGIN)

%%%%Populate the popImages menu with all the images in the folder

strFolder = './Images';

list = dir(strcat(strFolder,'/\*.jpg'));

l = length(list);

fname = cell(1,l);

for k=1:length(list)

fname{k} = strcat(strFolder,'/',list(k).name);

end

set(handles.popImages,'String', fname);

% Choose default command line output for main

handles.output = hObject;

% Update handles structure

guidata(hObject, handles);

% UIWAIT makes main wait for user response (see UIRESUME)

% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.

function varargout = main\_OutputFcn(hObject, eventdata, handles)

% varargout cell array for returning output args (see VARARGOUT);

% hObject handle to figure

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure

varargout{1} = handles.output;

% --- Executes on selection change in popImages.

function popImages\_Callback(hObject, eventdata, handles)

% hObject handle to popImages (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popImages contents as cell array

% contents{get(hObject,'Value')} returns selected item from popImages

% --- Executes during object creation, after setting all properties.

function popImages\_CreateFcn(hObject, eventdata, handles)

% hObject handle to popImages (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.

% See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

% --- Executes on selection change in popEnhance.

function popEnhance\_Callback(hObject, eventdata, handles)

% hObject handle to popEnhance (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popEnhance contents as cell array

% contents{get(hObject,'Value')} returns selected item from popEnhance

% --- Executes during object creation, after setting all properties.

function popEnhance\_CreateFcn(hObject, eventdata, handles)

% hObject handle to popEnhance (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.

% See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

% --- Executes on selection change in popEnhance.

function popupmenu4\_Callback(hObject, eventdata, handles)

% hObject handle to popEnhance (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popEnhance contents as cell array

% contents{get(hObject,'Value')} returns selected item from popEnhance

% --- Executes during object creation, after setting all properties.

function popupmenu4\_CreateFcn(hObject, eventdata, handles)

% hObject handle to popEnhance (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.

% See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

% --- Executes on button press in btLoadImg.

function btLoadImg\_Callback(hObject, eventdata, handles)

% hObject handle to btLoadImg (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

%%%Load the image

contents = cellstr(get(handles.popImages,'String'));

fname = contents{get(handles.popImages,'Value')};

global img;

global img2;

img = imresize(imread(fname),[800 600]);

img2 = img;

axes(handles.axes1);

handles.axes1 = imshow(img);

set(handles.axes1,'ButtonDownFcn',@ImageClickCallback);

function ImageClickCallback ( objectHandle , eventData )

global img;

h = figure;

h = imshow(img);

% --- Executes on button press in btApplyE.

function btApplyE\_Callback(hObject, eventdata, handles)

% hObject handle to btApplyE (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

global img;

global img2;

if (get(handles.popEnhance,'Value') == 1)

img2 = img;

else

img2 = enhance(img,get(handles.popEnhance,'Value') -1);

end

axes(handles.axes2);

handles.axes2 = imshow(img2);

set(handles.axes2,'ButtonDownFcn',@ImageClickCallback2);

function ImageClickCallback2 ( objectHandle , eventData )

%global img2;

h = get(objectHandle,'parent');

F=getframe(h); %select axes in GUI

figure(); %new figure

image(F.cdata); %show selected axes in new figure

% --- Executes during object creation, after setting all properties.

function popLocate\_CreateFcn(hObject, eventdata, handles)

% hObject handle to popLocate (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.

% See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

% --- Executes on button press in btLocate.

function btLocate\_Callback(hObject, eventdata, handles)

% hObject handle to btLocate (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

global img2;

global plate;

try

if (get(handles.popLocate,'Value') == 1)

axes(handles.axes3);

plate = PlateIsolation(img2);

if (~isempty(plate))

plate = plate{1};

handles.axes3 = imshow(plate);

set(handles.axes3,'ButtonDownFcn',@ImageClickCallback2);

else

set(handles.edit2,'String','Error');

end

else

imgs = PlateLocalization(img2,get(handles.popLocate,'Value')-1);

plate = PlateIsolation(img2,imgs{1});

plate = plate{1};

axes(handles.axes3);

handles.axes3 = imshow(plate);

axes(handles.axes2);

img2 = imgs{1};

handles.axes2 = imshow(img2);

set(handles.axes2,'ButtonDownFcn',@ImageClickCallback2);

end

catch

set(handles.edit2,'String','Error');

end

function edit2\_Callback(hObject, eventdata, handles)

% hObject handle to edit2 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text

% str2double(get(hObject,'String')) returns contents of edit2 as a double

% --- Executes during object creation, after setting all properties.

function edit2\_CreateFcn(hObject, eventdata, handles)

% hObject handle to edit2 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.

% See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

% --- Executes on selection change in popSegmentation.

function popSegmentation\_Callback(hObject, eventdata, handles)

% hObject handle to popSegmentation (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popSegmentation contents as cell array

% contents{get(hObject,'Value')} returns selected item from popSegmentation

function popLocate\_Callback(hObject, eventdata, handles)

% --- Executes during object creation, after setting all properties.

function popSegmentation\_CreateFcn(hObject, eventdata, handles)

% hObject handle to popSegmentation (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.

% See ISPC and COMPUTER.

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

% --- Executes on button press in btSeparate.

function btSeparate\_Callback(hObject, eventdata, handles)

% hObject handle to btSeparate (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

%%%%Extract the letters in the plate

global plate;

global letters;

[s1 s2 ~] = size(plate);

i2 = edge(rgb2gray(plate),'canny',0.3); %increase the edges

%separate one letter from another

se = strel('square',2);

i3 = imdilate(i2,se);

i4 = imfill(i3,'holes');

%Connect components

[Ilabel num] = bwlabel(i4,4);

Iprops = regionprops(Ilabel);

Ibox = [Iprops.BoundingBox];

Ibox = reshape(Ibox,[4 num]);

axes(handles.axes3);

out = cell(1,num); %worse case it will have num images as letter candidates

for cnt = 1:num

%the size and area of the image has to make sense...

%the image was resized to [100 NaN]

if (Iprops(cnt).Area > 100 && Iprops(cnt).Area < 1000)

aux = Ibox(:,cnt);

if ((aux(3) < s2/4) && (aux(4) > s1/3) )

%w = waitforbuttonpress;

rectangle('position',Ibox(:,cnt),'edgecolor','r');

out{cnt} = imresize(imcrop(plate, aux),[45 20]);

%%figure; imshow(subImage);

end

end

end

letters = out(~cellfun(@isempty,out)); %eliminate empty cells

% --- Executes on selection change in popRecognition.

function popRecognition\_Callback(hObject, eventdata, handles)

% hObject handle to popRecognition (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns popRecognition contents as cell array

% contents{get(hObject,'Value')} returns selected item from popRecognition

% --- Executes during object creation, after setting all properties.

function popRecognition\_CreateFcn(hObject, eventdata, handles)

% hObject handle to popRecognition (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles empty - handles not created until after all CreateFcns called

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))

set(hObject,'BackgroundColor','white');

end

% --- Executes on button press in btRecognize.

function btRecognize\_Callback(hObject, eventdata, handles)

global letters;

if (~isempty(letters))

axes(handles.axes4);

s = '';

for k=1:length(letters)

handles.axes4 = imshow(letters{k});

pause(1);

s = strcat(s,im2char(letters{k}));

set(handles.edit2,'String',s);

end

else

set(handles.edit2,'String','Error');

end