



University of
Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization: Cybernetics/Signal processing	Spring semester, 2011 Open / Restricted access
Writer: Andreas Waal (Writer's signature)
Faculty supervisor: Tom Ryen External supervisor(s):	
Titel of thesis: Optical Character Recognition(OCR) on Electrical Specification Plates	
Credits (ECTS): 30	
Key words: OCR, pattern recognition, feature extraction, Image processing, segmentation, classification, direction extraction, k-nearest- neighbor	Pages: 31 + enclosure: 0 Stavanger, Date/year

Abstract

In this paper we have developed an Optical Character Recognition(OCR) system to be used on electrical specification plates. The project is given by the Stavanger based company Verico AS(Verico). Verico performs large scale of Asset Documentation, where *photo documentation* is one of the main tools. Today the data collection done from the photo documentation are performed manually, which is a monotonous and time consuming process. Our system is developed to streamline their data collection process, by automatically reading specification data from images captured of electrical specification plates.

The system contains two main sections, a preprocessing part and a character recognition part. The preprocessing part performs several image processing operations including background segmentation, character segmentation and several operations to prepare the image for classification. Background segmentation is performed using *Otsu's thresholding method*. For character segmentation we use vertical histogram analysis. We also present a modified version of vertical histogram analysis for splitting of connected characters.

The character recognition consists of two main parts, feature extraction and classification. We use *direction extraction* as our feature extraction method. This method looks at direction transitions in thinned version of the character one wish to classify. This method is originally presented as a method for recognition of handwritten characters, in our work we present the accuracy we obtained using this method against electrical specification plates. Classification is done using *k-nearest-neighbor* method.

Contents

Abstract	i
1 Introduction	1
1.1 Project background	1
1.2 Employer	1
1.3 Motivation	1
1.4 Paper outline	1
2 Problem	2
2.1 Background to the problem	2
2.2 Problem outline	3
3 Optical Character Recognition (OCR)	5
3.1 History	5
3.2 State of art	5
3.3 Commercial available products	6
3.4 Our problem and commercial available systems	6
4 Preprocessing	7
4.1 Background segmentation	7
4.1.1 Segmentation methods	8
4.1.2 Thresholding tests	9
4.2 Character segmentation	10
4.2.1 Vertical histogram for character segmentation	10
4.2.2 Modified vertical histogram	11
4.3 Image preparation	13
4.3.1 White on black	13
4.3.2 Removing frames	13
4.3.3 Character thinning	14
5 Character recognition	15
5.1 Feature extraction	15
5.1.1 Direction extraction	15
5.2 Classification	21
5.2.1 K-nearest-neighbor	21
5.2.2 Distance measurement	22
5.3 Training data	22
6 Experiments and testing	23
6.1 Available data	23
6.2 Our training data	24
6.3 Test bench	24
6.4 Experiment results	25
6.5 System computation performance	27

6.6	Discussion of test results	27
7	Conclusion and further work	29
7.1	Conclusion	29
7.2	Future work	29

1 Introduction

1.1 Project background

This thesis is written in the subject “Cybernetics and Signal Processing” at the University of Stavanger. The work performed are done spring 2011. A pilot project to the thesis was performed autumn 2010, in a subject which was weighed 10 ECTS points.

1.2 Employer

This problem is given by Verico AS(Verico). Verico is a company based in Stavanger, Norway, with Asset Data Management and Software Development as their primary areas of work. Their main field of business are directed against Transmission System Operators (TSO), such as Lyse and Statnett SF, where they in addition to the Asset Data Management and Software Development also performs Asset Documentation. Much of this documentation are done through *photo documentation*, which basically are documentation done though capturing images.

1.3 Motivation

The motivation for this paper is to automate parts the photo documentation processes used by Verico. Photo documentation are used as an important tool when documenting electrical installations, components etc. This is done through capturing images of specification plates. The data captured from the specification plates are manually inputted into the Asset Data Management systems developed by Verico. When the amount of specification plates are in the tens of thousands this makes is a time-consuming and monotonous task.

If one could find a method to automate this process it would be a significant automation of the system.

1.4 Paper outline

This paper is divided in seven chapters.

1. Introduction - Introduction to the employer and the motivation for this paper.
2. Problem - Detailed description of our problem.
3. Optical Character Recognition (OCR) - Introduction to OCR, historically and on a general basis.
4. Preprocessing - This includes background segmentation, character segmentation and several other image processing operations needed to prepare the image for the recognition stages.
5. Character recognition - Covers the character recognition processes. This includes feature extraction and characters classification.
6. Experiments and testing - Presentation of our test methods and results.
7. Conclusion and further work - Our conclusion and suggestions to further work.

2 Problem

2.1 Background to the problem

Verico's clients includes many major Transmission System Operators (TSO) which possesses large quantity of assets. These assets range from installations that are several decades old to brand new installations. This means that one got a wide range of components, manufactures and other variations. When working with this task Verico provided us with images of 30000 specification plates divided among 3000 classes.

In addition to the variation in the specification plate classes one got the human factor, since all images are manually captured, one will rarely find a perfect captured specification plate. Because of the physical limitations, such as the placement of the specification plates, it is not always possible to capture an image from a straight forward position. This means that a large percentage of images are captured from a warped position. This is major problem for the further process of streamlining the storage process, and as a result of this Verico has developed a software for warping images to a straight forward position, using templates customized to individual plate classes. An example of a warped specification plate can be seen in figure 1.

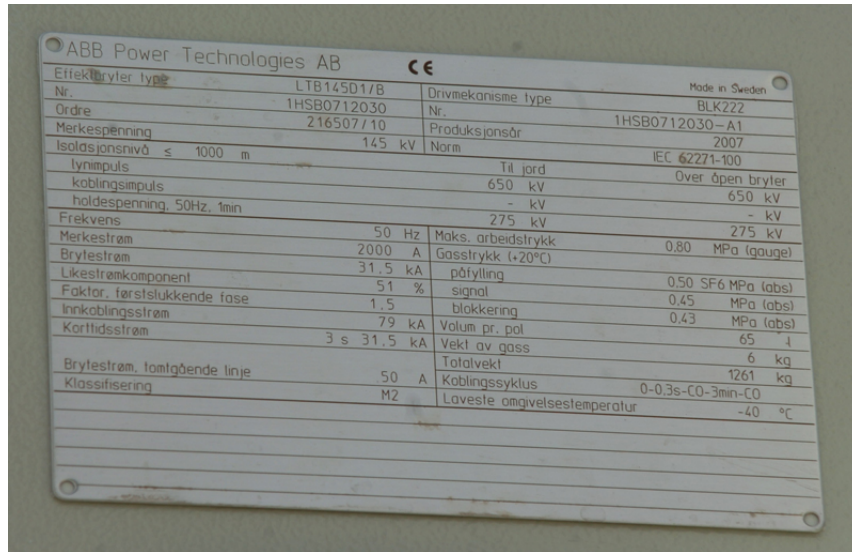


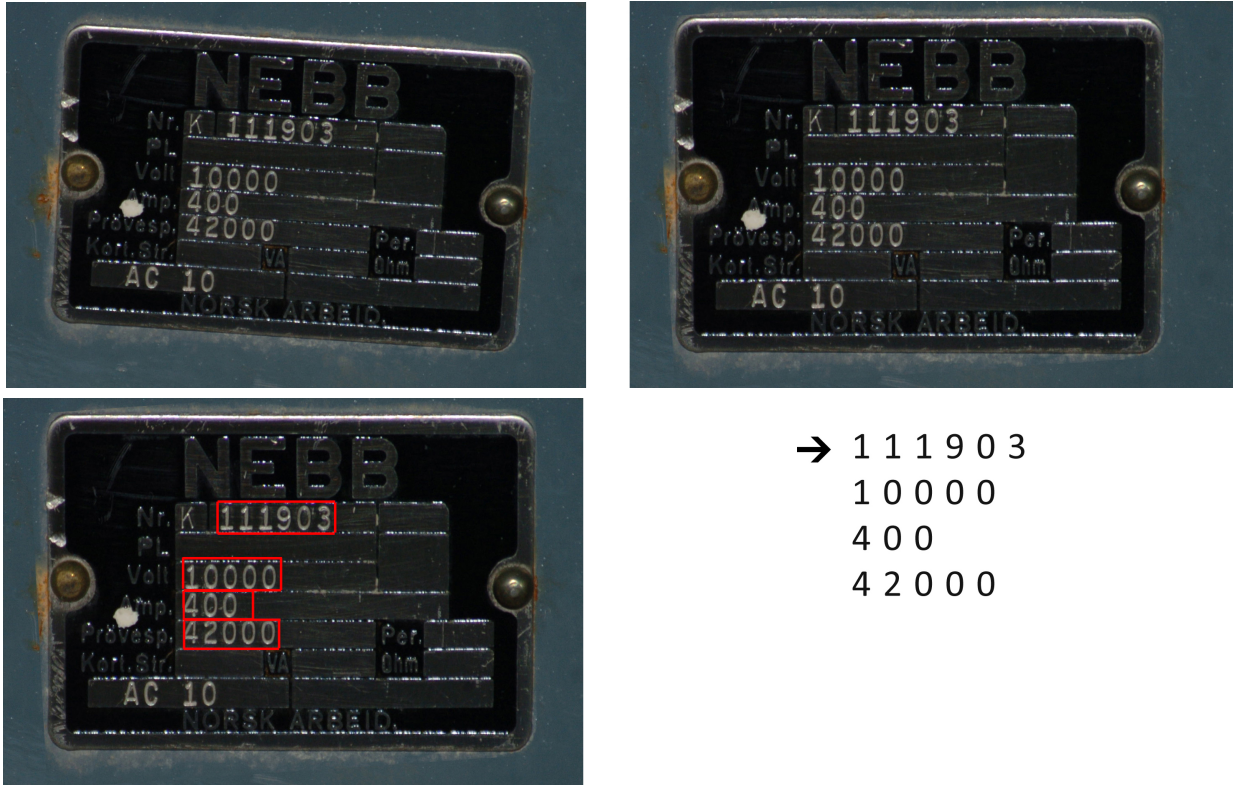
ABB Power Technologies AB		CE		Made in Sweden	
Effektbryter type	LTB14501/B	Drivmekanisme type			
Nr.	1HSB0712030	Nr.		BLK222	
Ordre	216507/10	Produksjonsår		1HSB0712030-A1	
Merkespenning	145 kV	Norm		2007	
Isolasjonsnivå ≤ 1000 m				IEC 62271-100	
lynimpuls		Til jord		Over åpen bryter	
koblingsimpuls		650 kV		650 kV	
holdespenning, 50Hz, 1min		- kV		- kV	
Frekvens	50 Hz	275 kV		275 kV	
Merkestrøm	2000 A	Maks. arbeidstrykk		0.80 MPa (gauge)	
Brytestrøm	31.5 kA	Gasstrykk (+20°C)			
Likestrømskomponent	51 %	påfylling		0.50 SF6 MPa (abs)	
Faktor, førstslukkende fase	1.5	signal		0.45 MPa (abs)	
Innkoblingsstrøm	79 kA	blokkering		0.43 MPa (abs)	
Korttidsstrøm	3 s 31.5 kA	Volum pr. pol		65 l	
		Vekt av gass			
Brytestrøm, tomgående linje	50 A	Totalvekt		6 kg	
Klassifisering	M2	Koblingsyklus		1261 kg	
		0-0.3s-CO-3min-CO			
		Løveste omgivelsestemperatur		-40 °C	

Figure 1: Example of an electrical specification plate captured from a warped position.

The templates assigned to each plate class contains information of plate size and the location of data fields of interest. When a template is chosen to the given specification plate, an operator selects three corners of the plate using a Graphical User Interface(GUI) program. Using the data from the template, the image is warped and fields of interest is marked.

The last step of this process is our problem, find a method to automatically read the content of the marked fields and translate this into machine encoded text. This is illustrated in figure 2. If one manage to solve this problem, the total process of storing the

data in one plate would be three mouse clicks. This would be a major efficiency of the process.installations



$$\begin{array}{c|c} a & b \\ \hline c & d \end{array}$$

Figure 2: (a)The captured image, (b)warped image, (c)data fields of interest marked, (d) data translated to machine encoded text.

2.2 Problem outline

To be able to automatically recognize and translate the characters, we need to do two things.

The first is a preprocessing part, where the image is prepared for character recognition. After this process, one would have an image containing separated characters against a white background.

The second problem is the character recognition. This part performs the recognition and outputs the result as computer encoded text.

In most cases the characters needed to be recognized are numbers. Thus we will only work to recognize number and not letters in this project.

Part 1: Preprocessing - Image Processing

In the first part of the problem we wish to prepare our input images for the character recognition process. This can again be divided into two sub-problems.

The first problem is background segmentation. In [6] segmentation is summarized as “*Segmentation subdivides an image into constituent regions or objects*”. In our case we wish to segment the characters from the background.

The *major* challenge regarding background segmentation is to find a method that works uniformly, and can be applied to the wide range of different plates we will be working with. The method chosen must also be able to perform segmentation, without removing vital information from the characters.

The second part is character segmentation. This is the process of dividing the characters apart from each other. When this is done we want to have a region, or sub-image, containing each of the characters.

Part 2: Character recognition

This part contains of two main sections. Feature extraction and classification.

The feature extraction section is the main part of our problem. Extraction of features are how we describe characters, and thus how we separate them. Good features are absolute crucial in this project.

Classification is the decision of which character we are dealing with. This section uses pattern recognition to decide which class a character belongs to, based on the extracted features.

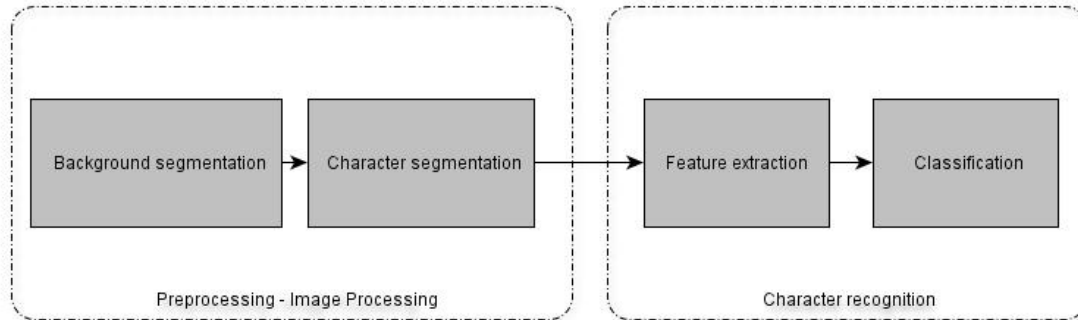


Figure 3: The main parts of our problem.

3 Optical Character Recognition (OCR)

OCR is the process of translating images, documents and other physical medium into computer encoded character.

3.1 History

The first OCR systems was patented in 1929 [12] in Germany by Tausheck and in 1933 by Handel in the U.S., but due to lack of technology this ideas remained as a dream for a long time. When the computer age started in the 1950's the ideas patented two centuries earlier was revealed again. The first systems introduced were based on the principle of template/mask matching. The basic of this technique was a combination of mechanical and optical components. Light was passed through a mechanical mask and captured by a photo-detector and then scanned mechanically. When a match occurred the light failed to reach the detector and the system recognized a match. Even though the methods existed, the technology suffered strong limitations due to lack in technology.

One of the first fields that applied OCR on a lager scale was postal services, as early as 1965 the United States Postal Service started using OCR machines to sort mail. Later the technology has been applied to many other fields such as reading of tax forms, bank checks and various application forms.

As a result of computer technology becoming common property and its computing power increasing, desktop OCR systems has been introduced[11]. This is applications used by end users for converting various documents of daily use into coded form. Examples of this is scanned documents, letters etc.

3.2 State of art

A study [7] from 2009 conclude that typewritten text is still not 100% accurate even where clear imaging is available. Further on, they conclude that even though most commercial system claims an accuracy of 99% and better, these recognition rates are based on documents of perfect quality and are often much lower in applied settings. Tests showed an accuracy of 71% to 98.02% using commercially available systems.

Since 100% accuracy not yet has been managed, many commercial available system use different post processing algorithms to increase accuracy. An example of post processing are spell checking system who controls all recognized words against a dictionary and uses this information to increase accuracy.

Other areas like recognition of hand printing, cursive handwriting and printed text in other scripts(such as East Asian languages) are still subject of active research[2].

One of the field where OCR systems are applied in a larger scale is Automatic Number Plate Recognition(ANPR) systems [4]. This is a mass surveillance method that uses OCR technology on license plates on vehicles. This is applied in many different setting such as police enforcement, average speed cameras, traffic control systems and electronic toll collection.

As a result of technology and computation power growing, and becoming more available to the public, OCR systems are transferred to new platforms. One of the latest platforms for OCR are Smart-phones, and in 2010 *Quest Visual* launched the application *Word Lens*

for the *Apple iPhone*[10]. This application scans a text using the iPhones video mode, recognizes it and translates it to a chosen language. This application uses the video mode to capture multiply images of the same characters, and uses this to increase accuracy.

3.3 Commercial available products

It is a numerous of available commercial OCR software, from simple and free software that is able to recognize simple documents in good quality, to more sophisticated products that manage advanced page layouts.

In our research of the problem, we have found that in the later years it has been developed two different main areas for OCR software.

The first is directed against document digitization for the home user. The main purpose of these products are to digitize documents. These systems often use intelligent learning combined with spell checking algorithms for further improving of accuracy. Tests from 2008 [14] points out *ABBYY FineReader* and *OmniPage 16* as the leading softwares for document digitization.

The second area of OCR software is customized systems used in industrial settings. An example this is a system used to recognize license plates in road toll systems[4].

3.4 Our problem and commercial available systems

Most of the commercial available programs are intended for recognition of characters in documents. Many of these programs can show excellent accuracy in recognition rate, but got strict requirements to the input source.

These programs are designed to work in a uniform setting and to be applied to any given situation without the need, or the possibility, of large scale training and customization to the given settings. *ABBYY FineReader* and *OmniPage 16* both got the possibility to train against new characters[13, 1], but both of them limit training to a few characters at once, and not to a larger set of training data.

Contrary to document recognition we will only be working with smaller blocks of characters, and not an entire document. This gives us the possibility to perform image processing operations in a more significant manner then when dealing with an entire document. We will also be able to train our system against the same type of data that we are going to recognize.

4 Preprocessing

This chapter describes the preprocessing of character images for the later stages. The goal of this process is to localize the individual characters that are to be classified in the later stages, from a raw image. This is done through background and character segmentation. In addition to this, we perform several processes to prepare the image even more.

4.1 Background segmentation

The first step of the preprocessing is background segmentation. This is the process of removing the background from the image, with the goal of creating an image only containing the characters we wish to recognize. We also want the result after this stage to be a binary image. The process of transferring an image from grayscale to binary image is done using thresholding[6].

The basic equation for thresholding can be seen in equation 1.

$$g(m,n) = \begin{cases} 1 & \text{if } f(m,n) > t \\ 0 & \text{else} \end{cases} \quad (1)$$

In equation 1, f is our input image, g is the binary image after thresholding, m and n being pixel coordinates. In the equation, t is our threshold value. The threshold value is in the range $t \in [0, 1]$, which gives us the intensity range we wish to distribute to black or white.

$$t = \frac{\text{Intensity value}}{\text{Intensity range}} \quad (2)$$

The threshold value is calculated using equation 2. In a grayscale image the *intensity range* would be 256, and the *intensity value* would be the given pixel value. All pixels with a value above the threshold value are set to 1, representing white, while the rest of the pixels are assigned the value 0, representing black.

The process of thresholding is actually a segmentation process since by modifying the thresholding value can remove unwanted parts of an image. This can be done manually by analyzing the image or using algorithms that automatically set the threshold value. Examples on segmentation using different thresholding values can be seen in figure 4.

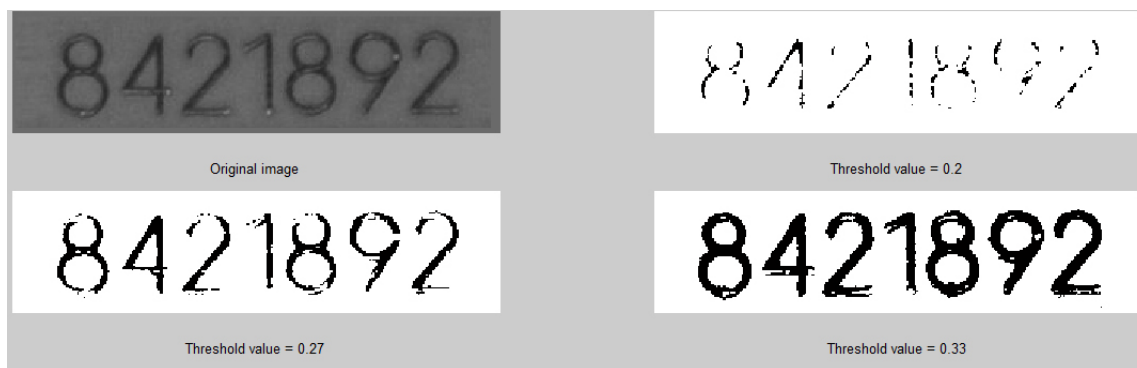


Figure 4: Examples on thresholding for background segmentation.

4.1.1 Segmentation methods

The formula presented in Equation 1 is the foundation for segmentation through thresholding, but as mentioned earlier, this formula alone requires the thresholding level to be set manually. Since we are going to process a large scale of data we wish to create a system that do not require any input from user, except during the warping process.

Litterateur [6] got several methods for automatically setting of thresholding values, we will look at some of the most applied methods.

Basic Global Thresholding[6] is a simple, but effective thresholding method. This method is performed using Algorithm 1.

Algorithm 1 Basic Global Thresholding algorithm.

1. Select an initial estimate for the global threshold, T .
2. Segment the image using T . This gives us two groups of pixels, G_1 which consists of pixel with intensity values above the threshold and G_2 with pixels below the threshold.
3. Compute the m_1 and m_2 , which is average value of regions G_1 and G_2 .
4. Compute a new threshold with the following equation

$$T = \frac{1}{2}(m_1 + m_2)$$

5. Repeat step 2 through 4 until the difference between T and the previous T is smaller than predefined value, ΔT
 6. Segment the image using the final T value.
-

Optimum Global Thresholding Using Otsu's Method[9, 6] is a method much similar to Basic Global Thresholding, but instead of just calculating average values it chooses the threshold to minimize the intraclass variance of the black and white pixels. This method is implemented in the Image Processing Toolbox in Matlab under the function *graythresh()*. For more documentation see [9, 6].

Improved Global Thresholding using Edges[6] is another thresholding method we will look at. This method is based on the Basic Global Thresholding method, but it uses edge detection for improving the variations in the histogram.

Algorithm 2 Improved Global Thresholding using Edges

1. Compute an edge image from $f(x, y)$, using an edge detector such as Sobel[6], Prewitt etc.
 2. Specify a threshold value, T , from $f(x, y)$ using *Basic Global Thresholding*, *Otsu's method* or other thresholding methods
 3. Threshold the edge image from step 1 using the threshold value from step 2. This creates a marker image $g(x, y)$.
 4. Compute a histogram using only the pixels in $f(x, y)$ that correspond to the locations of the 1-valued pixels in $g(x, y)$.
 5. Use the histogram from step 4 to segment $f(x, y)$ globally.
-

4.1.2 Thresholding tests

The methods discussed above are general segmentation methods and we do not know how they will work against our problem. For testing we ran initial tests against 20 different specification field, from four different plates. The purpose of this test was to see if thresholding was suited for our problem, and to see if any of the methods are better than the rest. Two of the test images and the threshold value chosen by the method, can be seen in figure 5 and 6.



Figure 5: Test of segmentation methods.



Figure 6: Test of segmentation methods

After examination of the test result, we concluded that images segmented using Otsu’s method were the best, with overall very good results. The method managed to deliver good segmentation while persevering the shape and information of the characters in a very good manner. *Basic global thresholding* gave in most cases similar results to Otsu’s method, but removed more information from the character. *Improved Global Thresholding using Edges* gave variable results, in some cases it delivered very good result, but in many cases the method either removed too much or too little of the character.

Our conclusion is that Otsu’s method gives the best background segmentation of the threshold methods we have tested, and will be the applied method in our work.

4.2 Character segmentation

Character segmentation is the process of dividing an image containing characters into sub-images containing characters or words, depending on the OCR system. This process can be done for both gray-scale and binary images[3], with several different methods for both image types. A system with bad segmentation may create problems for the next stages of the OCR system. A good example of this is the letters “cl” and “d”. If the system segments “cl” as one character instead of two this may be classified as a “d” later in the system.

Literature [3] concludes that we got three “pure” strategies for character segmentation. These are *dissection*, *recognition-based* and *holistic segmentation strategies*. We will work with dissection methods in our problem.

When performing dissection segmentation we decompose an image into a sequence of sub-images using general features. OCR problems vary a lot in complexity, so do the complexity in segmentation strategies.

4.2.1 Vertical histogram for character segmentation

A widely used method when working with spaced or partially spaced characters is *vertical histogram*[3]. This method performs a count of white pixels in each column of the picture and gives a histogram from this. This histogram may be used directly for segmentation, which will work great when working with spaced characters. When dealing with spaced characters the vertical histograms will read zero between characters and give us the location where

one character ends. Using this information we can precisely segment characters. It is still possible to use vertical histograms for segmentation in cases with connected characters, but we need a method that is modified accordingly to this. An example of a vertical histogram may be seen in figure 7. In our project we will only work with white on black images.

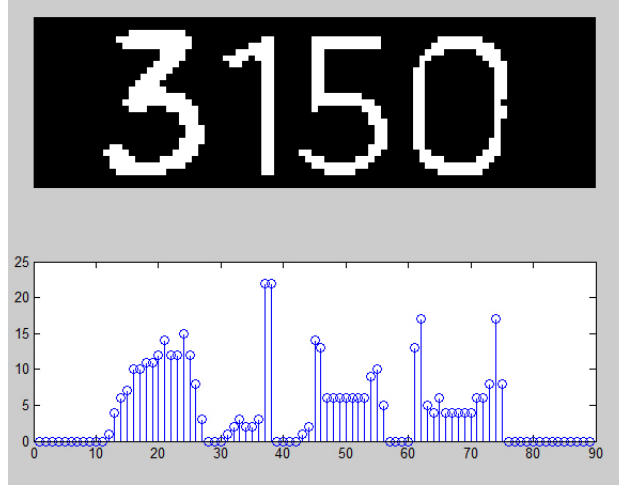


Figure 7: Example of vertical histogram

When working with electrical specification plates, we will be working with both connected and spaced characters. The rate of connected and spaced occurs is unknown, but we know that it is a majority of spaced characters. When vertical histogram segmentation is applied in cases with connected characters, we will get a multiple of characters detected as one.

To solve this we need a way to identify cases where multiple characters have been detected as a single character, and a method for separating them. One way to find out if a detected character actually contains more than one character is by looking at the height against width ratio of the character. This gives an opportunity to use standard vertical histogram segmentation in most cases, and when encountered connected characters apply other methods for this.

Even though characters are connected it is possible to see where one character ends and the next one starts using a vertical histogram. After testing several methods, we managed to find a method for separating the characters.

4.2.2 Modified vertical histogram

After regular splitting using vertical histogram, we inspect the width against height ratio for each of the segmented characters. If a ratio out of the ordinary occurs, one performs a modified vertical histogram algorithm. In our test we found the ratio that gave the best result to be 1.09. The connection of two characters are usually represented by the lowest values in a vertical histogram. At this position we perform a splitting of characters. An example of this can be seen in figure 8c.

By removing all pixel values corresponding to histogram values below a given percentage,

in our test we start at 5%, one provokes a splitting of characters. This percentage starts at a low value and are stepwise increased. After each removal, ratios of sub-images are checked and the process is continued until all ratio values are below the acceptable value. Example of the process can be seen in figure 8. A summary of this process can be seen in algorithm 3.

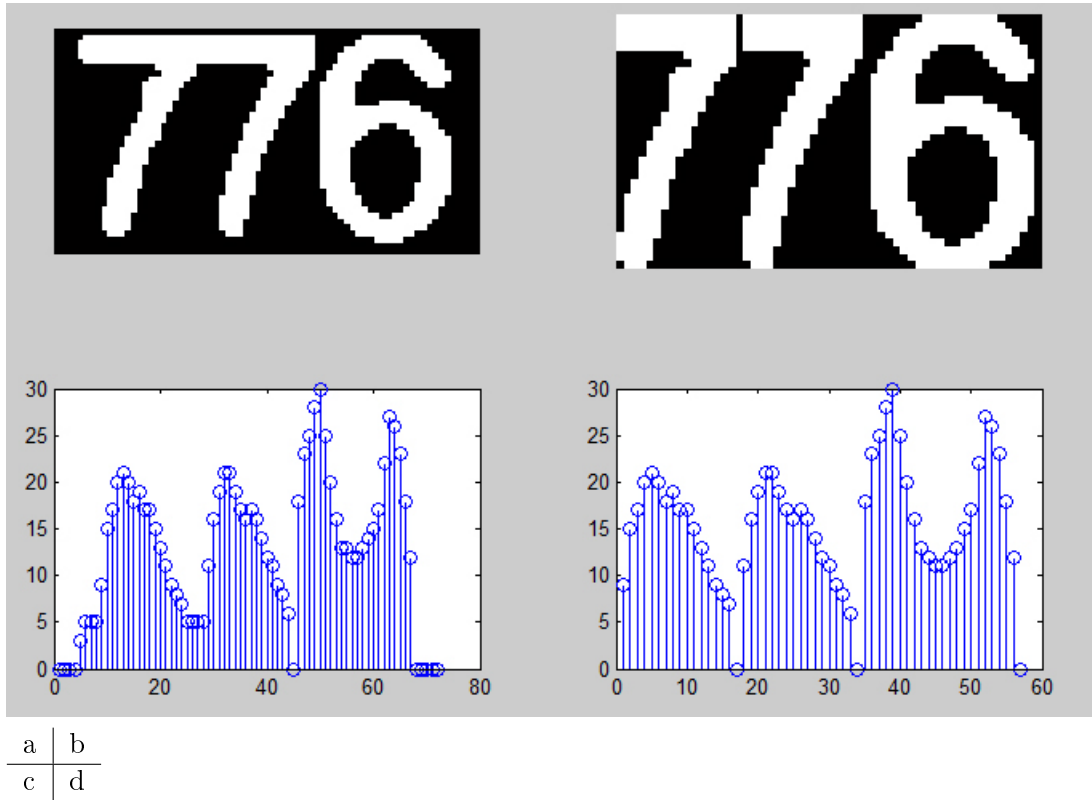


Figure 8: (a) Connected characters (b) Characters after splitting (c) Vertical Histogram before splitting (d) Vertical histogram after splitting

Algorithm 3 Character splitting using modified histogram

1. Perform vertical histogram segmentation
 2. for(all splitted characters)
 - 3a. Calculate height:width = ratio
 4. While(ratio < preset_ratio = 1.09)
 - 4a. Split using modified histogram with percentage = 5%
 - 4b. Vertical histogram segmentation
 - 4c. Calculate new height:width = ratio
 - 4d. percentage = percentage + 1%
 - 3b. Store sub-image
-

4.3 Image preparation

To realize the segmentation with a satisfying result we need several operations that are not directly segmentation related.

4.3.1 White on black

The characters we wish to recognize can be both white and black after background segmentation. Due to the nature of binary images in Matlab, white being represented with the value “1”, one wish to have white characters on black background. This makes it convenient to work with since one easy can find ratios of character representation in rows, lines or image parts by summing all pixel values. To ensure that characters is represented by white, we got an algorithm that looks at the ratio of white pixels against the total number of pixels. This is performed after the thresholding process on the entire field of data. If this ratio is higher than a preset value the image is inverted, in our tests the best result was obtained with a value of 0.64.

4.3.2 Removing frames

Since the data fields of interest are selected from templates made to fit a specific plate class and not cut to fit each individual plate, it may happen that we get a frame surrounding our data. To prevent this an algorithm detecting and removing frames is developed.

The algorithm scans rows in a given percentage of the top and bottom section of the image and calculate the ratio of white pixels against the total number of pixel in the given row. If this ratio exceeds a given value, in our case we use 0.7, the value of the entire row is set to black. An example of this function may be seen in figure 9.

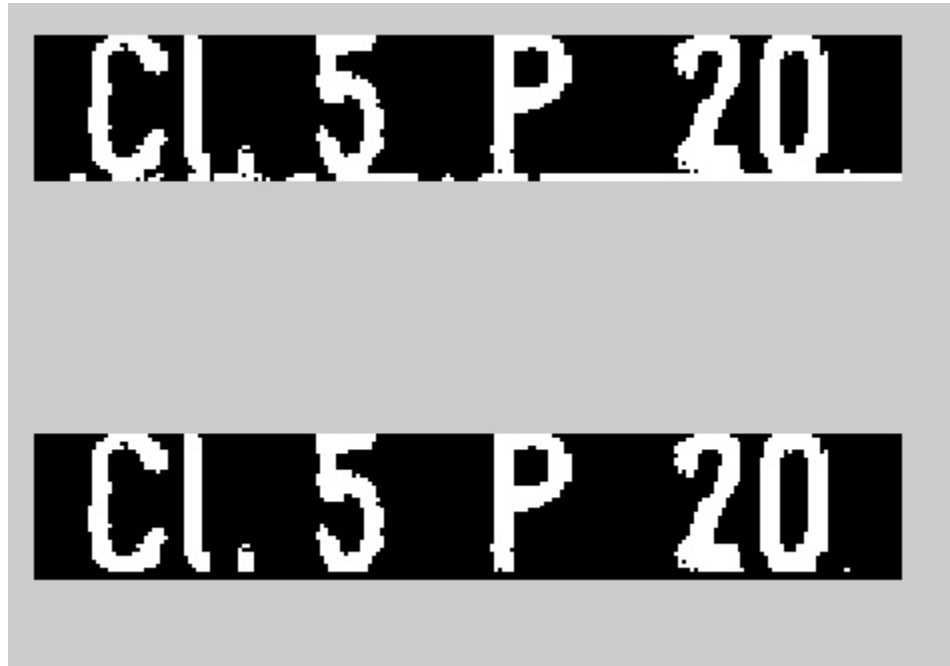


Figure 9: Example of *frame removal algorithm*

4.3.3 Character thinning

Thinning of a character is the process of removing pixels so that an object without holes shrinks to a minimally connected stroke[9], one pixel wide. Example of a thinned character can be seen in figure 10.

After thinning the character image will contain a frame of blank pixels, as shown in figure 10, resulting from the removed pixels. The size of this frame contains some information about the character because different characters and fonts will give different size of frame. Because of this we will, in chapter 6.4, run experiments on test sets with the frame intact, with top and bottom frame removed and entire frame removed, in order to find what gives the best results.

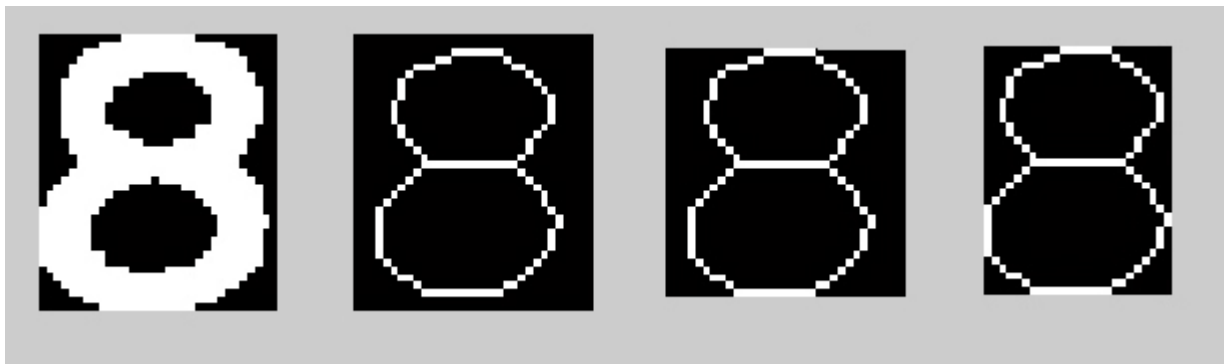


Figure 10: Example of thinned character

5 Character recognition

In this section we perform the character recognition, which includes the feature extraction and classification sections.

5.1 Feature extraction

To recognize a character we need a method to describe the character. This is done by extracting features from the character we wish to recognize[11]. Due to the difference in complexity in OCR problems, it is also a wide range in feature complexities. An OCR system constructed for recognition of machine-printed characters of a known font requires a much less complex feature extraction method than a system made for recognition of handwritten characters.

5.1.1 Direction extraction

This method was first presented in litterateur by Blumstein et.al.[2], and was presented as a method for recognition of segmented handwritten characters. Blumstein et.al.[2] managed to achieve a recognition accuracy of 70% on lowercase handwritten characters and 80% on uppercase handwritten characters.

Since we in our problem will be working with characters deformed as a result of warping, background segmentation and other process, we found that characters in many cases would be more similar to handwritten characters than machine typed, and as a result of this we decided on a method originally designed for handwritten characters.

The basis for this method is a thinned version of the character, which is divided into zones. Each of the zones are then divided into segments according to a set of rules. Finally each of the segments is labeled with directions. The process of direction extraction need the following steps to be implemented.

- a. Finding intersections
- b. Line localization and line segmentation
- c. Direction labeling
- d. Finding main directions
- e. Zoning
- f. Feature vector extraction

a. Finding intersections

Intersections are point where two or more lines meets. This can be identified by a pixel with three or more neighbor pixels, neighborhood pixels are defined as the four or eight surrounding pixels to a pixel[6].

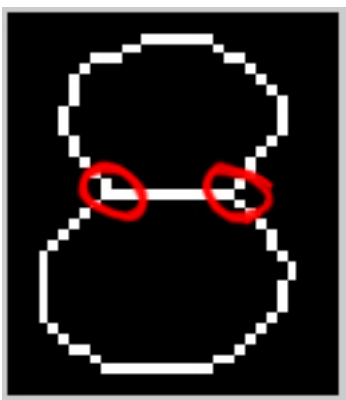


Figure 11: Example of intersections in a thinned character, intersections marked with red circles

b. Line localization and line segmentation

After localization of intersections, we wish to divide the thinned character into smaller line segments. One line segment is defined as a line in between two intersection points. In cases with characters not containing intersections, such as a zero, we only get one line segment.

These line segments are localized by first removing the intersection points from the characters such that the line segments are physically separated. When this is done the line segments are localized by finding all connected regions. The intersections are added to the image later in the process.

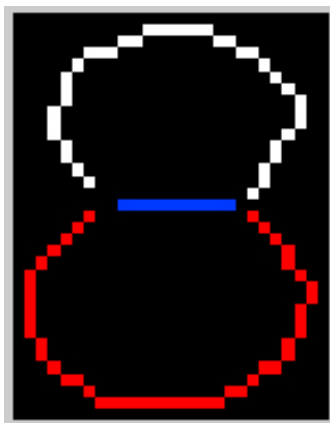


Figure 12: Example of the line segments in a character

c. Direction labeling

The thinned character are now divided into line segments. At this step the line segments are labeled according to direction. This is done by looking at the direction transition from one pixel to the next one. We operate with four directions, and two special cases labeled as

shown in Algorithm 4. The values chosen to label direction, starting point and intersections are just for identification and do not serve any other purpose than to be a known value.

Algorithm 4 Labeling values

- 2: Vertical transition
 - 3: Right diagonal transition
 - 4: Horizontal transition
 - 5: Left diagonal transition
 - 8: Starting points
 - 10: Intersections
-

The process of labeling transitions is done by first locating the starting point of a line segment. A starting point is a pixel with only one neighbor pixel. When locating the starting point, the algorithm search the image from the lower left corner. This is an important detail. In cases where characters do not have any starting points, such as a zero, the first pixel encountered when searching from lower left corner are considered to be the starting point.

After the starting point are localized, the algorithm follows the line and labels the transition from one pixel to the next one. This is done by creating an 8-connected neighborhood to each pixel. The 8-connected neighborhood of a pixel is the 8 pixels surrounding this pixel[6]. Except from the previous pixel, each pixel will only have one neighbor. This is the next pixel of the line segment, and the direction of this pixel gives us the direction of the transition. Figure 13 shows the 8-connected neighborhood to a pixel and the direction labels described in Algorithm 4.

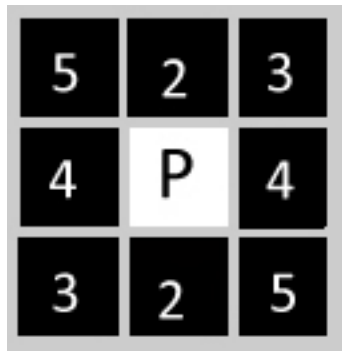


Figure 13: Labeling using the 8-connected neighborhood

Example of a labeled line segments can be seen in figure 14, the line segments shown are the line segments marked white in figure 12.

f. Feature vector extraction

For each of the nine zones of the character we create a vector with the following values.

1. Horizontal line marker
2. The length of horizontal lines
3. Right diagonal lines marker
4. The length of right diagonal lines
5. Vertical lines marker
6. The length of vertical lines
7. Left diagonal lines marker
8. The length of left diagonal lines
9. Intersection marker

The marker values(1,3,5,7,9) got a basic value of 1, and are subtracted by 0.2 if the direction occurs in the current zone. The intersection marker(9) got a default value of 0, but in cases where intersections are detected it are set to $1 - (0.2 * \text{number of intersections})$.

The line lengths(2,4,6,8) is calculated as in Equation 3. This formula is very important for the system since it scales the feature vector against the image size, and thus making the system invariant to image size.

$$length = \frac{\text{number of pixels in particular direction}}{(\text{window height}) * 2} \quad (3)$$

Having nine zones, these calculations give us a total of nine vectors with nine elements in each, that are sorted in a matrix of the size 9 x 9. In figure 17 three of the zones from figure 16 with normalized values and corresponding feature vectors are shown.

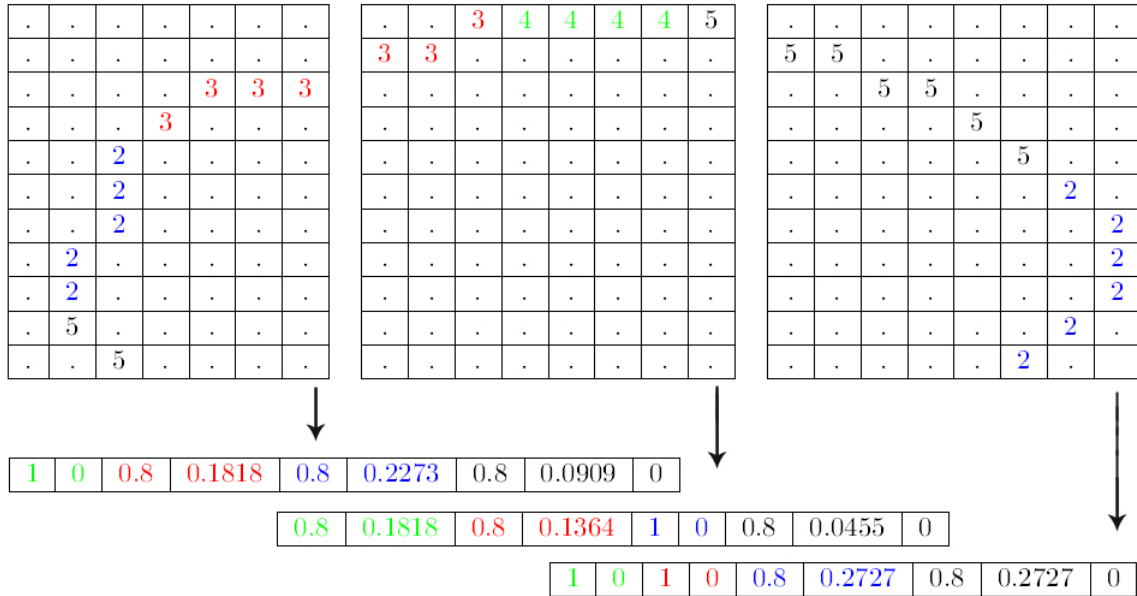


Figure 17: Example of vector extraction

5.2 Classification

The goal of this stage is to classify, and assign a machine encoded character to the unknown character we wish to recognize. Throughout the years many different methods for pattern recognition and classification have been tested for character recognition. In a study by Mo-hiuddin and Mao [11] they conclude that the methods *k-nearest-neighbor* and *feed-forward neural networks* are the most applied and most successful methods, with neither of them being a clear winner. This is based on tests done in a different setting than ours, so we cannot conclude that the same is the case for us. But, since k-nearest-neighbor is the one easiest implemented, we choose to use k-nearest-neighbor and see if it could be applied successfully in our application.

5.2.1 K-nearest-neighbor

This is a simple, but very efficient method. K-nearest-neighbor is an instance-based learning algorithm[5], which compares a new problem, in our case an unknown character, against instances. The algorithm needs stored training data to be able to classify. This means that the result of each classification relies totally on the training data.

This basic principle of this method is to compare the unknown class against all training data. This is usually done by a measurement of distance, and then look at the *k*-nearest occurrences to the unknown class. With *k* being the variable that decides the number of neighbors to include. After measuring distances between the unknown class, in our case the unknown character, and the training data, one classifies by analyzing the k-nearest occurrence to the unknown class.

In our problem we classify the unknown class to the class with the most occurrence. In cases with equal occurrences from two or more different classes, we choose the class which got the occurrence with the shortest distance.

5.2.2 Distance measurement

In our work we use the *Euclidean distance*[5] as our distance measurement. The basic formula for Euclidean distance between two vectors can be seen in Equation 4, where d is the distance between the features, q and p being the vectors we wish to calculate distance between.

$$d = \sqrt{\sum_i^M (q_i - p_i)^2} \tag{4}$$

We want to calculate the distance between the unknown characters feature matrix and each of the training feature matrices available. This is performed using Equation 5, where d is distance between the features, a_{ij} is the unknown character feature matrices and b_{ij} is the training data feature matrix we wish to calculate distance against.

$$d = \sqrt{\sum_i^M \sum_j^N (a_{ij} - b_{ij})^2} \tag{5}$$

5.3 Training data

The purpose of training data is to have a set data to compare an unknown class against, either by comparing directly as with instance-based algorithms or by training the system in advanced. In either cases the training data are what makes the system able to recognize and classify an unknown class, there are exceptions such as systems recognizing using decision rules and decision trees[8], but in the majority of systems either k-nearest-neighbor or a neural network, training data are needed to train the system. An example of training data in a character recognition system would be characters of the font or type one expect the system to work with.

6 Experiments and testing

A major challenge when designing an OCR system is to design and perform proper testing. The main goal of testing is to find out how the system would perform in a real setting. For us this is a crucial process because the result from testing would decide if our system can be put into real use or not.

6.1 Available data

For testing Verico has provided a large database of specification plates, containing images of over 30000 specification plates from around 3000 different component types. The size of the different classes varies greatly with the smallest classes containing only a few images to the largest containing hundred of images. Even though we got as many as 3000 different components, lots of specification plates are very similar.

Choose data classes

In the massive database available from Verico, we selected four specific plate classes for testing our OCR. The reason we chose these four classes, is because they are specification plate types used by some of the largest manufacturers and together they represent the largest classes of specification plates.

Examples from these four classes can be seen in figure 18.

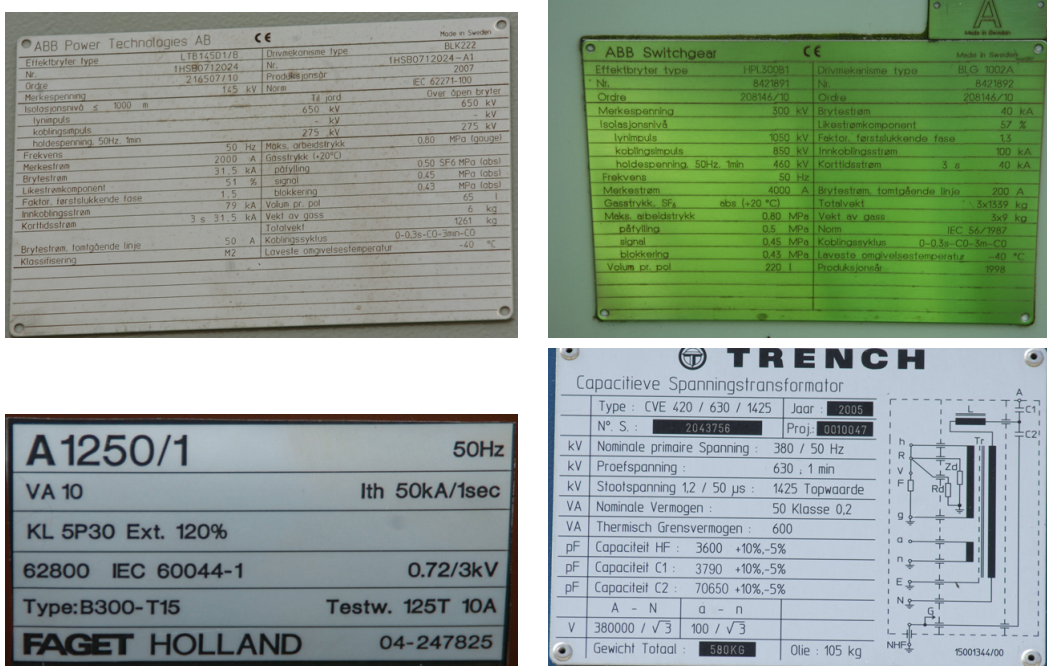


Figure 18: Specification plates from the four classes used in the test bench.

In a real setting our system is the second step in a process where step one warps the image and marks the fields of interest. Since we in this stage primarily wanted to test the

OCR accuracy of our system, we performed the warping and field of interest marking in advance.

6.2 Our training data

When training our system we needed to find a standardized method of inputting training data to the system. As a result of our choice of classifier the system required *a-priori* information about the true value of all the training data. We found that one way to realize this was to feed the system with images containing training characters together with the true value of the character. By doing this the system could extract features from the training data and store them in a register categorized using the *a-priori* information collected together with the training data. Since we already had method for splitting characters one could include multiple characters in one single image, instead of having one image for each character the system was to be trained against.

This left us with two alternatives for inputting training data. The first option was to feed the system with character region collected direct from the specification plates. The second option was to manually collect characters from the specification plates and sort them into a predetermined order. This meant more work editing, but since the order of characters already was known we did not have to provide the system with *a-prior* information about the characters.

We went with option two since this gave us better control over how many training data one included from each class. Since one training set would include all the numbers the system would work against, one would get exactly the same amount of training data to each class. Using this method, each training set would be an image containing the number zero to ten, which would be split ted using the methods discussed earlier.

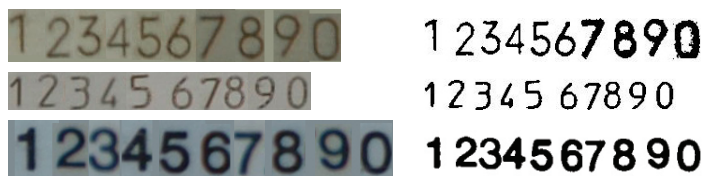


Figure 19: Examples of training sets.

Due to the limited available characters in each specification plates we found it to be natural to create one training set for each specification plate we trained the system against. When training the system this way, using real data, it gave us the possibility to train the system against many different light conditions and other outer interference.

6.3 Test bench

The testing phase was not only to measure the accuracy of the system, but also as a tool to test the system during development, in order to get the best performance. To do this, we created a test bench, which made it possible to test the system against the same problem, using the same test and training data. This way we could see how small changes in our system would affect the accuracy of the system.

The test data used in the test bench was collected from the four classes described above, and contained 923 characters.

For training we used in total 64 training sets collected from the four specification plates. These training sets were collected from same classes, but different specification plates than the test data. In addition to this, we included 24 training sets of common fonts available, including Arial, Calibri and etc. These training data were created using an text editor and had a perfect quality, opposite to the training data collected from the specification plates. This gives us in total 88 training sets.

6.4 Experiment results

Using the test setup described above, we performed multiple tests in order obtain the best accuracy.

The first issue we wanted to test was how the frame obtained during the thinning process affects the results. This is covered in detail in section 4.3.3, the three different cases we wish to test is:

1. No frame removed
2. Top and bottom of frame removed
3. Entire frame removed

For each of the cases we created training data and run tests using the setup described for the test bench. For each case we ran in total of 15 test, where we adjusted the k , of the k -nearest-neighbor classifier, value from 1-15(this is covered in section 5.2.1). Since the setup are identical from case to case the result can be compared directly.

k	No frame removed	Top/bottom side removed	Entire frame removed
1	96.64	96.64	97.51
2	96.64	96.64	97.51
3	96.86	96.86	97.40
4	96.86	96.86	97.40
5	96.86	96.86	98.05
6	96.64	96.64	97.94
7	96.94	96.64	97.29
8	96.97	96.64	97.40
9	96.64	96.97	97.18
10	96.86	96.86	97.18
11	96.86	96.86	97.07
12	96.97	96.97	96.97
13	96.97	96.97	96.64
14	96.97	96.97	96.64
15	96.32	96.32	96.64

Table 1: Test results for the cases “no frame removed”, “top and bottom removed” and “entire frame removed”, for $k = 1-15$

The best result accuracy during tests was **98.05 %**. This was obtained when removing the entire frames were and with a k value of 5.

Best results obtained when not removing or partially removing frame was **96.97 %**. The tests show that the results from these two methods are almost identically. Further on tests show that results obtained when removing the entire frame gave the overall best result.

Character	1	2	3	4	5	
Successful classifications	69	88	91	83	90	
Misclassification	0	1	3	3	1	
Percentage success (%)	100	98.88	96.87	96.51	98.90	

Character	6	7	8	9	0	Total
Successful classifications	81	61	29	19	294	905
Misclassification	5	0	1	1	3	18
Percentage success (%)	<i>94.19</i>	100	96.67	95.00	98.99	Average: 97.59%

Table 2: Detailed recognition stats for case with “*entire frame removed*” with $kn = 5$

When examining the results from the test with the highest accuracy one can see that the distribution between the different number represented in the test data variates greatly. The character “9” is represented fewest times, while character “0” is represented with the far most occurrences. This makes the overall result weighted more by “0” then the other classes. Since our test data are collected from real specification plates, the distribution of the characters one wish to recognize in a real setting would be weighted in a similar manner. This makes the overall accuracy presented still valid as what to expect, even though accuracy for individual characters being lower. The detailed statistics can be seen in Table 2.

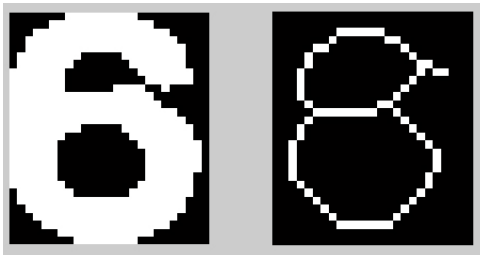


Figure 20: Character with true value “6” that was misclassified by the system.

In figure 20 one can see an example of one of the characters that was misclassified by the system. The true value of the characters is “6”, but the system classified it as a “8”, with the 5-nearest-neighbors being [8 8 8 8 8]. The reason for the misclassification in this case, we believe is the thinning process. The thinning version of the character shows similarity to a “8”.

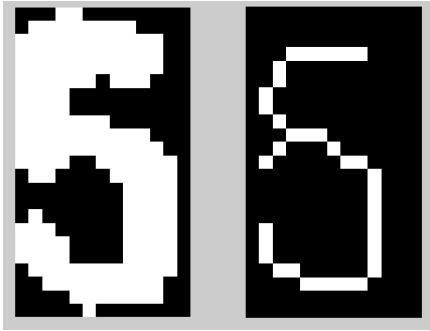


Figure 21: Character with true value “5” that was misclassified by the system.

In figure 21 one can see another example of a misclassified character. The true value of the characters is “5”, but the system classifies it as a “6” with the 5-nearest-neighbor being [5 6 6 6 6]. The k-nearest-neighborhood shows that the training data with the nearest distance was a “5”, which was the true value, but the four next distances was to the character “6” and thus the misclassification.

6.5 System computation performance

All test bench runs were performed in Matlab R2007b on a Intel Core I3, 2.13 GHz processor. The average time for one run of the test bench, 923 characters, was 124 seconds. This equals 0.1343 seconds for each character. A typical specification plate would contain from 20-40 characters, which makes the recognition of one specification plate ranging from 2.5 to 5 seconds. This is using our test bench, which was created without any kind of performance optimization.

6.6 Discussion of test results

When starting the test phase we were unsure about what one could expect using this method. We knew that the *Direction extraction* had been used to obtain 80% accuracy on handwritten characters and we expected that we could expect higher accuracy since our problem was less complex than an OCR on handwritten characters. The final result was 98.04% which we look at as a good result. In [7] the author tested many of the leading OCR software available, against historical newspapers, and concludes that a OCR accuracy of 98-99% was to be considered as a good result. This is however a setting with larger quantity of characters than we will work with.

One important factor to note is that we got a very wide range of different specification plate classes. The complexity in these classes also vary a lot. For us this means that for some classes we might be able to obtain close to perfect recognition rate while we in other cases might manage lower recognition rate. In figure 22 one can see an examples of a specification plate with good quality and specification plate with bad quality.

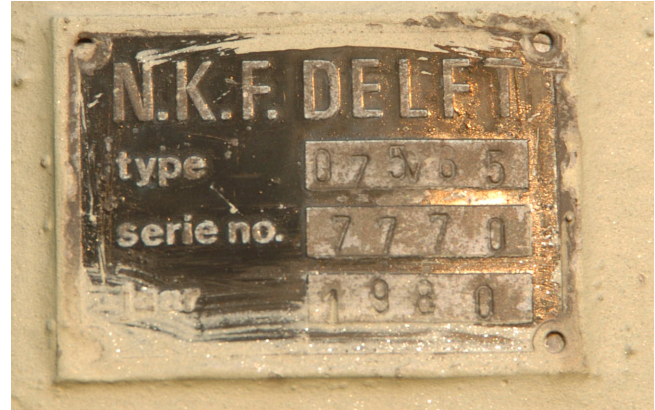
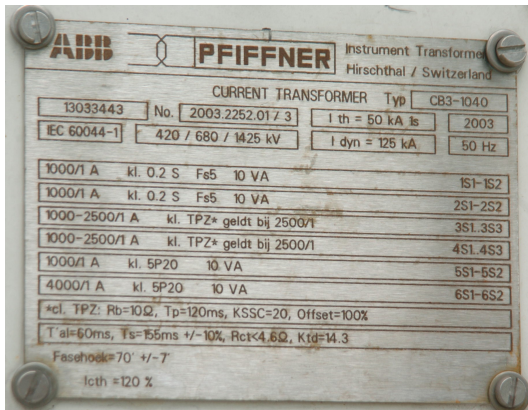


Figure 22: Examples of specification plates with good and bad quality.

In our test bench we used the same training data to classify test data from multiple different classes. In a future application it might be possible to create individual training data to individual classes.

7 Conclusion and further work

7.1 Conclusion

In this paper we have created an Optical Character Recognition(OCR) system to be applied on electrical specification plates. This project proposal was given by Verico AS as a step in streamlining their process of data collection using photo documentation.

The system is customized to a given setting and is to be implemented together with a system that warps the source images and selects the data fields of interest. Four major steps perform all processes regarding the OCR system. The first two steps are based in image processing; background segmentation and characters segmentation. The background segmentation was performed using Otsu's method for thresholding, which delivered overall good results. Character segmentation was performed using modification of vertical histograms. This method gave good results in those cases where splitting was needed.

The core of this paper, and in our system, is the feature extraction process. We used a feature extraction method called *direction extraction*, this method was originally presented as a method for recognizing handwritten characters, and the best result found in literature using this method was 80%, so when we started our work it was uncertain what we could expect from this method.

After testing we archived an accuracy of 98.04%, which is a good result based on our expectation for the method chosen, but we cannot conclude that the accuracy is good enough to be used in a real setting.

Classification was performed using *k-nearest-neighbor* method. By using this method we found that a *k* value of 5, gave the best results. The processing time used to classify one single characters was *0.1343* second in our setting, which we find acceptable.

7.2 Future work

- Field detection - At this moment the system is depending on templates to know where the data fields of interest are. If we managed to find a method of auto-detecting the fields of interest this would be a major improvement of the system.
- Neural Network - As described the system uses *k-nearest-neighbor* to classify the characters. We do not know if the recognition accuracy could be increased using Neural Network, but it would be interesting to test. This would at least decrease the processing time during classification, but would increase the training time.
- Recognition classes - In this paper our system recognizes numbers. In the further work we could train the system against letters, special symbols and other characters we might need to recognize. Since the framework of the system exists, one should be able to train the system against new characters without major problems.
- *A-priori* information - The way we have designed the system we do not use *a-priori* information. By including this one might be able to obtain a much higher recognition rate than we have managed in this paper. One type of *a-priori* information would be to weight the result according to what number is expected. An example might be

when reading a component voltage, if one know the lowest and highest voltage this component can work with, we can exclude all numbers beyond this range.

References

- [1] ABBYY. Abbyy fine reader user guide, training user patterns. <http://finereader.helpmax.net/en/advanced-features/recognition-with-training/training-user-patterns/>.
- [2] M. Blumenstein, B. Verma, and H. Basli. Blumstein 2003, a novel feature extraction technique for the recognition of segmented handwritten characters. *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*, pages 137–141 vol.1, 2003.
- [3] R. G. Casey and E. Lecolinet. A survey of methods and strategies in character segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 18(7):690–706, 1996.
- [4] Mike Constant. An introduction to anpr. *CCTV Today*, http://www.cctv-information.co.uk/i/An_Introduction_to_ANPR.
- [5] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2 edition, November 2001.
- [6] Rafael C. Gonzalez, Richard E. Woods, and Steven L. Eddins. *Digital Image Processing Using MATLAB, 2nd ed.* Gatesmark Publishing, 2nd edition.
- [7] Rose Holley. How good can it get? analysing and improving ocr accuracy in large scale historic newspaper digitisation programs. *D-Lib Magazine*, 15 Number 3/4, 2009.
- [8] G. S. Lehal and Chandan Singh. Feature extraction and classification for ocr of gurmukhi script. *Vivek*, 12:Pages: 2–12, 1999.
- [9] MATLAB. *Documentation, version 7.10.0 (R2007a)*. The MathWorks Inc., Natick, Massachusetts, 2007.
- [10] Mark Milian. Cnn tech - new iphone app translates foreign-language signs. http://articles.cnn.com/2010-12-20/tech/word.lens.iphone.app_1_iphone-app-android-foreign-language.
- [11] K. M. Mohiuddin and Jianchang Mao. Wiley encyclopedia of electrical and electronics engineering - optical character recognition, 1999.
- [12] C.Y.; Yamamoto K.; Ricoh Co. Ltd. Yokohama Mori, S.; Suen. Historical review of ocr research and development. *Proceedings of the IEEE*, 80 Issue:7:1029 – 1058, 1992.
- [13] Communications Nuance. Omnipage user guide - training. http://www.nuance.com/imaging/resources/userGuides/OPUserguide/chapter5/ch5_7.asp.
- [14] PCMAG.com. The best ocr solutions, test 08/2008. <http://www.pcmag.com/article2/0,2817,2327834,00.asp>.