

RoadEye

by

Marcelo Dias Avelino (0840416)



CMI-Program *Computer Science* – Rotterdam University of Applied Sciences

July 13, 2014

First supervisor *Mrs. T. Ubert*
Second supervisor *Mr. A. van der Padt*

Abstract

Nearly everyone carries a smartphone around nowadays. These devices offer a variety of possibilities to tackle a multitude of problems. One of those problems is the difficulty law enforcement faces when trying to find stolen vehicles. This might be solved by crowd-sourcing information gathering in the form of parsing license plates' information using a smartphone's camera coupled with a computer vision algorithm. Using this idea, this thesis is the result of research on the practicability of recognizing license plates using such a device and how to implement such software.

Acknowledgements

First of all I would like to thank Igor Lemm and Heino Peters for giving me the chance of working on this subject. Furthermore I would like to thank Edwin Essenius, Ron Valk and Tanja Ubert for providing me guidance during this project and being there to answer all and any question that I had. To my fellow graduates Samuel Cheung and Nilson Xavier da Luz for making this experience very enjoyable. For all my friends at the computer science course at the Rotterdam University of Applied sciences for managing to help me survive this course for past four years.

Contents

Abstract	ii
Acknowledgements	iii
Introduction	1
Problem	1
Objective	2
Scope	2
Requirements	3
Methodology	4
Research method	4
Development method	4
Algorithms	6
Plate localization	6
Text recognition	7
Design	8
Privacy	8
Application	9
Implementation	11
Bands	11
Plates	13
Text recognition	14
Database matching	16

Validation	17
Distance	17
Angle	17
Reliable area	19
Real scenario	20
Adjacent vehicles	21
Conclusion	23
How can license plate information be gathered from the images of a smartphone camera using software?	23
How can the software be optimized to work in a correct way from within a mobile device?	23
What are the limitations of such an application?	24
Recommendations	25
Distance and speed	25
Angle	25
Text recognition	26
References	27
Evaluation	28
A Legends and acronyms	29

Introduction

This project has been made available by CGI Nederland BV. CGI is a company with a steadfast commitment to corporate social responsibility. Their website states:

“CGI is a company that cares. As stated in our commitments—and as demonstrated through our actions—we take seriously our obligation to best serve our stakeholders and to improve the social, economic and environmental well-being of the communities in which we live and work.” [1]

As a result of this drive, the idea for this project arose: How can people be given the ability to help track stolen cars, and by extension improve their community? Since everyone nowadays carries a smartphone around in their pocket, there is a lot of untapped processing power which could be used for this purpose. This led to the invitation of a student to research and implement this project as the topic for a bachelor thesis. I chose this project because of its relation with Computer Vision, a computer science topic I have worked with before and find interesting.

Problem

Even nowadays, it is still difficult for law enforcement to follow up on every case of stolen property, e.g. cars, simply because they cannot be everywhere at once. A possible way to improve law enforcement’s efficiency would be by including the aid of civilians through the voluntary provision of gathered information from one’s environment. This used to be a very difficult issue because it required people to carry dedicated, and often large, hardware devices which made the gathering of information possible. Since the breaking of the smartphone age, a large number of people carry a computer in their pockets. This Internet-connected all-purpose device allows for a whole range of new possibilities and with the right software, gives the user the ability to help his own community and therefore make it a better place. By using the smartphone’s camera to capture video images, that information can then be paired with a Computer Vision algorithm to give the user ability to help find stolen cars by parsing the information on their license plate. This project will therefore be focused on applying the Computer Vision discipline to solve this problem and try to answer the following question: *How can license plate information be gathered from the images of a smartphone camera using software?*

But this only concerns the problem of this project in a more general way, without taking any requirements or conditions into account. Because the software must run on a mobile platform it must take all the limitations of such a platform or the environment where it will operate into consideration, i.e. limited processing power, battery life, unstable images or car distance. To keep these conditions in mind while developing the application, the following questions will be answered: ‘*How can the software be optimized to work in a correct way from within a mobile device?*’ and ‘*What are the limitations of such an application?*’.

Objective

The objective of this project consists in creating a smartphone software application for the Android platform that is able to locate license plates in video images gathered from the smartphone's camera. From these images, the application must be capable of reading the alphanumeric text displayed on the plates. This information will then be compared to a list of license plate information, which was fetched from a website beforehand. If there is a match, the application must inform a central application of the said match, along with its position when the image was captured and how reliable the recognition is.

Scope

The scope of this project will encompass the development of the Android application which will recognize license plates using the images from the phone's camera, the communication with the web page where the license plate information will be fetched from and the communication with the main application where information over the recognition will be sent to. It will only take into account Dutch rectangular yellow car license plates and might therefore not work with foreign plates. It will take into account, implementation and design wise, privacy concerns according to the Dutch law. The software will only be written and tested for Android version 4.2.2, running on an HTC One X.

Requirements

Even though a general goal and vision are important for guiding which direction the application goes towards, they must be broken down into smaller, more concrete and measurable objectives. These are called requirements and are used to determine the project's progress and whether it was completed successfully or not. The following requirements were deduced from the project's problem and objective:

- **The application must be able to locate the license plate within an image.**

One of the most important functionalities of the application, without this requirement most of the following requirements can not be accomplished. Must find a plate within an image where a plate is present and ignore everything else if no license plate is present.

- **The application must be able to recognise the text displayed on the license plate.**

The other requirement which ties the whole application together. Once the plate has been located, the text contained in the plate must be parsed so the application possesses information to work further.

- **The application must be able to retrieve a list of license plates from a designated website.**

For the application to be able to provide useful information about a certain vehicle, it must have information about stolen vehicles to match the recognised plate text against. Therefore it must be able to retrieve that information from the place where it is available, in the case a website.

- **The application must be able to match the recognised text in the license plate to the retrieved list.**

The application must be able to match the retrieved information of stolen vehicles with the recognised text.

- **The application must be to notify a central application whenever a match has been made.**

When there's a match between the parsed information and the information retrieved from the website, the application must be able to send a notification about this information to a central application.

- **The application must be not save any information/images longer than necessary.**

As to minimise the chance any privacy law will be broken, the application will not save any form of data which might be used to identify someone. The reason for this is explained in a later chapter.

Methodology

Before starting the development, the methods must be found which will be used to find viable ways of solving the problems presented by the requirements and how to implement them once the correct solutions have been found. For this reason some time was first spent searching for possible research and development methods. In the end the following choices were made.

Research method

The method applied for researching the problems of this project and its possible solutions is called *literature review*. This method consists of researching what has already been published, which might be in the form of scientific or engineering papers, journals, thesis, etc., by accredited scientists, scholars or engineers concerning this assignment's topics. This method is applied for searching for potential algorithms which can be used to solve the problems created by the requirements. Once a group of the most suitable algorithms has been found, the best one must be chosen and the reasoning for this choice must be explained. When the choice has been made, the algorithm is then implemented using the chosen development method.

Development method

This project will be developed using the *Iterative Application Development* (IAD) method. This development method works by dividing the project into smaller ‘sub-projects’, called *cycles*, and incrementing them to past cycles, which will ultimately lead to a complete system. Each cycle consists of three phases, which can be repeated multiple times if necessary, called *iterations*. These iterations are: *definition*, *development* and *deployment*.

During the definition phase the goals, limitations and conditions for the current cycle are examined and described. If a previous cycle has been completed, it will be evaluated during this phase. This phase is intended for thinking towards the completion of the project and to achieve a more clear picture of the system as a whole. After defining the objective for the new cycle, the software will be developed. After finishing, the software is then integrated with the software developed in the previous cycles and becomes therefore part of the general project.

This method of software development brings multiple benefits: The complexity of the project is decreased by breaking down the problem into smaller chunks, which allows for faster and more concrete results and makes it therefore easier to get better feedback or to solve critical bottlenecks by being able to discuss them at the end of each cycle. The project development also becomes more flexible by having the possibility to review the requirements and strategies every cycle.

Each cycle lasts 2 weeks and at the end of each cycle the evaluation of the past cycle and the objective for the coming cycle will be discussed with the organisation's mentor.

Algorithms

After reasoning about what kind of main algorithms and researching which software could be used to develop the application, the problem was then broken down into two different categories: localising the plate within an image, because it is a broad problem which can be solved in a different number of ways, and parsing the text within the plate, which is a more specific problem and there are available libraries which can be promptly used to solve this problem. After this realisation and researching the possibilities, the following choices were made:

Plate localization

When searching for possible algorithms which make it possible to find license plates in an image, two main types came forth from the research: feature detection and edge detection.

The feature detection algorithms work by finding so called *features* in a image, which are used to recognise the first image within a second one. These features are segments of an image which must be uncommon, as to reduce the possibility of retrieving a false positive when applying the algorithm, and also consist of something which can be objectively described to a computer. Because of these requirements, the features extracted from images are usually corners since corners usually only match themselves when compared to other segments in an image. This opposed to flat surfaces or lines, which may appear multiple times in multiple places in the same image. Because this algorithm focuses on detecting the uniqueness of an image and using those attributes to detect themselves in different images, it is difficult to use feature detection for the recognition of license plates for the reason that every license plate contains unique text. The considerable collection of diverse shapes that exist in the Latin alphabet create false positives which are often detected in random and incorrect locations. One possible approach to use this algorithm would be by creating a feature database of every possible alphanumeric character and then finding the highest concentration of text as a possible location.

The other possible algorithm is mostly based on edge detection. This kind of algorithm works by applying an edge detection filter to a grey scale version of the image where the car is present. This creates a binary image where the edges of every shape present in the image are displayed. Because of the nature of one of the characteristics common to every license plate, which is the presence of text, an area with a high density of edges is created. Although license plates are not the only objects which might have such a property, e.g. a fence, it is the most common one which might be encountered while driving. By applying this filter to find horizontal edges it is possible to find the vertical location of the plate and then vice-versa to find the horizontal location and by extension the plate itself. Due to little information on the performance of the feature detection algorithm and a healthy amount of information regarding this one, this algorithm was chosen.

This algorithm was implemented using Open Computer Vision (OpenCV) library which is a library that contains optimized state-of-the-art computer vision algorithms. This library is maintained by Intel, a company known for its mathematical knowledge, is completely free, as in beer and as in freedom, and has an Android version. Because of these reasons and prior experience working with this library, OpenCV was chosen. Although the application was implemented using this library, this thesis explains the algorithms usage in a general, OpenCV-agnostic way as to be easier to understand the algorithms without the need for the library.

Text recognition

Text recognition is a specific problem which will only vary slightly from project to project, which usually means it has to be trained to recognise a different font or the text to be recognised has a special orientation, e.g. vertical or diagonal. There are libraries which have this specific functionality along with enough customisability to adjust the library to the specific needs of a project. There is for this reason no need to create an own text recognition algorithm. From the available options in OCR technology, the Tesseract Optical Character Recognition (OCR) was chosen because it is maintained by Google, a company known for its OCR technology, because it is one of the most accurate free OCR libraries available, and because it also has an Android version. It was chosen because no there are no other free library available which can deliver the same type of results. The implementation of the text recognition is Tesseract specific because, as opposed to the plate localisation algorithm, it uses a very specific software package instead of a collection of algorithms.

Design

Before the application can be built, some sort of blueprint must be created where the functionality, the flow, and a variety of other specifications are incorporated and described. This is called the design of the application and it outlines what each component does, how they are interconnected, and the reasoning behind each choice.

Privacy

Privacy has always been a hot topic within the computer science circle of professions with computers becoming faster each year and all sort of algorithms being created which have the potential to invade the privacy of thousands if not millions of people. As of the writing of this thesis, a chain of events have gotten this kind of controversy to a new all-time high. To prevent that this application will ever create any privacy concerns, some research was done on the Dutch privacy law of personal information as of 2014. According to this law, it is only possible to transgress if, while collecting license plate information, the following three requirements are met. The information must be:

1. personal information

The information gathered must qualify as personal information. This means the information must directly link to a certain person, e.g. phone number, address or name but also pictures of people's faces. The only instance which has access to the personal information linked to a license plate is the *Rijksdienst voor het Wegverkeer*, which is a government instance, and therefore the only instance which would have to comply to the privacy law. It is possible to access that information through their services, but you need the information's owner's permission and have to pay a fee, which means accidental transgression of the law impossible is.

2. processed

Processing information is a very broad concept, which can be anything from using the information for a website or just matching it against a database.

3. saved in a file

This can be a simple spreadsheet or a full-fledged database.

This application will only meet the second point, which means it will not violate the Dutch privacy law. To add some additional certainty, the application will only save the license plate information for as long as necessary, will not save any pictures taken after they have been processed and the user will not have direct access to that information.

Application

The top-level flow of the system consists of three individual components: the website where the license plate information of stolen cars is available, this android application which uses the information from the website to match that information with captured video images, and a central server where, if a match has been found between the previous two stages, the match information is sent to. A diagram of this flow can be seen in Figure 21.

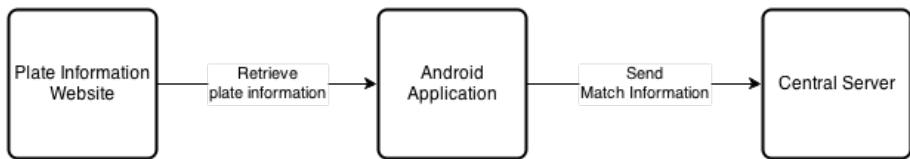


Figure 1: Top-level diagram.

Although all those three components are part of the whole system, this project only focuses on the application itself. A flowchart illustrating the general design and flow of the application is displayed in Figure 2. Within an Android application all components run in the same thread by default, which is also known as the ‘main thread’, and is identified in the flowchart as ‘UI’ because it is used mainly to regulate all the processes related to the user interface. One of those processes, and also one of the most fundamental running in this thread, is the process which captures images from the device’s camera. The process captures image frames at a set rate in order to avoid the unnecessary capture of very similar images and, because image processing algorithms consist of a large number of calculations and therefore might take a long time to finish, to avoid overwhelming the application with too many frames. These images are then pushed into a buffer, called ‘Frames buffer’, where they are kept until they are able to be processed. For the same reasons the camera only captures frames at a specific rate, this buffer has a limit of how many frames can be stored within but with the intention of only preserving images which are as recent as possible, before a new frame is added the oldest remaining frame in the buffer is removed. Further, the UI thread also manages the retrieval of the license plate information from the specified website. This process is repeated after a specified amount of time, the default being 30 minutes. This process runs until the application is terminated.

As explained before, image processing is rather heavy and it is necessary to use every tool available to increase the application’s performance. One of these tools is multi-threading and apart from the UI thread this application uses two other kinds of threads, up to a total of four threads which is also the number of physical cores in the used device and therefore gives every thread a dedicated core. The processes called ‘Find bands’ and ‘Find plate’ use a type of thread called a *One-Call Thread*, which are threads used for a short period of time and only run when called. The other kind of thread is called a *Permanent thread*. This type of thread is started during the initiation of the application, runs parallel to the UI thread and lives as long as the application itself. This process is responsible for checking whether the bands and plates buffers have available items and for calling the respective processes to process them. Further, if a plate is found then this thread applies text recognition to retrieve the displayed text and tries to match it with the database entries. If there is a match, the application will save the plate information for a defined amount of time, the default being 30 minutes, to prevent sending the same or very similar information about the whereabouts of the vehicle every time the application recognises the plate. Furthermore,

the plate's information will be bundled together with the current location of the smartphone, the current time and the confidence of the recognition. This bundled message will then be sent to a central server where it can then be distributed to the correct authorities. After this has been done or if there wasn't a match, the system starts again from the beginning.

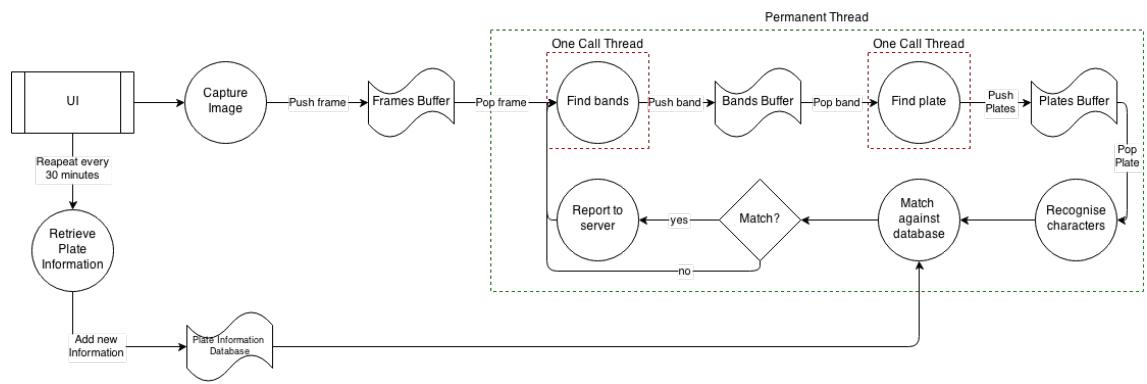


Figure 2: Application flow diagram.

Implementation

The application consists of three main components: the algorithms for finding the vertical and horizontal locations of the license plates, and the text recognition component. Both localization algorithms are based on the work of Martinský Ondrej [2], with some slight modifications, which are explained in the following chapters, to fit the requirements of this application. The algorithms use the car in Figure 3 to demonstrate how they work.



Figure 3: Source car image. [3]

Bands

The first step to locating a license plate is to find its vertical location, which is also known as a band. Using one of the most characteristic features of a Dutch license plate, which is its yellow colour and can be seen on Figure 4, it is possible to reduce the area of the image where the license plate might be located through the application of colour segmentation.



Figure 4: Example of a Dutch license plate. [4]

Colour segmentation consists of fetching only the areas of an image which fall within a specified colour range and is usually used on an image which uses the HSV colour space representation. As seen in Figure 5, HSV stands for Hue, Saturation, and Value and it is a way of representing colours on a computer. This in contrast to the most widely used colour space, the RGB colour space (Red, Green, Blue) which uses a combination of those very colours to represent a specific

colour. The HSV colour space on the other hand uses the H-value to represent the pixel's colour and is defined in degrees (0° - 360°), the S-value to represent how bleak or how colourful the pixel is and is defined with a percentage (0% - 100%), and the V-value to represent its brightness or darkness and is defined the same way as the saturation. This provides a system where it is simple to choose a colour range for the colour segmentation algorithm and is also the standard system for these kind of operations. The range chosen for the license plate lays between 40° and 50° , which is broad enough to take into consideration the deviations in the colour of the plate caused by shadows and reflections.

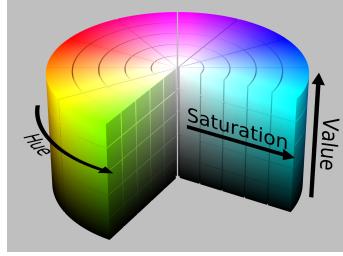


Figure 5: HSV colour space. [6]

As a result of segmenting the yellow colour from the source image, the image in Figure 6a is created, where the plate's location is quite visible. Often the segmented image also contains speckles from various small objects present in the image. These speckles are also visible in the previously mentioned image and might cause incorrect results when trying to locate the plate. To prevent this, a morphological closing operation [7] is performed. This operation works by dilating the image, which actually dilates its background and therefore removes small objects but also reduces the area of the plate, followed by eroding the dilated image, which does the inverse of the previous operation and returns the plate to its original size while the small objects are still gone. Further, because there are such things as yellow cars and other yellow objects, an extra step must be taken to decrease the chance of finding random objects. This is accomplished by making use of a characteristic of text and by extension of license plates, which is a high number of edges. By applying an edge detection algorithm, in this case the Sobel operator [8] was used and its output is visible in Figure 6b, we still get a visible area where the plate is located but in case the source image contained large yellow areas, these areas are now mostly gone.

Once that is done, the application can start analysing the previous image for the license plate's possible location. This is done by summing up the intensity of every pixel on each row and therefore creating a graph where the density of edges on each row is displayed. Before making any decisions concerning the plate's location on the graph, a filter must be applied to the data to remove unwanted data and increase the desired area slightly which is necessary for a later stage. This filter is called a *mean-filter* and works by summing up the values of a specific number of the previous and next data values, in the case of this application seven values are used, with the current value and diving it by the total number of values. The results in the 'smooth' graph displayed in Figure 6c. The next step and last step is finding the potential plate areas and this is done by searching for a number of the highest peaks in the graph, with three being the default. The algorithm tries to find multiple possible locations in case an object in the image happens to have a higher edge density than the plate, and therefore increasing the chance of still finding it. The way the application finds peaks is by finding the highest value in the data set and then searching for the right and left boundary of the peak by iterating towards its respective side until it finds a value

which is equal to or smaller than 10% of the peak's value. The coordinates of both boundaries are then saved and the area in between both coordinates is made equal to zero, or *zeroised*. By rinsing and repeating this process for the chosen number of times, the bands displayed on Figure 6c are found.

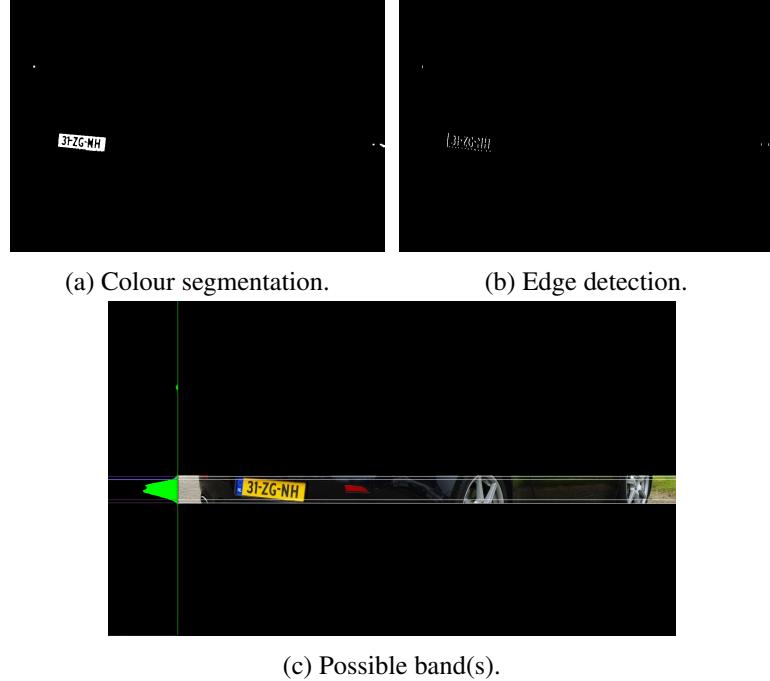


Figure 6: Phases from band localization.

Plates

After finding the bands, these can be further processed to find the exact location of the plate. Using the binary version of the band's image, a graph can be generated just like before, but instead of summing up all values on each row, the values of each column are now summed. Once the graph has been created, the mean-filter is also applied for the same reason as before. Using the filtered data the plate is then located by searching for the largest cluster of peaks, as shown in Figure 7, which are equivalent to the letters on the plate. The algorithm works by iterating the graph until a data cluster is found, from which the algorithm then saves the start coordinate and keeps iterating the remainder of the graph until it reaches the end of the current cluster. The size of the cluster is then saved and after repeating this process for every data cluster present in the graph, the largest one is chosen as the plate's location.



Figure 7: Horizontal localisation of the plate.

This gives an image of the plate with a rather larger border of undesired parts of the image, as shown in Figure 8a. The reason for this is that depending on the angle at which the image was captured at, the plate might be skewed and the text won't therefore be at a 90° angle. This makes it difficult for the OCR software to recognise the text properly. To correct the skewness the bounding box of the plate must be calculated, which is a quadrangle that surrounds the plate's area and from where its four vertices can be retrieved. The box can be seen in Figure 8b. Using these points, the angle at which the plate is skewed can then be calculated and the image can then be de-skewed using the warp perspective algorithm [9] as displayed in Figure 8c.

Finally, only a few last steps are necessary to complete the localisation and processing of the plate before the OCR software takes over. But before, a simple filter is applied to the plate candidate to filter out wrongly detected plates. This is done by calculating the ratio between the width and the height of the plate. A rectangular Dutch car license plate has a size of 52cmx11cm [10]. This means the width to height ratio is around 4.7. By using a range between 4 and 5 to compensate for some deviations caused by the angle the image was captured at, most incorrect plate candidates are filtered out. If the plate's ratio is within the right range, the plate can be further processed. Using the de-skewed plate, a binary image must then be created to allow for the removal of unwanted artefacts still present on the image, e.g. borders. The binary image is created by applying a threshold algorithm onto a greyscale version of the plate, which converts every pixel with an intensity value above a certain threshold to a black pixel and everything else to white. The used threshold algorithm is an adaptive threshold algorithm, which gets its name because of its ability to adapt the threshold value of a pixel according to the neighbouring pixels. In this case the algorithm calculates the mean value of the sum of the pixels around the threshold pixel. This gives a better result than using a fixed value, because the lighting condition is not the same on every image, or a threshold algorithm like Otsu's method [11] which requires a distinct difference between foreground and background, which is not always available in the case of a license plate. After applying the threshold algorithm, a binary image is created containing the background in white while the text and eventually some artefacts are in black. The last step left to take is to discriminate the text from everything else and therefore creating an image fit for text recognition. This is accomplished by applying a contour finding algorithm [12], which finds the outline of every item in the image. By using only the contours which are enclosed within another contour and in turn do not contain other contours within themselves, as is the case of the letters' contours as they are located within the contour of the license plate and contain no further contours inside them. This is possible because the contours algorithm creates a hierarchy list of every contour and whether that contour has a parent and/or child. One last filter is applied to the characters to filter out unwanted artefacts or images which are not plates. This is done by sorting the contours by size and only displaying the largest six contours present. Six was chosen because Dutch license plates only consist of 6 alpha-numeric characters. If there are more than six contours, the unwanted contours are then filtered out, and if there are less than 6 contours, there's a large chance that it's not a license plate. This results in Figure 8e.

After this whole process, the plate is now ready to have the OCR software applied to it.

Text recognition

Although the Tesseract library works really well most of the time, it requires some training in some specific cases. The text displayed on license plates is one of those cases because it uses an unique font designed specifically to be used on plates. There is no official version of this font publicly

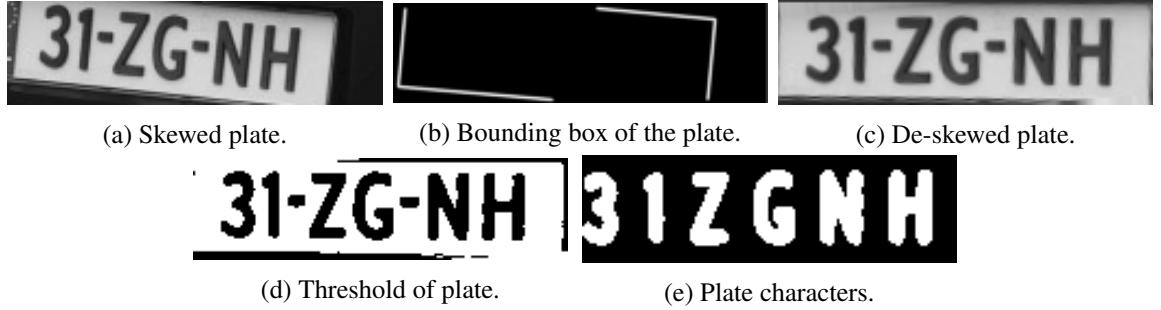


Figure 8: Phases from the plate de-skewing process.

available, but using LeFly's [13] interpretation it was possible to train Tesseract to recognise this type of text. The font used for the training can be seen in Figure 9. Not every character has been added to the training text simply because not every character is used in license plates. The reasons for this ranges from characters being reserved for special plates, e.g. the royal family or diplomats, to characters that are too alike, e.g. the 'I' and the '1' or 'O' and the '0'.

BDFGHJKLMNPQRSTVWXYZ0123456789

Figure 9: License plate font.

Using the image displayed above and the external tools provided by the Tesseract library, training data can be generated. Tesseract does this by segmenting every item on the image and creating a file with its best guess at which character is represented on the image, its location on the image, and its size. This data is displayed in Figure 10. After this file has been checked for possible mistakes, it can then be used to generate further necessary files where information on the shape of the characters, etc. are stored. Finally all the data is then bundled into a single file which can later be used with the recognition software.

Char	X	Y	Width	Height
B	104	109	20	31
D	130	109	22	31
F	158	109	18	31
G	182	108	21	32
H	209	108	22	32
J	237	108	15	32
K	259	108	21	32
L	287	108	18	32
M	311	108	26	32
N	342	108	22	32
P	371	109	19	31
R	396	109	22	31
S	424	108	21	32
T	450	109	21	31
V	476	108	22	32
W	503	108	22	32
X	530	108	22	32
Y	558	108	21	32
Z	584	109	19	31
O	610	108	22	32
I	638	108	8	32
1	654	108	20	32
2	679	108	18	32
3	703	108	22	32
4	732	108	20	32
5	758	108	21	32
7	785	109	19	31
8	810	108	21	32
9	837	108	21	32

Figure 10: Font data.

```
1  {
2      "Plate": "XX-123-Y",
3      "Date": "01-01-1970 00:00",
4      "Latitude": "76.92061352",
5      "Longitude": "41.04492187"
6  }
```

Figure 11: JSON annotation example

Now if Tesseract is applied to the plate in Figure 20b using this training data, the results will be much better than without it.

Database matching

Once the plate has been found and the text has been parsed, the only thing left to do is to see if it's one of the cars the application is looking for. This component consists of two separate parts: matching the parse text with the database and updating the database.

First the application must retrieve a list with information about license plates of cars which are being searched for. Without this step it won't be possible to match the parsed text. This list is retrieved from a specific website using an HTTP socket. The website used for this application was a local dummy website. The information was stored in the JavaScript Object Notation (JSON) format. This is a simple way of storing and sending information objects between servers and clients. Once the information was retrieved from the website, it is stored in a file on the smartphone's storage. This process runs on a timer, which repeats itself after a specified amount of time, 30 minutes being the default.

The second part of this component is very simple. Once the text has been parsed, the application runs the parsed information through the database looking for a match. If a match is indeed found, the application will then retrieve the current GPS location and bundle it together with the plate information and the current time in a new JSON object and will then send it again via a HTTP socket. An example of a JSON annotation is displayed in Figure 11.

Validation

To define where the strengths and weaknesses of the software lie, some validation tests must be performed. These tests will provide information which in turn will be used to answer the research questions stated at the beginning of this thesis and, by extension, draw a conclusion on whether the project was successful. These tests are focused on three elements: the maximum distance and angle at which the application is able to present reliable results and how well it performs while inside a driving car.

Distance

The validation phase where the distance limitations of the application are tested, perform these tests at 4 different distances while standing straight in front of the vehicle: 3, 4, 5, and 6 meters. The results are displayed in Figure 12. On Figure 12a, the text recognised is rather sharp and the OCR software has absolutely no problem parsing it. But the further away we go, the less and less sharp the text gets. Figures 12b and 12c still look clear and are parsed well, but once we get to the around 6 meters of Figure 12d the parsed text starts getting some inconsistencies. Although it might still be possible the application will parse the text correctly, more often than not it will return incorrect results. The maximum reliable distance lies therefore between 5 and 6 meters.

Angle

To find what the maximum angle is at which the application still works reliably, various angles were tested in an incremental way. Starting at the 6 meter 0° mark, because 6 meters was the last position the application worked at this angle, a small step was continuously taken in a circle around the vehicle and the application was tested. If no angle worked at this distance, the same was then repeated at a closer distance. By repeating this process, the boundary of the reliable working area was found. As shown in Figure 13a, the application was able to parse the text correctly, but in Figure 13b which is only a step further to the right, the application didn't work correctly. This point was located at a distance of 4 meters from the car and at an angle around 47° .

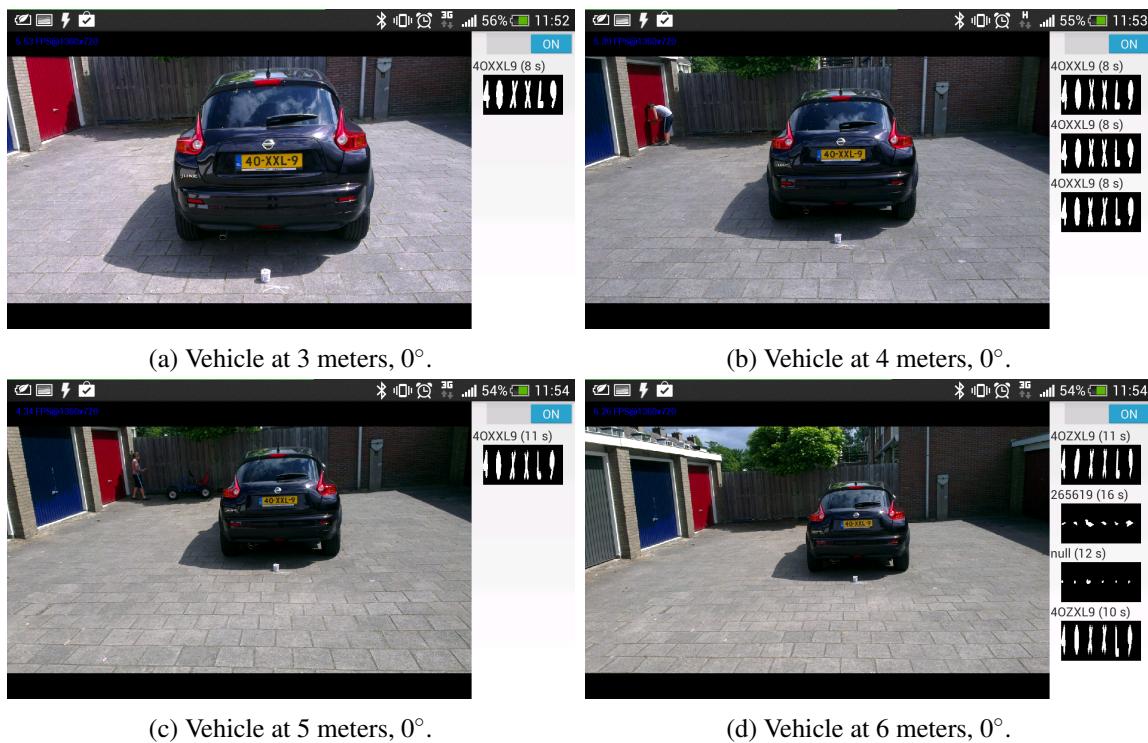


Figure 12: Results at varied distances.

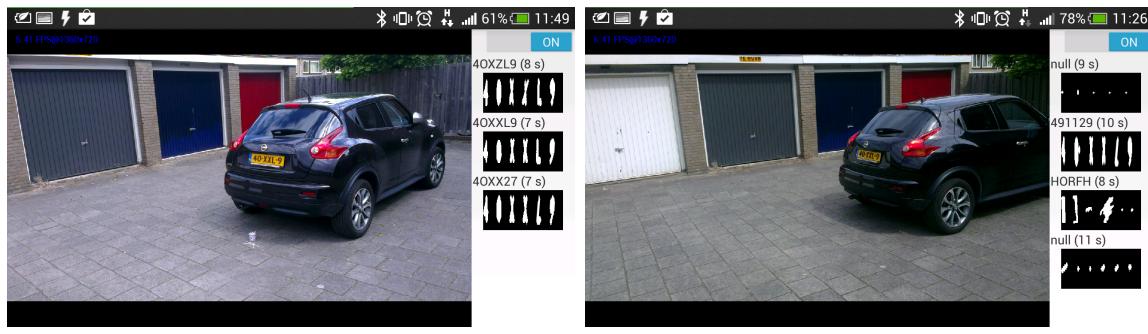


Figure 13: Results at varied angles.

Reliable area

Using the information gathered using the past two tests, an area can be derived within which the application works well and reliably. This area is depicted in Figure 14 and the scenario was created using the road design criteria from the province of Zuid-Holland [14]. Looking at this figure it's clear that the application can process the plates of vehicles both in the same lane, at a distance of 5 meters, and in the adjacent lanes, at a distance of 4 meters, while driving at a safe distance on an inside road or during a slow commute on the highway.

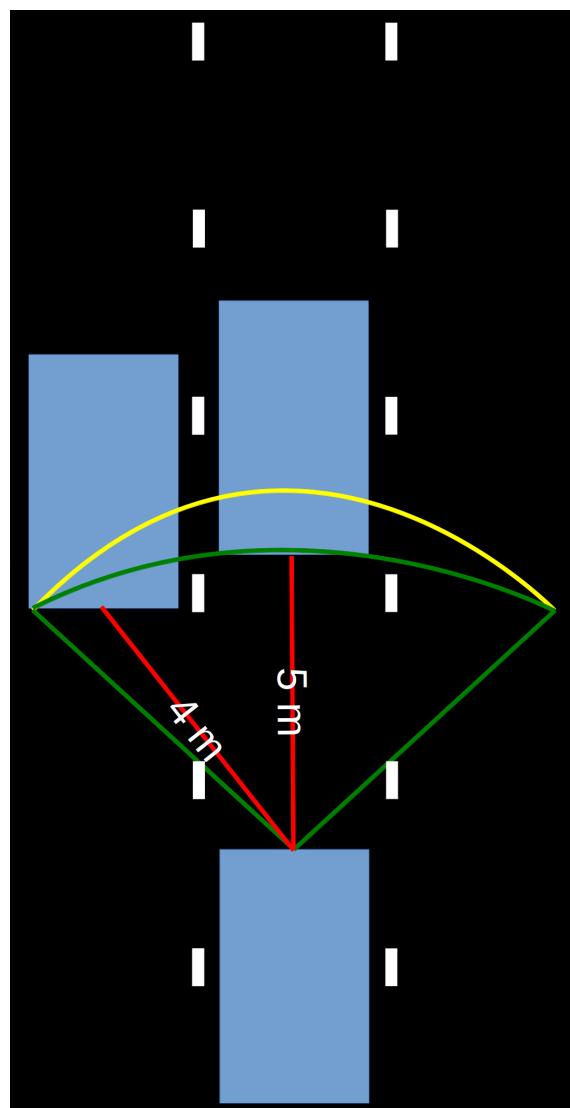


Figure 14: Reliable detection area.

Real scenario

Once the distance and angle limitations of the application have been found, they must be tested on a real scenario to see how well the application handles real usage. This was tested while inside a car driving normally on an inside road. One of the situations encountered is displayed in Figure 15a. The vehicle in front of the camera is within the distance limit of 5 meters and the application is able to find the plate perfectly. But even though the plate is within the reliable detection area, the text is incorrectly parsed. The application mistakes the letter ‘S’ for the number ‘5’ or even the number ‘3’. This exposes a limitation that might impact the accuracy of the application.

One other situation the application isn’t able to handle correctly is displayed in Figure 15b. Even though the car in the lane left of the application is within the reliable detection area, the application is not able to find the location of the plate, let alone parse its text which in this case is not even visible. This because the plate on the image appears as only an yellow blur without any visible text. This is a result of the combined speeds of both the car where the application is located and the incoming car making it impossible for the smartphone’s camera to capture a useful image.

The last situation, depicted in Figure 15c, proves that if a car is indeed at a distance larger the calculated reliable distance, that the application is not able to find its location.

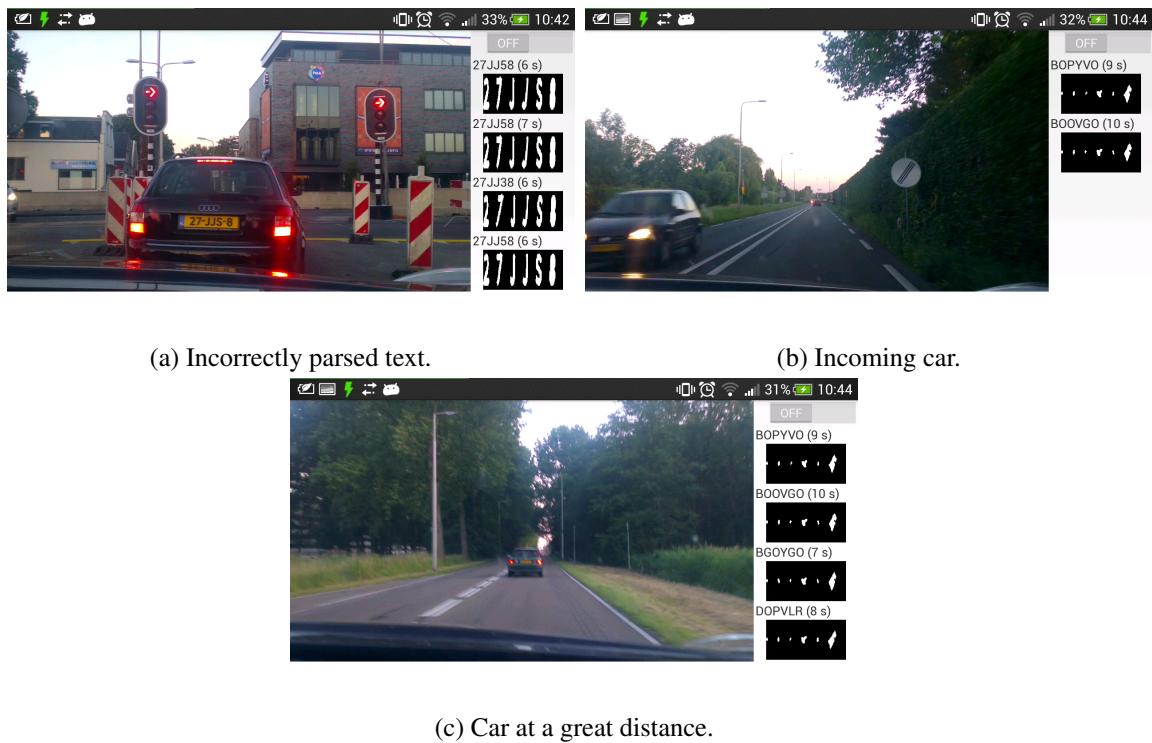


Figure 15: Results at varied angles.

Adjacent vehicles

One other possible problem of this application is attributed to the way its plate location software works. By only retrieving the largest ‘plate’ on each band, it is possible that the application will miss a plate if the vertical locations of two car plates happen to overlap. To test this problem an image was created, displayed in Figure 16, with two different cars next to each other with their license plates at around the same height. Further, another image was created, displayed in Figure 18, with the same cars but then at different heights to test if both plates are located if the vertical locations of both plates do not overlap.



Figure 16: Reliable detection area.

Starting with the image with the overlapping plates, this image was processed using the application and the result is displayed in Figure 17. Because the algorithm only looks for the largest group of peaks in each band, it also only finds one of the bands present in this band. In this case the right plate happens to be larger than its left counterpart and is therefore indeed the only one chosen. This reduces the amount of plates retrieved from camera images in these kind of scenarios.



Figure 17: Reliable detection area.

To test whether the application is able to find multiples plates within an image if those plates’ vertical locations do not overlap, the second image was processed using the application.



Figure 18: Reliable detection area.

Because the plates' vertical locations do not overlap, they end up in different bands. Both these bands are displayed in Figure 19. As it's possible to see in both displayed images, the application is undoubtedly able to locate both plates. This reduces the possible scenarios where the application will only find one of the multiple plates present to the scenarios where the plates are located in overlapping vertical locations.

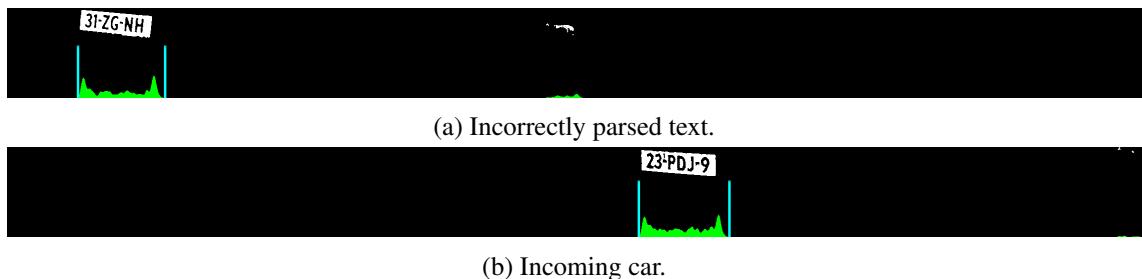


Figure 19: Results at varied angles.

Conclusion

At the start of this thesis the following questions were asked: ‘How can license plate information be gathered from the images of a smartphone camera using software?’, ‘How can the software be optimized to work in a correct way from within a mobile device?’ and ‘What are the limitations of such an application?’. Using the information gathered during the course of the project, these questions will now be answered.

How can license plate information be gathered from the images of a smartphone camera using software?

Because this project was focused only on Dutch yellow rectangular car license plates, the algorithm was therefore specifically created to recognise this type of plates. The first step is to remove everything in the image that is not yellow. This leaves a binary image where in most cases a clear area where the plate is located is present. Yet because there are yellow cars or other objects with large yellow areas, before grabbing that area as the plate’s location, an edge detection algorithm is applied to the the binary image to retrieve only the edges of the objects. The reason for this is to remove those large yellow areas. To increase the chance of finding the correct location, one of the characteristics of a license plate is used; a high density of edges thanks to the presence of text on the plate. Once the edges image has been created, all the values are summed up per row. This creates a graph with which the vertical location of the plate can be found by searching for the broadest group of peaks. By applying this method to the three broadest group of peaks present in the graph, three possible vertical locations of the license plate are found, also known as ‘bands’. The reason for multiple bands is that by looking at multiple places on an image, the chance of finding the correct location of the plate rises. By applying the same method but now for every column on each band and applying some filters to filter out non-plate objects, the full location of the plate is found. It’s possible the plate is rotated depending on the angle the image was captured at. To solve this, the angle of the bounding box of the plate is calculated and transformed. The plate has then a contour finding algorithm applied to so the characters within the plate can be retrieved. Once the characters have been retrieved, the image is then passed to the text recognition software to parse the text.

How can the software be optimized to work in a correct way from within a mobile device?

Because smartphones are limited devices when it comes to processing power, when creating such an application these limitations must be taken into account. This was solved using two different

approaches: multi-threading and buffers.

Starting with multi-threading, this technique allows an application to share its processing load with every processor core in the smartphone. The application uses four threads for its main four components: the UI thread, the band localization thread, the plate localization thread and the text recognition thread. This allows for, in the case of the smartphone used to test the application, a dedicated core for every thread. Every time there is a new item available in one of the buffers, the respective thread is executed with the new data.

The buffers approach takes into consideration that a smartphone does not have enough processing power to process images at real-time. The outputs of every thread, with the exception of the text recognition thread because its output is handled immediately, are therefore stored within a buffer so they can be processed in their own time. There are 3 different buffers: the frames buffer, the bands buffer, and the plates buffer. The frames buffer stores the video frames capture with the smartphone's camera and has a maximum size of 1. This prevents the accumulation of frames which were captured shortly after each other which leads to a high chance of processing the same plate multiple times. The other two buffers store bands and plates respectively and have no size limitations because when the initial frame has been processed, the follow-up data is considered important and is processed evenly.

Using this two methods it's possible to create a functional application despite its need for a large amount of processing power.

What are the limitations of such an application?

Because of the nature of this application, it brings a couple of problems to its usability. One of these problems is the distance at which the application can still process a plate reliably. After performing tests to find this maximum distance, the results indicated the application was still able to provide reliable results at the distance of 5 meters, while at the distance of 6 meters it was still able to sometimes provide correct results but not in a reliable manner. Another problem is the angle at which the results are still processed correctly. Tests demonstrated that the maximum angle is located at around 47° at a distance of 4 meters. Using this information, a model was created consisting of a lane with the car the application is inside of and two adjacent lanes. This model was created using the correct ratios between each item and shows an area where the application is able to detect license plates reliably. This model also proves that the application is able to read the plates on cars on the same lane at a maximum distance of 5 meters and on adjacent lanes at a maximum distance of 4 meters.

After this detection area was found, some tests were performed while driving inside a car and these tests revealed some additional limitations. One of those limitation is that even though a car is within the reliable detection area and the plate can be found correctly, it is still possible the plate's text is parsed incorrectly. This is owed to some characters used in license plates still being to similar to each other, e.g. the 'S' and the '5', and depending on the scenario, the application might have issues distinguishing between them and therefore providing incorrect results.

Another of the found limitations was when trying to retrieve the text displayed on the plate of an incoming car. Because of the speeds of both the car where the application is located and the incoming car, it's impossible for the application to get a proper image where the plate of the incoming car is visible. Usually the plate displayed on that image is just a yellow blur with no text visible and it's there impossible to parse its text.

Recommendations

Distance and speed

The distance and speed at which the application is able to capture images is entirely dependent on the hardware of the smartphone. A better camera allows for either a very high resolution to be used at a decent frame rate, which would improve the distance and the quality of the text recognition, or a lower resolution with a higher frame rate, which would allow the capture of plates from cars which are driving in the opposite direction. Beware that with a higher resolution, the processing time of the application would rise significantly.

Angle

To improve the angle at which the application is able to retrieve the characters correctly, aside from a better camera, a Principal Component Analyser (PCA) algorithm could be applied. This kind of algorithm is used to find a linear pattern within a dataset and if that dataset happens to be a character, it calculates the direction it is “pointing” to. Using that information the angle α which the character is slanted can be calculated. Using the calculate angle, a shear transform can then be applied to remove the slant of a character, as displayed in Figure 20a and 20b. Although it works most fine most of the times, there are some characters from which the angle is incorrectly calculated and results worse off than before, e.g. 20c and 20d. To solve this the algorithm must be modified to work better with characters.

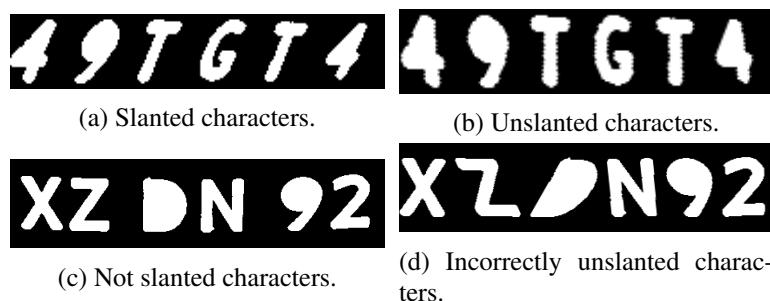


Figure 20: Process to unslant characters.

Text recognition

One way to solve the problem of similar shape characters being wrongly recognised is to create a dictionary where similar characters are grouped together. Every time one of these characters is recognised, the other characters in the group can be given as a possible alternative and a confidence percentage can be calculated according to which characters are present in the text.

References

- [1] Csr at cgi. <http://www.cgi.com/en/corporate-social-responsibility>. (accessed 16-06-2014).
- [2] Martinský Ondrej, V Zboril Frantisek, and Drahanský Martin. Algorithmic and mathematical principles of automatic number plate recognition systems. *BRNO University of technology*, page 10, 2007.
- [3] Car source image. http://pics.auto-commerce.eu/ac1414/foto/large_139452873607.JPG. (accessed 04-06-2014).
- [4] Dutch license plate. http://upload.wikimedia.org/wikipedia/commons/thumb/6/6c/46-GZB-8_license_plate_of_the_Netherlands.JPG/1920px-46-GZB-8_license_plate_of_the_Netherlands.JPG. (accessed 04-06-2014).
- [5] Alvy Ray Smith. Color gamut transform pairs. *SIGGRAPH Comput. Graph.*, 12(3):12–19, August 1978.
- [6] Hsv cylinder figure. http://commons.wikimedia.org/wiki/File:HSV_color_solid_cylinder.png. (accessed 04-06-2014).
- [7] Jean Serra. *Image analysis and mathematical morphology*. Academic Press, Inc., 1983.
- [8] I Sobel and G Feldman. A 3x3 isotropic gradient operator for image processing (1968). *a talk at the Stanford Artificial Intelligence Project*.
- [9] Warp perspective opencv. http://docs.opencv.org/modules/imgproc/doc/geometric_transformations.html#warpperspective. (accessed 04-06-2014).
- [10] Dutch car license plate measurements. <http://www.rdw.nl/Particulier/Paginas/Soorten-kentekenplaten-.aspx>. (accessed 17-06-2014).
- [11] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27, 1975.
- [12] Satoshi Suzuki et al. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, 1985.
- [13] Lefly license plate font. http://robospace.files.wordpress.com/2013/10/pca_line.png. (accessed 04-06-2014).
- [14] Road design criteria zuid-holland. <http://www.zuid-holland.nl/documenten/opendocument.htm?llpos=355202489&llvol=0>. (accessed 15-06-2014).

Evaluation

	<i>analyse</i>	<i>advise</i>	<i>design</i>	<i>realisation</i>	<i>administration</i>
<i>user interaction</i>					
<i>company processes</i>					
<i>software</i>					
<i>infrastructure</i>					
<i>hardware interfacing</i>					
<i>level 3</i>					
<i>level 2</i>					
<i>level 1</i>					

Figure 21: Competences matrix.

During the project which led up to this thesis, the following competences that were stated in the mandate were used. For the software category, the analyse competence was used to create the requirements necessary for the development of the application. The advice competence was used when writing the recommendations on how the application could be further improved. The design competence was necessary to design an efficient system which would fulfill the requirements. Once a design for the system has been created the application must be developed, and this is where the realisation competence comes in. Last but not least, as to prevent the loss of source code, a version control system was used. This is part of the administration competence.

Only two competences were chosen from the infrastructure category: design and realisation. To design the way the application communicated with the information website and the central application, the design competence was used. And to implement that design, the realisation competence came into play.

The last competence used was the administration competence from the hardware interfacing category. This competence was used to program the application in a way that worked within the Android platform.

Appendix A

Legends and acronyms

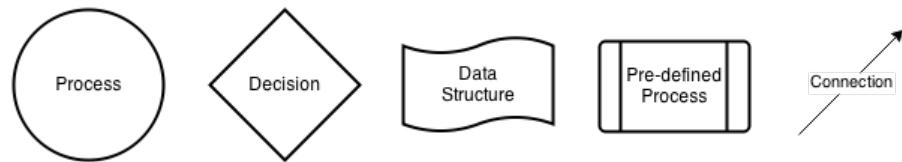


Figure A.1: Flowchart legends.

- **OCR** - Optical Character Recognition
- **IAD** - Iterative Application Development
- **PCA** - Primary Component Analyser
- **OpenCV** - Open Computer Vision