License Plate Recognizer
Numerical Calculations for Engineering

International Semester

Pelle Reyniers

May 6, 2019

Due Date: 6 May 2019

Class : International Semester

Abstract

TODO:Write abstract of about half a page

Introduction

This document is a report describing a project. The project being reported is part of the course *Numerical Calculations for Engineering*, element of the *International Semester* offered at *Escuela Técnica Superior de Ingeniería y Diseño Industrial*, part of *Universidad Politecnica de Madrid*. This course is presented by *ALBARRACIN SANCHEZ RICARDO* and *CASTANO SOLIS SANDRA*.
The course consists of mutiple lectures and examples all related to Matlab and more imported the link between Matlab and the engineering world. Everything in the subject is teached and showed with real life examples.
The evalution of this course consists of multiple smaller assignments and one large final project. This document describes that final project.
TODO: WRITE ABSTRACT

Contents

Listings

List of Figures

1 Problem Introduction

The general information about the project reported in this documented is presented in this section.

1.1 Description

In recent years or decade, a new way of traffic/speed control is introduced. The so called trajectory control. This way of traffic control exists in monitoring traffic at entry and exit points of a given trisect. With monitoring is meant the identification of vehicles. In this case cameras are used with a computer system that recognizes license plates. By comparing times at entry and exit point an average speed can be calculated. In general this method of speed control is better than normal one point control due to the avoidance of the break-and-accelerate syndrome. This system can be used for a lot more than just speed control. By analyzing data, conclusion can be drawn why people change lanes, etc. Maybe it will become possible to predict traffic jams, accidents etc. Both present systems and futuristic extension relay on the basic principle of licenseplate recognition from images. This is a very interesting and also fundamental subject in the modern world.

1.2 Matlab functionality

The recognizing of license plates is no more than an Image processing problem. This is possible with Matlab because in digital terms a photo is a matrix of values and can be analyzed. Matrix manipulation is very straight forward in Matlab, you could say Matlab is build for Matrix manipulation. In this way is this a perfect example as a Matlab project.

1.3 Goals and Objectives

For a project of any size it is very important to clearly define the goals. This gives a clear view off direction, wheter the project consists of research, development or even experiments.

- Creating a script that manipulates the images so the license plate becomes clear.
- Creating a script that returns the licenseplate when (in string format) when a picture is given as input.
- Creating a script capable of recognizing the land code.
- Provided enough test data and results to confirm all stated conclusions.

1.4 Possible Extensions

Further in this report is a larger section devotad to possible extensions, this is only a fortaste.

1.4.1 Moving Images

In real life this system works with cameras so with image processing on frames. An extension can be giving an input of video files instead of just pictures.

1.4.2 CUDA

CUDA is a programming language built on top of C that lets the user control the Graphic processor. Because image processing contains a lot of parallel calculations, it can be interesting to see what the gain is when executing on a GPU.

2 Problem Breakdown

In this section the general problem is brought down to simpilar and smaller sub problems. The goals of this section is not making more objectives but making a clear and good dividence in the project. Important in breaking down a problem is th clear definition off the sub problems. It has to be very clear what every part consists off and most important when a sub problem is solved. Out off this last defintion follows that it has to be possible to test every different sub problem in a convenient way. To summarize: a project consists off different sub problems which all have the following.

- Title
- Clear and simple objective.
- Orientation whitin the whole project.
- Test conditions.

Multiple sub problems can be combined into a project phase. When talking about project phases, the following phases can be defined, there is a large resemblance with the goals defind in the previous section.

- Image manipulation: Manipulate the input image to a form where it is easy to start the proces of finding the license plate.
- The search: The search off the interire (manipulated) image to the license plate.
- Evaluation: Confirmation or denial.

These project phases each consist of multiple sub problems, these are defined in the following subsections.

2.1 Image manipulation

As earlier described consists this project phase of preparing the image for the search off a license plate. The preparation of this image consists of multiple steps. Each of these steps can be considerd a sub problem.

a. **Uniform:** Transform the image to a uniform dimension. In this way all images from this point on have accactly the same characteristics.

b. **Greyscale:** Transform the image to a greyscale version off itself.

c. **Noise:** Removal of possible noise in the picture.

d. **Edges:** Detecting off all the possible edges in an image.

e. **Clear edges:** Clear the image to make detected lines the only visible things.

f. **Fill edges:** Detect non straight edges and fill them.

2.2 The Search

Before getting into the search, first a few words of explenation about the divide between the image manipulation and search process. We live in a world were a lot of things are the same in the whole world, a few of them are: the use of license plates, the need for speed control, and things in these categories. When looking at license plates, although the concept of a license plate seems universal, it is easy to recognize that they are different around the world. Some of them have only letters, others have two letter and four numbers, etc. Even inside the european union there are a lot of differences. When building a license plate recognizer it is clear to see that a big part of the process will not change: this is the regnizing of edges and filling possible areas. This will always result in an image with intersting areas coloured in, no matter the composition of the licinseplate to be found. It is because of this fact that I divided search and image manipulation. No matter which (country) licenseplate that has to be found. The image manipulation part will always be the same.

a. **Find candidates:** this subproblem is the deliverance of X containers that are possibly part of the license plate.

b. **Test:** testing of the different candidate sets.

c. **Choice:** Make a choice which solution is the most predictable.

These are only three sub problems but they are larger in proportion than the subproblems handeld in the Image Manipulation section. The span multiple functions and files, it is possible to say that each of these problmens have own sub problems. Therefor are they maybe not in line with how I defined subproblems through this project. Further in this subsection follows a more in detail division for every subproblem.

2.2.1 Find Candidates

This subproblem can be furthet divided into the following subproblems:

a. **Regions:** Which regions with containers are intersting.

b. **Selection:** Filter every intersting region.

It is difficult to furder split all the possible subproblems because of the overlap.

2.3 Evaluation

The evaluation part is straight forward. There is only one goal:

a. **Evaluate:** define a function that really tests all parts of the matlab script.

This were all the sub problems for this project. In the next section is every problem going to be discussed in a proper way. It is normal that a lot of the causes are not clear and completely understood. The main objective of this section was giving a brief overview and also some structure to relate back to in future sections.

3 Component Overview

3.1 Project Overview

In this subsection follows a description about the structure of the scripts. The main script is called *ReadLicensePlate.m* and can be found in listing 1 on page 19. In this script are a few things defined:

- The name of the input file.
- The amount of characters in a license plate.
- The amount of letters in a license plate.

The three different parts of this project are really visible in this script. The first part: Image Manipulation is done in a different script but started from *ReadLicensePlate.m*. The second part: The search, also started in *ReadLicensePlate.m*, is also mostly done in other scripts. The last part: evaluation is done outside this script. The Image manipulation script can be found in listing 3 on page 20. This script uses only one custom made function, defined in listing 2 on page 20. Part two starts with calling the function findIndices, the script with the matlab code can be found in listing 4 on page 21. The *findIndices.m* function returns a possible license plate, to get to this license plate the function makes use of an other matlab function called *eliminateOptions.m*. The code corresponding to this function can be found in listing 5 on page 23. Afterwards everything returns to the main script *ReadLicensePlate.m*. This is in large lines the structure of the project. Ofcourse there are other scripts used for extra functions like matching a container to a letter, but the large parts are explained above.

3.2 Image Manipulation Components

Here follows an implementation of all the subproblems introduced in the previous section. Before starting with discussing the different sub problems, the objective of this phase is transforming the image to a new image were the following thing has happend: edgde detection to define possible areas where the license plate can be. To do this, the subproblems defined in the previous section, subsection image manipulation are implemntated. This implementation is shown in the current sub section. The corresponding matlab code can be found in listing 3 on page 20.

3.2.1 Image Preparation

First steps are making the image uniform and removing the color without losing elements in the picture. The color removing is called greyscaling. In the project is a seperate script present that reads and greyscales the image. Therefore can subproblems 1 on page 3 and 2 on page 3 be combined when talking about the implemntation. The matlab code corresponding to these subproblems can be found in listing 2 on page 20. These subproblems are solved in the custom made matlab function called *[rgbImage,greyImage] = greyscale(inputFileName)*. This function has an input argument and produces two output products. As an input the name of the file to read can be found. By making this a variables it is easy to change the change the file name, what is only good for the dynamic carater of the application. When looking at the output arguments, both an rgb and greyscale image can be found. This happens to keep the original in the system while executing the program. When looking at the body of this function, the try catch is present to catch any possible error. Because the project consists of a lot of components, it is a very handy feature to always have a more detailed explenation when the program crashes for one reason or another. While the first lines take care of the importing of the picture and ashoring that it is an rgb version. Line 15 in the code solves subproblem 1 on page 3. The image is resized using the command *imresize()*, from now on all the images are the same size. The lines 18 to 25 take care of the greyscaling. Greyscaling an image is done by seperating the r, g and b channels, multipling each channel by a certain factor and recombining the new value. New value, as in singluar value, because in a greyscale image are the r,g and b values equal for each pixel. The factors used can be found in equation 1.

$$Y_{linear} = 0.2126R_{linear} + 0.7152G_{linear} + 0.0722B_{linear} \tag{1}$$

The value of these factors are defined in the *CIE 1931.* When everything is executed as supposed to, the function now returns both the greyscale and rgb image, otherwise the inputimage is returned twice.
From this point on, a greyscaled image is used in the script. Before starting with edge detection it is usefull to remove the noise from a picture. The noise removal is done with the command *medfilt2()*. This function also accepts an gpuarray input, this is a better and faster way because the matrix manipulations will be done by the GPU. The gpu has a significant more cores to calculate results. The gpu functionality does relie on the CUDA compatability of the gpu in the computer it is running on. Because this is not generally supported, I made the decision to not go forward with gpu implementation in the standerd version of my project. With removing the noise, subproblem 3 on page 3 is solved.

3.2.2 Edge Detection

After preparing the image, it is now ready to start the edge detection. Before discussion the implementation, a short explenation what edge detection is and how it can be achieved. An edge in an image is an area where the rgb pixel values drasticly change. For this reason it was important to remove the color without losing any value of the image.
Imagin not using a greyscaled image but a full colored rgb image. Looking for edges is far more complicated because we have three different channels to take into account. When using a greyscaled image, the r g and b channels have the same value what makes the detection a lot easier. One of the most used detection technics is both dilating and erroding the image. The difference between the two resulting images will result in a very good edge detection.
To succesfull dilate an image, it is necesairy to have a neighborhoud search area, this can be created using a *strel* function. Imagine to have a neighborhoud that looks like a cicrle with radius of a few pixels. This is shown in figure 1 on the facing page. This *strel* is moving over an image, when it covers an aerea with the same rgb value, the system nows it is in an area without a border. But when the *strel* is in an area that is covered with more than one different value, it is clear that their is a border present. When there is a pixel of the *strel* that has a lighter value [1] than the center pixel, the value of this center pixel becomes this lighter value. When this is performed on an image, the lighter areas on the image will become bigger. This is shown in figure 2 on the next page.
To succesfull erode an image, it is necesairy to have a neighborhoud search area, this can be created using a *strel* function. Imagine to have a neighborhoud that looks like a cicrle with radius of a few pixels. This is shown in figure 1 on the facing page. This *strel* is moving over an image, when it covers an aerea with the same rgb value, the system nows it is in an area without a border. But when the *strel* is in an area that is covered with more than one different value, it is clear that their is a border present. When there is a pixel of the *strel* that has a darker value [2] than the center pixel, the value of this center pixel becomes this darker value. When this is

[1]Lighter in greyscale value means a higher value
[2]Darker in greyscale value means a lower value

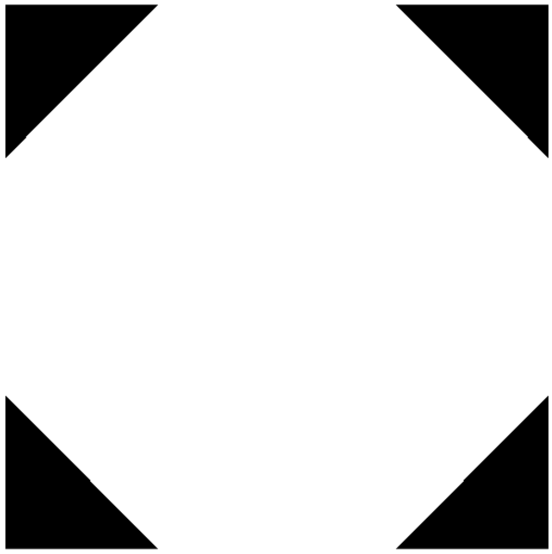

Figure 1: Example of a strel neighborhoud.

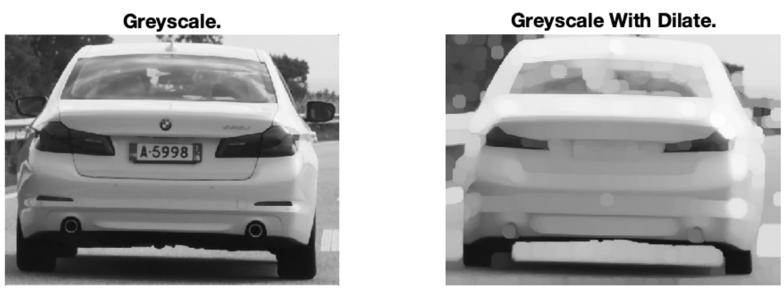

Figure 2: Example of a dilated image.

Figure 3: Example of a eroded image.

performed on an image, the lighter areas on the image will become smaller. This is shown in figure 3.

In the previous paragraphs, dilating and eroding is exagerated. When the radius of the *strel* is reduced to 1, the edges can be exact deteremt when the the results are subtracted. The result of this is shown in figure 4 on the facing page. With this result we can conclude that subproblem 4 on page 3 is solved.

With a quick recap we can recognize the fact that at this point we have an image with recognized edges. However the goal of the image manipulation part is delivering interesting regions where to search for a license plate. To get to that point there are two more necesairy sub problems to solve: clear edges and delete non interesting edges, fill the interesting edges. We start with clearing the image. The start point is an image as shown in figure 4 on the facing page. To make a difference between the different kind of edges, it is necesairy to embrighten the edges and completely endarken the rest of the picture. This is possible by making the contrast bigger and than convert the image to a binary map. By using a binary map, there are only two options: a border or no border, a 1 or a 0. When converting the binary map back to a "normal" image we are now shore we have an image with only the borders/edges present. The result at this point is an image with only clear edges, therefor it is safe to say that the objective of subproblem 5 on page 3 are achieved.

Figure 4: Example of an all edge detection

3.2.3 Filling

As stated in the previous paragraph, the objective of the image manipulation is the presentation of interesting regions. What are interesting regions in the picture? Regions that possible contain a number or letter. A property of numbers and letters is that they mostly do not have a lot of straight lines. especially no single straight lines. With this property in mind we can use another *strel* function to remove straight lines. Instead of using a disk size strel, it is possible to use a linear strel, this will help detect the non interesting edges. With the function *imfill()*, it is possible to fill enclosed regions in the image. This is applied to the image, in theory, every enclosed region can be a letter or a number. Non filled edges are thinnend out with the result that the difference between them becomes bigger. Results off these manipulations are shown in figure 5 on the following page. These manipulations are combinend with the earlier talked about linear edge removal. An example of the final result after image manipulation is shown in figure 6 on the next page. Before starting with the search part, one adition is done to the image manipulation. The function *Iprops()* is called on the final result from the image manipulation. This function produces containers over all the the different interesting parts in the pictures. Two parts of this function are very intersting for the use in this case. The tag 'Bounding Box', this produces practical information about the container in the form of four elements:

- Starting X coordinate.
- Starting Y coordinate.
- Height.
- Width.

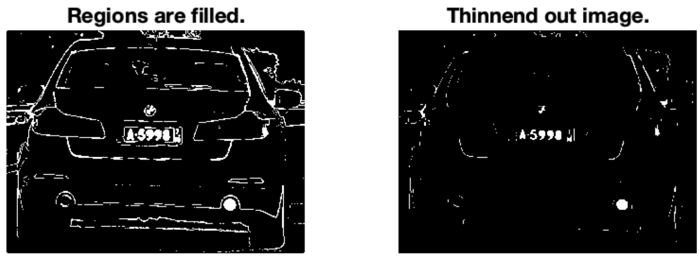

Figure 5: Example of region filling and thinning out the image.

Figure 6: Example of the final result after image manipulation.

In that way are the complete dimensions of the container given. The second tag is 'Image', this produces the image corresponding to the container dimensions. Now all the elements are there to start the search operation.

3.3 The Search

The use of making a difference between image manipulation and the search is explained in the "Problem Breakdown" section. In this subsection follows a detailed explenation of how the search works and how to adapt the search to different licenseplate layouts.

3.3.1 Find Candidates

The main objective is also the biggest question, how do start searching in the picture. We know we have a picture with the regions selected but how can we confident say which regions to skip? The answer to this question can partly be found in the knowledge we have about the containers. We can assume the following: the containers corresponding to the different characters of the licenseplate will be in the samen Y dimension. So if it is possible to filter the containers on Y dimension it will be clear which regions are intersting and have possibly the licenseplate.
From the main script *ReadLicensePlate.m*, listing **??** on page ??, the function *findIndices* is called. In this function, the sorting in regions start. This is possible using th *hist()* function, with parameter the Y starting point of the different containers. The hist function makes a histogram, standard dividing the complete range into 10 parts. This function uses a bucketsort mechanism.[3] After executing this function it is possible to see how much containers there are in every region. Because the specifics of the license plate are given in the main script, it is justified to say that all regions with less containers than total license plate caracters are not interesting. In this way we only keep the regions with a lot of containers.
Only filtering on Y coordinate is not enough. It is safer to make a safety mechanism with another factor. By multiplying the Y coordinate with the width of the container we get a new factor to filter. This second factor is used as a backup when the first filter is unable to give usefull results. After the filtering, all the regions with at least as many items as ther have to be characters in the license plate, are selected. Every region is looked into. The histogram corresponding to the example used in the image manipulation can befound in figure 7 on the next page. The next major challenge is that regions can have more containers than that there are characters. A selection procedure is necesairy to select which of these containers are not part of the licenseplate. This selection is done in the function *eliminateOptions*. The function gets the selection of containers as input as well as the images corresponding to these containers, the amount of characters in a license plate and the amount of letters in a licenseplate. This function is run in a for-loop on every interesting region.

3.3.2 Testing

The testing happens in the function *eliminateOptions*, can be found in listing 5 on page 23. All containers in the region are matched to a character, by the function *readLetter*. This function not only returns the letter that is matched to the corresponding container but also the correlation factor. This correlation factor is very important because after the matching, the filtering has to start. The filtering of the containers is done in a while loop and continous as long as there are more characters left than that there are in a licenseplate. In every iteration of the while loop will the character with the lowest correlation be tracked down and deleted. In this way it is safe to say that the best option for this region will come out of this function.

[3]BucketSort is a fast but not so precise sorting algorithm.

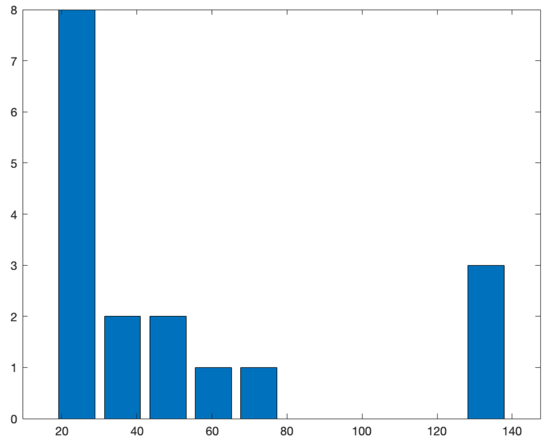

Figure 7: Histogram corresponding to the picture used in the image manipulation example.

3.3.3 Choice

This subproblem is solved in the function *ReadLicensePlate*. After the for-loop done on all possible regions, we have a possible license plate for every interesting region. Except for the license plate, the function *findIndices* also returned an average correlation factor for the complete licenseplate. This average factor is used to select the licenseplate with the highest possiblity. At this point the system returns the licenseplate found in the original picture and this is therefor the end of the system. The system returns the licenseplate in ascii code, what means that these are not the character values but this is easier if the matlab scripts are ever going to be used in another software program. Also most languages support an easy conversion from ascii code to character value.

4 Tests and Results

The main objective of this section is proving my system works. The explenation of how it works is done in previous sections. To prove how the system works I have tested with a couple of license plates to see what the result is.

5 Development proces

In this section the development proces is discussed. Before getting into the details, first a short description on why this is important. Although *Matlab* is a scientific software tool, making projects with *Matlab* can be compared to a software development proces. In software development processes, a very important part is planning and way of developping. The way of developping has a direct impact on the chances to succeed, or chances to meet the objectives. Persnally I have some experience with (small) software projects so I started this project with my experience in mind.

The first and very import stage is the preparation and planning. This is a big stage in the project because the better this is done, the easier it is to make and develop the project. With better is not meant more into detail, with better realistic and thoroughly are the real goals. The proper way of planning can easily be found in section 2 Problem Breakdown. There a devidence in different parts with each own sub problems is clearly visible. This way of working is necesairy to keep track of the complete project and state clear objective goals.

After reading a lot online and in a very interesting book: TODO link image manipulation book, I fastly came to the conclusion that the image manipulation was the best way to start working on this project. In the case of libraries and concepts, this was also the most new part for me. Before working on this project, my matrix manipulation mostly concentrated on signal processing or audio filtering. Without having the exact necesairy knowledge, it was still possible to define an end goal for this part, this because the end goal is straight forward: An image with all parts detected.

The next part, reffered to as the search, is not so much based on new knowledge but more implementation of my personal ideas how this would work. By defining the sub problems as stated, it gives the developper (me) clear small objectives, what is most of all important in parts were the projects builds on own knowledge.

The same planning tactic can also be used for writing the report. Write objectives for your report: list all necesairy items, design a template, make the base structure. Then write the first version, puzzeled into the designed structure.

To keep track of changes and as a safety procedure, I did put the project (code and report) on *GitHub*. Working with a Git server not only provides a back up but the luxury of going back in time when certain mistakes are made.

6 Critical Reflection

7 Conclusion

8 Possible Extensions

References

A Matlab Scripts

Listing 1: The matlabscript corresponding main matlab script.

```
% Pelle Reyniers
% This script is used to extract the license plate out of a picture.
% This is done by following a lot of steps, general control of the
    whole
% process is done in this script.
clear all;

% Part 1 --------------------- Image Manipulation ---------------------

% image name is loaded into the workspace.
imageName='pic4.jpg';
% information about the license plate
totalCharacters = 6;
letters= 3;
% image manipulation script is used.
imageManipulation;

% The funciton regionprops returns information about the containers
    with
% capturing interesting regions in the picture.
% Boundingbox: info about the container
% Image: image for all the containers
ImageContainers=regionprops(final,'BoundingBox','Image');
ICMatrix=cat(1,ImageContainers.BoundingBox); % Matrix conversion

% Part 2 ------ Starting the search for parts off the license plate
    ------
Images={ImageContainers.Image};
[plates,corrs]=findIndices(ICMatrix, totalCharacters, letters,Images)
    ;

if ~isempty(plates)

    [M, I]=max(corrs);
    licensePlateInt=plates(:,I);
    licensePlateInt=licensePlateInt';

    fid = fopen('licensePlate.txt', 'wt');
    fprintf(fid,'%s\n',licensePlateInt);
    fclose(fid);

else
    fprintf('Operation failed, the characters could not be matched to
        make a correct license plate.\n');
end

% Part 3 ------ End of the script, evaluation of result manually
    ------
```

Listing 2: The matlabscript corresponding to the custom made greyscale function.

```
% Pelle Reyniers
% Function used to greyscale an rescale and greyscale an image.
% Funciton returns resized colored and greyscaled image.

function [rgbImage,greyImage] = greyscale(inputFileName)
try
    % read input image
    [X,map] = imread(inputFileName);
    % transform input image to rgb if needed.
    if ~isempty(map)
        rgbImage = ind2rgb(X,map);
    else
        rgbImage = X;
    end
    rgbImage=imresize(rgbImage,[400 NaN]); % Resizing the image.
    [rows, columns, numberOfColorChannels] = size(rgbImage);
    if numberOfColorChannels  == 3 % this doesn't work with RGBa
        % split the different color channels
        redChannel = rgbImage(:, :, 1);
        greenChannel = rgbImage(:, :, 2);
        blueChannel = rgbImage(:, :, 3);
        % make grayscale image
        greyImage = .299*double(redChannel) + ...
                    .578*double(greenChannel) + ...
                    .114*double(blueChannel);
        % Backscale using unint8 for readebility.
        greyImage = uint8(greyImage);
    else
        % image isn't RGB -> return input image.
        greyImage = rgbImage;
    end
catch ME
    errorMessage = sprintf('Error in function %s() at line %d.\n\
       nError Message:\n%s', ...
    ME.stack(1).name, ME.stack(1).line, ME.message);
    fprintf(1, '%s\n', errorMessage);
    uiwait(warndlg(errorMessage));
end
end
```

Listing 3: The matlabscript corresponding to the image manipulation.

```
% Pelle Reynierss
% Part 1: imageManipulation
% This script is ustrelDisk1d to manipulated the input image.
% After this script the image will be in a form that can be read and
% interpreted by the other Matlab scripts later in the project.

% Image manipulation consists of the following steps:
% - Read the image into the system
% - Greyscale the image
% - Remove possible noise
```

```
% - Dilate and Errode the image
% - Detect regions of interest in the picture

% -- Image preparation
% Greyscale the image -> custom made greyscale function
[rgbImage,greyImage] = greyscale(imageName);
% Remove noise -> cpu removal (universal okay), consider gpu removal
    when
% using a dedicated system.
greyImage=medfilt2(greyImage,[3 3]);

% -- Edge detection
% The use of strelDisk1el is explained in the report. High level:
    used for edge
% detection.
strelDisk1=strel('disk',1);
% Image dilation and eroding, use is explained in the report. Both
    are
% combinend to detect all edges.
greyImageImdilate = imdilate(greyImage,strelDisk1);
greyImageImErode = imerode(greyImage,strelDisk1);
% Substraction of the two previous items.
initialEdgeDetection=imsubtract(greyImageImdilate,greyImageImErode);

% -- Clear edges
% Converting the class to double -> increase brightness -> convert to
% logical values -> become a perfect only edges image.
initialEdgeDetectionDoubleValues=mat2gray(initialEdgeDetection);
initialEdgeDetectionDoubleValues=conv2(
    initialEdgeDetectionDoubleValues,[1 1;1 1]);
initialEdgeDetectionDoubleValues=imadjust(
    initialEdgeDetectionDoubleValues,[0.5 0.7],[0 1],0.1);
logicalEdges=logical(initialEdgeDetectionDoubleValues);

% -- Filter edges and fill areas
% Clearing the image from non interesting edges and fill interesting
% regions.
er=imerode(logicalEdges,strel('line',50,0));
out1=imsubtract(logicalEdges,er);
% Filling
F=imfill(out1,'holes');
% Thinning the image to ensure character isolation.
H=bwmorph(F,'thin',1);
H=imerode(H,strel('line',3,90));
% strelDisk1lecting all the regions that are of pixel area more than
    100.
final=bwareaopen(H,100);
```

Listing 4: The matlabscript corresponding to the function used to find the correct containers.

```
% Pelle Reyniers
function [plates, corrs]=findIndices(input, totalCharacters, letters,
     Images)
```

```
% Function findIndices gets as an input a vector with container
    information.
% This information contains location and size of the containers
    containing
% the interesting regions in the image. This function will make an
    educated
% guess on which containers are probably part of the license plate an
      which
% containers probably aren't.
% The output values is an array with indices of the corresponding
% intersting containers.

[n,xout]=hist(input(:,4));
bar(xout,n);
ind=find(n>=totalCharacters);
for a=1:length(input)
    combo(a)=input(a,2) * input(a,4);
end
input2=cat(2,input,combo');
[n2,xout2]=hist(input2(:,5),20);
ind2=find(n2>=totalCharacters);

plates=[];
corrs=[];

if length(ind)>=1
    for x=1:length(ind)
        % find middle point, find width, make box, add result to
            output.
        MP=xout(ind(x));
        binsize=xout(2)-xout(1);
        container=[MP-(binsize/2) MP+(binsize/2)];
        temp=takeboxes(input,container,2);
        [p,c] = eliminateOptions(temp, totalCharacters, letters,
            Images);
        plates = cat(2,plates,p');
        corrs = cat(2,corrs,c');
    end

elseif length(ind2)>=1
    for x=1:length(ind)
        % find middle point, find width, make box, add result to
            output.
        MP=xout2(ind2(x));
        binsize=xout2(2)-xout2(1);
        container=[MP-(binsize/2) MP+(binsize/2)];
        temp=takeboxes(input,container,2);
        [p,c]=eliminateOptions(temp, totalCharacters, letters, Images
            );
        plates = cat(2,plates,p');
        corrs = cat(2,corrs,c');
    end
end
```

```
end
```

Listing 5: The matlabscript corresponding to the function used to eliminate unlikely options.

```
% Pelle Reyniers
function [licenseOut, corrOut]=eliminateOptions(candidate,
    totalCharacters, letters, Images)
% This function is designed to guess every character in a license
    plate and
% reduce the license plate to the size previously stated.
% return the possible license plate and average corr.
tempPlateCh= [];
tempPlateCo= [];
for v=1:length(candidate)
    N=Images{1,candidate(v)};
    [letter corr]=readLetter(N);
    while letter==79 || letter==30
        if v<=letters
            letter=79;
        else
            letter=30;
        end
        break;
    end
    tempPlateCh = [tempPlateCh letter];
    tempPlateCo = [tempPlateCo corr];
end
% eliminate characters with lowest corr until restrictions are met
while length(tempPlateCh)>totalCharacters
    [M,I] = min(tempPlateCo);
    tempPlateCh(I) = [];
    tempPlateCo(I) = [];
end
licenseOut = tempPlateCh;
corrOut = sum(tempPlateCo)/length(tempPlateCo);
end
```