



Karlheinz Wohlmuth, BSc

Nameplate Detection and Classification

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme
Computer Science

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Horst Bischof
Institute for Computer Graphics and Vision

Advisors

Univ.-Ass. Dipl.-Ing. Manuel Hofer, BSc
Institute for Computer Graphics and Vision

Dipl.-Ing. Michael Maurer, BSc
Institute for Computer Graphics and Vision

Graz, Austria, Jan. 2015

Abstract

Nameplates are attached to power supply equipment and describe their electrical properties, which are needed to maintain and repair these devices. The current practice of manually transcribing these values into the testing application is time consuming, susceptible to typing errors, and prone to omit important values. A number of related approaches to locate and extract content from license plates, business cards, as well as forms exist, but they rely either on a simple layout of the object, or require low noise and clutter free input. These requirements are not guaranteed to be valid in our case of weathered and stained nameplates, that usually contain a considerable amount of content. We propose an algorithm to locate, segment, classify, and extract information from rectangular plates, available as human readable text, that is applicable on a mobile device. A multi-view approach ensures robust text extraction. We first locate the plate, segment it using Grab-Cut, extract features and determine the type using a Random Forest, and finally extract the text using a state of the art optical character recognition engine. Additional images are matched to the existing one and all visible text regions are processed. Further, we created a prototype implementation for the Android operating system that implements the proposed steps, and use it to conduct several experiments under different lighting conditions with a number of plates. The results show the applicability of the proposed approach, as well as the robustness of the plate classifier even for weathered plates and difficult lighting conditions, such as reflections or shadows.

Keywords. Nameplates, Segmentation, Classification, Optical Character Recognition, Mobile Device

Kurzfassung

An einem Großteil aller Energieversorgungsgeräte befinden sich Typenschilder, die deren elektrische Eigenschaften beschreiben und nötig sind um die Geräte zu warten und zu reparieren. Zur Zeit werden die benötigten Daten manuell in die Wartungswerkzeuge übertragen. Dieses Vorgehen ist zeitaufwändig, fehleranfällig und wichtige Daten werden möglicherweise ausgelassen. Für andere Arten von Typenschildern oder vergleichbaren Objekten, wie zum Beispiel Kfz-Kennzeichen, Visitenkarten oder Formulare, gibt es bereits bestehende Ansätze um die Position und den Inhalt zu bestimmen, die jedoch auf ein simples Layout aufbauen oder Eingabebilder mit wenig Störeinflüssen voraussetzen. Diese Voraussetzungen sind in unserem Fall von großteils verwitterten und verschmutzten Typenschildern, die üblicherweise größere Informationsmengen enthalten, nicht gegeben. Wir stellen ein auf Mobilgeräten einsetzbares Verfahren vor, um die Position des Typenschildes zu ermitteln, es auszuschneiden, den Typ zu bestimmen, sowie darauf vorhandenen Text zu extrahieren. Weitere Aufnahmen aus mehreren Blickwinkeln werden verwendet um eine robuste Textextraktion zu gewährleisten. Zu Beginn wird das Schild lokalisiert und mithilfe von GrabCut segmentiert. Weiters werden Merkmale extrahiert und mit einem Random Forest klassifiziert. Die Textextraktion wird mit einer frei verfügbaren Texterkennungs-Software durchgeführt. Zusätzliche Aufnahmen des Schildes werden zum bestehenden Bild zugeordnet und alle sichtbaren Textregionen werden verarbeitet. Mithilfe eines Prototyps für Android Geräte, der das vorgestellte Verfahren umsetzt, werden Versuche mit mehreren Schildtypen unter verschiedenen Beleuchtungsbedingungen durchgeführt. Die Ergebnisse zeigen die Anwendbarkeit des Algorithmus und die Robustheit der Klassifikation bei verwitterten Schildern und unter schwierigen Beleuchtungsbedingungen, wie zum Beispiel Reflexionen und Schatten.

Stichwörter. Typenschilder, Segmentierung, Klassifikation, optische Zeichenerkennung, Mobilgerät

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

The text document uploaded to TUGRAZonline is identical to the presented master's thesis.

Place

Date

Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

Ort

Datum

Unterschrift

Acknowledgments

Over the last year a number of people helped me to complete this thesis. I would like to thank my thesis advisers DI Manuel Hofer and DI Michael Maurer for their patience, valuable input and guidance, as well as reviewing draft versions of this thesis.

Further, I would like to thank my supervisor Prof. Horst Bischof, and OMICRON electronics GmbH for the cooperation that allowed me to work on this thesis. Moreover, OMICRON supplied a number of images, used for development and testing, and a number of actual plates.

Finally, I want to thank my family and friends for their moral support.

Contents

1	Introduction	1
2	Related Work	5
2.1	License Plates	5
2.2	Business Cards	8
2.3	Scanned Objects	10
2.4	Form Recognition	12
2.5	Summary	16
3	Nameplate Detection and Classification Approach	17
3.1	Nameplate Detection and Extraction	18
3.1.1	Hough based Methods	18
3.1.2	Line Primitive based Methods	20
3.1.3	Feature based Methods	21
3.1.4	Our Approach	21
3.1.5	Plate Extraction	24
3.2	Plate Classification	28
3.2.1	Nameplate Feature Vector	29
3.2.1.1	Plate Color	31
3.2.1.2	Local Binary Patterns	31
3.2.1.3	Size Ratio	32
3.2.1.4	Final Feature Vector	33
3.2.2	Classification using a Random Forest	33
3.3	Text Detection	34
3.3.1	Character Extraction using Maximally Stable Extremal Regions	36
3.3.2	Character Grouping	37

3.3.3	Noise Filtering	38
3.4	Text Extraction	41
3.5	Guided Image Acquisition	44
3.6	Summary	45
4	Evaluation and Experiments	47
4.1	Evaluation of the Classifier	49
4.1.1	Parameter Selection using Grid Search	49
4.1.2	Plate Color	50
4.1.3	Local Binary Patterns	54
4.1.4	Size Ratio	55
4.1.5	Combination of all Features	56
4.2	Evaluation of the OCR	61
4.3	Application	63
4.3.1	Edge Cases	69
5	Conclusion and Future Work	73
5.1	Summary	73
5.2	Further Work	74
A	List of Acronyms	77
	Bibliography	79

List of Figures

1.1	Examples for common nameplates.	2
1.2	Images of weathered nameplates.	3
2.1	License plate detection using MSER	6
2.2	License plate detection using CSER	7
2.3	Business card preprocessing steps	9
2.4	Scanned input image and segmented objects	11
2.5	Sample grid, graph representation and production rules.	14
3.1	Flowchart of the proposed algorithm.	18
3.2	Nameplate detection steps for sample image 1.	22
3.3	Nameplate detection steps for sample image 2.	23
3.4	Major steps of the connected component extraction algorithm.	24
3.5	Nameplate extraction steps for sample image 1.	25
3.6	Nameplate extraction steps for sample image 2.	26
3.7	Plate detection results for sample image 1.	27
3.8	Plate detection results for sample image 2.	28
3.9	Sample images of different nameplates.	29
3.10	Color features extracted from the two sample images.	30
3.11	LBP codes extracted from the two sample images.	32
3.12	Schematic of the values used to calculate the size ratio.	33
3.13	Text extraction, grouping, and matching for sample image 1.	36
3.14	Text extraction, grouping, and matching for sample image 2.	37
3.15	Criteria for character grouping.	38
3.16	OCR preprocessing steps illustrated on printed text.	40
3.17	OCR preprocessing steps illustrated on embossed text.	40

3.18	Matches between the existing plate and a new image.	44
4.1	A sample image for each plate type.	48
4.2	Mean and median precision and recall plots for different grid sizes in the CIE L*a*b color space.	50
4.3	Mean and median precision and recall plots for different grid sizes in the RGB color space.	51
4.4	Training times for different grid sizes with median color values in the CIE L*a*b color space as features	51
4.5	Original and blurred images for different median blur kernel sizes, that are only classified correctly with the complete feature vector.	58
4.6	Nameplate class where the sample images contain dark regions and reflections.	59
4.7	Nameplate class with three samples that is never correctly classified.	60
4.8	Screenshots of the Android application.	64
4.9	Plate detection steps for a plate of type k on an Android device.	65
4.10	Plate detection steps for a plate of type l on an Android device.	66
4.11	Plate detection steps for a plate of type m on an Android device.	67
4.12	Input images acquired at different angles relative to the plate.	69
4.13	Samples with problematic lighting conditions.	70
4.14	Classification results for badly illuminated plates.	71

List of Tables

3.1	Tesseract output for the samples from Figure 3.16 and Figure 3.17.	42
3.2	Sample regions where the OCR output improves after preprocessing.	42
3.3	Preprocessed sample regions where part of the text is removed.	43
4.1	Overview of the dataset.	47
4.2	Grid search results for the feature extraction parameters.	49
4.3	Grid size compared to number of trees in the random forest with only median CIE L*a*b color value features.	52
4.4	Confusion matrix created with median color features extracted from the CIE L*a*b color space.	53
4.5	Confusion matrix created using only LBP features.	54
4.6	Precision and recall for varying LBP radii and number of neighbors.	55
4.7	Confusion matrix created only with the size ratio as feature.	55
4.8	Results for common amounts of active split size variables.	56
4.9	Classification results for pairwise combined features.	56
4.10	Classification results for median blurred images with the complete feature vector and LBP features alone.	57
4.11	Confusion matrix calculated using the final feature extraction parameters.	59
4.12	OCR recognition results of the provided and self trained language file.	62
4.13	OCR recognition results trained from the ISOCP font.	62
4.14	List of incorrectly recognized text regions.	63
4.15	Execution times of the different stages of the Android application.	68

Our modern society highly depends on electric power. Therefore, a huge network of electric power supplying devices, such as power pylons, circuit breakers, and transformers, is spread all over the world. To ensure the constant availability of electric power, the underlying infrastructure has to undergo regular servicing. Therefore, attached nameplates provide important information for all kinds of maintenance routines, as well as documentation purposes. The content, which is present on these nameplates, usually consists of textual information, such as serial numbers, manufacturing year, or vendor names, as well as schematic drawings, logos, or even barcodes. In most cases, nameplates are of rectangular shape, and made of metal or synthetic material.

Figure 1.1 shows two examples of such nameplates. These plates differ in size, layout, the used font and material, as they originate from different vendors. The first example nameplate, shown in Figure 1.1a, consists of aluminum, has a tarnished surface, and contains a barcode amongst the human readable text. The second one, displayed in Figure 1.1b, is made of synthetic material, has a polished and reflective surface, contains schematic drawings, and is significantly larger than the first plate.

Up to now, the information on the nameplate is transcribed manually by the operator during maintenance tasks. This transcription is prone to errors, such as typing errors or omission of important information, which might result in severe damage of the respective device, or even trigger blackouts. Furthermore, this process is quite time consuming because of the complexity of most nameplates.

The two examples of nameplates, shown in Figure 1.1, are recent production samples. As such, they are new, were never mounted to a device, and therefore are in pristine condition. In contrast, Figure 1.2 shows two metallic nameplates that have spent a few decades outdoors, mounted on a device. These images illustrate some of the problems that complicate automated content extraction, as well as manual transcriptions. The nameplates are exposed to environmental conditions for decades, and therefore become weathered and stained. On very old nameplates it is even possible that the paint chips off, rendering the text virtually unreadable. Further issues arise from the outdoor location,

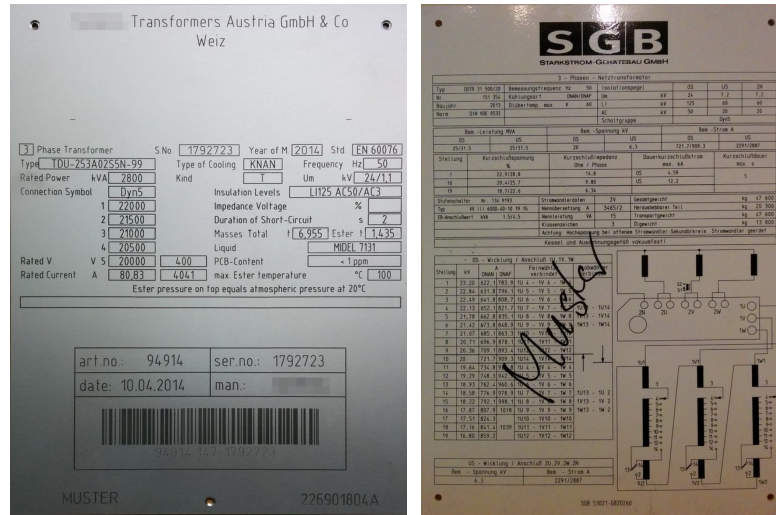
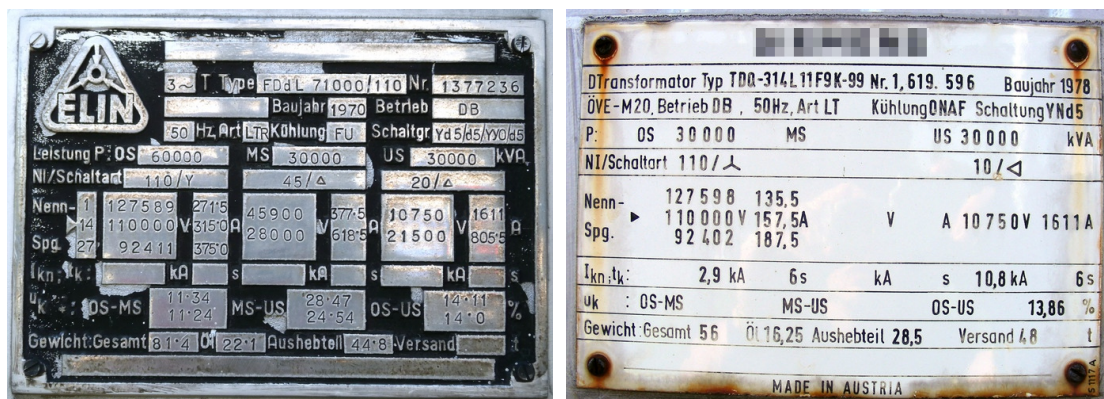


Figure 1.1: Examples for common nameplates. Some vendor names have been obfuscated.

and therefore are likely to include strong shadows, reflections, illumination changes and partial occlusions, that may become problematic when capturing images for automatic processing. An example of this can be seen in the plate displayed in Figure 1.1b, which contains a specular reflection at the top.

In order to extract the semantic information which is present on the nameplate, one could convert images of the text into machine readable text. This process is called Optical Character Recognition (OCR). There already exist a number of approaches that aim at solving this task, as well as readily available *OCR* software implementations. However, most of the available systems are designed and trained to process scanned sheets of paper. Considering this, the four presented nameplates show another difficulty, namely each of the four plates has a different font that differs from the ones used in text processing. Moreover, one plate contains text that is embossed and therefore has a smaller stroke width and lower contrast than painted or printed text. Hence, the existing *OCR* solutions will most likely generate unsatisfactory results.

We propose a novel algorithm to detect, classify, and extract text from nameplates. More specifically, nameplates that are attached to electric power supply equipment, with the requirement that the nameplates are rectangular and contain human readable text. Schematic drawings and barcodes are not processed. Our approach consists of the following steps: First, the nameplate is located in the input image, extracted, and warped into an up-right position. Then, we extract features and determine the type using a machine learning approach. For each type, a list that contains the designations and positions



(a) Nameplate with embossed text (b) Weathered nameplate with rust stains

Figure 1.2: Images of aged, weathered nameplates. Some vendor names have been obfuscated. (a) A nameplate with embossed text. The borders of the plate have been partially overpainted. (b) A nameplate with printed text. It is partially covered with rust stains from the screws.

of the text regions that should be extracted is assumed to be available. Each region is preprocessed to remove noise and clutter, caused by dirt, parts of neighboring text, or frames around the text. Finally, the filtered regions are input into an *OCR* engine. If the recognition score of one or more regions is below a threshold, the user is instructed to acquire a new image of the plate region where the problem occurred. The new image, which may only contain a part of the plate, is matched to the existing one and the *OCR* is executed again. This multi-view approach ensures robust text extraction when occlusions or reflections are present. The final output of the algorithm consists of the detected plate type, the recognized text, its position on the plate, as well as its confidence. The goal of the practical task is to create a prototype of an Android¹ application that allows the user to acquire images, on which the previously listed steps to extract the content are performed.

The main focus of the evaluation is placed on the plate classification, where we test the features, extracted from the plate, individually and combined. Further, we show the impact of training the *OCR* specifically for the fonts used on the plate, as well as the applicability of the Android application on three different plate types. Finally, we show the influence of different lighting conditions on the plate classification and text extraction stages. This includes a number of extreme cases that lead to failure.

OMICRON electronics GmbH supplied a number of images of nameplates from various transformers and similar devices, as well as a number of sample plates to capture more images.

¹<http://www.android.com/> (last visited January 21, 2015)

This chapter provides an overview of several related areas of research. Since no other algorithms have been proposed that exactly match our goal, a number of similar tasks are examined, especially the detection and text extraction from license plates and business cards, as well as the detection of objects in scanned images and the classification of forms. In general these algorithms consist of the detection of plate-like objects, followed by the extraction of information contained in the form of human readable text, similar to what we are aiming at.

2.1 License Plates

The task of detecting and extracting content from license plates exhibits a number of similarities to our task of nameplate detection and classification. License plates have a rectangular shape and contain human readable text. Therefore, to extract their contents it is necessary to locate them in the image and an Optical Character Recognition (OCR) is applied to the individual characters.

Donoser et al. [16] propose a system to detect and track license plates for traffic management. They use Maximally Stable Extremal Regions (MSER) [29] to detect the license plate in the input image (for a short overview of *MSER* see Section 3.3.1). The proposed algorithm uses bright regions surrounded by a dark boundary (MSER+) to detect the license plate, while dark regions with a bright boundary (MSER-) are used to detect the characters on the plate. The *MSER* output of three sample license plates is displayed in Figure 2.1. A region is detected as a license plate if a number of MSER- regions are found inside a larger MSER+ region. Furthermore, the dark regions must be of approximately the same size and their center points must lie on a straight line. The average height of the dark regions must also be approximately the height of the surrounding bright region. The authors use Support Vector Machines (SVM) [11] with the one-vs-one strategy to perform character recognition, where the inputs of the classifier are the individual characters represented by the MSER- regions. Since they also track the license plate over a



Figure 2.1: License plate detection using *MSER*. (a) Input image with various license plates from different countries. (b) The individual characters of the plates detected using *MSER*- regions. (c) The license plates detected as a *MSER*+ region. (Images taken from [16])

number of frames, multiple images of the plate are available, which are used to improve the classification rate. The final detection result of the license plate text are the best individual character results from a majority vote over all frames.

Matas and Zimmermann [30] propose a similar approach to detect text in natural images. However, they use extremal regions, a superset of *MSER*, to define Category-Specific Extremal Regions (*CSER*). The proposed algorithm enumerates all extremal regions by thresholding the image at each gray level, starting at level one. For each intensity level, connected components are extracted that represent the extremal regions and descriptors are calculated. The authors note that there are only three possible ways that existing extremal regions change: they grow larger, two previously separated ones merge, or a new region appears. This incremental update behavior is exploited by choosing descriptors that are also incrementally computable, which keeps the runtime complexity low. They propose the use of normalized central algebraic moments, compactness, Euler number (the number of holes in the objects subtracted from the number of objects), entropy of the cumulative histogram, number of convexities, and the area of the convex hull (although the last two descriptors are not incrementally computable). These descriptors are then input at each intensity level into a classifier whose output indicates whether the region is of interest or not. This means that rotated or slanted characters must be incorporated into the training samples, to be detected. It should be noted, that this approach detects text in natural images. Therefore, to only obtain license plates, a separate grouping and filtering step must be employed. The *CSER* detection pipeline and an example result are shown in Figure 2.2. In addition to the license plate, the characters of the country of origin sticker above the license plate are detected as well. Furthermore, this approach is not only limited to text, but may also be used to detect other objects. For example in

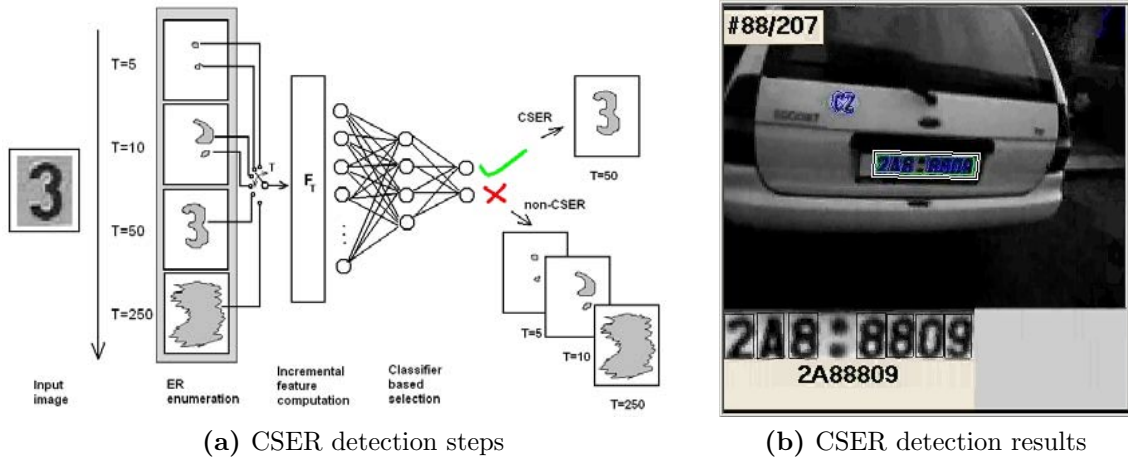


Figure 2.2: License plate detection using *CSER*. (a) *CSER* detection algorithm. The classifier determines if an extremal region contains relevant content or not. (b) *CSER* detection result. Since all text in the image is detected, the letters on the country of origin sticker are found as well. (Images taken from [30])

[30], the authors also explore how to detect traffic signs.

In contrast to the previous methods, Chang et al. [9] propose an approach that uses the colors of the plates. This consists of two steps, license plate location and license number identification. The authors define that only the four colors white, black, red, and green are used on the license plates that will be detected. Hence, they require a color image as input. They propose a color edge detector that responds only to edges caused by specific color combinations, which results in very few edges in sections of the image that are not part of a license plate. First, the image is converted into the Hue, Saturation and Intensity (HSI) color space and the edge map is calculated. Fuzzy maps that model specific color transitions and encode the probability that a region contains a license plate are generated from each plane of the image, as well as the edge map. Given that license plates contain a number of repetitive edges with high gradient magnitude, the maps are combined into a single map, which is used to perform the detection step. To extract the license text, adaptive thresholding is performed and connected components are extracted and filtered by their aspect ratio. The centers of the connected components must form a straight line, connected components that deviate are removed. Since the number of characters on supported license plates is exactly eight, connected components are removed, starting from the smallest one, until that number is reached or a significant jump in size occurs. Should the number be lower, the algorithm tries to find new characters starting from the outermost ones in direction of the known characters. The set of detected characters is also checked against the format of the license text. To perform the character recognition, they first separate numerical and alphanumerical characters using the known format of the license text. Next, they perform topological sorting using the number of holes and the

node types as features. This is done to decrease the number of templates to which the candidate must be compared to in the next and final step. Each character is compared to the remaining templates and the best one is determined via a self organizing recognition model. A neural network is used, where the weights for the nodes are derived from the input character. The template is then input into the network and after it stabilizes, the sum of weight changes in the network is used to derive a measure of similarity.

In summary, the examined approaches to license plate detection first detect the plate in the input image, extract it and employ optical character recognition to extract the license text. In contrast to the expected input images for our task, these images are very cluttered, with only a small portion of the image being occupied by the actual license plate. Furthermore, the layout of a license plate is much simpler. It consists of at most two lines of text and the set and diversity of characters as well as the number of fonts is limited. Since the examined approaches rely on the known layout and (in part) on the colors and license text format, they do not fit our task of nameplate detection, although individual steps can be adapted to extract text from the nameplates.

2.2 Business Cards

Another related topic is the detection and text extraction from business cards as they are also rectangular and contain information in the form of human readable text. In contrast to license plates there is no common layout and a variety of fonts are used, but the content usually consists of a name and address information. There exist a number of papers that cover the detection and text extraction specifically tailored to mobile devices.

Luo et al. [28] propose a system to extract the contact information from business cards on mobile devices with limited processing power. The image from the built-in camera is first converted to grayscale and scaled down to one-fourth of the width and height, which reduces the computational- and memory requirements. Next, the edge map is extracted using the Canny edge detector and the skew angle is calculated on the scaled down image. Since business cards usually only contain non overlapping rectangles of text, everything that does not fulfill this criterion is discarded. This yields the location of the business card, by thresholding. To perform the text recognition, the authors propose a template based algorithm that uses a two stage classifier, where the first stage is used to limit the number of possible templates. The features used for classification of each character are the width/height ratio, as well as the number of foreground and background pixels of the input divided into a 4×4 grid. The second stage additionally incorporates the average distance of the outermost foreground pixels to the borders, and the number of foreground pixels in a 4×4 grid in four directions. If the classification score is low, the algorithm tries to split the input into multiple characters, assuming they were not properly separated during preprocessing. The points where the characters are divided, are determined by analyzing the positions of the extreme points of the character contour. Finally, possible recognition errors are corrected during post-processing. Characters that deviate from the

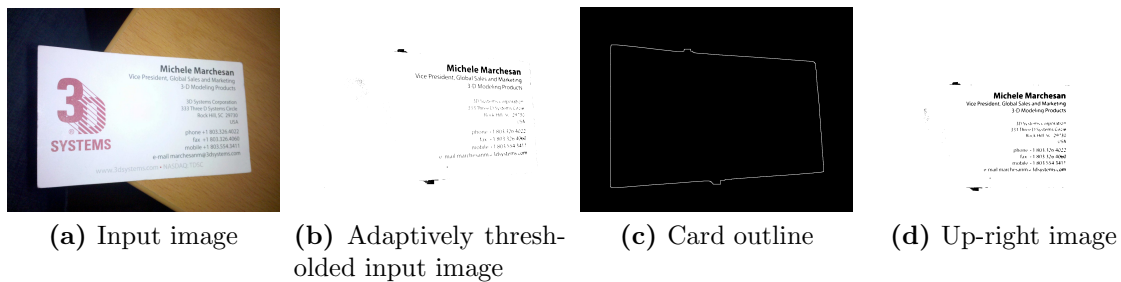


Figure 2.3: Business card preprocessing steps. (a) Sample input image. (b) Adaptively thresholded image. (c) Difference between the eroded and dilated image which is used as input for the calculation of the Hough transform. (d) Warped business card, which is the final output of the preprocessing stage. (Images taken from [2])

top baseline are discarded and the remaining ones are grouped into words. Furthermore, no numbers are allowed in a word, and text correction is applied.

An approach similar to the previous one is proposed by Mollah et al. [31], which also targets computationally limited mobile devices, although they only cover the segmentation of the individual characters. The input image is separated into non overlapping cells, for each one the variance is calculated and used to remove the background. Next, everything that is not recognized as text is discarded. This includes graphics, lines, and noise. From the remaining structures generated by text, connected components are extracted. The authors note that multiple distorted or narrowly placed lines of text may merge and result in a single connected component, whereas one well spaced line of text is represented by one connected component. The skew angle for the text lines is determined by calculating the distance of the first pixel at the bottom of the connected component, to an axis parallel line. Skew angles are calculated at the borders and the middle of the connected component. Should they differ too much, the procedure is repeated, although the pixels from the top of the connected component are used. The final skew angle is the mean of the three values. To separate the up-right connected components into individual lines, the region is thresholded and a horizontal histogram is calculated. The region is split at approximately same sized intervals, as indicated by the histogram. To separate a line into characters, the same approach is employed by calculating and analyzing a vertical histogram, however this means that italic or cursive text cannot be segmented properly. The result consists of the separated characters of the individual lines, which may then be input into an *OCR*.

Bhaskar et al. [2] propose a business card reader that runs on Android devices. In contrast to the other solutions, they do not develop their own *OCR*, but use Tesseract¹ instead, which is also our approach. First, they apply adaptive thresholding to the image and then a number of morphological operations, which results in white regions inside the

¹<https://code.google.com/p/tesseract-ocr> (last visited January 21, 2015)

business card and black regions outside. To detect the boundaries of the card, another morphological operation is applied and the Hough transform [17] of the thresholded difference of the eroded and dilated images is computed. They use the strongest peaks in the Hough accumulator to detect the four boundary lines, with the constraint that all corners, calculated by intersecting the lines, must be inside the image. Should all detected lines violate the constraint, the input image is used as it is, and the perspective transform later on is omitted. The authors note that often business cards do not have a uniform background but instead contain graphics or color gradients, which can lead to wrong outline detection results. To detect and correct such cases, they further analyze the text flow. The bounding box that contains all black pixels of the card is split into six bars and in each one, the angle of the text flow is calculated. The angles of the outermost bars should be very similar to the angle of the corresponding boundary lines of the card. Should they differ too much, the text flow angle takes precedence. The authors note that this is only possible for the horizontal lines because business cards usually do not contain vertical text, which could be used to calculate the angle for the vertical lines. The preprocessing is completed by warping the image into an up-right position, which removes the background. Figure 2.3 shows the preprocessing steps for a sample business card. The extracted image is adaptively thresholded again and used if the number of black pixels changes significantly compared to first thresholding result. Otherwise the previous result is used. To improve *OCR* performance, different text blocks are separated by splitting the image into two parts at corridors that are at least twenty pixels wide and contain black pixels at each side. This is done by sweeping the image at different angles starting from different points originating from the top line of the card. Finally, all split images are passed to Tesseract (see Section 3.4 for an overview of the Tesseract *OCR* engine).

The considered approaches for business card readers primarily rely on the uniform color of the business cards to segment them. Furthermore, the amount as well as the content of the text, present on business cards, is limited to names and address information, which allows the usage of dictionaries or text correction to limit *OCR* errors. The size of business cards is also restricted as they must fit into wallets, which also keeps them in mint condition. However, a large number of fonts and font faces may be used. Unfortunately some of the examined approaches just fail with cursive and italic input.

2.3 Scanned Objects

The approaches considered in this section aim at detecting rectangular objects in preview scans, so that multiple objects may be put onto the scanner, but each one is automatically stored into an individual file. This topic only relates to our first step, namely the plate detection and extraction. Since these approaches consider overlapping and approximately rectangular objects, they are relevant because the plates we need to detect are usually not perfectly rectangular or not well separated from the background.

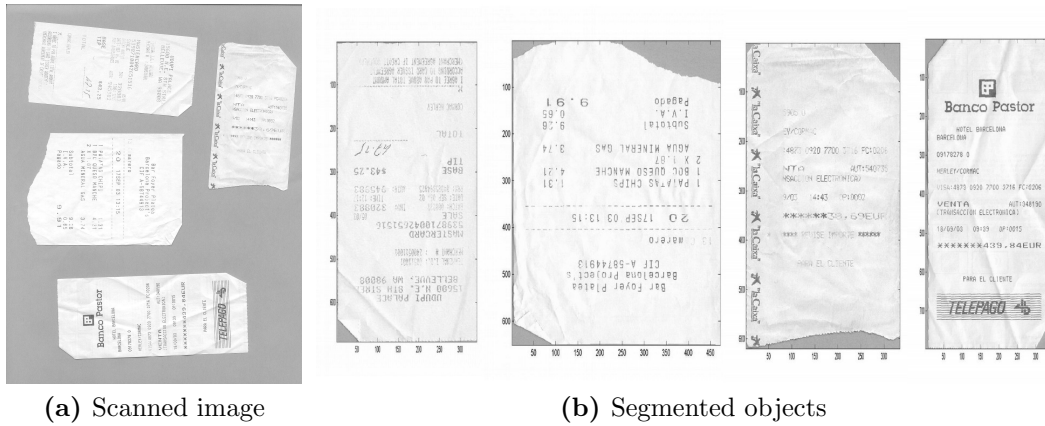


Figure 2.4: Scanned input image and segmented objects. (a) Input image with multiple objects, several of them do not consist of four corners and four straight lines. (b) Segmented objects. (Images taken from [24])

Herley [24] proposes a computationally efficient algorithm to segment rectangles where corners may be missing or edges may be ragged. An example of such objects is shown in Figure 2.4a. First, the input image is split recursively into smaller images, such that the resulting images only contain one object and little background. Next, they calculate the number of foreground and background pixels in the split images and the center of mass of the object, which is only an approximation of the real center because the objects are not perfectly rectangular. The approximate angles of the corners of the object are estimated by calculating the distance from the center to the borders. The authors note that the corners of rectangular objects appear periodically. Therefore, their angle can be estimated from the calculated distance values and refined by applying the precondition that the objects must contain at least two parallel edges. Finally, to obtain the bounding rectangle, the authors grow a rectangle located at the already known center with the previously calculated angle. The growing rectangle is split into four quadrants. Each of them is validated after every growing iteration with respect to the number of pixels not belonging to the object. They terminate if three quadrants are above a threshold. The results of the input image are depicted in Figure 2.4b.

Guerzhoy and Zhou [21] propose an algorithm to segment rectangular objects that may overlap and are placed on a lightly textured background with unknown color, which is the usual output of scanner preview images. The authors aim at detecting the background color, which is challenging when the image consists primarily of differently colored foreground objects, but it allows them to segment the objects more easily. To retrieve the background color, they first separate the image into line segments which have the same tint, by calculating the neighboring color differences. Under the assumption, that the background color segments are long, a voting scheme is used to extract possible background colors. For each of the background color candidates, the edge strength at the boundaries

to non background colors are calculated. The authors observed that the gradient between background and object colors is larger than it is between foreground colors, and that the number of edge pixels correlates with the number of foreground objects. Therefore, they determine the true background color by only using line segments that exceed a certain length, are of the most frequent color, and have a large number of edge points with high gradient values. From these edge pixels, connected components are extracted and used to fit lines, with weights derived from the edge strength. Neighboring pixels, located in the direction of the line are added until a stopping criterion is reached. Orthogonal lines are used to calculate the corners of the objects, as well as the line segments which form the rectangle. To eliminate wrong line segments, the median and mean color differences between foreground and background along the line segments are calculated and used to derive the parameters of a score function from a number of automatically generated images. Next, for each rectangle candidate, the same values used for line fitting and a score that relates to the number of background pixels in the rectangle are input into an Adaboost classifier, which determines if the rectangle candidate is a proper object or not. To find better matches, the rectangle candidates are also shifted in the local neighborhood. A candidate is accepted when at most 10% of the pixels inside the rectangle are classified as background.

In summary, all approaches are specifically designed to be fast and detect potentially imperfect rectangular objects with possible overlap or missing corners. However, they rely on uniform background color which also must be distinct from the majority of the content found in the foreground objects. Both conditions are usually not satisfied in our application, as plates are mounted on arbitrary background, and unknown lighting conditions may cause significant brightness changes. Also, the color of the plate is not necessarily different from the background. Hence, the only way of distinguishing them from the background is by the border of the plate.

2.4 Form Recognition

Classification and content extraction from preprinted forms is also a closely related research topic. Instead of manual sorting and transcription of filled forms, a number of automated approaches that process scanned documents have been presented. Forms are templates, that consist of text describing its purpose, captions for the blank regions, and tables or cells that must be filled in or are already preprinted. Examples include administrative forms, bank checks, payment slips, and many others. Since the use of forms is widespread, a number of different approaches have been developed over the years to automatically process them. In [14], an overview of the topic is given, including images of sample forms and descriptions of a number of popular approaches. The general layout and the small variability within the form types, where the only differences are the filled-in regions, are almost identical to our problem.

In a large number of forms the text must be entered in rectangular regions. Therefore,

a number of approaches exist that exploit this fact, such as Shinjo et al. [36]. Their approach is designed to work with low quality scans, that produce broken lines and noise. First, lines are detected and skew correction is performed. Next, the 16 types of line crossings and endings that occur in tables and cells are extracted. Cells are detected by checking the computed features, as they are formed by L, T and + shaped crossings. To correct errors introduced by broken lines and noise, the extracted cells are checked for inconsistencies under the assumption that the lines of cells and tables start and end in other lines of cells and tables. The consistency is checked using three rules in the following order: a line must end on another line. If this is not the case, the line is extended until it crosses another line. The outermost lines of tables must end in a line, otherwise they are also extended until they do. And finally, crossing points are only allowed in cell corners. The current structure is modified using these rules until no more rules are violated, where matching rules with higher precedence are preferred. The final result consists of the table and cell structure from which the values can be extracted.

Fan et al. [19] propose an algorithm that uses the line structure of the document to classify it. Their first step involves removing all characters, since their information is not used. This is done by thinning out all structures, followed by extraction and clustering of features. Finally, all clusters that belong to characters are removed, leaving an image that consists solely of lines. Next, three matrices are generated. The first contains the line intersections, where each vertical line is represented as a column, and each horizontal one as a row. When two lines intersect, the first matrix contains a number at the appropriate location, representing the intersection type. The other two matrices encode the distances of the lines to each other, one matrix for horizontal and one for vertical lines. To remain scale invariant, the authors assign numbers to distances relative to the document size. Matching a new document against a known one starts by calculating per element and matrix differences that must be above a threshold, which eliminates completely unfitting forms. Should the matrix sizes differ, a sub matrix of the larger one is used. A matching score that is derived from the similarity of the line crossing types and the distances, is calculated for the remaining candidates. This also requires that the matrices are of the same size. Should this not be the case, the authors extract a sub matrix with the proper size, that has the best match with the full version. The document type with the best matching score is the determined and returned document type.

Sako et al. [35] propose an approach that performs the form classification using only keywords and their location on the form. They extract all characters from the form and perform *OCR*. Then a previously created keyword database is used to classify it. The database contains keywords and their locations, which are unique to each form type. A matching score is calculated by comparing the positions and string similarity. Finally, the form type with the best match determines the result type. To counteract *OCR* inaccuracies, the authors match the strings by counting the number of insertions and deletions needed to transform the first string into the second. To retrieve the form contents, a list of regions is available for each form type. Those are the contents of interest which should be

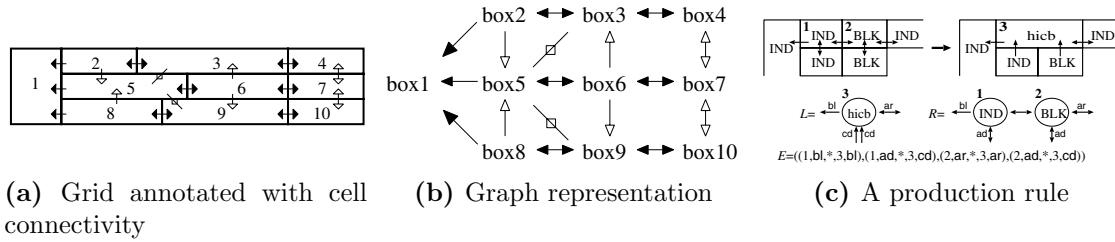


Figure 2.5: Sample grid, its graph representation and an example production rule. (a) A sample grid. The arrows indicate the box connectivity (\rightarrow indicates inclusion, \leftrightarrow indicates same height and $-$ indicates no connection) (b) The graph representation of the grid. (c) A production rule of the graph grammar. (Images taken from [1])

extracted. Each of these regions is thresholded separately to improve the *OCR* precision, and additional constrains, such as string composition suffixes, are used to extract the part of the region (excluding descriptive text) that is of interest for further processing.

An approach that employs line and text positions and is applicable to forms without tables or tabular structure is proposed by Ting and Leung [41]. They use fourteen different primitives that occur on forms, such as text, lines, vertical distances between adjacent rows, intents (positive and negative), and nine types of corners that are encoded into a string by iterating over the objects that are sorted by their vertical position. To match a new document, the number of common tokens of the learned blank and the new incoming form are counted with and without the text blocks and combined into a score. A measure without text is also incorporated, because learning is performed with empty forms.

Contrary to the considered approaches that are trained on empty forms, there exist a number of methods where the structure of the document is specified using various flavors of grammars which are then used to parse a new form. One such approach is proposed by Amano and Asada [1], which uses a graph grammar to describe forms. The authors first define an XML based description for grids, where the rows and the indentation of the cells are described, that can also be used as the result of the form analysis. Grids can be described using graphs, where the cells are the vertices and the edges represent the connectivity between them. Figure 2.5a shows a grid, where the arrows indicate the connectivity and Figure 2.5b displays the resulting graph. The graph grammar consists of the node and edge labels, the starting symbol and the production rules. A production rule consists of a left and righthand side and a conversion rule. A sample rule is displayed in Figure 2.5c, where the upper case labels indicate terminal symbols and the lower case ones indicate nonterminals. The authors note that this grammar can be processed by a conventional graph grammar parser, although since it is context sensitive, parsing is not efficient. To speed up processing and make it suitable for real world application, the authors assign priorities to the rules.

Couïason [12] proposes another grammar based system that is robust against noise.

They employ a method named Description and Modification of Segmentation (DMOS) that uses Enhanced Position Formalism (EPF) to describe the structure of the input, and was previously used to extract musical notes, mathematical formulas, and military application forms from the 19th century. Lines and symbols are extracted from the input image and processed according to the rules described in *EPF*, where the parser may change the structure during parsing. The authors note that there are two sources of noise: The image itself, as well as parts of the document structure that are not formalized in the grammar. Also, other grammar based approaches are not robust against noise because the next element that is parsed is the next token, whereas using their approach, the parser selects the next token to parse from the entire image by itself. *EPF* includes a number of operators such as the position operator (which specifies that an object is located relative to another object), the factorization operator (used to factor common parts of rules), save operators (that allow to store and later load a nonterminal), the declaration operator (that allows to refer to a number of rules by name), and the space reduction operator (that limits the space where rules may be applied). To cope with noise, they propose two new operators that work on terminals and have pre- and postconditions to skip noise, and a new find operator, that applies the given rule to each token until it matches or a stopping condition is satisfied. To prove the robustness against noise, the authors present a complete grammar to detect the seven lines that make up a tennis court [12]. They note that it is not necessary that the complete tennis court is visible and that the grammar is scale invariant. The implementation of the parser that can execute a given grammar is described in [13]. An extension of Prolog named λ Prolog is used to translate the grammar into a λ Prolog program, which then performs the recursive descend parsing and bindings to C are used to segment and extract the tokens from the input image.

Conclusively, a number of approaches have been proposed over the years to classify and process forms which solve problems similar to our task. However, methods that extract cells and grids from the image are not applicable, since not all plates contains them, while approaches that extract a grid like structure from the text blocks encounter issues with complex layouts. Approaches that further incorporate text blocks and keywords, by performing *OCR* at the beginning, are also problematic for plate types that are weathered or contain embossed text, where *OCR* performance without preprocessing is not satisfying and slow to execute on a mobile device. When matching a new form against the known form database, approaches that use string or graphs, consider the new form a noisier version than the initially learned one, which might also lead to misclassifications on very similar forms. Most grammars used to describe forms are either limited in expressiveness or very complex to write, and very sensitive to noise [12]. An exception is for example *DMOS*, but writing the grammars for each type is time consuming and cannot be done by a regular user. Finally, each grammar must be translated to λ Prolog and executed to check if the form matches its type. Overall, scanned forms usually contain a very low amount of noise and are well segmented, the problematic cases are merged objects and non continuous lines, where the amount of noise found on weathered plates is much higher.

2.5 Summary

What all of these approaches have in common is that they detect approximately rectangular objects and, with the exception of the scanned objects section, extract information printed on them in the form of human readable text, which is then stored or handed on for further processing. The detection step of these methods either detects the contents and exploits the simple structure to group them together to locate the objects, or further constrains such as uniform background or foreground color must be fulfilled. In case of processing forms, no detection is needed at all, but only the type of the form must be classified. Furthermore, the format of the content is usually known and a limited set of characters and fonts are used, which simplifies the content extraction and optical character recognition. A notable exception are business cards where a large number of fonts may be used. This however also causes issues in the examined approaches. Forms are also often filled by hand, but handwriting recognition is a field of its own and such cases are not covered in the examined papers. Since these approaches exploit a number of specific properties of the objects for which they are designed, which are not guaranteed to be valid in our case, none of these complete approaches can be used as it is. Hence, we propose our own method to detect and classify nameplates, utilizing individual steps of some of the examined methods. The next chapter describes all steps of our proposed algorithm in detail.

Nameplate Detection and Classification Approach

This chapter describes our approach to nameplate detection and classification of rectangular nameplates, that are attached to electric power supply equipment. All necessary steps, which are needed to extract the desired information from a known set of nameplates are explained, with the restriction that only human readable text is processed. Schematic drawings and barcodes are not in the scope of this work.

Figure 3.1 shows a flowchart of the proposed algorithm. First, the nameplate is detected, extracted, and warped. The result is an image that contains only the up-right nameplate, which is then used to determine the type of the plate. Since there is no standardized layout for such nameplates, each manufacturer produces different layouts for different product lines. Therefore, the number of existing plate types is large, which makes it desirable that the training process of the classifier is fast, so that new types can be added quickly. This is achieved by using Random Forests [4], which are also fast during classification and therefore of advantage when used on a mobile device. To generate the features for classification, the image is split into cells by overlaying a grid. For each cell of the grid, we compute a feature vector that consists of the median color value and histograms of Local Binary Patterns (LBP). We concatenate the values of all cells and add the size ratio of the plate as final feature. For each type, a list containing the designations and positions of the text regions, that should be extracted, is usually available, because we deal with a set of known nameplates. As all plates of a specific type share the same layout, knowing the positions of the text regions simplifies the content extraction, since the boundaries of the fields do not need to be determined. Further, the mapping of the content from a specific region to a label is available, which is needed to meaningfully use the values later on. The extracted regions are preprocessed, which removes artifacts caused by dirt and other structures on the plate. Preprocessing consists of size filtering, morphological opening and closing, and character grouping. Finally, the preprocessed text regions are fed to the Tesseract Optical Character Recognition (OCR) engine. If the recognition score of one or more regions is below a threshold, the user is instructed to acquire a new image of the plate region where the problem occurred. This new image is matched to the existing

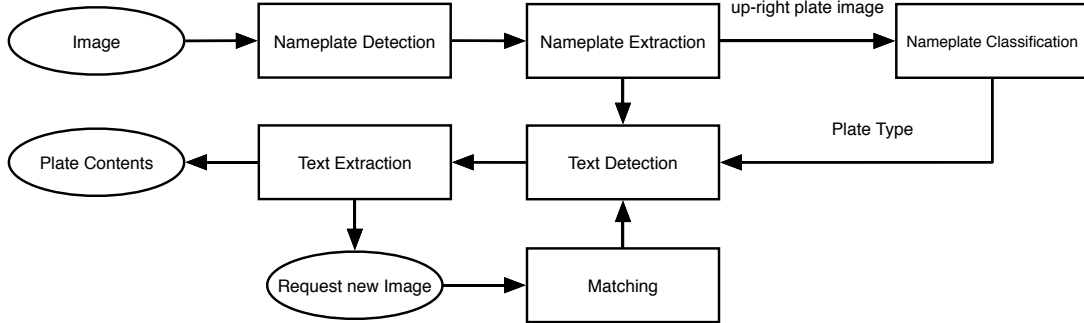


Figure 3.1: Flowchart of the proposed algorithm to detect, classify, and extract text from nameplates.

one and the *OCR* steps are repeated. When the scores of all regions are sufficiently high or the maximum number of retries has been reached, the results are stored for further processing. The final output consists of the detected plate type, the recognized text, as well as its confidence and position on the plate.

This chapter is structured as follows: In Section 3.1 the detection and extraction of the nameplate is explained. Section 3.2 describes the machine learning algorithm and features that are used to classify the plate. Section 3.3 describes the detection of the text regions on the plate, as well as the noise filtering. Section 3.4 gives a short overview of *OCR* systems and describes our choice. Finally, Section 3.5 explains the guided image acquisition approach to improve the final output.

3.1 Nameplate Detection and Extraction

This section details our approach to detect and extract the rectangular plate from the input image. We assume that the plate is completely visible and placed predominantly and roughly centered in the image, with some amount of background around it. We start the section with an overview of a number of related approaches, which are designed to detect arbitrary rectangular objects in images. Then, our approach to detect, extract, and warp the plate into an up-right position is explained in detail.

3.1.1 Hough based Methods

There exist a number of related approaches that use the Hough transform [17], or derivations, to detect lines. To obtain rectangles, the located lines are grouped in various ways.

Wu et al. [42] propose an algorithm to detect rectangular panels in realtime. Their goal is to process large images, for which the standard Hough transform is too slow. Therefore, they use the Progressive Probabilistic Hough Transform (PPHT) to detect the lines. In each iteration, the *PPHT* selects a point randomly, performs the hough voting,

and removes it from the list of unprocessed points. If a cell in the accumulator exceeds the detection threshold, a line has been found. The pixels voting for the line are removed from the input, as well as the accumulator and the line is stored if it exceeds the minimum line length. The authors state that the line detection results are not satisfying, because of distortions from the camera, resulting in lines that are not perfectly straight. To detect such distorted lines, they propose to replace each pixel with a cross-shaped object. Furthermore, they define a mapping from a pixel to its corresponding cross structure and vice versa. This mapping is used to check line candidates only at their cross centers, as well as to remove the entire cross structure from the input list if it voted for an accepted line. This is done because the surplus pixels would result in additional detections of the same line. The detected lines are grouped into rectangles by extracting parallel lines of the same length, which are then combined with orthogonal pairs to complete the rectangle.

Hartl and Reitmayr [22] propose an efficient rectangle detection system, intended to be used on mobile devices. First, they extract the edge map using the Canny edge detector [6]. The threshold parameters are automatically selected. To improve the robustness and increase speed, all edges produced by text are removed. Consequently, a lower number of lines need to be processed. They do this by extracting connected components and calculate the bounding box for each one. The aspect ratio, height of the bounding box, and number of pixels in relation to the bounding area are used as filtering criterions. Next, the filtered image is used to extract lines by applying the Hough transform [17]. Two lines form a line bundle if their angle does not differ more than fourteen degrees, which accounts for some amount of perspective distortion. The authors then group line bundles to rectangle hypotheses, if all line intersections are inside the image. Finally, they compute the edge support on the dilated edge image, where the most dominant rectangle hypothesis with the highest edge support is the resulting rectangle. The edge map is dilated to increase robustness against camera distortions, causing lines that are not completely straight.

A more generic approach to extract rectangles using a windowed Hough transform is proposed by Jung and Schramm [25]. They use a ring shaped sliding window, where the outer diameter approximately equals the size of the largest and the inner diameter the size of the smallest rectangle that can be detected. Next, the Hough transform of the region under the sliding window is calculated, where the discretization depends on the outer ring diameter. To detect the peaks in a robust manner, a modified butterfly evaluator (see Equation 2 in [25] for more details) is applied to the accumulator. To retrieve the lines, the accumulator is thresholded. The authors select the threshold to be half of the inner ring diameter, which only detects lines of at least half the required length. This yields a number of peaks, which are grouped to extended peak pairs if the following conditions hold: The two lines are approximately parallel, have the same length, and are symmetrically placed in the accumulator space. An extended peak pair forms a rectangle hypothesis if the orthogonal angle difference is within a threshold. Finally, the authors perform non-maximum suppression by calculating an error measure that incorporates the values used for grouping the peaks into extended peaks. They note that a shifted rectangle

is visually more distinctive than a slightly wrong rotation angle, therefore they assign a larger weight to the distance measures.

All these approaches apply a Hough transform to extract lines which are grouped into rectangles if a number of conditions are satisfied. As noted in a number of these papers, the Hough transform is quite slow and requires a lot of memory. Therefore, to use it on a mobile device, the input image must be downscaled and the resolution of the Hough accumulator reduced. Furthermore, the content of nameplates is often contained in a grid structure or cells. A number of schematic drawings are also frequently present, which results in a large number of lines, that cannot be easily removed beforehand. With the examined approaches, grouping and checking the hypotheses is time consuming. The printed lines inside the plate may also exhibit stronger edge characteristics than the actual separation between the plate and the background, thus leading to wrong detection results.

3.1.2 Line Primitive based Methods

A different approach is to use line primitives, which are extracted directly from the edge map and are grouped and formed into rectangles. Since they do not apply the Hough transform, these approaches are faster.

Lagunovsky and Ablameyko [26] propose an approach, where extracted lines are subsequently grouped into rectangles. First, they detect the edges of the input image and calculate the edge strength and direction. These values are used to extract contours and cluster the line primitives. A line primitive is added to an existing cluster, if it is located next to a primitive that is already in the cluster. They must have the same orientation and the length of the line must be approximately the same as the average line length of the cluster. If the conditions are not fulfilled, a new cluster is created. Lines, separated because of noise, are recombined if they have approximately the same angle and the distance of their nearest points is below a threshold. To extract rectangles, all lines are represented with their slope, length, and distance to the coordinate center. The authors note that in this representation, approximately parallel lines are located close to each other. To detect the other two orthogonal lines of a rectangle, the authors generate a hypothesis from the already known parallel lines, by increasing the slope by 90° . The search region is defined by the end points of the parallel lines. Next, they extract quadrangles by extending the four lines, until they intersect. Four lines form a quadrangle if the maximum distance from all line end points to the line intersections is below a threshold. To retrieve the final rectangle, the line pair with the greater length is adjusted to their averaged slope and the shorter line pair is modified to be orthogonal to the other one. Each one of the adjusted lines goes through the center of its corresponding unmodified line.

A similar method, that is also based on line grouping, is proposed by Tao et al. [40]. They use the Canny edge detector to extract the edge map and a splitting arithmetic to retrieve linear elements, which are represented by their start and end points. Next, all lines parallel to each other are grouped and parallel pairs of lines are combined into prim-

itive structures. These primitive structures are then used to generate rectangles, taking into account that not all four lines may have the same length. Finally, non-maximum suppression is performed to remove duplicate rectangles.

In contrast to the Hough-based approaches, line-based methods require less memory and have lower computational requirements. Since the grouping is very similar among both types, they share the same disadvantages when applied to nameplates. A large number of rectangles are returned when the plate contains cells and grids, which might provide a better detection score than the actual boundary between plate and background. They are specifically designed for aerial images and printed circuit inspection with low amounts of perspective distortion, which renders it unsuitable for our task.

3.1.3 Feature based Methods

The approaches examined in this section use specific features of the objects to detect them. Several approaches were already elaborated in their entirety in Section 2.1. In this context, only the object detection is of interest, not the following processing steps.

As previously described, Chang et al. [9] exploit specific color transitions of the plates, used in a number of countries, to locate plate candidates. The final detection step checks if the characters in the candidates follow a specific layout and format. In contrast, the approach proposed by Matas and Zimmermann [30] uses Category-Specific Extremal Regions (CSER) to detect characters in the input image. They locate the license plate by grouping extracted objects, using knowledge of the content format and geometrical layout. Similarly, Donoser et al. [16] detect the plate background and foreground objects using Maximally Stable Extremal Regions (MSER) and report a license plate if the regions meet a number of geometrical conditions.

The reasons why these approaches are unsuitable for our use case, are detailed at the end of Section 2.1. In short, they exploit a number of structural properties of license plates and their contents that are not valid in our use case, as nameplates have a more complex and diverse layout.

3.1.4 Our Approach

In our approach, we first determine whether a nameplate is present in the image, and should this be the case, retrieve its approximate location. We assume that only one nameplate is located in the image. Furthermore, we require the plate to be entirely within the image and the most predominant or centered object. Figure 3.2a, as well as Figure 3.3a show images where these conditions are fulfilled.

First, the input image is proportionally scaled to a size of $w_d \times h_d$, where w_d is fixed to a value of 400. The size of the rescaled image is sufficient to detect the plate and improves the performance considerably. Next, the image is converted to grayscale, median blurred with a window size of three, and adaptively thresholded. This is done by calculating the threshold for each pixel, by averaging the values of the neighboring pixels, contained

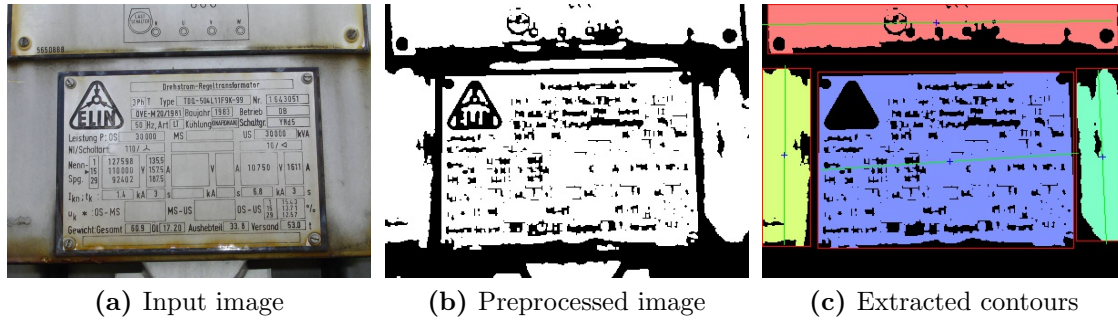


Figure 3.2: Nameplate detection steps for sample image 1. (a) Input image acquired by the user. (b) Preprocessed input image. Median filtering, adaptive thresholding, and morphological opening have been applied. (c) Extracted components after filtering. They are the remaining plate candidates.

in a square window with the size $h_d/2 \times h_d/2$. Under our assumption that the image consists mostly of the plate, this window size ensures that the individual pixel thresholds are adjusted to possible brightness changes, resulting from the illumination conditions. Yet, the size is large enough that regions in the image that contain no or very little structure are not larger than the averaging window. In such regions, bright pixels caused by noise or dirt would produce a very noisy thresholding result. The next step consists of morphological opening with a square 3×3 structuring element, which ensures that the plate is one connected object. The size of the structuring element is chosen to bridge small gaps caused by dirt or noise. Testing has shown, that a larger size would merge the plate with the background. Figure 3.2b shows the preprocessing output for the first sample input image, and Figure 3.3b shows the output for the second image. From the preprocessed image, connected components are extracted using the approach proposed in [8]. The approach operates very efficient, as each row in the image is scanned from top to bottom. Background pixels are visited once, and contour pixels a constant number of times. Therefore, the time required is linear in the number of pixels in the image. The authors state that their algorithm consists of the following four principal steps: When an unlabeled foreground pixel is encountered, the entire outline of the connected component is assigned the same label until the starting point is reached again (Figure 3.4a). If an already labeled pixel is encountered, the current row is followed until a background pixel is reached and all pixels are assigned the label of the initial pixel (Figure 3.4b). Should the next background pixel belong to an unlabeled internal contour, the entire contour is labeled until the starting point is reached (Figure 3.4c). Finally, if a labeled internal contour is encountered, the pixels in the row are followed and assigned the same label as the first pixel until a background pixel is found (Figure 3.4d).

We filter the extracted connected components, where only regions with an area that is larger than 6% and smaller than 90% of the image area are accepted. Next, the remain-

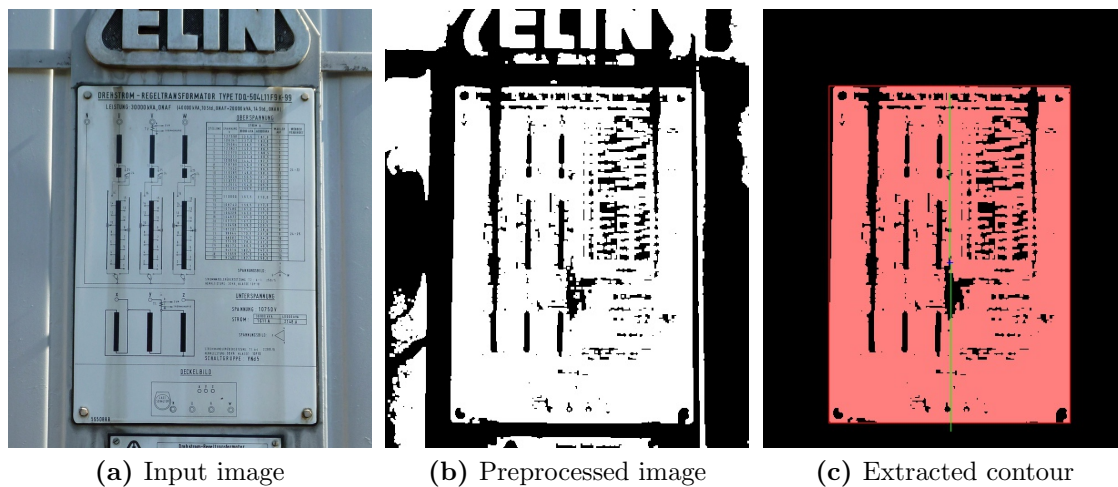


Figure 3.3: Nameplate detection steps for sample image 2. (a) Input image acquired by the user. (b) Preprocessed input image. As for the first sample image median filtering, adaptive thresholding, and morphological opening have been applied. (c) Filtered extracted components. Only a single contour remains.

ing connected components are approximated through a polygon. The maximum allowed distance between a contour pixel and the approximation is set to the perimeter of the connected component divided by the fixed value 20. A connected component is accepted if the approximation has three, four, or five corners. Triangles and pentagons are accepted to account for inaccuracies caused by the morphological closing. If the number of corners is not in this range, the shape is not similar enough to a quadrangle and the contour will not be processed further. Figure 3.2c shows the filtered contours, extracted from the preprocessed image, of the first sample. As we can see, four candidates remain after filtering. The two candidates at the right and left side are caused by the background, while the candidate at the top originates from another plate that is only partially visible. Further, Figure 3.3c displays the contour extracted from the second sample image. Only one connected component remains after filtering, which originates from the actual plate.

Since the color of the plate is not known, the thresholded image is inverted and the entire procedure repeated, which allows detection of plates with bright and dark background. Should the combined contour list contain more than one plate candidates, the object where the distance between its centroid and the image center is smallest is chosen. This removes cases where parts of the background form large connected components next to the plate, that have either a larger width or larger height than the centered object. This means, that a selection purely based on size or area cannot be used. Finally, if the absolute value of the object angle does not exceed ten degrees, the rotated object corners are returned as approximate plate location. Otherwise, the up-right bounding rectangle is returned. The angle is limited, because it is calculated using moments and therefore,

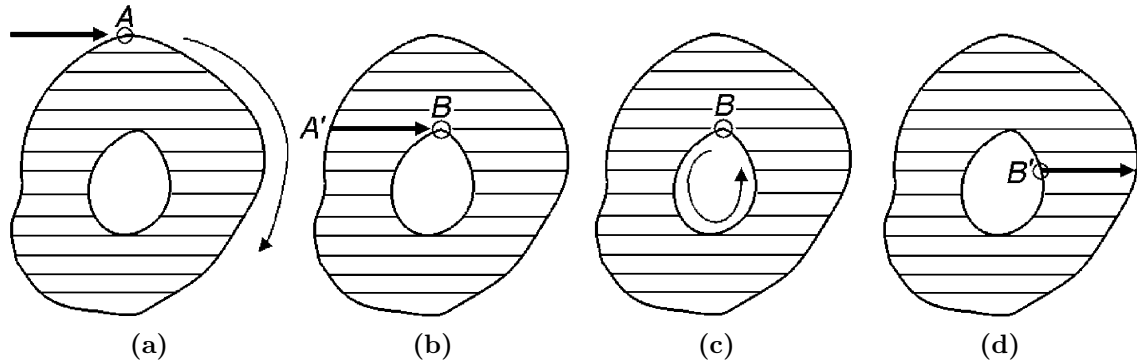


Figure 3.4: Major steps of the connected component extraction algorithm. (Images taken from [8])

concavities on the connected component may cause large and entirely wrong rotation angles. This constraint can be applied, as large angles are unlikely to occur since the user will likely keep the imaging device leveled when capturing the plate.

In summary, the outlined steps were chosen to allow efficient detection of quadrangles. A small amount of perspective distortion is likely present, but it does not require explicit handling, unlike in some related approaches which are based on line grouping. Furthermore, there are no assumptions about the plate color or content, only about its rectangular shape and central location in the image. Our approach is efficient and has low memory requirements, which makes it suitable for usage on mobile devices.

3.1.5 Plate Extraction

After the presence of a nameplate and its approximate location have been determined, the plate should be extracted as accurately as possible to simplify further processing. The boundary returned from the previous step is unsuitable, because of inaccuracies introduced by thresholding the image and the morphological opening. Furthermore, the rotation of the returned rectangle is only an estimate based on the extracted connected components and therefore unlikely to be accurate. The following steps operate on an input image with the same size $w_d \times h_d$ as the plate localization, and result in an image that contains only the up-right nameplate, extracted from the unscaled input image.

The first step consists of the retrieval of the plate outline using GrabCut [33]. The input for the GrabCut algorithm consists of a color image, as well as a mask that specifies for each pixel if it contains foreground, background, maybe foreground or maybe background. The authors use two Gaussian Mixture Models (GMM), one for the background and one for the foreground. Each iteration of the minimization algorithm first assigns the *GMM* components to the pixels, learns the *GMM* parameters and segments the image using a minimum graph cut. After convergence, this results in a segmentation with hard borders. To generate a visually pleasing result, the authors apply a step called border matting. The

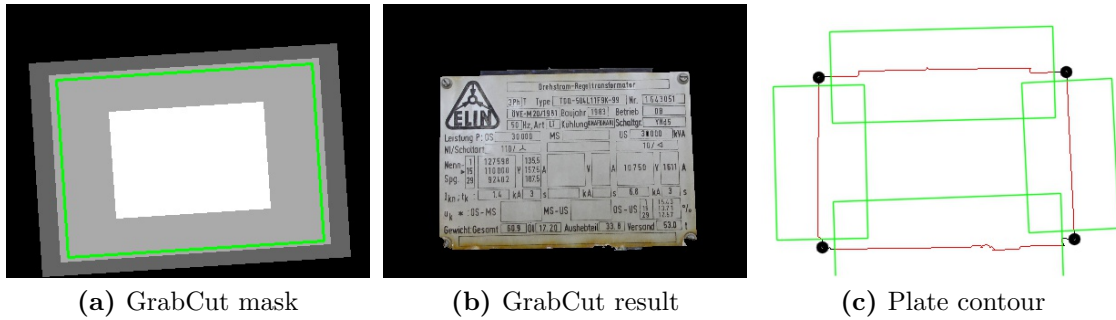


Figure 3.5: Nameplate extraction steps for sample image 1. (a) Mask generated for GrabCut. The white region indicates foreground, the light gray region probable foreground, the dark gray region probable background and the black region background. The green rectangle is the plate detection result. (b) GrabCut segmentation result. (c) Connected components of the GrabCut segmentation. The black circles are the corners of the approximated quadrangle, calculated from the contour. The pixels of the contour inside the green rectangles are used to fit lines.

contour of the thresholded object is retrieved and each pixel in the neighboring regions, orthogonal to the contour, is assigned an alpha value depending on the distance to the contour.

We use the approximate location of the plate (determined in the previous step) to generate the mask for GrabCut. The center of the approximate rectangle remains unchanged, only the size is scaled. A scale factor of 60% is used to indicate foreground, a factor of 105% indicates probable foreground and a factor of 120% probable background. The remaining parts of the image are marked as background. These values have been chosen empirically to maximize the probability of a proper segmentation, even if the approximated rotation angle is wrong. Figure 3.5a displays the generated mask for the first sample image. It shows the four different options starting from foreground (white) to background (black), as well as the result from the previous plate detection step as a green rectangle. Figure 3.6a displays the mask for the second image. In this case, the estimated angle of the plate, with a value of 89.8° , exceeded the plausibility threshold. As a result, the plate detection step returned the up-right bounding rectangle.

In case the previous step failed and no rectangle was detected, a statically generated mask with a fixed size is used. The backup mask is generated according to our assumption that the plate is centered and fills most of the image. Therefore, a rectangle scaled to 55% of the image size indicates foreground, a rectangle scaled to 75% indicates probable foreground and at a scale of 97% probable background is indicated. As before, the remaining pixels indicate background. Testing has shown, that if the input image meets our assumptions and the plate is within the mask regions that indicate foreground, the resulting segmentation is usable. Using the generated mask, one GrabCut iteration is performed.

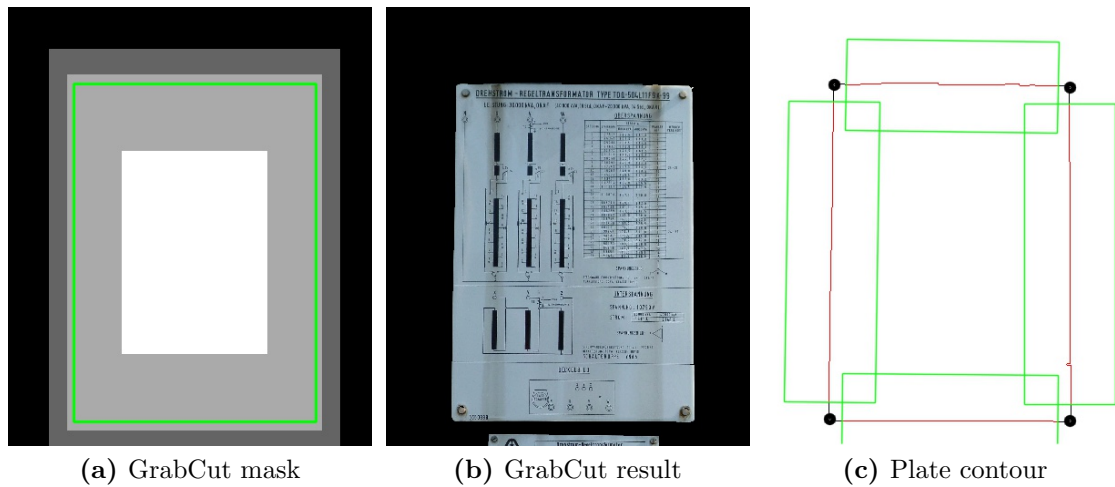


Figure 3.6: Nameplate extraction steps for sample image 2. (a) Mask generated for GrabCut. The color coding is the same as described in the figure for the first sample image. The green rectangle is the plate detection result. As the estimated angle of the connected contour is too large, the up-right bounding rectangle is used. (b) GrabCut segmentation result. At the bottom, separated from the nameplate, another structure is visible. (c) Connected components of the GrabCut segmentation. The red pixels, inside the green rectangles, are used to fit lines.

Since no user interaction to correct the mask is desired, the parameters were chosen accordingly. As such, one iteration is sufficient to retrieve the plate and more iterations would result in over segmentation. Figure 3.5b and Figure 3.6b display the segmentation for the two sample images. The segmentation result for the second image contains the visible portion of another plate at the bottom. The two plates are not connected, but the second one is included because they have the same texture and color. The GrabCut output is then used to retrieve the connected components of the segmented objects. Since the segmentation is usually not perfect, a number of contours are detected. At the borders, a number of small confined regions are often generated. The connected components are filtered and only the largest contour is processed further. To retrieve the corners, the contour is approximated as a polygon, starting with a small allowed distance between the connected component pixels and the approximation. If the resulting polygon does not have four corners, the allowed approximation error is increased and the approximated points are processed again. If the approximation is not a quadrangle after fifty iterations, the procedure aborts and the plate detection has failed. The approximated quadrangle is unlikely to be precise, because dirt that has accumulated at the borders, results in discolorations that throw off GrabCut. Furthermore, differently colored screws that are used to fasten the plate cause jagged contours and corners. However, the four corners are only used to retrieve all points from the original contour, contained in a 100 pixel wide search window,

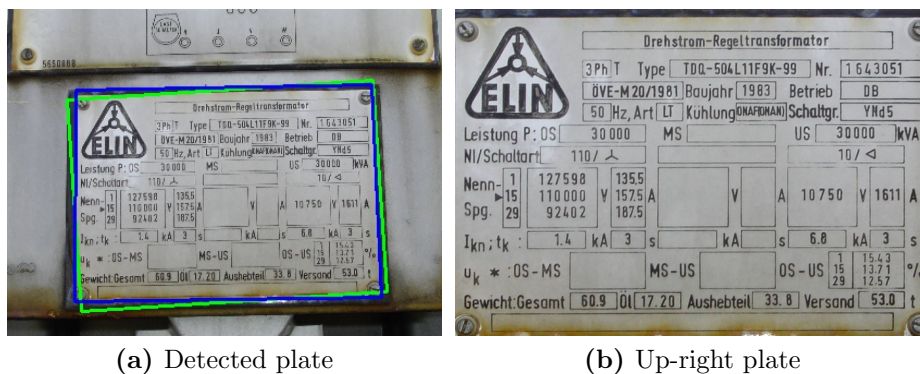


Figure 3.7: Plate detection results for sample image 1. (a) Plate detection result (green) and extracted rectangle (blue). The estimated rotation of the plate detection output is not correct. (b) Extracted, up-right plate. At the top, a portion of the plate border is included.

centered at the line connecting two corners. The euclidian distance between the corners is divided by a fixed value of 20, and the resulting value is used to offset the start and endpoints of the search rectangle from the detected corners. The offset and window size ensure that outliers, as well as the degraded corners are ignored. Furthermore, it prevents retrieval of contour points that belong to another side if the plate is rotated. Figure 3.5c and Figure 3.6c show for each sample image the connected component retrieved from the segmentation, the search windows, and gathered points of the contour. The extracted points are used to fit lines. These steps are performed for all four sides of the quadrangle, and the intersection of the lines results in the corners of the plate. Finally, to retrieve an up-right image of the plate, the corners are sorted clock wise. The maximum width and height are used as output size and a perspective transform is applied. Figure 3.7a shows the located plate positions of the detection (green) and extraction (blue) steps for the first sample image. The rotation of the green rectangle, which is the result of the plate detection, is not correct. This causes the inclusion of the plate border at the top, as can be seen in the final plate image, displayed in Figure 3.7b. The results for the second sample image are displayed in Figure 3.8. In this case, the result contains only the plate.

We should note that image segmentation using GrabCut is slow compared to the extraction of the approximate rectangle. However, since texture information is used, the resulting segmentation is far more accurate than the simple connected component based approach. Line fitting, using the points of the plate contour of the GrabCut output, results in accurate plate corners. Extracting the corners solely from the contour would be challenging because of the often degraded corners, caused by the fixation screws or dirt. After this step, the plate is available in an up-right position with the same size as in the unscaled input image.

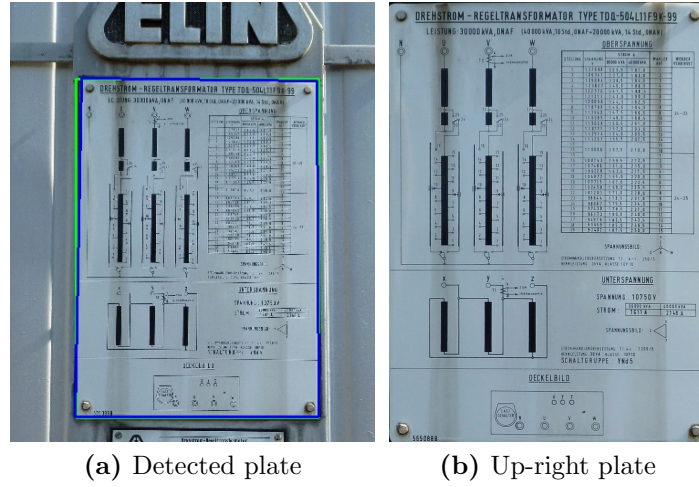


Figure 3.8: Plate detection results for sample image 2. (a) Plate detection result (green) and extracted rectangle (blue). (b) Extracted, up-right plate. The entire plate is included in the result.

3.2 Plate Classification

After the plate has been located and extracted, its type must be determined. We conquer the issue of numerous different types using a machine learning approach. In the literature, several related methods for the task of image classification have been presented. The goal of these approaches is to assign a category to the entire image. Therefore, no a priori assumptions about the content of the input image can be made, which differs from our situation. Nevertheless, the extraction and selection of image features is of interest.

Bosch et al. [3] propose an approach which classifies an image based on the categories of the objects it contains. The image is represented at different levels by dividing it into cells, arranged in a grid, where a higher level increases the number of rows and columns in the grid. In each cell, the authors compute features. Scale-Invariant Feature Transform (SIFT) features with four different radii, extracted at equally spaced points, represent the appearance of the objects in the image. These points are clustered into visual words. The shape of the objects is represented by histograms of oriented gradients. These values are then calculated at each level over the entire image, and used to detect regions of interest. A region of interest is a visually similar region that appears in multiple test images. Only these regions are used to train a random forest.

Zhou et al. [43] propose a classification framework that, like the previous one, operates with features extracted from cells in a grid. The first step of their approach consists of transforming the calculated descriptors, extracted at regular intervals, into a feature vector. The authors state that their coding allows the original and non linear function, as described by the feature vector, to be approximated by a linear function. They further explain, that any descriptor, such as *SIFT*, that takes the local neighborhood into account

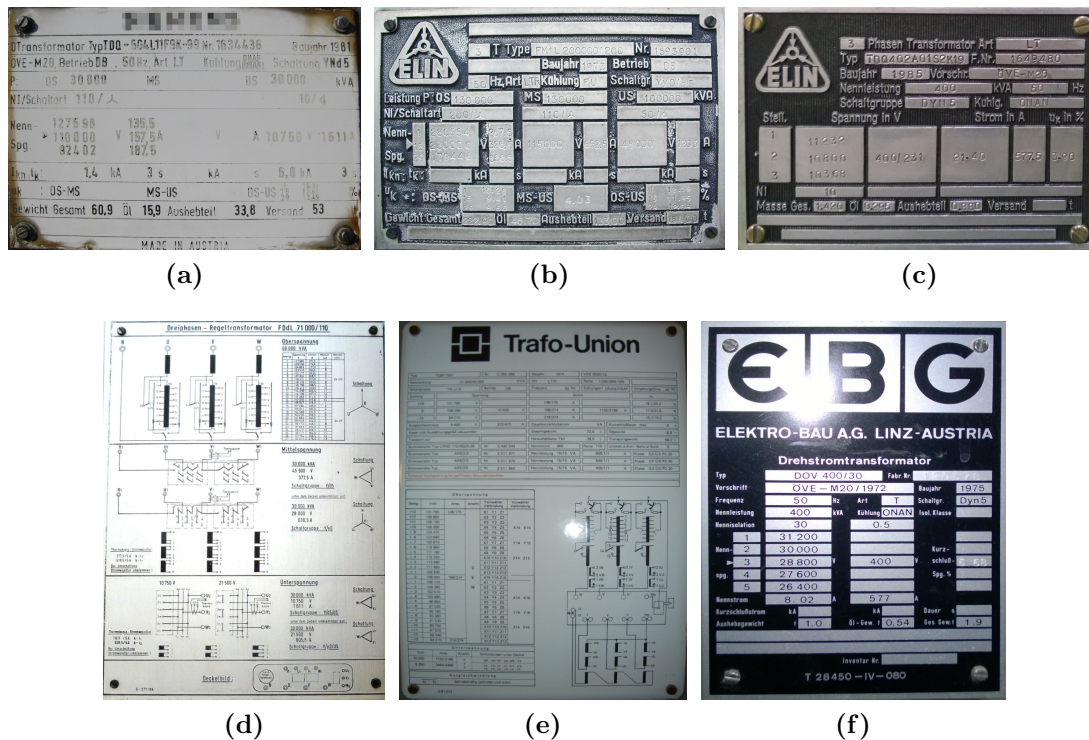


Figure 3.9: Sample images of different nameplates. Some vendor names have been obfuscated.

can be used. In the next step, the image is divided into cells, where feature vectors in the same cell are combined. The authors also combine feature vectors of different cells to retrieve descriptors that cover the entire image. Finally, linear support vector machines are used to perform the classification.

These approaches divide the image into small regions and compute descriptors, that encode the texture in each one. The descriptors are often *SIFT* or Speeded Up Robust Features (SURF), which are expensive to calculate. Since these approaches aim at classifying entire images into categories, no assumptions about the input image can be made. However, in our case a few assumption can be made: Plates usually contain a number of empty regions, and the non-empty content consists of text and line shaped structures. The positions of these regions do not change for plates of the same type. Therefore, we can more easily compute a feature for the entire plate that includes the spatial relationships of these regions. The extracted feature vector and the machine learning algorithm used in our approach are detailed in the following paragraphs.

3.2.1 Nameplate Feature Vector

Our feature vector contains information about the plate color, which makes it possible to differentiate nameplates with the same structure but different color, and histograms of

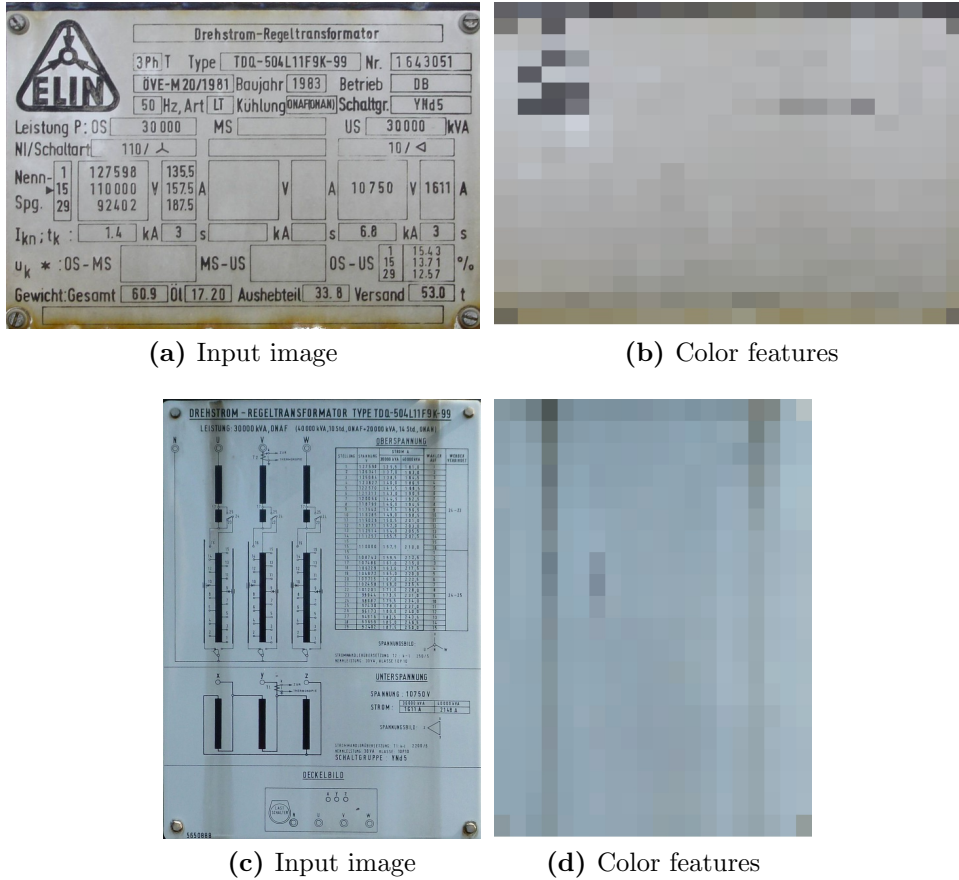


Figure 3.10: Input image and visualization of the color features, used for classification, of the two sample images. A 20×20 grid is used. The images that contain the color features have been converted from the CIE L^*a^*b color space to RGB for this visualization. (a) Extracted plate image for the first example. (b) Median color features extracted from the first sample plate. (c) Extracted plate image for the second example. (d) Median color features extracted from second sample plate.

local binary patterns, that encode the texture of the plate. The final feature is the size ratio of the nameplate, which helps to distinguish plates with similar content but different size proportions. For classification, we use a random forest. The feature extraction and random forests are described in detail in the following sections.

The image, that we use to determine the type of the plate, must only contain the nameplate in an up-right position. Such images are delivered by the previous plate extraction step. Some examples are illustrated in Figure 3.9. Initially, the image is resized to a fixed width of 800 pixels, where the new height is chosen to preserve the aspect ratio. This results in an image with a size of $w_c \times h_c$, that is used to extract all features.

3.2.1.1 Plate Color

The plate color is an important feature, because the content of many plates is laid out in similar structures, but the colors are different. To extract the color features, we first convert the input image into the CIE L*a*b* [10] color space. In this color space, the L-channel represents the lightness, the a-channel the color position in the red to green range, and the b-channel the color position in the yellow to blue range. We use the CIE L*a*b* color space, because it is designed to mimic human perception. This means, that a change in the perceived color creates a proportional change in the color values in this representation. Therefore, if the perception of the plate color only changes slightly, because it is weathered or stained, the color values also only change slightly. This is not the case in the standard RGB color space. Next, we divide the resized image into *numColsRows* columns and rows. This results in a number of non overlapping cells, depending on the size of the input image. For each cell and channel, the median color value is calculated, and the resulting three values are appended to the feature vector. We chose the median to be robust against outliers, regularly caused by dirt or an inaccurate plate segmentation at the borders.

Figure 3.10 shows two input images and the resulting color values that are added to the feature vector for the extracted plates of the test images. In the two samples, a 20×20 grid is used to retrieve the cells. Large dark regions, for example the vendor logo present in the left upper corner of the first plate, are noticeable in the extracted color features.

3.2.1.2 Local Binary Patterns

The next feature, added to the feature vector, consists of histograms of Local Binary Patterns (*LBP*) [32]. This feature encodes the texture of the plate. The *LBP* codes are computed by comparing the values of a fixed number of neighboring pixels to the value of the current center pixel. The neighboring pixels are evenly spaced around the center pixel, in a circle with a fixed radius. For each neighbor, the corresponding bit in the code is set to zero if its value is larger than the value of center pixel. If the value of the neighboring pixel is smaller, the bit is set to one. The output image has the same dimensions as the input image, and contains a *LBP* code for each pixel.

We extract the L-channel from the plate image that was converted to the CIE L*a*b color space, as it contains the intensity values. Next, we apply Gaussian blurring with a kernel that is 7×7 pixels large, which removes noise that would otherwise be incorporated into the *LBP* features. The blurred image is then used to calculate the *LBP* codes. In our implementation, we use *numLBPNeighPixels* neighboring pixels and a radius of *LBPRadius*. Next, the same grid structure that was used to extract the color features is applied to the image that contains the *LBP* codes. For each cell, we calculate a histogram of the codes, which contains *LBPHistSize* bins. The cell histograms encode the number of occurrences for each the direction, that are present in the corresponding area of the plate. The placement of the histograms in a grid structure is important, because it preserves the

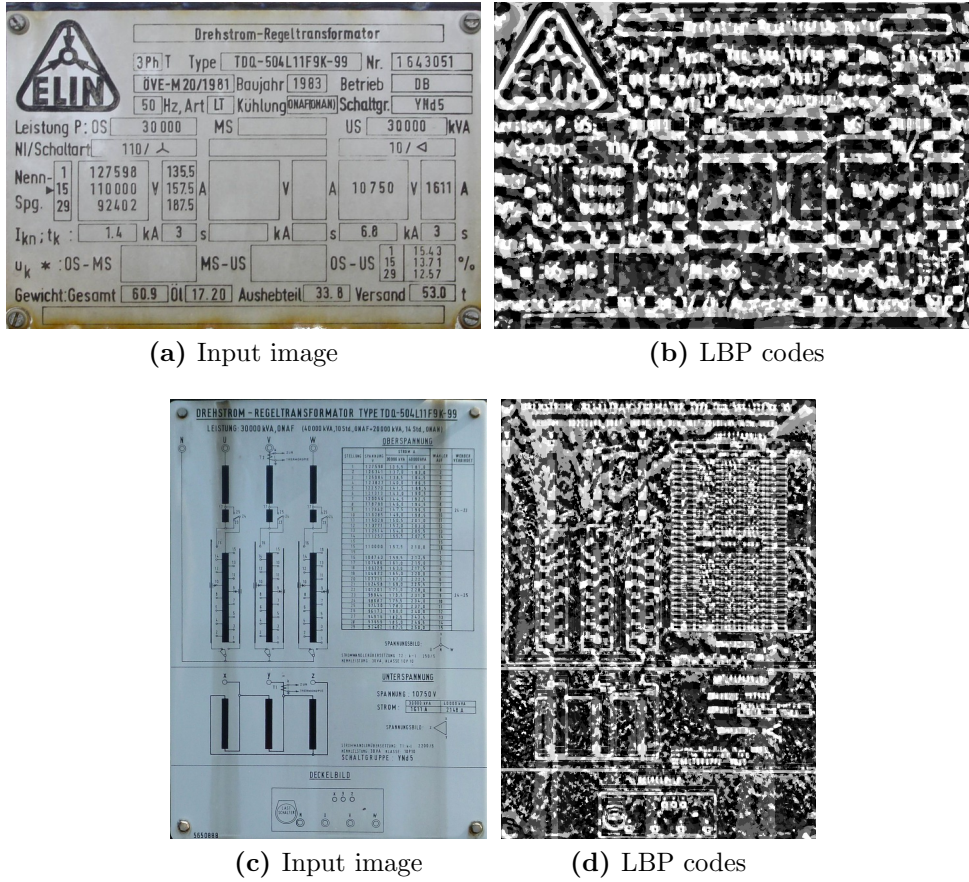


Figure 3.11: Input image and *LBP* codes used for classification, extracted from the two sample images. The radius is set to 16 and 4 neighbors are checked. The codes are normalized to the default grayscale range of 0 to 255 for display. (a) Extracted plate image for the first example. (b) *LBP* codes extracted from the sample plate. (c) Extracted plate image for the second example. (d) *LBP* codes extracted from the second plate.

spatial information that would otherwise be lost, if a single histogram would be calculated over the entire image. The *LBP* codes for the two sample plates, as well as the original input images, are displayed in Figure 3.11. To generate these images, a radius of 16, with 4 neighbors was used. The structure of the plate content is clearly recognizable in the two images.

3.2.1.3 Size Ratio

The final feature that is added to the feature vector consists only of a single value. We add it to distinguish plates that have a similar content structure and color, but different

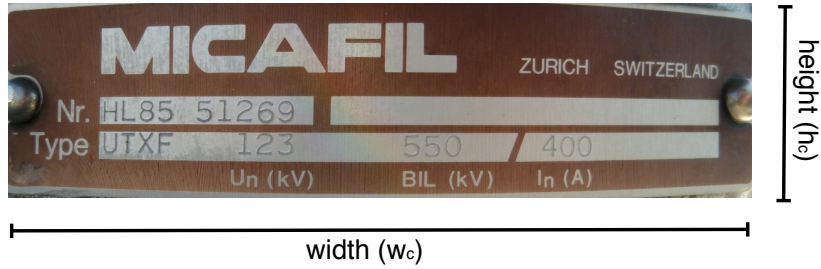


Figure 3.12: Schematic of the values used to calculate the size ratio.

size proportions. It is the aspect ratio of the plate image, calculated as

$$ratio = \frac{w_c}{h_c}. \quad (3.1)$$

We use the width (w_c) and height (h_c) values from the image, that contains only the extracted plate, as shown in Section 3.1.5. Figure 3.12 displays a schematic that contains the used values. We do not use the actual width and height values of the unscaled image, as this would add a dependency of the feature to the resolution of the camera used to capture the image. This is important, because the camera resolution varies with different capturing devices.

We should note that the size ratio is the least important of the three features. Nevertheless, it represents a characteristic of the nameplate that helps to distinguish types with similar content, but a different size ratio.

3.2.1.4 Final Feature Vector

In conclusion, the final feature vector that is calculated for each image, used for training and classification, contains median color values, histograms of *LBP* codes, and the aspect ratio of the plate image. The selection of the parameters used to extract the different features, referenced only with symbolic names (*numColsRows*, *numLBPNeighbors*, *LBPRadius*, and *LBPHistSize*), is described in Section 4.1.1. The following section describes and justifies our chosen machine learning algorithm, a random forest.

3.2.2 Classification using a Random Forest

Random forests were proposed by Breiman in [4]. A random forest consists of a number of decision trees, where each tree is trained on a different subset of the training set. The elements in the subsets are generated by bagging, which chooses elements randomly and with restitution. Therefore, a sample may appear multiple times. When training the trees, a random subset of the features in the feature vector is used to split the samples for

each node. In our case, the number of features used for the splits is half the square root of the number of all features. From all calculated splits, the one that causes the largest decrease of entropy and therefore the largest information gain of the label histogram is used. In our case, the learning is completed if *numRFTrees* trees are generated, or the estimated classification error is smaller than 1%. The error is estimated during training by classifying samples that were not chosen by the bagging procedure. The trees in the random forest are not pruned. For classification, each tree generates a result from the input data and a majority vote is used to derive the final class label.

We use random forests because they have a number of advantages. Breiman [4] states that increasing the number of trees does not cause over fitting. Therefore, a large number can be used safely, as it increases accuracy. However, with an increasing number of trees, the improvement in classification accuracy declines at a certain point. Furthermore, a larger number of trees increases the time needed for training, and linearly increases the time for classification. Training and classification are also very fast, since each tree can be processed in parallel. Finally, a very important factor is the classification performance. The supervised machine learning survey presented in [7] compares a number of popular approaches, including support vector machines, boosted trees, random forests and others. Eleven datasets that contain binary classification problems are used for testing. The authors conclude that there exists no universally superior algorithm, as no one excelled at each problem. Their test results show that calibrated boosted trees, random forests and bagged trees generally deliver the best performance. Another survey [38] compares machine learning algorithms for traffic sign recognition. Their test set consists of 43 classes and over 51,000 test images. The results list random forests on the third place of the machine learning algorithms. We evaluate our use of random forests in detail in Chapter 4.

To sum up, our approach to classify the plates uses a feature vector that consists of histograms of local binary patterns, median color values in the CIE L*a*b color space, and the plate aspect ratio. The features contain texture and color information of the plate and can be calculated efficiently, which is important when used on a mobile device. Random forests are used to classify the plates, as they are fast during training and classification and deliver good classification performance. The machine learning approach allows an operator to easily and quickly add new types. The major drawback is, that a relatively high number of images for each plate type are needed for training, in order to achieve desirable classification accuracy.

3.3 Text Detection

The next step consists of locating the text regions on the plate and matching them to a predefined list. For each plate type, a number of regions that hold information which should be extracted are defined beforehand. The located text regions are mapped to the user defined ones, and the image regions are passed to the *OCR* to convert them to

machine readable text. This section starts with an examination of popular text detection methods and subsequently our approach is explained in detail.

There exist a number of methods, that are specifically designed to extract text from natural images. One well known example is the Stroke Width Transform (SWT), proposed by Epshtein et al. [18]. The authors exploit the fact, that the stroke width of characters remains almost constant. The output of the first stage is an image of the same dimensions as the input image, that contains the stroke width for each pixel. First, they compute the edge map of the input image using the Canny edge detector and initialize all the stroke width result values to infinity. For each returned edge pixel, the gradient direction is followed until another edge pixel is found. If the gradient direction of the second edge pixel roughly points back to the starting pixel, all stroke width values on the line connecting them are set to the distance between the starting and the end pixel, but only if they do not already have a lower value. When all edge pixels are processed, all start pixels where an end pixel was found are revisited again. The stroke width values on the lines connecting them are set to the median value of all pixels on the line. The authors note that this is necessary to get correct results in corners. Next, they extract connected components, where neighboring pixels are assigned to the same contour if their stroke widths are similar. These contours are filtered to retrieve text regions. A contour is discarded if the stroke width variance of its enclosed pixels is too large, or when its aspect ratio or diameter to stroke width is not within a threshold. Furthermore, the bounding box of a contour must be properly sized and not contain more than two inner contours. The resulting letters are grouped into pairs if they have a similar stroke width, height, and color and are not located too far apart. Finally, the character pairs are merged into text lines and word boundaries are detected using a histogram of the character distances.

Gomez and Karatzas [20] propose another approach to detect text in natural images. They first detect character candidates using *MSE*R, which are filtered by size, aspect ratio, stroke width, and number of holes. For the remaining regions, the following features are calculated: Number of pixels, bounding box area, diameter of the bounding circle, mean gray and color value in the L^*a^*b color space for the region itself and its boundary, the stroke width as described above, and the mean gradient value of the outer boundary. Next, they calculate a dendrogram for each feature. For each node of the dendrogram, the probability that a number of regions share the same feature is calculated using the binomial distribution and used for clustering. Then, a co-occurrence matrix over all clusters is calculated. The final grouped regions are obtained by applying the previous clustering steps again to the co-occurrence matrix. An Adaboost classifier is used to prune regions that do not contain a character-shaped object. Finally, a second Adaboost classifier is used to remove clustered regions that do not contain text.

Two other approaches were already examined in Section 2.1 as part of a larger system. The first is described in [30] and uses category specific regions and a classifier that determines if the input region is a character or not. The detected text regions are grouped into license plates by incorporating knowledge of the layout. The second approach [16]

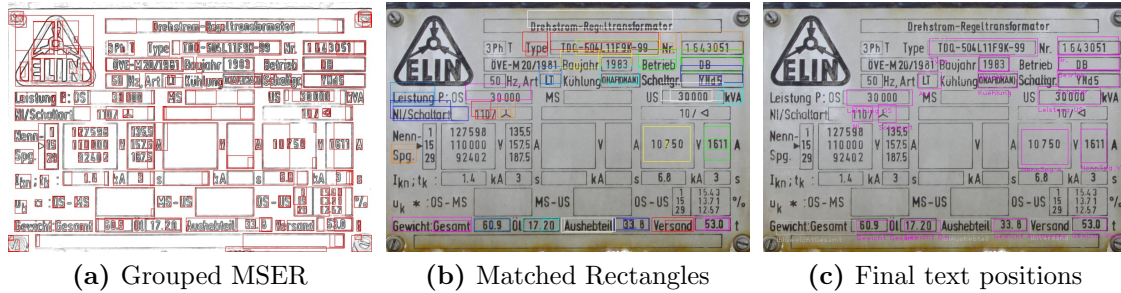


Figure 3.13: Text extraction, grouping, and matching for sample image 1. (a) Extracted *MSERs*, grouped into words. (b) Matches between detected and user-defined text regions. (c) Final text positions used for *OCR*. The text regions were moved to the bottom to account for the plate border at the top.

employs maximally stable extremal regions to locate license plates. The license plates are located by searching for a number of smaller regions placed within a larger one.

In summary, most of these systems are designed to detect text in natural images. Since our plate images do not contain nearly as much clutter as found in natural images, such sophisticated approaches like the *SWT* are not needed. Instead, we employ a similar *MSER* and grouping based approach to extract characters and words from nameplates, as presented in [16] for license plate detection. We have the extracted up-right plate image, as well as a list that contains the position, content type, and name of the regions that should be extracted. Since the plate is not always extracted perfectly, we need to adjust the given regions, so that they align with the content of the plate.

3.3.1 Character Extraction using Maximally Stable Extremal Regions

First, the gradient magnitude of the input image is calculated and normalized to the 0 to 255 range. To remove the majority of noise from this image, caused by rust stains and dirt, we set all pixels that have a value larger than 50 to the new value of 255. The threshold has been determined experimentally, so that it removes noise and the characters remain intact. The resulting image is used to extract *MSER* features.

Maximally stable extremal regions were proposed by Matas et al. [29]. The input image is thresholded at each intensity level, starting at zero which results in a completely white image. For each level, connected components are extracted and stored in a tree structure. As the threshold increases, black regions appear, increase in size, and finally merge, until the entire image is black. Each node in the tree represents an extremal region and its subtree contains the nested components. Finally, regions that are maximally stable are located and returned. A region is maximally stable, if its size remains mostly unchanged over a number of thresholding levels. The stability is calculated at each level i using the regions at the levels $i + \Delta$ and $i - \Delta$, where Δ is a user provided parameter. The authors

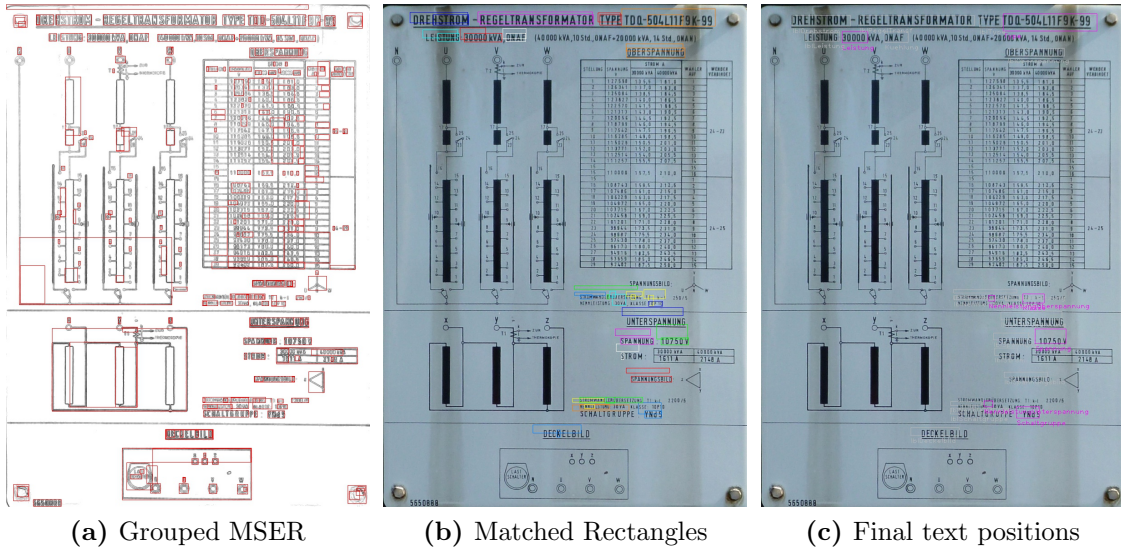


Figure 3.14: Text extraction, grouping, and matching for sample image 2. (a) Extracted and grouped *MSERs*. (b) Matches between detected and user-defined text regions. (c) Final text positions.

note that *MSER* is very fast to calculate, invariant against affine transformations of image intensities, and offers multi scale detection.

We set the Δ parameter to 5, the smallest allowed area of a region is $0.005 \cdot h_c \cdot 0.00125 \cdot w_c$, and the maximum area is $0.05 \cdot h_c \cdot 0.05 \cdot w_c$. Furthermore, all regions with an aspect ratio smaller than 0.2 are discarded.

3.3.2 Character Grouping

The regions are grouped into words, where two regions are combined if their height difference is smaller than 30 pixels, the distance on the x-axis between their bounding boxes is smaller than 5 pixels, and their bounding box bottom positions on the y-axis differ by less than 15 pixels. The iterative grouping procedure stops if no more regions can be merged. Figure 3.13a and Figure 3.14a show the grouped regions of the two input plates, while Figure 3.15 shows a schematic of the grouping conditions. Next, the grouped regions are mapped to the user provided list, which contains their position in a reference plate, the content type, and the names of the regions that are used to store the results. A detected region is assigned to the defined region, where the euclidian distance between the two bounding box centers is the smallest. Since this simple mapping method is likely to produce incorrect matches, usually caused by missing text regions on the plate or wrong character grouping, the correspondences are used to estimate a rigid transform. The coordinates of all user defined regions are warped using the calculated transform, resulting in text regions with the user specified size and positions adapted to the detected

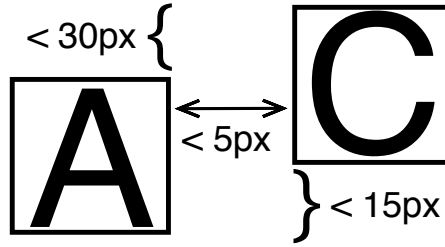


Figure 3.15: Criteria for character grouping.

regions. Figure 3.13b and Figure 3.14b show the matched regions and Figure 3.13c and Figure 3.14c the final text positions for the two sample plates.

At this point, we have a list of text regions and their positions on the full size plate image. The regions contain images of the desired content, which are transformed by the *OCR* into machine readable text in the next step.

3.3.3 Noise Filtering

The images of the text regions frequently contain structures that are not part of the text, caused by borders around the text or neighboring text regions, as well as dirt on the nameplate. Since most *OCR* systems are developed to process scanned text, which mostly consists of well segmented characters, these additional structures would deteriorate the output significantly. Therefore, preprocessing to remove them is applied. The listed steps are applied to each extracted text region, which are provided by the previous stage. The text regions are extracted from the unscaled plate image. As such, the size $w_f \times h_f$ of a region depends on the image size. An input region may also contain multiple lines of text.

First, the absolute gradient magnitude of the input image is computed. Using the gradient instead of adaptive thresholding yields better results if the image contains a gray value gradient, which is frequently caused by rust stains. In such cases, simple thresholding would create large black regions unsuitable for further processing. The minimum and maximum gradient values are determined and used to calculate an adaptive threshold for filtering, where all values smaller than the threshold are set to zero. To retrieve the threshold, a histogram with 256 bins is calculated over the magnitude values and the bin with the largest amount of values is located. This is done under the assumption that the largest number of gradients is produced by text. A fixed thresholding value is unsuitable because embossed text has very low contrast and would be removed by blurring the image before calculating the gradient. For further processing, the filtered image is normalized to the 0 to 255 range and adaptively thresholded, where the block size equals half the image height, resulting in a binary image. Figure 3.16b shows a sample binary image for printed text, where the rust stain from the input image is not visible anymore. Further, Figure 3.17b shows another example for embossed text, that contains a lot of noise.

Next, possible horizontal and vertical lines, caused by borders around the text, are

removed. To remove horizontal lines, the image is morphologically opened using a structuring element with the size $w_f/2.5 \times w_f/2.5$, that contains a one pixel wide line at the center. The opened image is subtracted pixel-wise from the original image. Since the character width is small in relation to the structuring element, they remain intact. Figure 3.16c illustrates this for the first sample region. As the lines are not leveled, some parts of them remain. However, using the same approach to remove vertical lines would also remove most of the text. To prevent this, the presence and location of borders is determined first, and only those regions are filtered. At the first 20% of the left and right side of the image, peaks are detected. A peak is reported if 70% of all pixels in a column are black. Since we assume that the text is roughly centered, the distance of the first detected peak on either side to the image border should also be approximately the same. If a peak on each side has been detected and the distance differs by more than $w_f/12$, no filtering is done. This condition is also often triggered by the first and last letter of the text. Figure 3.17c shows the detected border at the left side of the second sample region. No borders are reported for the first image, hence no border removal is necessary. The sides of the image where a filtering region has been found are morphologically opened using a structuring element with the size of $h_f/6 \times h_f/6$, that contains a one pixel wide centered vertical line. Again, the opening result is subtracted pixel-wise from the original image. Since the removed lines usually are not perfectly horizontal or vertical, some debris of them remains. Figure 3.17d displays the filtering result for the second region. To remove the noise, we extract and filter connected components. A connected component is removed if its bounding box touches the borders of the image, the bounding box width and height are smaller than three pixels, or the area of the connected component is smaller than $(w_f \cdot h_f)/800$. The filtered regions are shown in Figure 3.16d and Figure 3.17e.

Next, the individual words are grouped into lines of text. Therefore, we first sort all detected words by the y-coordinate of the centroid of their bounding box in ascending order. A new line is started if the y-coordinate differs from the previous one by more than 20% of the text region height (h_f). To assign commas and accents to the correct line, the overlap with the position of the starting contour is checked. After all characters have been assigned to lines of text, each line that contains more than five characters is checked for remaining border fragments that were not discarded earlier, because they are skewed. The bounding box of the first and last characters of the line are checked against the median bounding box contour height. If the height difference to the median contour height is larger than 25%, the bounding box does not overlap with the bounding box of the neighboring contour, and the bounding box width is smaller than 33% of the text region width (w_f), the contour is removed. At this point, the preprocessing is complete. However, the characters only consist of borders, as all processing was done on an image created by thresholding the gradient magnitude (e.g. Figure 3.16d). Such images cannot be processed by the Tesseract *OCR*. To retrieve filled characters, the original unmodified input image is adaptively thresholded with a block size of $w_f/7$ and the filtered image is morphologically opened with an 20×15 ellipsoidal structuring element. Figure 3.16e shows

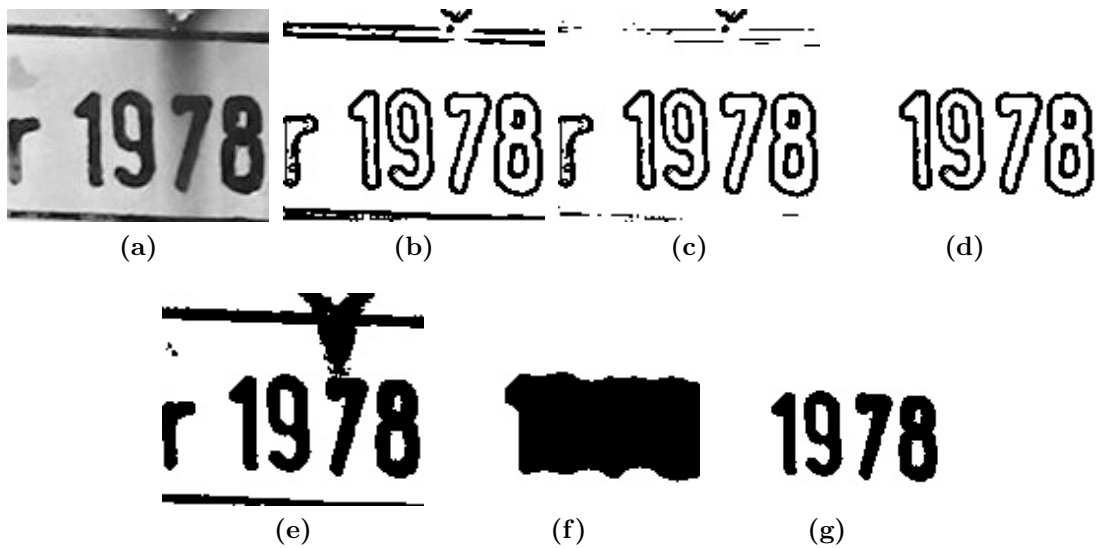


Figure 3.16: *OCR preprocessing steps illustrated on printed text.* (a) Input image. The leftmost character does not belong to the region. (b) Binary image generated from the gradient magnitude. (c) Horizontal and vertical line removal result, no vertical borders were detected. Some parts of the horizontal lines remain. (d) Remaining structures after size and location filtering. (e) Adaptively thresholded input image. The discoloration above the digits remains visible. (f) Mask, generated from the filtered gradient magnitude image. (g) Masked regions, copied from the adaptively thresholded image.

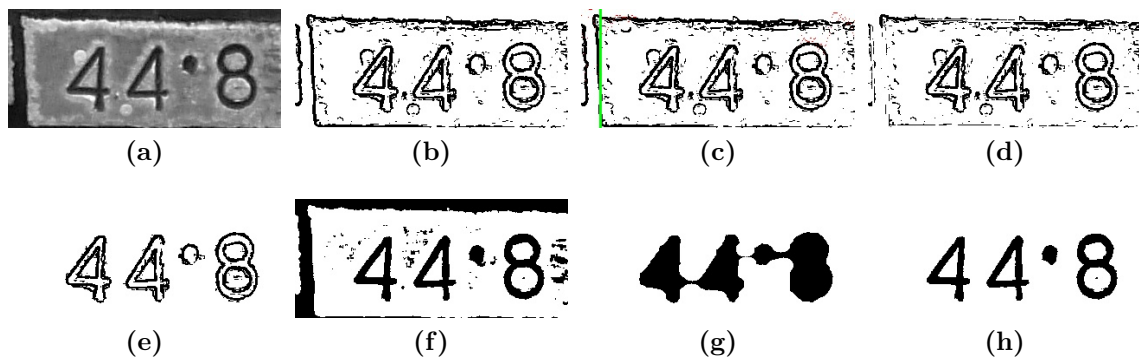


Figure 3.17: *OCR preprocessing steps illustrated on embossed text.* (a) Input image. The quality and contrast is good for embossed text. (b) Binary image generated from the gradient magnitude. The uneven metal results in noise. (c) A border in the image was detected at the left side. (d) Horizontal and vertical line removal result. Since the lines are not straight, some parts remain. (e) Remaining structures after size and location filtering. (f) Adaptively thresholded input image. (g) Mask, generated from the filtered gradient magnitude. (h) Masked regions copied from the adaptively thresholded image.

the thresholding result for the first image. The large black region is caused by the rust stain. The thresholding result for the second regions, shown in Figure 3.17f, also contains some clutter. The masks for both images are shown in Figure 3.16f and Figure 3.17g. The opened image is used as a mask, when copying from the adaptively thresholded image. The results for both regions that are passed to Tesseract are displayed in Figure 3.16g and Figure 3.17h. As we can see, everything that does not belong to a character has been removed.

3.4 Text Extraction

The process of assigning the character class to an image that contains a rendering of the character, is called Optical Character Recognition (*OCR*). A number of papers examined in Section 2 contain their own *OCR* approaches. Chang et al. [9] use the format of license plates to separate digits from characters and topological sorting to reduce the number of templates, to which the input is compared. A neural network is used to classify the image. Luo et al. [28] employ a two stage classifier that determines the result by comparing a number of extracted features from the input to a set of templates. Finally, Donoser et al. [16] use a support vector machine [11] to classify the extracted *MSE*R regions of the license plate. As previously mentioned, these approaches are unlikely to provide good results in our setting, because the authors exploit a number of constraints that do not hold for nameplates. Examples are a known format of the string, only digits or no digits in words, and a limited number of fonts and character classes. For our approach, we decided to use the Tesseract *OCR*, designed to process scanned sheets of text.

The Tesseract *OCR* is well known, produces good results, and is widely used. It initially started as a research project, but it is now available under an open source license. The Tesseract algorithm is described in [37]. First, connected components are extracted and grouped into lines. The line slope is tracked over the entire image region, therefore deskewing of the input image with its associated loss of resolution is not needed. For each line, the initial baseline is calculated using characters that are continuously placed. Tesseract is able to process curved baselines which occur frequently in scanned documents. Next, the lines are separated into characters. In case of fonts with a fixed character width, the gaps are analyzed and split. Otherwise, a limited region above the baseline is analyzed, separated, and classified. The words with the lowest recognition score are split into further characters, until the result improves. If this does not improve the recognition score, contours are merged and classified again to check the change in the recognition score. The features for classification are extracted from small fixed size patches over neighboring contours and matched against the trained character representations. The authors note that this approach enables Tesseract to easily handle characters broken into multiple parts, which would not be possible if each contour is processed separately. The following classification is done in two steps, the first selects possible candidates which are compared to each other in the second stage to retrieve the result. The detection result is




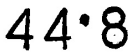
Original	Text	Score	Preprocessed	Text	Score
	@	54%		1978	81%
	i 44;8s	61%		44'8	86%

Table 3.1: Tesseract output for the samples from Figure 3.16 and Figure 3.17. The value in the score columns is the Tesseract recognition confidence.





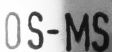



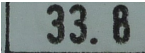









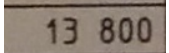

Original	Text	Score	Preprocessed	Text	Score
	-9-8-	51%		56	56%
	3-979-	33%		1970	74%
	os-I	78%		OS-MS	82%
	No output	N/A		FU	89%
	93498	34%		33. 8	84%
	TSESE	51%		3Ph	80%
	330000	73%		30000	90%
	No output	N/A		20 800	80%
	5 73-	2%		2013	74%
	'7 3 -16609	37%		13 800	82%

Table 3.2: Samples of typical text regions where the *OCR* output improves after preprocessing. For each sample, the unmodified image, the preprocessed region, and the corresponding Tesseract output are listed. The value in the score columns is the Tesseract recognition confidence. For each sample, the output after preprocessing is correct.

also improved by incorporating a dictionary. Furthermore, while the text is processed, a second classifier is trained with the output of the first one. After the first classifier has processed the entire page, the second classifier is used to refine the result.

Since Tesseract is designed for scanned text, preprocessing as shown in Section 3.3.3 is necessary to obtain good results. The output of Tesseract for the examples shown in



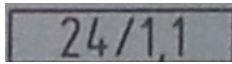
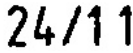



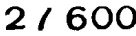
Original	Text	Score	Preprocessed	Text	Score
	9 m	31%		Lr	60%
	C27+E	27%		24/11	68%
	20-4/25,7	72%		20 4/25 7	84%
	27600	43%		21600	73%

Table 3.3: Text regions where the preprocessing removes parts of the text. The unmodified and preprocessed regions, as well as the resulting *OCR* output is displayed. The value in the score columns is the Tesseract recognition confidence. In each region the preprocessing removes some part of the content, specifically commas and portions of characters.

Figure 3.16 and Figure 3.17 can be seen in Table 3.1. The table contains the unmodified and preprocessed text regions. For each region, the corresponding Tesseract output, as well as the recognition confidence are listed. Further examples of typical text regions are shown in Table 3.2. In general, the output for the original images is always incorrect or even missing. In contrast, the output for each preprocessed region is correct. However, there also exist cases where the preprocessing modified the actual content. A number of such cases are shown in Table 3.3. In two examples, parts of the characters are removed, while in the other two examples, the commas are removed. It should be noted, that only in one case the output of the unmodified image is correct, while the preprocessed one is not.

It should be noted that we disable the internal dictionary, as most of the strings we need to process are arbitrary combinations of digits and letters. Therefore, using the dictionary would worsen the result by altering the correctly detected characters. The Tesseract project provides a number of language files, trained with fonts that are common in scanned documents. Unfortunately the shape of the characters of the fonts used for training are different from the ones used on most nameplates, which results in wrong output. Furthermore, the thresholding algorithm, applied internally by Tesseract to the input image, is not designed to handle uneven illumination and noise, which results in mostly unusable output. However, first problem can be tackled by training a new language with character samples from the unknown font. To reduce the issues caused by noise and problematic illumination, we apply our preprocessing step.

After the *OCR* is finished, we have the text in machine readable form and the confidence score for each region. If all confidence scores are larger or equal to 80%, the results are stored and entire plate detection and classification procedure is finished. However, it is possible that shadows and partial occlusions cause low scores. Our approach to handle such cases is presented in the following section.

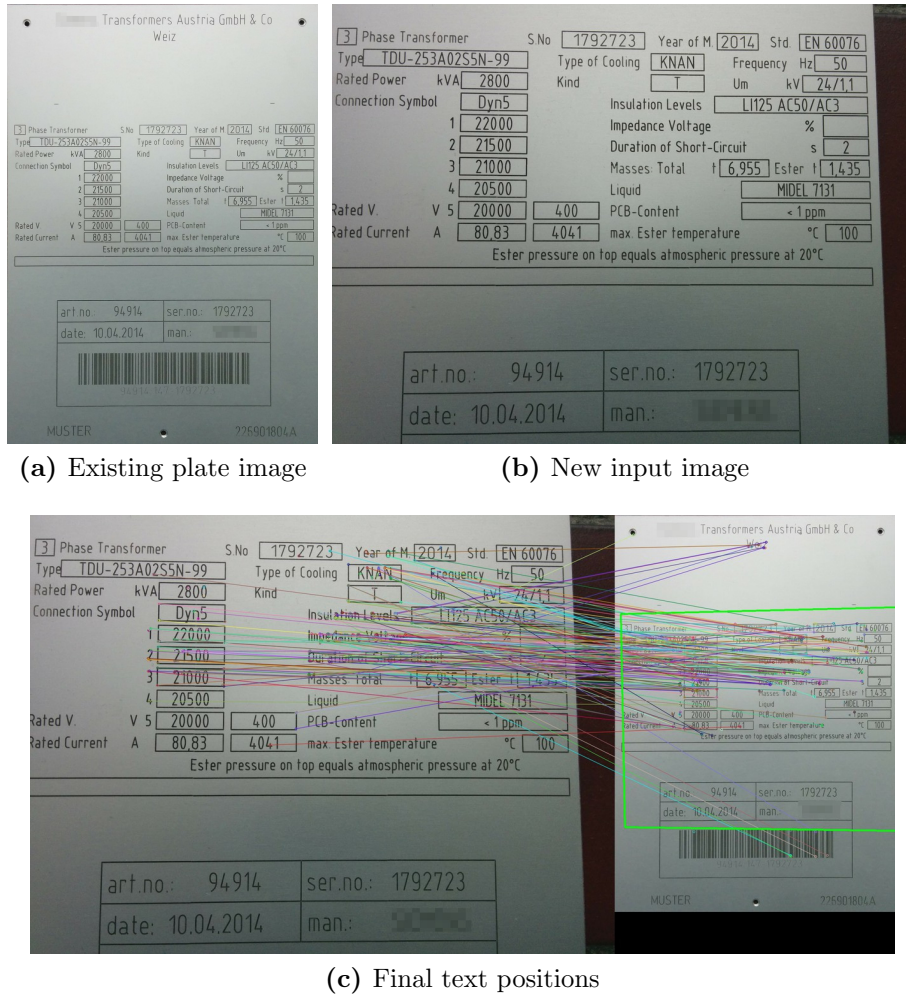


Figure 3.18: Matches between the existing plate and a new image. (a) Image of the existing up-right plate image. (b) New image of the right plate area. (c) Visualization of the keypoints, determined matches, and location of the new image in the existing plate.

3.5 Guided Image Acquisition

The final step of our approach consists of requesting and processing further images of the nameplate, if the *OCR* confidence score for at least one text region is low. The user is prompted with a request to acquire a new image of the approximate region of the plate, where the most problems occurred.

The extracted plate and the initial *OCR* results are stored as reference. Should the averaged recognition score of at least one region be lower than 80%, a new image is requested. The image is separated into four disjunct parts (top, bottom, left, and right) and a new image is requested for the region, in which the most errors occurred. Keypoints

are extracted from the new image and the stored plate using the Oriented FAST and Rotated BRIEF (ORB) [34] detector, and Binary Robust Invariant Scalable Keypoints (BRISK) [27] descriptors are calculated. *ORB* and *BRISK* are used because of their lower computational and memory requirements as well as comparable performance to *SIFT*. A homography, that maps the new image to the reference plate, is calculated using the best 200 matches and Random Sample Consensus (RANSAC). Figure 3.18 shows a sample of an existing up-right plate image, and a new input image that contains the right portion of the plate at a larger magnification. Further, the extracted keypoints, the calculated matches between the images, and the location of the new image in relation to the reference plate, are displayed. Each completely visible region is warped into its up-right position, where the size is approximately the same as in the new input image. Since each region is processed and warped separately, the process can be executed in parallel and less resources are needed since not the entire image must be processed. The preprocessing and *OCR* steps, as described earlier, are executed for each new region. The results are merged with the existing ones, where for each region only the one with the highest *OCR* confidence is kept. If there still exist regions with a low recognition score, and the maximum number of retries is not exceeded, another image is requested and processed as already described. Should the number of retries be exceeded, the process stops and the user is informed about the low score regions. The application experiments, presented in Section 4.3, show the input images and the *OCR* output for three plates, which also includes the additionally requested images.

3.6 Summary

In this chapter, all steps of our approach to detect, classify and extract text from nameplates were described. Contours are used to roughly locate the plate and to create a mask for its extraction using GrabCut. The feature vector used for classification of the extracted plate consists of local binary patterns, median color values, and the size ratio, and is used in conjunction with a random forest. The positions of the text regions that should be extracted for each type are mapped to the text regions of the extracted nameplate. Each region is preprocessed before it is passed to the *OCR*, which improves the detection result. Finally, should the *OCR* recognition score be low, a new image of the problematic regions is requested from the user and processed again. In the next section, the proposed steps are evaluated, with special attention placed on the nameplate classification part. Additionally, some results under different lighting conditions, as well as breakdown cases are shown.

Evaluation and Experiments

In the previous chapters we have shown our approach to locate, segment, and classify nameplates, as well as text localization and Optical Character Recognition (OCR). This chapter contains an evaluation of all these steps, where we primarily focus on the classification of nameplates, and show experimental results obtained using an Android device. First, we evaluate the individual features that are extracted from the plate. They are tested separately and combined, with different parameters for the machine learning algorithm. Next, the output of the *OCR* is compared to the actual plate contents. Since the used *OCR* is off-the-shelf software with no runtime tuning parameters (just training), only the effect of training with the actual fonts that appear on the plates is evaluated. Finally, we test three available plate types on an Android device and show the results. This includes different lighting conditions, such as shadows and reflections, and their impact on the processing steps. Furthermore, we show a number of breakdown cases, such as extreme angles when capturing the image, or blurriness caused by insufficient lighting.

The dataset used for the experiments consists of 125 images of nameplates from 15 classes. The number of samples is relatively low, because it is not easy to acquire images of real world aged plates for all the different types available. Table 4.1 lists the number of images per class, and Figure 4.1 shows a sample plate from each class. Eleven classes are composed of images of aged plates, that are mounted on deployed devices. These plates are for the most part thirty or forty years old. The final four classes (*k*, *l*, *m* and *n*) are composed of new plates and were photographed using an Android device. Due to the low number of samples, all tests are conducted using leave-one-out cross-validation, where each sample (from first to last) is classified while the rest is used to train the classifier.

Class	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	<i>h</i>	<i>i</i>	<i>j</i>	<i>k</i>	<i>l</i>	<i>m</i>	<i>n</i>	<i>o</i>
Samples	7	10	7	8	15	4	4	5	3	8	12	12	13	11	6

Table 4.1: Overview of the available set of nameplate images used to perform the experiments.

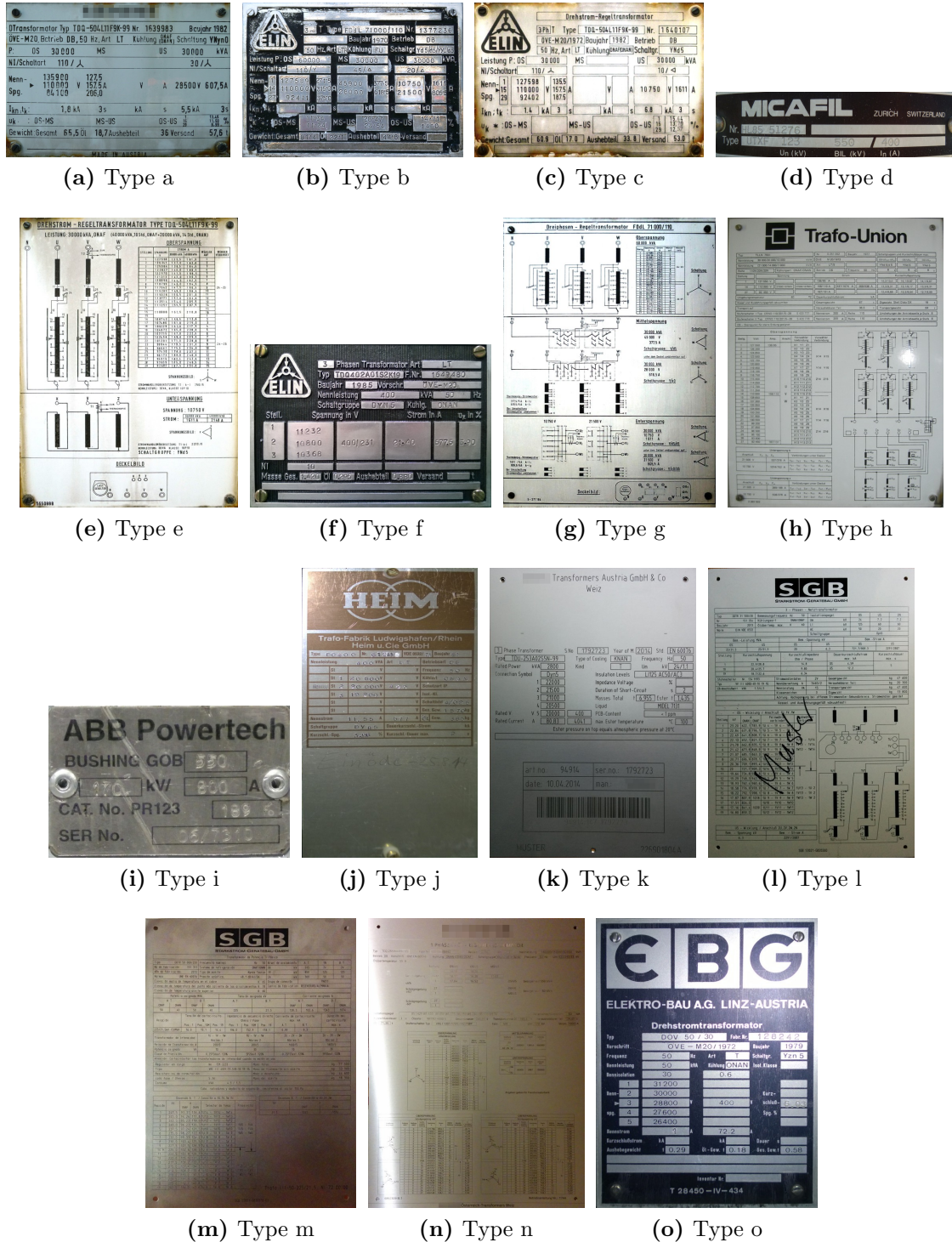


Figure 4.1: A sample image for each plate type. Some vendor names have been obfuscated.

Parameter	Possible values	Final value
<i>numColsRows</i>	10, 20, 30, 40, 50, 60	20
<i>numLBPNeighPixels</i>	4, 8, 16	4
<i>LBPRadius</i>	2, 4, 8, 16	16
<i>LBPHistSize</i>	30, 40, 50, 60	60
<i>numRFTrees</i>	100, 200, 300	300

Table 4.2: Grid search results for the feature extraction parameters. For each parameter, the possible values and the final value is listed.

This chapter is structured as follows: In Section 4.1, the features are used individually to classify the dataset, and then the results of the combinations are shown. Section 4.2 evaluates the output of the *OCR* and shows the impact of training the actual fonts used on the plates. Finally, Section 4.3 shows experimental results conducted on an Android device, as well as extreme cases that lead to failure.

4.1 Evaluation of the Classifier

First, the three different types of features (color values, Local Binary Patterns (LBP), and size ratio, as described in Section 3.2) are extracted from the plate and tested individually to evaluate their significance. Next, the features are calculated using their optimal parameters, which were determined using grid search. Then we combine them and examine their contribution to the final classification result.

The parameters of the random forest are set to generate at most 300 trees, with a maximum depth of 50 and a sufficient training accuracy of 1%. The split size is set to the square root of the number of features in the feature vector. All tests are executed on a notebook, which contains a 2.4GHz Intel Core 2 Duo processor and 4 GB RAM. The code is written in C++ and uses the OpenCV 2.4.9 framework¹ to store and process all images, as well as its random forest implementation. GPGPU capabilities, such as CUDA or OpenCL, are not utilized.

4.1.1 Parameter Selection using Grid Search

The previous chapter, that explained our approach in detail, referred to a number of parameters that are used to extract the features only in the form of symbolic constants. The following variables were used: *numColsRows*, *numLBPNeighPixels*, *LBPRadius*, *numRFTrees*, and *LBPHistSize*. We have selected the actual values for these parameters for the full feature vector using grid search. This includes the selection of a number of meaningful values for each parameter. To retrieve the best combination, all possible combinations are tested. In our case, we performed leave-one-out cross validation for each combination and

¹<http://opencv.org/> (last visited January 22, 2015)

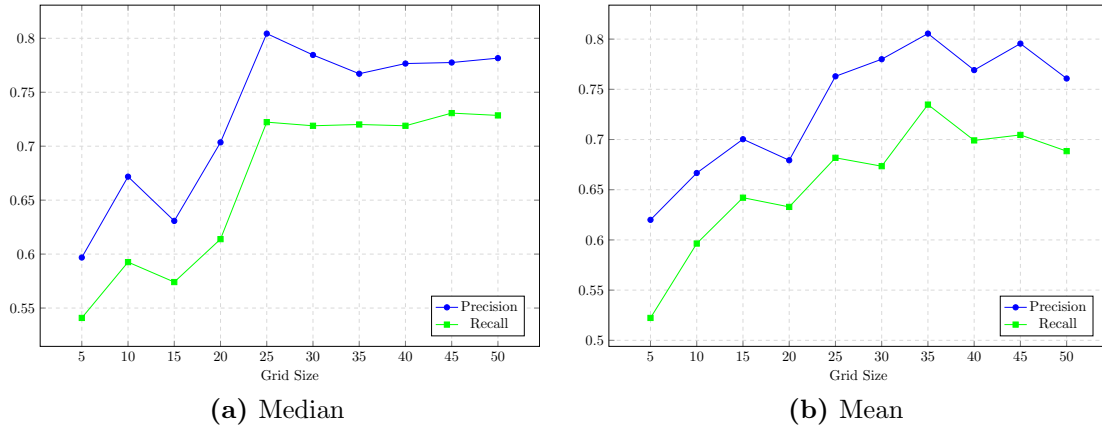


Figure 4.2: Precision and recall for different number of rows and columns in the CIE L^*a^*b color space. (a) Results when using the median cell values. (b) Results when using the mean cell values.

stored the precision and recall values. For the chosen values of the features, this resulted in 576 possible combinations, that required approximately a month of runtime to calculate. Table 4.2 contains the possible and best values for each parameter.

4.1.2 Plate Color

The color features are used to distinguish different types that have a similar content. We use the median color in the CIE L^*a^*b color space, as explained in detail in Section 3.2.1.1.

The plots shown in Figure 4.2 display the resulting precision and recall values for different numbers of rows and columns, where only the color information in the CIE L^*a^*b color space is used for classification. When calculating the median color value for each cell, the best result is achieved by splitting the image into 25 rows and columns, resulting in a precision of 80.43% and a recall of 72.22%, as displayed in Figure 4.2a. Using a larger number of rows and columns decreases the performance. The results of calculating the features using the mean instead of the median are shown in Figure 4.2b. The best classification result is achieved by dividing the image into 35 rows and columns, resulting in a precision of 80.55% and recall of 73.47%. Compared to the results obtained using the median, the precision increases by 0.12% and the recall by 1.25%. Further, Figure 4.3 shows the results when the RGB color space is used to extract the features. When extracting median color values, the overall best results are produced with 35 rows and columns, resulting in a precision of 80.91% and a recall of 73.79%. Compared to the best result obtained using the median in the CIE L^*a^*b color space, the precision increases by 0.48% and the recall by 1.57%. Finally, when using the mean RGB values, the best performance is achieved with 40 columns and rows, resulting in a precision of 78.87% and recall of 72.70%. Overall, this is the worst result and the feature vector needed has the highest number of dimensions.

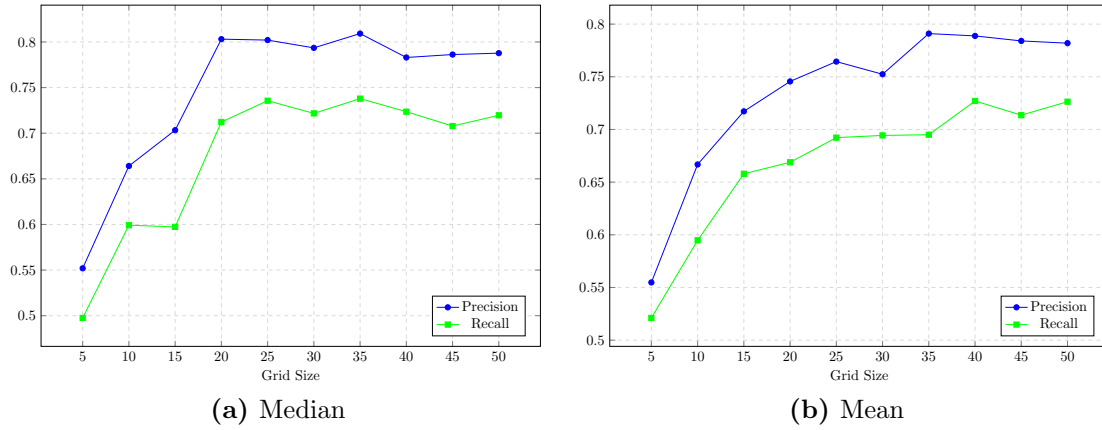


Figure 4.3: Precision and recall for different number of rows and columns in the RGB color space. (a) Results when using the median cell values. (b) Results when using the mean cell values.

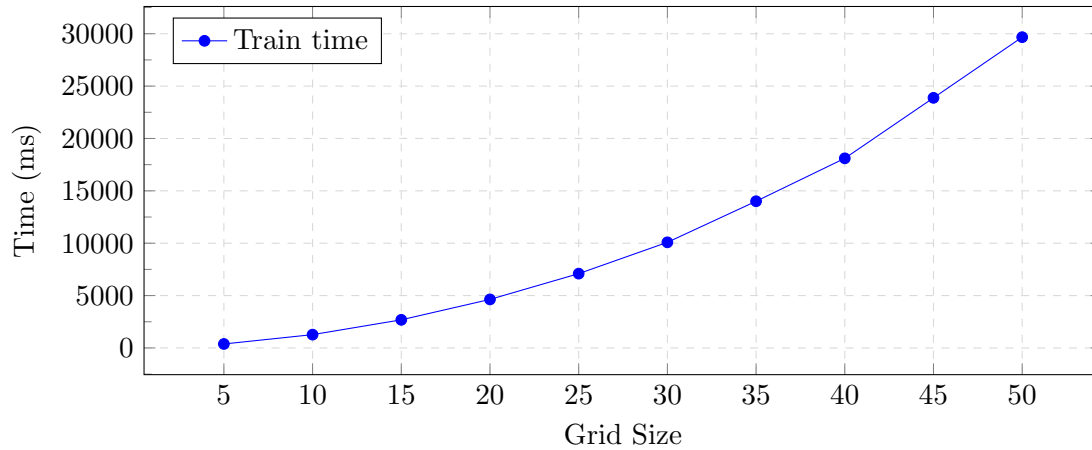


Figure 4.4: Training times for different grid sizes with median color values in the CIE L^*a^*b color space as features. The displayed values are the mean over five runs.

Since the number of trees in the forest influences the training and classification speed, as a larger number of trees has to be generated during training and the input feature vector must be evaluated by all trees during classification, a smaller number of trees is preferable. Table 4.3 lists the results of different numbers of maximally allowed trees, compared to different grid sizes. Median color values in the CIE L^*a^*b color space are used as features. We can see, that at least 200 trees are necessary to achieve the best results up to a grid size of 35 rows and columns. Furthermore, with an increasing number of features, smaller forests achieve good results and the difference between the best and the worst result, as well as the classification difference between the smallest and largest forest, diminishes.

We can see, that the best classification result, with only the color values as features, is

		Trees						Precision Recall
		50	150	200	250	300	350	
Grid size	5	48.69%	53.36%	56.47%	58.56%	59.68%	57.71%	
		50.47%	51.88%	52.06%	53.96%	54.08%	53.72%	
	10	59.03%	60.63%	65.44%	67.26%	67.17%	70.98%	
		54.32%	55.86%	58.70%	60.31%	59.25%	60.25%	
	15	64.35%	66.58%	68.08%	63.00%	63.07%	60.42%	
		57.10%	57.92%	59.14%	57.36%	57.40%	56.89%	
	20	67.07%	72.81%	72.01%	73.16%	68.47%	71.84%	
		62.03%	63.38%	63.79%	64.89%	62.42%	61.77%	
	25	72.43%	75.10%	74.16%	77.10%	80.43%	77.09%	
		67.69%	68.94%	70.21%	68.50%	72.22%	70.50%	
30	78.16%	75.57%	76.28%	77.83%	78.45%	79.46%		
	68.53%	68.81%	71.67%	71.60%	71.89%	73.79%		
35	67.79%	77.79%	77.05%	80.75%	76.71%	78.34%		
	67.04%	71.89%	74.48%	74.75%	72.01%	72.96%		
40	79.97%	80.57%	80.02%	77.65%	77.66%	79.60%		
	75.13%	74.75%	73.79%	72.67%	71.89%	74.81%		
45	77.60%	75.19%	74.62%	77.43%	77.75%	76.56%		
	74.73%	69.21%	70.65%	72.56%	73.06%	70.39%		
50	74.24%	75.81%	77.99%	78.23%	78.16%	77.48%		
	72.11%	70.88%	71.96%	71.96%	72.85%	73.62%		

Table 4.3: Precision and recall values for different numbers of trees in the random forest and varying grid sizes. The median CIE L*a*b color values are used as features.

achieved using RGB median values and a grid size of 35 rows and columns. The next best result is produced by CIE L*a*b mean values, again with 35 rows and columns, followed by CIE L*a*b median values with 25 rows and columns, and finally mean RGB features with 40 rows and columns. However, doubling the number of rows and columns results in a quadratic increase of features, which causes a large increase in time needed to train the forest. This can be seen in Figure 4.4, which displays the time needed to train a forest for different numbers of cells. The plot was created using median color values in the CIE L*a*b color space as features, where each displayed value is the mean over five runs. The timing differences of training the forest with a feature vector created using different combinations of median and mean value, as well as CIE L*a*b and RGB color space, are within three times of the largest standard deviation of the five runs for each grid size. Therefore, the choice of color space, as well as median or mean, appears not to have a significant impact on the training time. The difference in the training time between 25 and 35 rows and columns amounts to 6.911 seconds when using only color features. For comparison, the time difference becomes even larger with the complete feature vector that also contains the *LBP* values and the plate size, where it amounts to 164.40 seconds. The

		Predicted type														
		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
Actual type	a	1	0	0	0	5	0	0	0	0	0	1	0	0	0	0
	b	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0
	c	0	0	2	0	4	0	0	0	0	0	1	0	0	0	0
	d	0	2	0	3	0	0	0	0	0	0	1	0	1	1	0
	e	1	0	0	0	12	0	0	1	0	0	0	1	0	0	0
	f	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0
	g	0	0	1	0	1	0	1	0	0	0	1	0	0	0	0
	h	0	0	0	0	2	0	0	2	0	0	0	1	0	0	0
	i	0	1	0	0	1	0	0	0	0	0	0	1	0	0	0
	j	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0
	k	0	0	0	0	1	0	0	0	0	0	9	0	2	0	0
	l	0	0	0	1	0	0	0	0	0	0	2	9	0	0	0
	m	0	0	0	0	0	0	0	0	0	0	0	0	13	0	0
	n	0	0	0	0	0	0	0	0	0	0	2	1	3	5	0
	o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6

Table 4.4: Confusion matrix created with median color features extracted from the CIE L*a*b color space. The images are divided into 20 rows and columns.

relative speedup for the complete feature vector with 25 rows and columns, compared to 35, amounts to 3.74.

For systems like ours, the training time is usually not very important, as the classifier is trained once and then deployed. However, as previously mentioned, a very large number of plate types exist and new ones are added frequently as vendors introduce new devices. Also, only a low number of samples for a few types will be available initially, because it is difficult to retrieve images of the plates, as the devices are deployed globally and a number of plate types are certainly out of production, due to their age. Therefore, it is desirable to quickly add new types, where the trained classifier can then be deployed to the mobile devices. As the number of types, as well as sample images for each type increases, the time needed to train the classifier will also increase. As such, increased training speed is preferable. Combined with the fact that the *LBP* values contribute the most to the classification result, as will be shown in the following section, and considering that the median CIE L*a*b color space feature vector delivers results that are only 0.48% worse in precision and 1.25% in recall than the RGB median value feature vector, with 10 columns and rows less, we decided to use median values in the CIE L*a*b color space rather than RGB.

Table 4.4 shows the confusion matrix generated with only median color features, extracted from the CIE L*a*b color space. The precision is 70.35% and the recall 61.38%. The images are divided into 20 rows and columns, determined using grid search for the final feature vector. We can see that the types *b* (Figure 4.1b), *f* (Figure 4.1f), *j* (Figure 4.1j),

		Predicted type														
		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
Actual type	a	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	b	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0
	c	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0
	d	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0
	e	0	0	0	0	15	0	0	0	0	0	0	0	0	0	0
	f	0	1	0	0	0	3	0	0	0	0	0	0	0	0	0
	g	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0
	h	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0
	i	1	0	0	0	1	0	0	0	0	0	1	0	0	0	0
	j	0	0	0	0	0	0	0	0	0	7	0	0	1	0	0
	k	0	0	0	0	0	0	0	0	0	0	12	0	0	0	0
	l	0	0	0	0	0	0	0	0	0	0	0	12	0	0	0
	m	0	0	0	0	0	0	0	0	0	0	0	0	13	0	0
	n	0	0	0	0	0	0	0	0	0	1	0	0	0	10	0
	o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6

Table 4.5: Confusion matrix created only with *LBP* features. The histogram size is set to 60, 4 neighbors are checked with a radius of 16.

m (Figure 4.1m), and *o* (Figure 4.1o) are never misclassified. These types have colors that are distinct from the other types, where only the types *b* and *m* are misclassified as other classes. As expected, the errors are among similar looking types.

4.1.3 Local Binary Patterns

The next feature added to the feature vector consists of histograms of *LBP* codes which encode the texture of the plate. Details can be found in Section 3.2.1.2.

Table 4.5 shows the confusion matrix determined using only *LBP* as features. The precision is 89.65% and the recall 90.22%. As we can see, errors occur with four types: *f* (Figure 4.1f), *i* (Figure 4.1i), *j* (Figure 4.1j), and *n* (Figure 4.1n). For two of these incorrectly classified types, the dataset contains a low number of samples. For the type *f*, four images are available, and for the type *i*, three. The final two classes, where classification errors occurred, consist of a larger amount of samples. A sample from type *j* is reported as type *m*, and an image from the class *n* is reported as type *j*. As can be seen from the images, the three classes share some visual resemblance. Although there are a number of errors, the classification performance with only *LBP* features is close to the result of the complete feature vector.

Table 4.6 shows the precision and recall values for different radii and neighbor counts, with a feature vector that consists only of *LBP* values. It can be seen that using either four neighbors with a radius of eight, or eight neighbors with a radius of two deliver the best results. Since the plates contain mostly empty space and the existing structures are

		Radius				Precision Recall
		2	4	8	16	
Neighbors	4	89.60%	90.03%	91.05%	90.81%	
		87.38%	88.66%	92.50%	91.06%	
	8	91.25%	89.09%	90.91%	90.63%	
		91.66%	90.10%	91.66%	91.06%	
	16	88.48%	87.72%	88.55%	87.65%	
		85.09%	82.82%	87.44%	84.47%	

Table 4.6: Precision and recall for varying *LBP* radii and number of neighbors. Only *LBP* histograms with a histogram size of 60 are used as features.

		Predicted type														
		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
Actual type	a	0	4	2	0	0	1	0	0	0	0	0	0	0	0	0
	b	4	1	2	0	0	1	0	0	2	0	0	0	0	0	0
	c	2	2	2	0	0	1	0	0	0	0	0	0	0	0	0
	d	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0
	e	0	0	0	0	8	0	0	3	0	0	2	1	0	1	0
	f	1	0	2	0	0	1	0	0	0	0	0	0	0	0	0
	g	0	0	0	0	0	0	1	0	0	0	0	1	2	0	0
	h	0	0	0	0	1	0	0	0	0	0	2	1	0	1	0
	i	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0
	j	0	0	0	0	1	0	0	0	0	7	0	0	0	0	0
	k	0	0	0	0	2	0	0	2	0	0	4	1	1	1	1
	l	0	0	0	0	1	0	2	0	0	0	2	2	2	3	0
	m	0	0	0	0	1	0	1	0	0	0	2	3	3	1	2
	n	0	0	0	0	1	0	1	1	0	0	1	3	1	3	0
	o	0	0	0	0	0	0	0	0	0	0	2	1	1	0	2

Table 4.7: Confusion matrix created only with the size ratio as feature for classification.

for the most part rectangular, a large radius with a low number of neighbors is enough to represent the texture. Furthermore, a larger radius is less susceptible to changes in the text, as well as other noise.

4.1.4 Size Ratio

The last characteristic that is added to the feature vector is the size ratio of the plate image. Further details can be found in Section 3.2.1.3.

The confusion matrix, calculated with only the size ratio as feature per plate, shown in Table 4.7, results in a precision of 31.69% and recall of 30.87%. As can be seen in the table, only the class *d* (see Figure 4.1d) is never misclassified and no other image is reported as this class. This is caused by the very distinctive size ratio of this nameplate

m	$1/2 \cdot \sqrt{N}$	\sqrt{N}	$2 \cdot \sqrt{N}$
Precision	90.36%	89.96%	88.87%
Recall	92.12%	90.78%	87.95%

Table 4.8: Results for common amounts of active variables, chosen from all features (size N), when selecting the splits.

Features	Color + Size Ratio	Color + LBP	Size Ratio + LBP
Precision	75.63%	90.10%	90.12%
Recall	64.55%	92.12%	88.56%

Table 4.9: Classification results when combining the three features pairwise.

type, when compared to the other classes. As expected, the classification errors are among classes that have approximately the same size ratio. Compared with the other two features, the size ratio is the least important one. Nevertheless, the size ratio is characteristic for a nameplate type and may help to distinguish types with similar content structure and color, but different proportions.

4.1.5 Combination of all Features

In this section, we show the combinations of the three features that are tested individually in the previous sections. First, the features are combined pairwise and then the complete feature vector for the classifier is created by concatenating all three. For each combination, we show the classification results.

The feature vector, calculated for each plate and passed to the random forest for classification, consists of N individual features. During training, the random forest chooses m features from the feature vector at each node and uses the best split to split the data. The number of features m , which remains constant for the entire forest, influences the correlation between the trees and the classification strength of each tree. As shown by Breiman [4], the classification performance of the forest increases with decreasing correlation between the trees and an increasing number of strong classifiers. Further, increasing m increases the strength of the individual classifiers and the correlation between them, while decreasing it has the opposite effect. Table 4.8 lists the precision and recall values for three typical values of active variables when drawing the splits, where the square root of the number of variables is a frequently used value. As can be seen, using $1/2 \cdot \sqrt{N}$ active variables provides the best results. Therefore, we use this value from now on.

Table 4.9 shows the pairwise combinations of the three features. It can be seen that the *LBP* features contribute the most to the result, while the size ratio contributes the least. The precision difference between the final feature vector and only *LBP* features is 0.26%, while the recall remains the same. Since the dataset contains only a small amount of images, which are slightly blurry at the most, the color information contributes very

Kernel Size	9×9	19×19	29×29	39×39
LBP	124 (99.2%)	104 (83.2%)	60 (48.0%)	35 (28.0%)
	1 (0.8%)	21 (16.8%)	65 (52.0%)	90 (72.0%)
Full Feature Vector	125 (100.0%)	114 (91.2%)	73 (58.4%)	45 (36.0%)
	0 (0.0%)	11 (8.8%)	52 (41.6%)	80 (64.0%)

Table 4.10: Classification results for median blurred images with the complete feature vector and *LBP* features alone. The green values list the number of correctly classified images, while the red values wrong ones.

little to the classification result. To simulate the effect of blurred plates with distorted content, we apply median blurring to the images. Table 4.10 lists the classification results for the entire dataset with the complete feature vector, as well as *LBP* features only, for different median blur kernel sizes. For this test, we do not use leave-one-out cross validation. Instead, all images from the dataset are used to initially train the classifier. Then, all images are blurred before the feature vector used for classification is extracted. As such, every image is classified correctly with both tested feature vectors if no blurring is applied. The images are first resized to a width of 800 pixels, as described previously, and then the median blurring is applied to the scaled images. This is necessary to obtain comparable results, as the sizes of the images in the dataset vary greatly. The results show that the complete feature vector always outperforms the one that consists only of *LBP* features. With a blur kernel size of 9×9 , a single image is misclassified with *LBP* features, while with the complete feature vector, all images are recognized correctly. With increasing kernel size, the number of correctly classified images decreases for both feature vectors. However, for the three largest blur kernel sizes in the table, the difference of correctly classified images remains about the same. It is largest with a kernel size of 29×29 , where the classification result obtained with the complete feature vector includes 13 additional correctly classified images. The other two cases, with kernel sizes of 19×19 and 39×39 , show a difference of 10 images. Figure 4.5 shows an unmodified and a blurred image for each kernel size, where all blurred images are classified correctly with the complete feature vector, but are wrong with the feature vector that contains only *LBP* values. It should be noted, that with images blurred to such an extent, processing will fail in the later stages, as no text is readable anymore. Nevertheless, we included these results to show the robustness of the classifier. Furthermore, it is possible that such images are present during regular usage, e.g., if the camera is out of focus, or the operator moves while capturing the image, which results in motion blur. In these cases, the *OCR* will fail for all regions, but the user will be prompted to capture additional images, which can then be used to generate usable output.

The final classification results with the complete feature vector and the parameters determined using grid search, are shown in Table 4.11. The precision and recall values are listed in the first column of Table 4.8. Errors occur with the type n , where some of

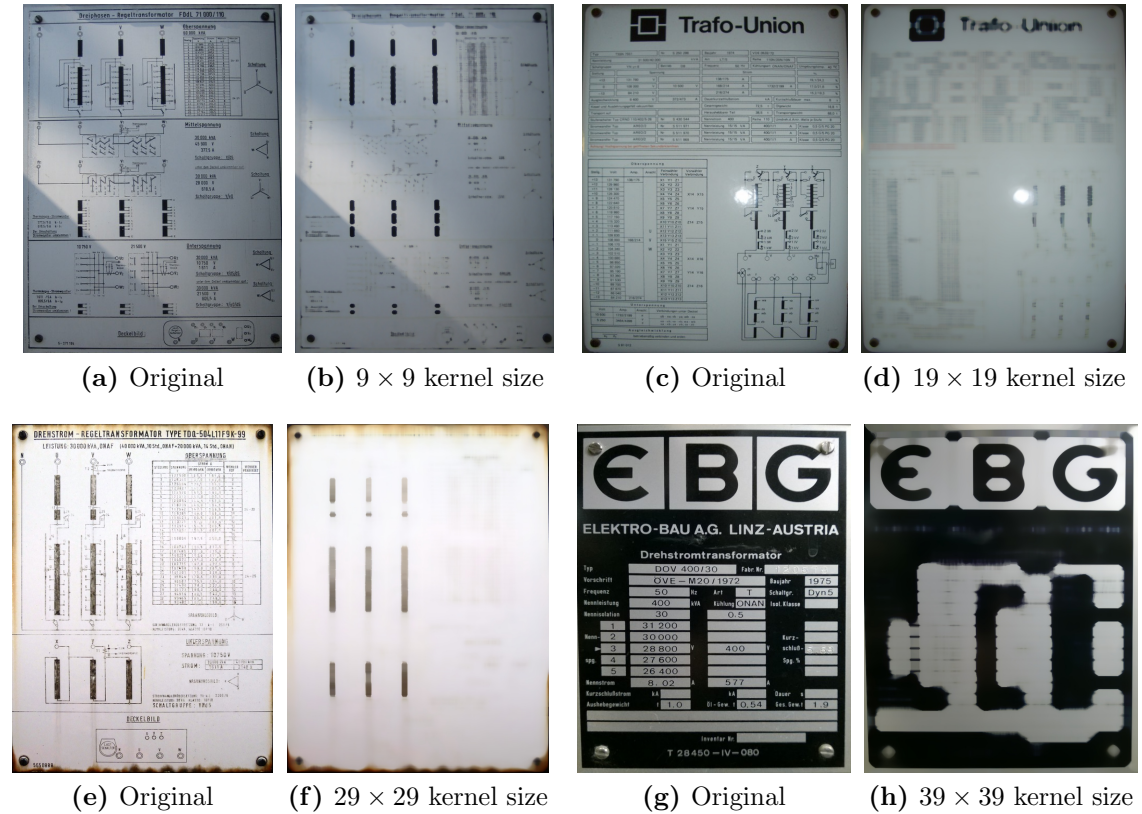


Figure 4.5: Original and blurred images for different median blur kernel sizes. All blurred images are classified correctly with the complete feature vector but not with *LBP* features alone.

the samples contain dark regions, that cover large portions of the plate, as well as bright reflections, which also cover a large area. As these conditions are present in most images, they cause large variations in the samples of this class. These issues are caused by the surface of the plate, which is highly reflective. Therefore, the quality of the captured image changes with the lighting conditions, as well as the region on the plate on which the camera focuses. Figure 4.6 shows four images from the dataset of this plate class, where the first two show reflections, and the last two dark regions at the borders. We should note that these images are less problematic for text extraction, as our gradient based approach is robust against brightness variations. Nevertheless, should the reflections or dark regions cause issues, more images will be requested where the operator can move to a different position, such that the text becomes readable. Also, if the plate is mounted on a device for a number of years, the surface gets less reflective and capturing images becomes less problematic. The other plate type that causes errors, to the extent that it is never classified correctly during leave-one-out cross validation, is class *i*. Figure 4.7 shows the three available samples. It can be seen, that there is very little content on these plates,

		Predicted type														
		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
Actual type	a	7	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	b	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0
	c	0	0	7	0	0	0	0	0	0	0	0	0	0	0	0
	d	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0
	e	0	0	0	0	15	0	0	0	0	0	0	0	0	0	0
	f	0	0	0	0	0	4	0	0	0	0	0	0	0	0	0
	g	0	0	0	0	0	0	4	0	0	0	0	0	0	0	0
	h	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0
	i	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0
	j	0	0	0	0	0	0	0	0	0	8	0	0	0	0	0
	k	0	0	0	0	0	0	0	0	0	0	12	0	0	0	0
	l	0	0	0	0	0	0	0	0	0	0	0	12	0	0	0
	m	0	0	0	0	0	0	0	0	0	0	0	0	13	0	0
	n	0	0	0	0	0	0	0	0	0	0	0	0	0	2	9
	o	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6

Table 4.11: Confusion matrix calculated using the full feature vector with the final parameters, determined using grid search.

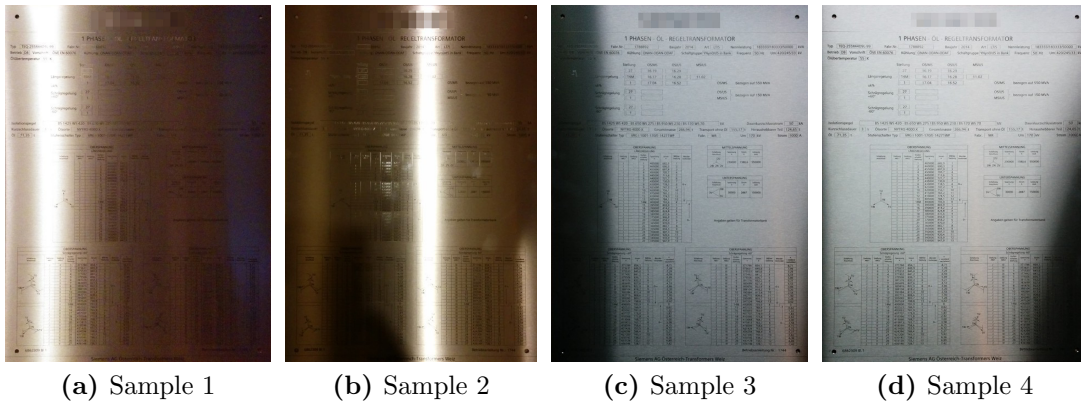


Figure 4.6: Nameplate class n where the sample images contain dark regions and reflections. They are caused by the reflective surface of the plate and depend on the lighting and on which part of the plate the camera focuses.

when compared to the other classes. Further, the plates in Figure 4.7a and Figure 4.7c are scratched and all three images have a slightly different color. We can see, that both classes, that cause classification errors, contain large variations in the training samples. Further, the dataset contains only three images for the class i . With a larger number of samples, that cover the variations, the classification results should improve.

A random forest can estimate the importance of the individual features in the feature

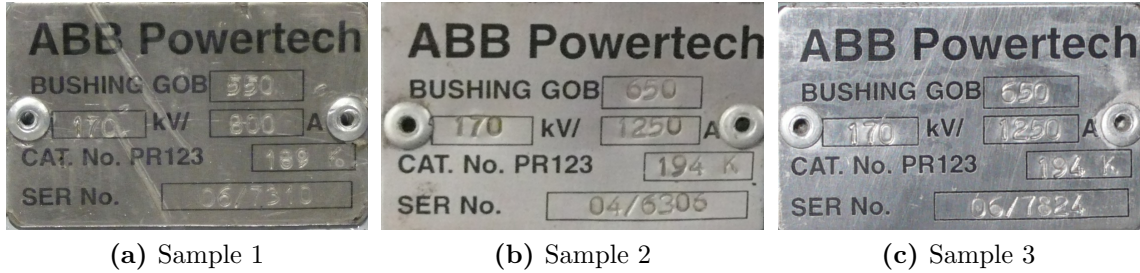


Figure 4.7: Nameplate class i that contains only three samples and is never correctly classified during leave-one-out cross validation.

vector after training. To calculate the variable importance, Breiman proposes four different measures, which are defined in the manual for the reference random forest implementation [5]. We use Method 2, as it is implemented in OpenCV. The values for the tested feature, of the samples that were not chosen by the bagging procedure, are randomly manipulated and classified again by the forest. For each tree, the number of votes for the correct class of the original and the modified samples are stored. The difference (i.e., the decrease in accuracy) is averaged over all trees and indicates the importance of the tested feature. However, as stated in [39], the approaches proposed by Breiman are problematic because they can produce misleading results. The authors perform a number of simulation studies, that show that the variable importance is influenced by the number of trees in the forest, which is selectable by the user, but not the sample size. Nevertheless, we chose to show the results of the estimated variable importance for informational purposes. Even if the values are not completely reliable, they still give an indication which of the three main features extracted from the plate, are ranked as most important by the forest. Examining the type of the first 100 highest ranked features shows that they consist of 89 *LBP* values and 11 color values. The size ratio is located at the end with a very low score, which matches our observations when combining the features pairwise. Furthermore, including the *LBP* features in the feature vector results in smaller forests, because the forest accuracy is reached before the maximum number of trees are generated.

The time (average over five runs) needed to train a forest with all samples amounts to 37.21 seconds. During classification, first the feature vector must be extracted. The time needed depends on the size ratio of the plate, as the image is resized to a fixed width, but the height is scaled to preserve the aspect ratio. Therefore, images that are higher than wide require more time compared to the reversed size proportion. Extracting the features from the plate type d (Figure 4.1d), requires the least amount of time with 56 milliseconds, whereas type j (Figure 4.1j) is the slowest, requiring 425 milliseconds (all values are the fastest times of five runs). The average time needed to extract features over the entire dataset is 247.59 milliseconds and the average classification time of the random forest is 0.172 milliseconds.

4.2 Evaluation of the OCR

The utilized *OCR* software, called Tesseract², is generally regarded as the best open source *OCR*. A comparison between the commercial ABBYY FineReader³ application and Tesseract is available in [23]. The authors use printed Polish historical documents as test input, both the original scan and a denoised version. They conclude that there is no clear winner, but Tesseract outperforms FineReader with good quality images, such as the denoised versions. They further state, that Tesseract requires more samples for training. Another widely used open source *OCR* is GOCR⁴. Dhiman and Singh [15] compare it to Tesseract, where they vary the brightness, image type, resolution, and font of the test images. They state that Tesseract outperforms GOCR in most cases. A short overview of the Tesseract algorithm [37] is presented in Section 3.4. However, the use case for which general *OCR* software, as the ones mentioned previously, are optimized, consists of processing sheets of scanned documents. The number of fonts used on documents is limited, and scanning produces images with even illumination and very little noise. This is not the case on our input images. Therefore, a preprocessing step outputs only filtered binary images, which are then passed to the *OCR*. The filtering approach is described in detail in Section 3.3.3.

Since Tesseract is off-the-shelf software, with little parameters to tune (only the use of a dictionary or layout analysis can be controlled) we evaluate the effect of training the font used on the nameplate. The Tesseract project provides a number of pre-trained language files. The default language file *eng* is created using 17 fonts, which are commonly used in word processing, including Arial, Comic Sans, Times New Romans, and Courier. However, these fonts are usually not utilized on nameplates. Moreover, the shapes of the characters of the fonts that are used, differ enough to be often classified incorrectly. For training, bitmaps of text rendered with the new font, as well as a file that contains the bounding box and content for each individual character are needed.

For the following evaluation, a plate of type k (see Figure 4.1k) is used. The language files provided by the Tesseract project contain a dictionary, which we disable. Further, we do not include a dictionary in the newly created language files. The use of a dictionary does not make sense in our case, as most strings present on the plates, such as serial numbers or abbreviations, are not regular words. Therefore, enabling the dictionary results in additional errors, as Tesseract tries to match the recognized words to known ones. As can be seen in the first column of Table 4.12, using the provided *eng* language file results in 57.14% of all regions and 86.25% of all characters being correctly recognized. The language file *plate* was trained using the characters of two images from the plate itself, resulting in 71.42% of all regions and 90.0% of all characters being correct. Finally, Tesseract allows to use multiple language files, which is shown in the final column. While the number of correct regions remains unchanged, the number of correct characters increases by 1.87%.

²<https://code.google.com/p/tesseract-ocr> (last visited January 22, 2015)

³<http://finereader.abbyy.com/> (last visited January 22, 2015)

⁴<http://jocr.sourceforge.net/> (last visited January 22, 2015)

Used Language	eng	plate	plate+eng
Correct Fields	16/28, (57.14%)	20/28, (71.42%)	20/28, (71.42%)
Correct Characters	138/160, (86.25%)	144/160, (90.0%)	147/160, (91.87%)

Table 4.12: *OCR* recognition results using the language files provided by the Tesseract project, the font extracted from the plate itself and the combination of both.

Used Language	ISOCP	ISOCP+eng	ISOCP+plate+eng
Correct Fields	20/28, (71.42%)	20/28, (71.42%)	19/28, (67.85%)
Correct Characters	145/160, (90.62%)	147/160, (91.87%)	146/160, (91.25%)

Table 4.13: *OCR* recognition results using a language file trained only using the ISOCP font used on the plate and combination with the other language files.

This is probably the case because Tesseract incorporates information about the occurrence of other characters in the vicinity. Therefore, the combination *plate+eng* provides the best recognition results.

However, generating character samples for training from plate images is cumbersome and there might not be enough occurrences for each character (at least five samples for rarely occurring characters are recommended). If the font is available, character samples can be generated using arbitrary text, which speeds up the procedure significantly and the characters can be chosen freely. The result of training a language file in this manner is shown in Table 4.13. The number of correct regions remains the same, while the number of correct characters increases by 0.62%, compared to the the language file trained with the characters directly from the plate. Combining the provided *eng* file and the one trained using the font yields the best results, with 71.42% of all regions and 91.87% of all characters being correct. Yet, the combination of all three decreases the number of correct regions and characters again.

As we can see, the combinations *plate+eng* and *ISOCP+eng* result in the same amount of correct regions and characters. This shows, that using samples, rendered using the font file, can produce the same recognition results as character samples extracted directly from the plate images. However, the number of character samples that are needed to produce these results differ for the two approaches. The *plate* file was trained using 995 individual character samples from two images of the plate. The file *ISOCP* was trained using 6320 character samples, where a dummy text (complemented with digits and decimal separators) was used to generate the training data. Although, more than six times the number of samples are used for the second approach, the time needed to compile the training data is only a fraction compared to the first approach, as no manual character annotation is needed. The results also show, that including the existing *eng* language file is beneficial, as it increases the output accuracy in both cases.

The regions of the plate, that contain at least one misclassified character, are listed in

Correct Text	Detected Text	OCR score
80,83	80 83	85%
Dyn5	Dn5	86%
SIEMENS	S1EMENS	72%
LI125 AC50/AC3	L1125 AE50/AC3	74%
1,435	1435	75%
24/1,1	21+/1.1	71%
10.04.2014	10 04 2014	85%
6,955	6 955	82%

Table 4.14: List of incorrectly recognized text regions created using a combination of the *ISOCP* and *eng* language files.

Table 4.14. It can be seen, that most errors result from missing punctuation characters. However, these regions have a high *OCR* confidence score. The reason is, that the pre-processing parameters were tuned for aged plates that include a lot of clutter, caused by dirt and stains or embossed text, because such plates will be processed more frequently than new ones. Hence, the punctuation characters are removed completely, if their size is below the filtering threshold. It is also possible that commas merge with the frames around the text, where the overlapping parts are then removed, along with the borders. The remaining part is either removed because it is too small or classified as a dot instead. The remaining incorrect regions do indeed have a low confidence score, where the user would have been prompted to acquire a new image of the problematic plate area.

4.3 Application

A goal of this thesis is to produce a prototype of an Android⁵ application that implements the previously described and evaluated algorithm. Figure 4.8 shows screenshots of all activities of the finished prototype. The first view, shown in Figure 4.8a, contains an overview of the previous scan results. Upon selecting an entry, another view (see Figure 4.8b) shows the detailed results. It displays all defined fields, including the detected text and *OCR* confidence score. Further, a visualization that includes the extracted and warped plate is available (see Figure 4.8c). The low confidence regions are framed with a blue color, and the high confidence regions with a green color. To capture images, the application displays a live camera image, where the user can start the processing with a tap on the screen. When an image is available, the plate segmentation, classification, and text extraction steps are performed. Should more images be required, the user is informed of the problematic plate region and a new image can be retrieved. Finally, if the procedure is finished, either because all *OCR* regions have a sufficiently high confidence or the maximum number of retries is exceeded, the results are visualized and stored in a file

⁵<http://developer.android.com/> (last visited January 22, 2015)

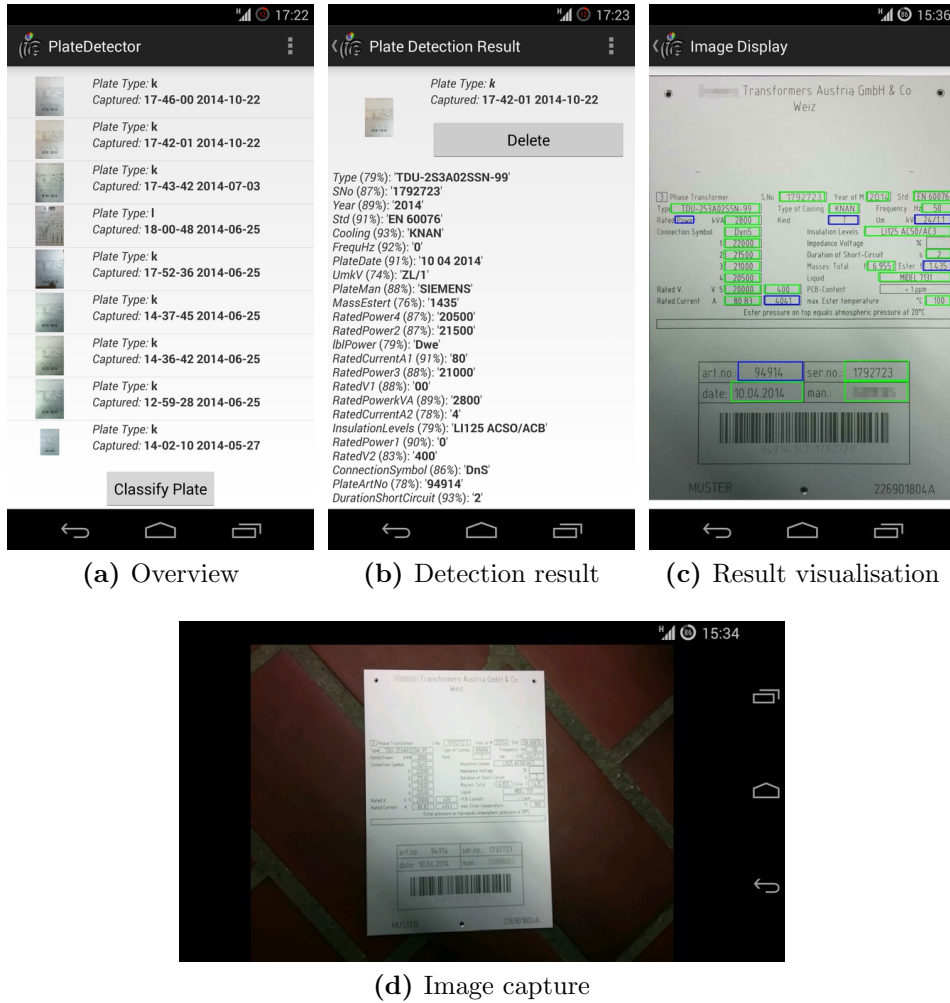


Figure 4.8: Screenshots of the Android application. (a) List of the previous plate detection results. (b) Detailed plate detection results. (c) Visualization of the plate detection results. (d) Image capturing view.

for further processing. The Android application uses the same C++ code for the image processing as the PC version, only the user interface is written in Java.

All tests of the Android application are executed on a LG Nexus 5, which contains a 2.26 GHz quad-core Snapdragon 800 Krait 400 SOC with 2 GB of RAM. As in the previous tests, the OpenCV 2.4.8 framework, but no GPGPU capabilities are employed. The trained random forest and Tesseract font are the same as in the evaluation from the previous section. The maximum number of retries is set to four, which prevents a large number of new image requests, caused by an improperly trained *OCR*. The visualization of the *OCR* result, displayed for each tested plate type, contains the adjusted positions for each text region. A green rectangle indicates an *OCR* score above or equal 80%, whereas blue rectangles signal a lower score. A red rectangle denotes an empty region where no

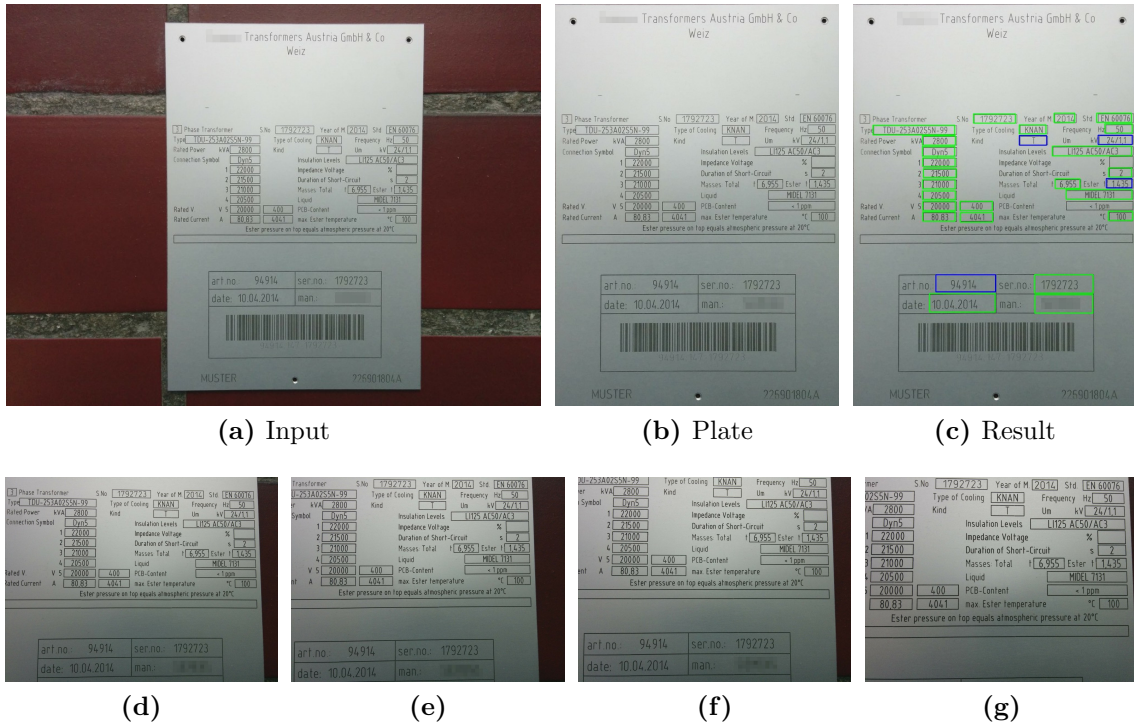


Figure 4.9: Plate detection steps for a plate of type k on an Android device. (a) Initial image of the plate. (b) Segmented and warped plate. (c) Visualization of the text regions and the OCR confidence. (d) First requested detail image of the right side. (e) Second requested detail image of the right side. (f) Third requested detail image of the right side. (g) Fourth requested detail image of the right side.

content was found.

Figure 4.9 displays the input image and detection results for a plate of the type k , that is made of aluminum. While the image was captured, the plate was illuminated using artificial and natural light. The plate is located and segmented properly, no manual intervention was needed. Further, all adjusted text regions are correctly placed at the actual text positions that are present on the plate. The maximum number of four additional images are requested from the operator, where each of the four times the requested direction is the right area of the plate. As can be seen in the visualization, shown in Figure 4.9c, the final result contains four regions with a low OCR score, where three of them are located on the right side of the plate. Table 4.15 lists the execution times of the individual steps for this plate in the first column. As expected, the plate segmentation using GrabCut, preprocessing and OCR , as well as keypoint matching require the highest amount of time. The time needed to filter and execute the OCR for the fourth retry is by far the longest, requiring 6.8 seconds. The reason is that the last image (see Figure 4.9g) was shot closer to the plate, which results in a magnification of the text regions. In addition to the larger

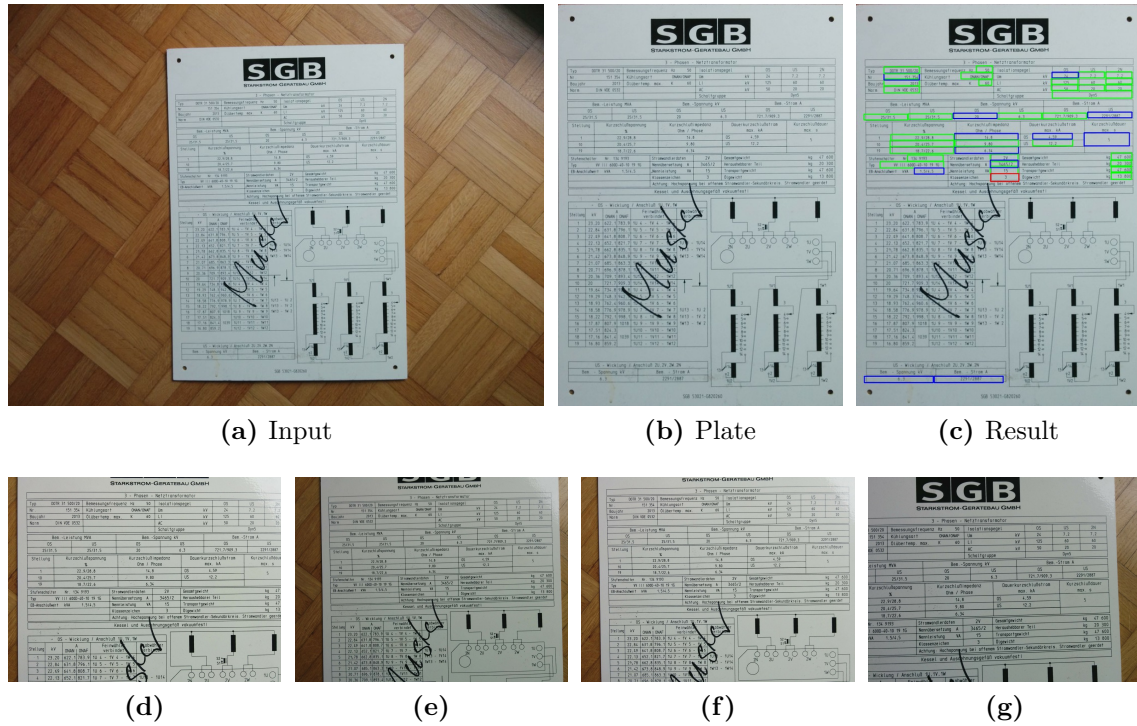


Figure 4.10: Plate detection steps for a plate of type l on an Android device. (a) Initial image of the plate. (b) Segmented and warped plate. (c) Visualization of the text regions and the *OCR* confidence. (d) First requested detail image of the left side. (e) Second requested detail image of the left side. (f) Third requested detail image of the left side. (g) Fourth requested detail image of the right side.

bitmaps that have to be processed, the image contains all regions, except of the lower four. In the final output, 20 out of 28 (71.421%) text regions are entirely correct. Overall, 149 out of the 160 (93.12%) characters present in the text regions are correct. There are six occurrences of missing or wrong (swapped dot and comma) decimal separators, as well as three cases where the digit '5' is recognized as letter 'S'. The remaining errors are the same as described in Section 4.2, where the same plate type was used.

The second plate has the type l and is made of plastic. The images for this plate were taken under daylight conditions. Figure 4.10 displays the input image, the segmented plate, the the text regions with the recognition confidence, as well as the four additionally requested images. As for the previous plate, the plate localization and segmentation did not need any manual intervention. Again, the maximum number of allowed retries are exhausted. The requested directions for the additional images are three times the left, and once the right area of the plate. This is expected, as the *OCR* is not trained for the font used on the plate and most text regions are in the upper half. The execution times of the individual steps are listed in the second column of Table 4.15 and are comparable

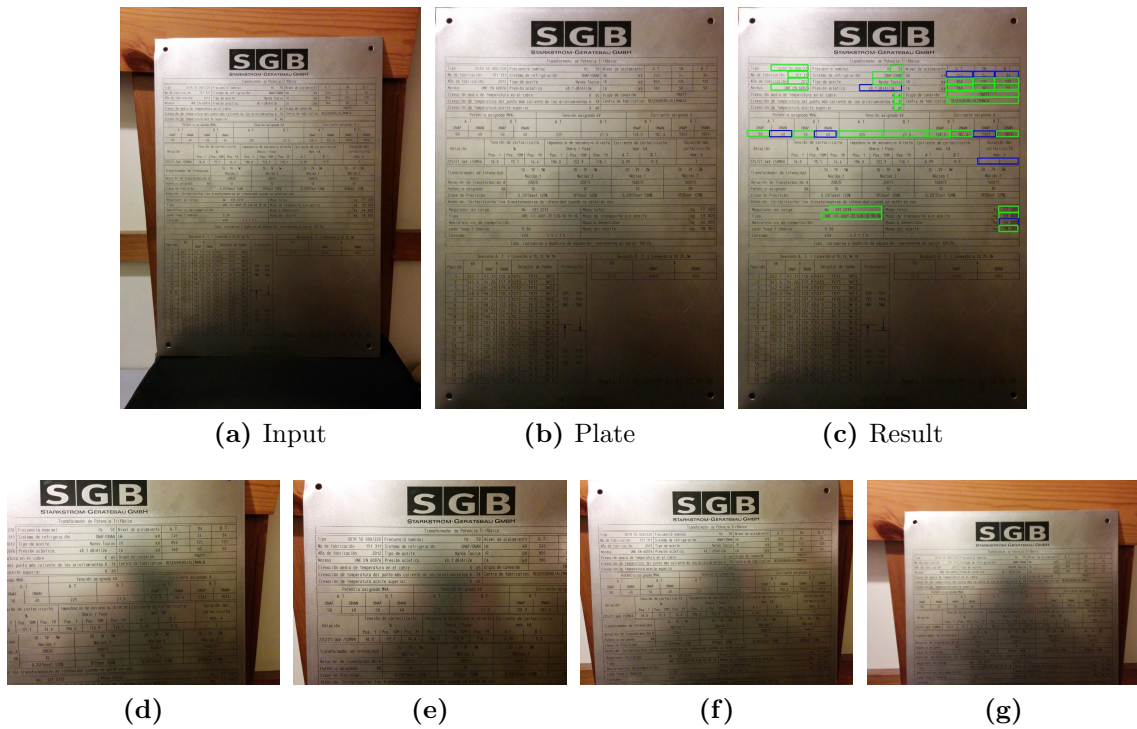


Figure 4.11: Plate detection steps for a plate of type m on an Android device. (a) Initial image of the plate. (b) Segmented and warped plate. (c) Visualization of the text regions and the *OCR* confidence. (d) First requested detail image of the right side. (e) Second requested detail image of the left side. (f) Third requested detail image of the left side. (g) Fourth requested detail image of the left side.

to those of the previous plate. The plate segmentation, preprocessing and *OCR*, as well as image matching are again the slowest operations. The final output consists of 17 out of 44 (38.63%) entirely correct regions, as well as 185 out of 244 (75.81%) correct characters. There are 5 instances where the characters '1', '/', or 'I' are reported as 'l'. This is a typical error that occurs when the *OCR* is not trained for a given font. Other frequent errors include mix ups of the digit '5' and the letter 'S'. Furthermore, there are 19 cases of missing or wrong (swapped dot and comma) decimal separators, caused by the preprocessing.

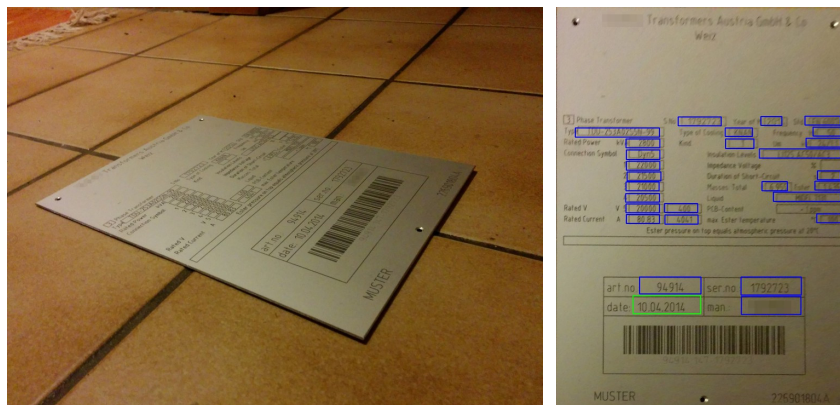
The last plate is of type m and is made of sheet metal. As such, the plate has a reflective surface. Contrary to the first two plates, it is illuminated solely by artificial light. The input image, extracted and warped plate, as well as the additionally requested images and the result visualization are shown in Figure 4.11. It should be noted, that this plate was not processed on the Android device, because the camera was unable to automatically focus on the reflective surface of the plate. As this focus method is the only one supported by the Android plate detector prototype, it is not able to capture an image that contains readable text. Therefore, a camera application that allows for more focus

Operation	Type <i>k</i>	Type <i>l</i>
Plate detection	42 ms	61 ms
Plate segmentation	893 ms	793 ms
Feature vector construction	300 ms	298 ms
Classification	0.356 ms	0.391 ms
Text region mapping	541 ms	570 ms
Preprocessing and OCR	1675 ms	2269 ms
Retry 1 matching	3112 ms	3621 ms
Retry 1 preprocessing and OCR	2636 ms	1518 ms
Retry 2 matching	1853 ms	1776 ms
Retry 2 preprocessing and OCR	2559 ms	1161 ms
Retry 3 matching	1798 ms	1866 ms
Retry 3 preprocessing and OCR	2807 ms	2569 ms
Retry 4 matching	1793 ms	2062 ms
Retry 4 preprocessing and OCR	6841 ms	2545 ms
Total	26850.356 ms	21109.391 ms

Table 4.15: Execution times of the different stages of the Android application.

control was used to acquire the images, which were then processed by the PC version of the application. For this reason, no timing values are listed as they cannot be meaningfully compared to the other measurements. As for the previous plate, the *OCR* is not trained for the font and all four allowed retries are exhausted. The first additionally requested image is from the right area of the plate and the following three from the left area. In the final output, 24 out of 39 (61.53%) text regions are completely correct, with 195 out of 223 (87.44%) correct characters. As for the first two plates, a number of errors are caused by the *OCR* preprocessing, resulting in missing or wrong hyphens and decimal separators. Moreover, a number of character mix ups, such as '0' and 'D', as well as '0' and 'O' are also present. These are again a consequence of the untrained *OCR* for this particular font.

In conclusion, we can see that the plate localization, extraction and warping, text region adjustment, as well as processing of additional images work properly for the images at hand. The biggest issues are caused by the *OCR* and preprocessing, where the inaccuracies are the result of two issues: The first one are interchanged characters, which are the result of an *OCR* that is not trained for the given font. The second issue is caused by the region preprocessing, which is tuned for aged plates and therefore repeatedly removes decimal separators and hyphens. Occasionally, if they are not filtered, the *OCR* confuses dots and commas. As described in Section 4.2, this is caused by the preprocessing, that removes part of the commas if they overlap with the frame around the text. The last two tested plates, with the types *l* and *m*, are labeled with a font where the comma character is only marginally larger than the dot character. When looking at the images, it occasionally requires effort to distinguish them. These problems are also the reason, that in all three cases, the maximum allowed number of additional images are requested. Finally, another



(a) Input

(b) Result



(c) Input

(d) Result

Figure 4.12: Input images acquired at different angles relative to the plate. (a) Input image. (b) Up-right extracted plate. The size ratio is correct, but the upper right parts of the plate are blurry. (c) Input image. (d) Up-right extracted plate. The estimated size of the extracted plate is wrong, which results in a squeezed image.

matter is the speed of the application. The processing times are acceptable for actual usage, but should be improved further.

4.3.1 Edge Cases

Finally, we show a number of sample images that were not captured under optimal conditions, as well their results. The examples include images where the camera is placed at an acute angle relative to the plate, as well as plates captured under bad illumination conditions.

Figure 4.12 shows two images, where the capturing device is located at different angles relative to the plate. Our approach to segment the plate fails for both images. Therefore, manual intervention which consists of annotating foreground and background areas is

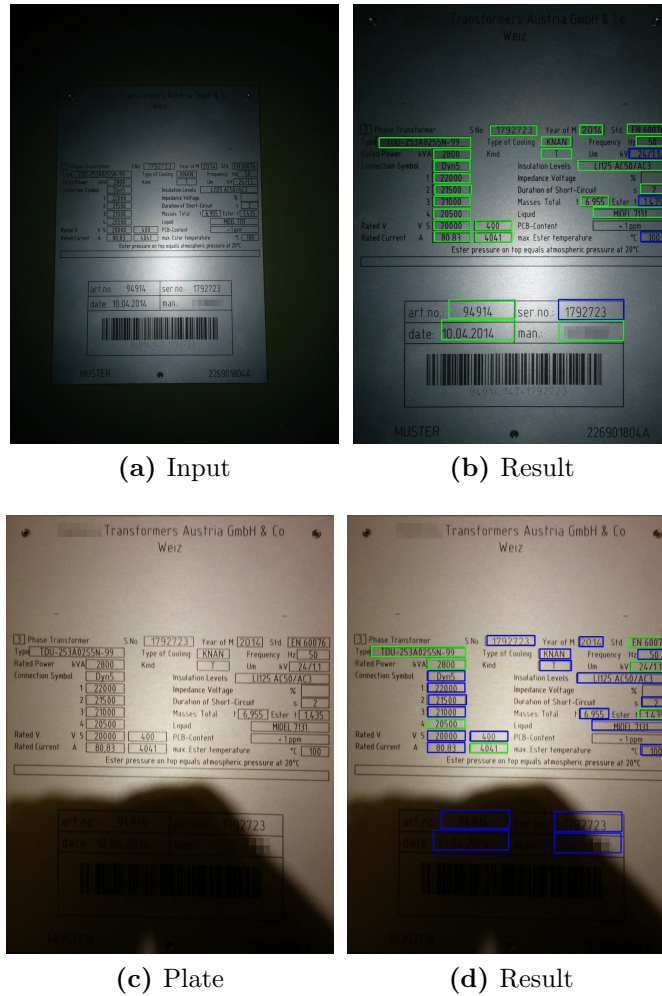


Figure 4.13: Samples with problematic lighting conditions. (a) Input image acquired in a dark room. The only light source is the flash. (b) Text regions and recognition confidence. The text extraction is unaffected by the illumination. (c) A plate that contains a strong shadow at the bottom. (d) Classification succeeds, the *OCR* output for the regions under the shadow contains a number of incorrect characters.

necessary. The first plate, displayed in Figure 4.12b, is extracted with the correct size ratio. However, the right upper region is blurry as the camera could not focus the entire plate at this capturing angle. The blurry text causes problems during preprocessing and *OCR*. For the second plate, the size is incorrectly estimated, causing a squeezed image with an incorrect size ratio. While the classifier determines the correct type and the text regions are adjusted to match, which can be seen in Figure 4.12d, the regions are offset by a few pixels to the left and bottom. Combined with the small size, this results in cropped text. As connected components that touch the border are removed during preprocessing,

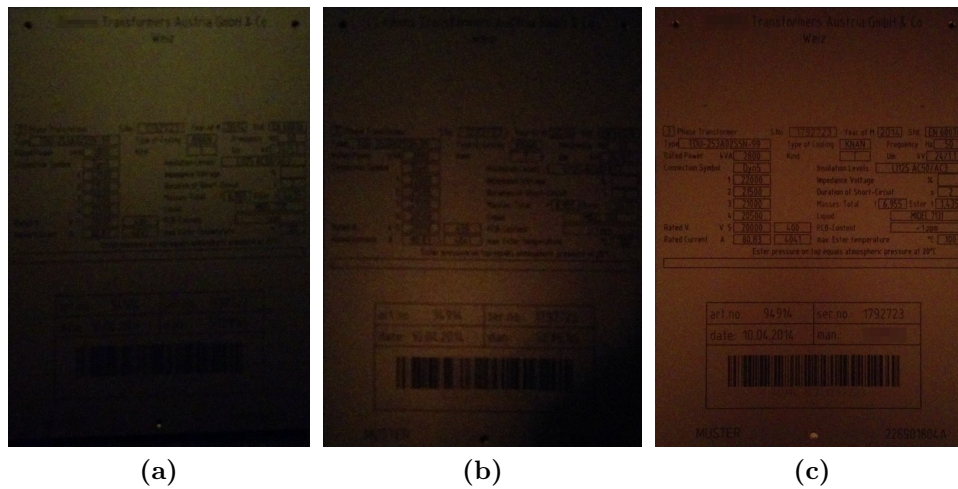


Figure 4.14: Classification results for badly illuminated plates. (a) The plate color is mostly unchanged. The image is classified correctly. (b) The image is noisy and is classified incorrectly as type *m*. (c) The image has wrong colors and contains a lot of noise. It is classified incorrectly as type *m*.

the *OCR* output for this image is not usable.

Problems are also caused by the lighting conditions under which the plate images are taken. Figure 4.13 shows two images of plates with lighting conditions that are likely to occur in real word usage. The first image, displayed in Figure 4.13a, was captured in a dark room, with the flash of the mobile device as the only light source. Since the flash is pointed at the center of the plate, the outer regions of the image are darker. This hides possibly present structures in the background, while highlighting the plate itself, which aids plate detection and segmentation. Furthermore, it should be noted that the images of this plate class, used to train the classifier, do not contain any shadows or modified colors. Also, the approaches used in the text extraction and *OCR* preprocessing steps are robust against color shifts and gradients. The image is classified correctly and we can see in Figure 4.13b, that the *OCR* reports four regions with a low confidence score. The second example is shown in Figure 4.13c. The image contains a shadow that covers a large region at the bottom of the plate. The plate is already extracted, because the shadow interferes with the plate segmentation, again requiring manual intervention. As in the previous example, the plate is classified correctly. However, the *OCR* output of the text regions, covered by the shadow, contain a number of incorrect characters. Figure 4.13d shows the text regions and the recognition confidence. As can be seen, the result contains 21 regions with a low *OCR* score (drawn in blue color). This is the case because the image is blurry, which is caused by the large shadow, that interferes with the autofocus of the camera.

Figure 4.14 shows three images of nameplates, that were captured under low light

conditions with no flash. This results in blurry images, that contain wrong colors and a lot of noise. The first plate, shown in Figure 4.14a, is noisy, the lower part is very dark, and the text is not readable. However the plate color is mostly unaffected and this plate is correctly classified. The next plate, shown in Figure 4.14b, is incorrectly classified as type *m*. In this image, the right and top regions are very dark and it contains about the same amount of noise as the first image. The text in the brighter regions is also not readable. The difference to the previous plate is the wrong color, which turned orange. The last plate, displayed in Figure 4.14c, is also classified incorrectly as type *m*. In contrast to the previous two images, no parts of the plate are dark and the entire text can be recognized by a human. However, the plate contains a large amount of noise, which affects the *LBP* feature extraction. In addition, when compared to the original color, the orange plate color is entirely incorrect. These images show that incorrect colors, that affect a large part of the plate, in combination with noisy and blurry images, result in misclassifications. Moreover, the text extraction steps, that follow the classification, fail for all three images. The first step that fails is the Maximally Stable Extremal Regions (MSER) extraction and grouping. For the shown images, there are none or a very low number of detected regions, which is caused by the low contrast. Even if the text region locations would be correct, the preprocessing and *OCR* would generate unusable output, as the text is only faintly visible and not readable by a human.

Conclusion and Future Work

5.1 Summary

In this Master's Thesis, a method to detect, classify, and extract text from a known set of nameplates was presented. The intended use case involved an operator that captures plates attached to electric power supply components, such as power transformers or circuit breakers, using a mobile device where the extracted content is transmitted to maintenance equipment. Our presented approach contains the following steps: The nameplate is localized in the input image, segmented, and classified using machine learning. The classification stage uses color information, histograms of Local Binary Patterns (LBP), and the plate size ratio as features to determine the type using a random forest. For each type, the layout of the plate and so the positions of the labeled text regions were given. The list of defined fields is matched against the detected fields on the plate to account for possible inaccuracies introduced by the plate extraction. The adjusted regions are preprocessed to remove noise, and passed to the Optical Character Recognition (OCR). The preprocessing of the *OCR* extracts connected components, filters them by size, and groups them into lines of text. The employed OCR is the well known open source project Tesseract. If the *OCR* reported a low confidence score for at least one text region, the user is signaled to acquire a new image of that part of the plate, which is then matched to the existing image and the text processing steps are executed again. The process completes, if all regions have a sufficiently high score or a fixed number of retries are exhausted. The output consists of the detected plate type, the list of regions including the extracted text and the *OCR* reliability.

To show the significance of the designed feature vector, we used a dataset of 125 images of 15 classes which represented a subset of all existing types of nameplates. It contains aged, as well as new types. The results showed, that the local binary pattern histograms contribute most to the classification result, followed by the color features, and finally the plate size ratio. Evaluation of the combined feature vector showed the robustness of the classification under different lighting conditions, such as the presence of shadows

and specular reflections, as well as blurriness and incorrect colors, caused by insufficient illumination. Furthermore, we demonstrated that training the *OCR* using the specific font of the test plate could reduce the number of recognition errors. The remaining errors were caused by the text region preprocessing, which removes hyphens, commas, and dots, in case they are small, compared to the text region, or merged with the borders around the text. This is caused by the filtering parameters, which are tuned for aged plates. The very restrictive text region preprocessing also removes letters that were partly chipped off or faded. Further, the plate localization may be inaccurate or fail if there is no clear boundary between the plate and the background. Such cases occur if the borders of the plate are overpainted, or dirt and moss accumulate.

Testing on an Android device with three available plates showed good results, if the *OCR* was trained for the fonts on the nameplates. Highly reflective plates are difficult to capture and require manual focus control, which is not implemented in the prototype. However, it is unlikely that many such plates are mounted to devices, because if a plate is exposed to environmental conditions for a time, the surface will tarnish quickly. Overall the developed prototype is able to extract the requested information from plates and execution time is suitable for actual usage.

5.2 Further Work

We have shown that the proposed algorithm is usable in real world conditions. However, we have some ideas how to further improve it. The segmentation could be enhanced by incorporating information acquired from the plate images that are used to train the random forest. Examples are the positions of text or logos, which can be used to estimate the plate boundary more precisely. Furthermore, the evaluation of the classifier shows that two classes are sometimes misclassified. The first class consists only of three visually distinctive images, which makes it very hard to classify them correctly. The second one also contains a lot of variance in the form of reflections. To improve the accuracy, no changes to the classification approach are necessary, but a larger number of samples are needed for training. Finally, some improvements are possible for the last stages, preprocessing and *OCR*. It is highly unlikely to always achieve perfect output, because some text (especially embossed text or content of very old plates) is difficult to recognize even by a human. A possibility would be to individually preprocess different types of marking. For example, embossed text is much fainter, but the metal around the letters contains a lot of noise that is caused by the imprinting. Further, knowledge about the format of the field contents could be used to automatically verify and correct the detected content, such as decimal separators of digits or hyphens in serial numbers. While the employed *OCR* is generally regarded as the best open source solution, it is not tuned for our intended use case. Therefore, another possibility would be to develop a new *OCR*, that is robust against noise.

Since the Android application is a prototype to demonstrate the functionality of the

developed algorithm, it should be integrated into a more consumer orientated user interface that, for example, includes better image acquisition controls, such as focus and flash. As shown in the previous section, the runtime requirements of the algorithm on a mobile device are well suited for regular usage. However, a more fluent workflow can be gained by exploiting the GPGPU capabilities, such as OpenCL, of modern smartphones. Especially image processing operations, such as the region preprocessing or keypoint matching, would gain performance due to the parallelism.



List of Acronyms

<i>BRISK</i>	Binary Robust Invariant Scalable Keypoints
<i>CSE</i>	Category-Specific Extremal Regions
<i>DMOS</i>	Description and Modification of Segmentation
<i>EPF</i>	Enhanced Position Formalism
<i>GMM</i>	Gaussian Mixture Models
<i>HSI</i>	Hue, Saturation and Intensity
<i>LBP</i>	Local Binary Patterns
<i>MSE</i>	Maximally Stable Extremal Regions
<i>OCR</i>	Optical Character Recognition
<i>ORB</i>	Oriented FAST and Rotated BRIEF
<i>PPHT</i>	Progressive Probabilistic Hough Transform
<i>RANSAC</i>	Random Sample Consensus
<i>SIFT</i>	Scale-Invariant Feature Transform
<i>SURF</i>	Speeded Up Robust Features
<i>SVM</i>	Support Vector Machines
<i>SWT</i>	Stroke Width Transform

Bibliography

- [1] Amano, A. and Asada, N. (2003). Graph grammar based analysis system of complex table form document. In *7th International Conference on Document Analysis and Recognition*, volume 2, pages 916–916. IEEE Computer Society. (page 14)
- [2] Bhaskar, S., Lavassar, N., and Green, S. (2010). Implementing optical character recognition on the android operating system for business cards. (page 9)
- [3] Bosch, A., Zisserman, A., and Munoz, X. (2007). Image classification using random forests and ferns. *Proceedings of IEEE International Conference on Computer Vision (ICCV)*. (page 28)
- [4] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32. (page 17, 33, 34, 56)
- [5] Breiman, L. (2002). Manual on setting up, using, and understanding random forests v3.1. http://www.stat.berkeley.edu/~breiman/Using_random_forests_V3.1.pdf. (page 60)
- [6] Canny, J. (1986). A computational approach to edge detection. *Transactions on Pattern Analysis and Machine Intelligence*, pages 679–698. (page 19)
- [7] Caruana, R. and Niculescu-Mizil, A. (2006). An empirical comparison of supervised learning algorithms. In *23rd international conference on Machine learning*, pages 161–168. ACM. (page 34)
- [8] Chang, F., Chen, C.-J., and Lu, C.-J. (2004a). A linear-time component-labeling algorithm using contour tracing technique. *Computer Vision and Image Understanding*, 93(2):206–220. (page 22, 24)
- [9] Chang, S.-L., Chen, L.-S., Chung, Y.-C., and Chen, S.-W. (2004b). Automatic license plate recognition. *IEEE Transactions on Intelligent Transportation Systems*, 5(1):42 – 53. (page 7, 21, 41)
- [10] CIE, Colorimetry (1986). Publication no. 15.2. *Bureau Central De la CIE, Vienna*. (page 31)
- [11] Cortes, C. and Vapnik, V. (1995). Support-vector networks. *Machine learning*, 20(3):273–297. (page 5, 41)
- [12] Coüasnon, B. (2004). Dealing with noise in dmos, a generic method for structured document recognition: An example on a complete grammar. In *Graphics Recognition. Recent Advances and Perspectives*, pages 38–49. Springer. (page 14, 15)

- [13] Coüasnon, B., Brisset, P., and Stéphan, I. (1995). Using logic programming languages for optical music recognition. In *3rd International Conference on The Practical Application of Prolog*, pages 115–134. (page 15)
- [14] Coüasnon, B. and Lemaitre, A. (2014). Recognition of tables and forms. *Handbook of Document Image Processing and Recognition*, pages 647–677. (page 12)
- [15] Dhiman, S. and Singh, A. (2013). Tesseract vs gocr a comparative study. *International Journal of Recent Technology and Engineering (IJRTE)*, pages 80–83. (page 61)
- [16] Donoser, M., Arth, C., and Bischof, H. (2007). Detecting, tracking and recognizing license plates. *Proceedings of Asian Conference on Computer Vision (ACCV)*, pages 447–456. (page 5, 6, 21, 35, 36, 41)
- [17] Duda, R. O. and Hart, P. E. (1972). Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15. (page 10, 18, 19)
- [18] Epshtein, B., Ofek, E., and Wexler, Y. (2010). Detecting text in natural scenes with stroke width transform. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2963–2970. IEEE. (page 35)
- [19] Fan, K.-C., Wang, Y.-K., and Chang, M.-L. (2001). Form document identification using line structure based features. In *6th International Conference on Document Analysis and Recognition*, pages 704–708. IEEE. (page 13)
- [20] Gomez, L. and Karatzas, D. (2013). Multi-script text extraction from natural scenes. In *12th International Conference on Document Analysis and Recognition (ICDAR)*, pages 467–471. IEEE. (page 35)
- [21] Guerzhoy, M. and Zhou, H. (2008). Segmentation of rectangular objects lying on an unknown background in a small preview scan image. In *Canadian Conference on Computer and Robot Vision (CRV)*, pages 369–375. IEEE. (page 11)
- [22] Hartl, A. and Reitmayr, G. (2012). Rectangular target extraction for mobile augmented reality applications. In *21st International Conference on Pattern Recognition (ICPR)*, pages 81–84. IEEE. (page 19)
- [23] Heliński, M., Kmieciak, M., and Parkoła, T. (2012). Report on the comparison of tesseract and abby finereader ocr engines. Technical report, Poznań. (page 61)
- [24] Herley, C. (2004). Efficient inscribing of noisy rectangular objects in scanned images. In *International Conference on Image Processing (ICIP)*, volume 4, pages 2399–2402. IEEE. (page 11)
- [25] Jung, C. R. and Schramm, R. (2004). Rectangle detection based on a windowed hough transform. In *17th Brazilian Symposium on Computer Graphics and Image Processing*, pages 113–120. IEEE. (page 19)

- [26] Lagunovsky, D. and Ablameyko, S. (1999). Straight-line-based primitive extraction in grey-scale object recognition. *Pattern Recognition Letters*, 20(10):1005–1014. (page 20)
- [27] Leutenegger, S., Chli, M., and Siegwart, R. Y. (2011). Brisk: Binary robust invariant scalable keypoints. In *International Conference on Computer Vision (ICCV)*, pages 2548–2555. IEEE. (page 45)
- [28] Luo, X.-P., Li, J., and Zhen, L.-X. (2004). Design and implementation of a card reader based on build-in camera. In *Proceedings of the 17th International Conference on Pattern Recognition (ICPR)*, volume 1, pages 417–420. IEEE. (page 8, 41)
- [29] Matas, J., Chum, O., Urban, M., and Pajdla, T. (2004). Robust wide-baseline stereo from maximally stable extremal regions. *Image and Vision Computing*, 22(10):761–767. (page 5, 36)
- [30] Matas, J. and Zimmermann, K. (2005). Unconstrained licence plate and text localization and recognition. In *IEEE Transactions on Intelligent Transportation Systems*, pages 225–230. IEEE. (page 6, 7, 21, 35)
- [31] Mollah, A. F., Basu, S., and Nasipuri, M. (2011). Segmentation of camera captured business card images for mobile devices. *CoRR*, abs/1101.0457. (page 9)
- [32] Ojala, T., Pietikäinen, M., and Harwood, D. (1996). A comparative study of texture measures with classification based on featured distributions. *Pattern recognition*, 29(1):51–59. (page 31)
- [33] Rother, C., Kolmogorov, V., and Blake, A. (2004). Grabcut: Interactive foreground extraction using iterated graph cuts. In *ACM Transactions on Graphics (TOG)*, volume 23, pages 309–314. ACM. (page 24)
- [34] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). Orb: an efficient alternative to sift or surf. In *International Conference on Computer Vision (ICCV)*, pages 2564–2571. IEEE. (page 45)
- [35] Sako, H., Seki, M., Furukawa, N., Ikeda, H., and Imaizumi, A. (2003). Form reading based on form-type identification and form-data recognition. In *12th International Conference on Document Analysis and Recognition*, volume 2, pages 926–926. IEEE Computer Society. (page 13)
- [36] Shinjo, H., Hadano, E., Marukawa, K., Shima, Y., and Sako, H. (2001). A recursive analysis for form cell recognition. In *6th International Conference Document Analysis and Recognition*, pages 694–698. IEEE. (page 13)
- [37] Smith, R. (2007). An overview of the tesseract ocr engine. In *International Conference on Document Analysis and Recognition (ICDAR)*, volume 7, pages 629–633. (page 41, 61)

- [38] Stallkamp, J., Schlipsing, M., Salmen, J., and Igel, C. (2012). Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural networks*, 32:323–332. (page 34)
- [39] Strobl, C. and Zeileis, A. (2008). Danger: High power! - exploring the statistical properties of a test for random forest variable importance. (page 60)
- [40] Tao, W.-b., Tian, J.-w., and Jian, L. (2002). A new approach to extract rectangular building from aerial urban images. In *6th International Conference on Signal Processing*, volume 1, pages 143–146. IEEE. (page 20)
- [41] Ting, A. and Leung, M. K. (1999). Form recognition using linear structure. *Pattern Recognition*, 32(4):645–656. (page 14)
- [42] Wu, Z., Kong, Q., Liu, J., and Liu, Y. (2011). A rectangle detection method for real-time extraction of large panel edge. In *6th International Conference on Image and Graphics (ICIG)*, pages 382–387. IEEE. (page 18)
- [43] Zhou, X., Yu, K., Zhang, T., and Huang, T. S. (2010). Image classification using super-vector coding of local image descriptors. In *Computer Vision–ECCV*, pages 141–154. Springer. (page 28)