RoadEye

by

Marcelo Dias Avelino (0840416)

CMI-Program *Technical Informatics* – Rotterdam University

May 23, 2014

First supervisor	*Mrs. T. Ubert*
Second supervisor	*Mr. A. van Raamt*

Abstract

ala

Acknowledgements

Wie kan je zoal bedanken? Denk aan de begeleiders en voorbereiders van je afstudeerproject, familieleden en andere personen die je geadviseerd of gemotiveerd hebben. Het is gebruikelijk om dit voorafgaande aan het verslag te doen. Dit bedanken mag ook in de inleiding gebeuren. Bijvoorbeeld: Bij het opstellen van dit verslag heb ik dankbaar gebruik gemaakt van 'metathesis' van *Donald Craig (donald@mun.ca)*.

Contents

Introduction

This project has been made available by CGI Nederland BV, which is a daughter company of the CGI Group Inc. located in The Netherlands. CGI is a multinational IT (Information Technology) company which provides consulting, systems integration, outsourcing, etc. on a multitude of different technological areas. Because of its affinity with these different kinds of technology, CGI researches various technologies to solve problems facing modern society. 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 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 possibility to help his own community and therefore help making it a better place. One of these possibilities is to capture video images using the smartphone's camera. These images when paired with a Computer Vision algorithm are able to produce amazing results. 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 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 an android smartphone software application 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 an 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.

Methodology

Before starting the development of the project, an important question became on how to find viable ways of solving the problems presented by the project's requirements and how to implement them once the solution has been found. For this reason some time was first spent searching for possible research and development methods and in the end the following methods were chosen.

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 facing the project. 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

When reasoning about what kind of main algorithms and/or software would be necessary to develop the application, it quickly came down to two categories: localising the plate within an image and parsing the text within the plate. 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 works by finding so called *features* in a image, which are used to recognize 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 an 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, e.g. the Sobel Filter [ref here] or Canny Edge Filter [ref here]. 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 first algorithm and an healthy amount of information regarding this one, this algorithm was chosen.

Text recognition

‘Explanation on training Tesseract’

Design

Because the application implemented is an Android application, its design was created with Android's design rules in mind. A flowchart illustrating the general design and flow of the application is displayed in Figure 1. 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. 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.

'add database and database entries retrieval'

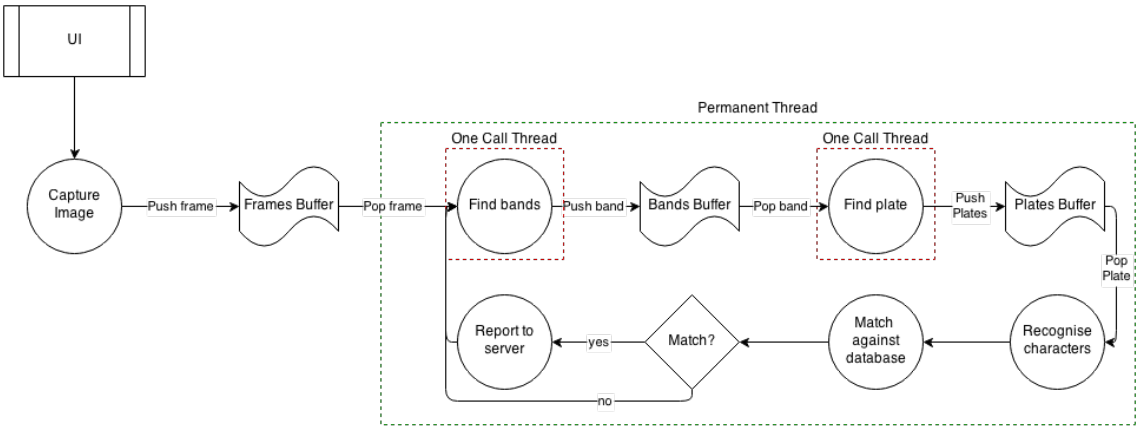


Figure 1: System flow diagram

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 [1], with some slight modifications to fit the requirements of this application. The car in Figure 2 is used to demonstrate how the algorithm works.

Figure 2: Source car image.

Bands

The first step to locating a license plate is to find its vertical location, which also known as a band. By making use of one of the most characteristic features of a Dutch license plate, which is its yellow colour and can be seen on Figure 3, it is possible to reduce the area of the image where the license plate might be located through the application of colour segmentation.

Figure 3: Dutch license plate.

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 4, 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 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. 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.

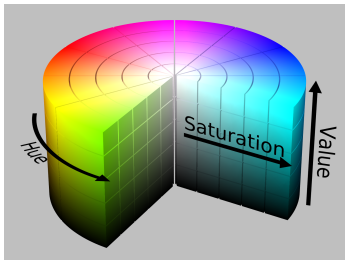


Figure 4: HSV colour space.

As a result of segmenting the yellow colour from the source image, the image in Figure 5a 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 invalid results when trying to locate the plate. To prevent this, a morphological closing operation [2] is performed

'part about removing the small speckles'. Because there are such things as yellow cars and other yellow objects everywhere, an extra step must be taken to decreased 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 [3] was used and its output is visible in Figure 5b, 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 5c. 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 5c are found.

(a) Colour segmentation.

(b) Edge detection.

(c) Possible band(s).

Figure 5: 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 6, 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 the every data cluster present in the graph, the largest one is chosen as the plate's location.

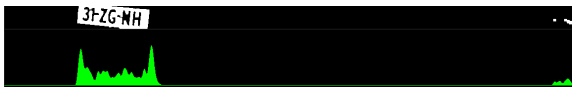
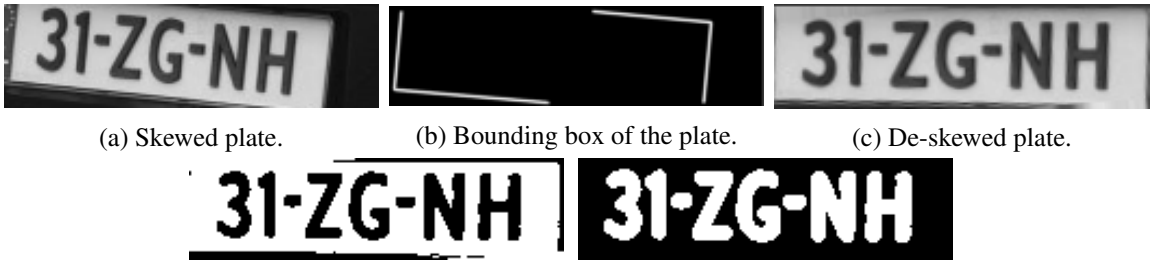


Figure 6: Horizontal localisation of the plate.

This gives an image of the plate with a rather larger border of undesired parts of the image,

as shown in Figure 7a. 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 (Optical Character Recognition) software to recognize 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 7b. 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 [4] as displayed in Figure 7c.

Finally, only a few last steps are necessary to complete the localisation and processing of the plate before the OCR software takes over. 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 every else to white, in this case this threshold is calculated using Otsu's method [5] and is displayed in Figure 7d. This method assumes the image consists of two types of pixels, foreground and background pixels, and uses that premise to search for the optimum threshold value. This applies perfectly to the case of a license plate because the desired section of the plate, i.e. the text, is in the foreground while everything else is part of the background. After applying Otsu's method, 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 [6], 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 an hierarchy list of every contour and whether that contour has a parent and/or child. Once the correct contours have been found they are then drawn on another image, which can be seen in Figure 7e.



(a) Skewed plate. (b) Bounding box of the plate. (c) De-skewed plate.

(d) Threshold of plate. (e) Contours of text.

Figure 7: Phases from the plate de-skewing process.

Conclusion and recommendations

Conclusies en aanbevelingen moeten verzameld worden in een apart en herkenbaar deel van het verslag. Hoewel in het hoofdverslag op diverse plaatsen conclusies getrokken kunnen worden, moeten de belangrijkste conclusies samengevoegd en samengevat worden.

Belangrijk is dat het verschil tussen objectief controleerbare conclusies en subjectieve aanbevelingen duidelijk wordt aangegeven. Ook is het aan te bevelen om de belangrijkste conclusies conform de opdrachtomschrijving te formuleren.

References

[1] ONDREJ, Martinský; ZBORIL FRANTISEK, V.; MARTIN, Drahanský. Algorithmic and mathematical principles of automatic number plate recognition systems. BRNO University of technology, 2007, 10.

[2] Jean Serra. 1983. Image Analysis and Mathematical Morphology. Academic Press, Inc., Orlando, FL, USA.

[3] Sobel and G. Feldman, "A 3x3 isotropic gradient operator for image processing," Stanford Artificial Project., Tech. Rep., 1968.

[4] Warp Perspective OpenCV:
`http://docs.opencv.org/modules/imgproc/doc/geometric_transformations.html#warpperspective`

[5] A Threshold Selection Method from Gray-Level Histograms," Systems, Man and Cybernetics, IEEE Transactions on , vol.9, no.1, pp.62,66, Jan. 1979

[6] Suzuki, S. and Abe, K., Topological Structural Analysis of Digitized Binary Images by Border Following. CVGIP 30 1, pp 32-46 (1985)

Evaluatie

In de evaluatie reflecteer je over je eigen afstudeerproces. Daarbij moet je vooral letten op de leereffecten. Welke competenties had je nodig? Welke competenties kwam je tekort en moest je zelf verwerven? Waren dit algemene of specifieke competenties? Voldeden de beroepscompetenties aan de standaard van het *HBO-I* (analyseren, adviseren, ontwerpen, realiseren en beheren)? Vielen de algemene competenties in de vijf categorieën van de *Dublin Descriptoren*[1] zoals het verkrijgen van kennis en inzicht, het toepassen van kennis en inzicht, het maken van onderbouwde keuzen (oordeelsvorming), het communiceren (schriftelijk en mondeling) en het verkrijgen van leervaardigheden?

[1] Dublin Descriptoren zijn eisen aan de competenties voor de bachelor en master studies aan universiteiten en hogescholen in Europa.

Appendix A

Legends and acronyms

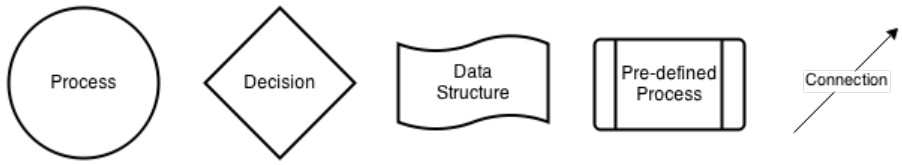


Figure A.1: Flowchart legends.

- **OCR** - Optical Character Recognition