Universität Passau
Fakultät für Informatik und Mathematik

Bachelor's thesis

Optical Character Recognition on supermarket receipts

Marco Ziegaus

July 19, 2016

Bachelorarbeit
am Lehrstuhl für Mathematik mit Schwerpunkt Digitale Bildverarbeitung
der Fakultät für Informatik und Mathematik
der Universität Passau

Erstgutachter: Prof. Dr. Tomas Sauer

Abstract

The objective of this thesis was the development of a proof of concept system that can recognize text on supermarket receipts and extract the information printed on the receipts. This information can be used to offer consumer orientied applications, for example for automatic budgeting and categorizing everyday expenses. Various challenges to OCR that are specific to receipts and do not occur on ordinary documents are identified, discussed and solutions are proposed. These challenges arise from the low printing quality of receipts, e.g. faded printing, small printing and joint characters and lines. The solutions include techniques for all steps of an typical OCR system and have been implemented for an evaluation. Both the pure character recognition as an isolated step and the whole system from preprocessing to data extraction have been evaluated with promising results. The pure character recognition, despite using a relatively simple template matching technique, reaches a recogniton rate of 97,36 %, the data extraction rate is 76,37 % due to subsequent errors from the character recognition. Several improvements to the system are possible, the most promising of them in the areas of intelligent autocorrection and data extraction.

Kurzzusammenfassung

Ziel dieser Arbeit war es, den Prototypen eines Systems zu entwickeln, welches Text auf Supermarkt-Kassenzetteln erkennen und die abgedruckten Informationen extrahieren kann. Diese Informationen können dazu verwendet werden, Endkundenapplikationen für den privaten Gebrauch zu entwerfen, zum Beispiel für das automatische Führen eines Haushaltsbuches und die Kategorisierung alltäglicher Ausgaben. Verschiedene Herausforderungen, die spezifisch für Kassenzettel sind und bei regulären Dokumenten nicht auftreten, werden identifiziert, beschrieben und Lösungen zu diesen Problemen werden vorgestellt. Diese Herausforderungen entstehen durch die geringe Druckqualität von Kassenzetteln, wie z.B. verblasste Schrift, kleiner Druck und miteinander verbundene Buchstaben und Zeilen. Die vorgestellten Lösungen zu diesen Problemen beinhalten alle Schritte eines typischen OCR-Systems und wurden für eine Evaluierung implementiert. Sowohl die reine Erkennung einzelner Buchstaben als isolierter Schritt als auch das ganze System vom Preprocessing bis hin zur Datenextraktion wurden mit vielversprechenden Ergebnissen evaluiert. Die reine Buchstabenerkennung erreicht trotz eines simplen Template-Matching-Verfahrens eine Erkennungsrate von 97,36 %, der Anteil der korrekt extrahierten Informationen ist aufgrund von Folgefehlern aus der Buchstabenerkennung 76,37 %. Das System bietet die Möglichkeit zu weitereichenden Verbesserungen, insbesondere im Bereich der intelligenten Autokorrektur und der Datenextraktion.

Contents

1 Motivation

Optical Character Recognition (*OCR*) is the technology of automatic reading of printed text, just as humans do. First attempts to recognize text were made in the 19th century with optical and mechanical template matching. In the 1950's, the first OCR machines became commercially available that were based on digital computer technology. Since then, OCR was more and more improved and especially used in industrial and office environment which they have been optimized for, e.g. reading addresses on mail, parcel ids on packaging and assistance for manual data entry. [1] [2] In recent years, OCR has been widely used to digitize books.

All these applications have in common that they are used in a commercial environment. There are few consumer orientied products that utilize OCR technology. This may be due to the fact that the cost of OCR is easier covered in commerical applications. However, consumer orientied OCR applications have quite some potential to improve everyday life and increase the individual productivity.

In this thesis, a proof of concept for such an application is presented. Supermarket receipts are widely used for documentation purposes in everyday life. Most people don't bother to even look at receipts at all, but a growing number uses budgeting apps to track their expenses and incomes. The data from supermarket receipts can be used to automate and refine the process of tracking expenses. As the data is only available in printed format, users would have to manually type in the values from the receipt. This is where OCR can be used to offer a fast and easy to use solution to automatically budget small expenses with high accuracy.

2 Introduction

In this thesis, first a short introduction to OCR in general is given before taking deeper look on the challenges that recognizing text on supermarket receipts has. Subsequently the details of the proposed solutions to these challenges and their implementation are discussed. Finally, the evaluation and proposed improvements will be considered.

2.1 Limitations and assumptions

In this thesis, only receipt images from a flatbed scanner (DR-7080C) are used and processed during development and evaluation. These scans are more straight and do not suffer from distortions as camera pictures would do.

Furthermore, only receipt images from a single store with cash payments are used. This simplifies the process due to the consistent font and layout structure.

2.2 Introduction to OCR

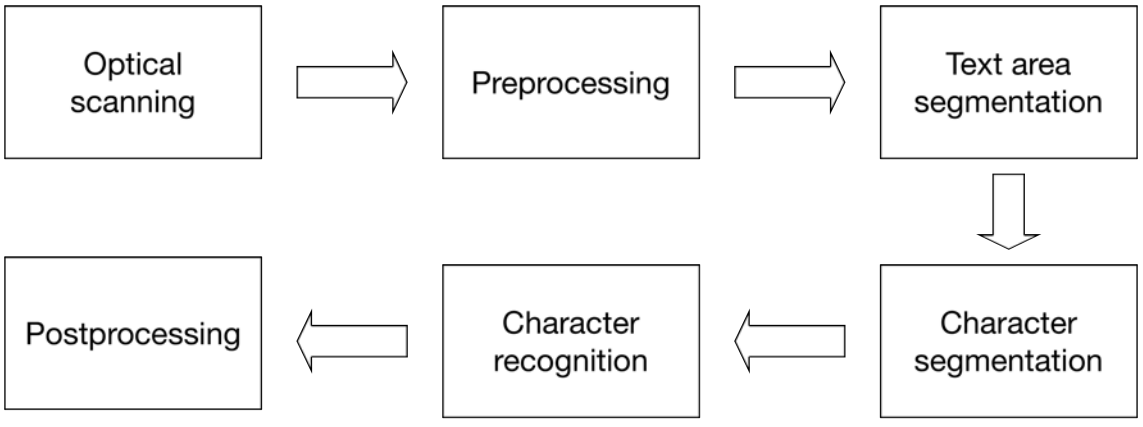

Figure 2.1: A typical workflow of an OCR system

OCR systems usually consist of several steps to be performed as depicted in figure 2.1. After scanning a piece of typed or handwritten text with a photo camera or an document scanner, the resulting digital image is preprocessed to enhance the input quality. This often includes binarization (see figure 2.2) of the image and filtering artifacts from poor printing or scanning of the document.

Figure 2.2: The original scanned image (grayscale) and the binarized counterpart

During the text area segmentation, areas with figures and blank space are separated from areas of actual text. After that, the individual characters are separated from each other. This task is crucial for the actual character recognition in the next step as usually only one character can be reconized at a time. The character recognition itself can be performed by a simple template-based approach or via an feature-based method. In the template-based approach, the difference between the optically scanned character image and a set of template images is calculated. Such a template image is depicted in figure 2.3 for an example. Usually, the character with the smallest difference is considered as the best match. For more details about a template-based approach, see sections 5.4 and 5.4.3. For a feature-based approach, a feature set is extracted from the scanned character. Possible features may include the ratio of black to white pixels or the number of times the black pixels are crossed by predefined vectors. After recognizing the individual characters, they need to compiled to a coherent text. Postprocessing can include correcting errors by using context information, dictionaries and frequency statistics as well as extracting information from the recgonized text.

Figure 2.3: A single extracted character and the associated template image (see section 5.4.2 for details). The extracted character is the first occurrence of an lowercase “a” in the receipt from figure 2.2 (in “Nibelungenplatz”)

3 Challenges of supermarket receipts

Performing OCR on supermarket receipts has specific challenges that must be met. As receipts are designed to be produced with lowest possible cost, most of these challenges result from poor printing quality.

3.1 Printing quality

Printing quality is a major issue when recognizing text on supermarket receipts. Usually thermal printing is used to keep printing and maintenance costs low. However, thermal printing often causes artifacts such as missing pixels, white flow marks or dark spots.

3.1.1 Missing pixels

In many cases, characters of the same type share only some parts of their shape because individual pixels or even blocks of pixels are missing due to low printing quality. This makes recognizing them correctly more difficult than on regular documents. In figure 3.1 the first six occurrences of an lowercase "a" from a receipt and the first six occurences from an ordinary document (printed by an regular laser printer) are shown. While the characters from the ordinary document are rather similar to each other, the letters from the receipt look quite different. As a matter of fact, the receipt images from figure 3.1 have 63 % of their pixels in common while the document images have 86.5 % of their pixels in common. This similarity of the images has been calculated by counting the number of pixels that are the same (black or white) on all 6 images of the receipt respectively the document. This includes both white and black pixels.

Figure 3.1: Left: Six lowercase "a" characters from the receipt in figure 2.2
Right: Six lowercase "a" characters from an ordinary printed document

3.1.2 White flow marks

White flow marks as depicted in figure 3.2 can cause character shapes to be disconnected. This results not only in large missing parts of the character but also makes it hard to distinguish characters in a character sequence. For example, the character "n" from figure 3.2 may also be two characters, an "r" and an "i".

These disconnected characters are especially a concern during the character segmentation. Many character segmentation strategies rely on a character's parts to be connected, or at least be overlapping horizontally. These characteristics are often destroyed by flow marks that are produced by receipt printers.

Figure 3.2: A receipt with white vertical flow marks (visible on the top logo) and characters extracted from this receipt

3.1.3 Dark spots

Due to dirt on the receipt paper and low quality of thermal printing dark spots are found within normally white areas, as depicted in figure 3.3. These spots will be extracted during character segmentation and be detected as characters that look alike, e.g. dots or asterisks.

Figure 3.3: A segment of a receipt with dark spots on blank areas (grayscale scan).

3.1.4 Faded printing

Thermal printing products tend to fade with time, especially when exposed to light or heat. This leads to pale characters with thin lines. Often these characters are hard to recognize even for humans when looked at without context (see figure 3.4).

Figure 3.4: Faded characters: “E”, “H”, “R”, “b”, “M”, “n”

Faded characters, even though they might not be as faded as in figure 3.4, cause difficulties during character recognition since many of the pixels that are dark on the template are white on the actual character. In figure 3.5 such an example is shown.

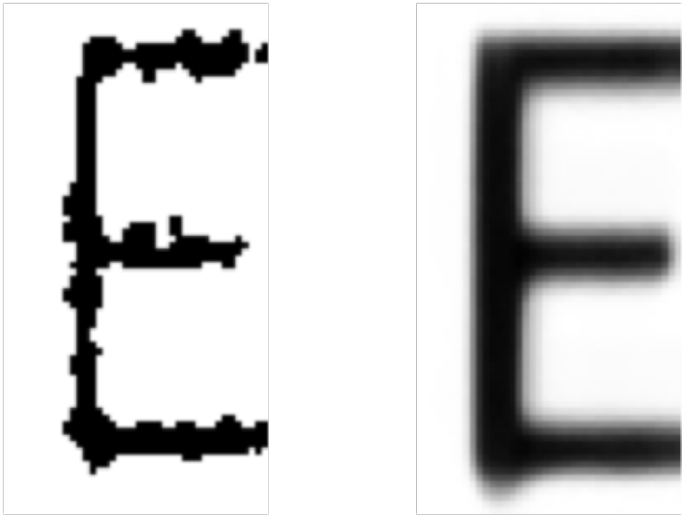

Figure 3.5: A faded uppercase “E” and the associated template image.

3.2 Small printing

Receipts are usually printed with small font and little line and character spacing. For bold fonts, this results in characters being connected to each other, as illustrated in figure 3.6. Connected letters are difficult to process during character segmentation. Often a sharp separating line cannot be determined as both characters merge on their edges. This leaves excessive black pixels on the extracted characters which decrease the recognition accuracy.

Figure 3.6: Small printing, little character spacing and bold font leads to horizontally connected characters. In this example, the letters B and A are connected in the word “BAR”.

In some cases, characters are connected or overlapping vertically due to small line spacing. This increases the difficulty to separate two lines from each other, which can effect the character recognition for all letters on both lines.

4 Advantages of supermarket receipts

Apart from having challenges, performing OCR on receipts has a couple of advantages over ordinary documents, too. They result mostly from the simplicity of the printing procedure.

4.1 Monospaced font

Receipts usually have a monospaced font. That means that each character will take up the same horizontal space regardless of their actual width. Thin characters, such as “i”, “l” or “1” have a larger padding on the left and right than wide characters like a “T”.

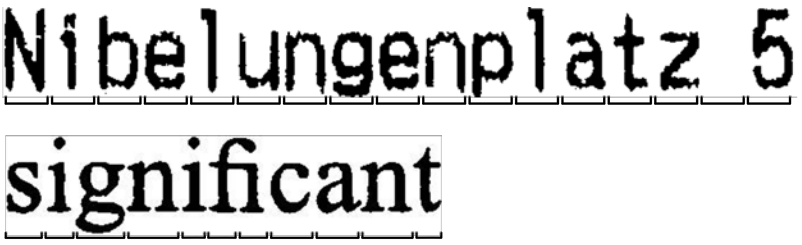

Figure 4.1: A short sequence from a receipt in monospaced font and from an ordinary document in proportional font.

A monospaced typeface is a great advantage for character segmentation. Given a single line of printed text, the individual characters can be separated from each other by slicing the line in constant intervals with the width of one character. Details on this technique can be found in section 5.3.1.

4.2 Machine readable layout

Receipts are usually structured in a table-like layout which makes it easy to interpret the chracters written on them. Technically, this is not a part of OCR; nevertheless it can be used to intelligently correct the output from optical character recognition.

For example, the prices are always printed on the right side of an receipt. This means that common errors, such as mistakenly recognizing a zero (“0”) as an uppercase

"O" or a one ("1") as an lowercase "l", can easily be corrected. If any uppercase "O" appears on the right side of receipt, it can be changed to a zero during postprocessing.

Besides, a tabular layout makes it easy to extract the desired information, such as the bought products and their quantity and prices, the date and time or the total sum, as explained in section 5.6.

5 Proposed solutions & implementation

To provide a showcase and proof of concept for the proposed solutions to the challenges described in chapter 3, an implementation of an OCR system for supermarket receipts has been developed in the course of this thesis, called *OCRAM*[1]. After a short review of existing open source implementations of OCR sytems, especially *Java OCR* [3] it was clear that regular OCR systems are developed and optimized for other purposes[2] than recognizing text on supermarket receipts with their special challenges and advantages shown in chapters 3 and 4. For example, running *Java OCR* on supermarket receipts resulted in inaccurate line and character segmentation which often leads to several lines and characters recognized as one, as depicted in figure 5.1.

Figure 5.1: Top: Two lines are segmented as one by *Java OCR*.
Bottom: Four characters, as they have been segmented by *Java OCR*.

As stated in chapter 1 of this thesis, the implementation is supposed to be a proof-of-concept for a consumer oriented application. As scanning a supermarket receipt on a stationary computer or laptop would be too laborious for a end user, the application targets mobile devices. Users could easily take a picture of the receipt with their smartphone and have the contents be recognized automatically.

Android [3] is currently the world's most used operating system for smartphones[4]. Its apps run in a Java Virtual Machine and are developed in Java. The proposed implementation is written in Java to keep the possibility open to extend it to consumer applications. Moreover, Java is a well known and widely used programming language[5] among software developers. Therefore *OCRAM* can easily be adapted, forked and maintained for future development.

[1] OCRAM has been choosen as a name as it is the reversed first name of the author of this thesis.
[2] For example digitalization of books.
[3] `http://www.android.com`
[4] `http://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems`
[5] See e.g. TIOBE index: `http://www.tiobe.com/tiobe_index`

This chapter is structured in the same way the processing of a receipt image is performed. Each section describes one step in the system and relies on the output of the previous step respectively section.

5.1 Preprocessing

Before text can be recognized on any kind of printed document, it usually needs to be prepared in some kind of way. This preprocessing usually includes binarisation and often contains further steps such as applying filters. During development of *OCRAM* it was found that apart from binarisation it was necessary to remove dark areas on the edges of the scanned image which were converted to a black block during binarisation.

5.1.1 Binarisation

After scanning, the receipt is available as a color or grayscale image. To recognize characters however, each pixel needs to be identified as a black or a white pixel. To simplify further processing, the scanned image is converted to a binary image with black and white pixels only. This accelerates the process, as the discrimination between a black and a white pixel has to be done only once and not several times during different steps of the OCR system (e.g. character segmentation, character recognition, etc.).

The basic idea of binarisation is to decide for each pixel of an image whether it should be treated as white or black. This can be achieved by a fixed, hardcoded threshold or a dynamically calculated threshold using Otsu's method. *OCRAM* supports both methods, which will be introduced in the following sections.

5.1.1.1 Fixed threshold

A fixed threshold is simple to implement but will suffer from calibration issues. Given a grayscale image with gray values ranging from 0 (black) to 255 (white), a fixed threshold means all pixels that have a grayscale value below or equal to the threshold will be considered black while all pixels with a grayscale value greater that the threshold will be considered white. The challenge is to find a threshold that detects all black pixels of characters as black while leaving blank areas white.

In figure 5.6 the core of the problem is depicted. A coherent choice for a fixed threshold would be 127, as it would set the threshold in the middle of the gray scale. However, a threshold as low as 127 leads to thin printing in the resulting binarized pixels as the pixels on the edge of single characters are usually lighter that the core pixels. When using a higher threshold like 200, the character's shapes are correctly binarized, but black spots are scattered among the characters and blank areas.

Figure 5.2: A typical receipt image. If not stated otherwise, all examples and figures in this chapter were generated from this receipt.

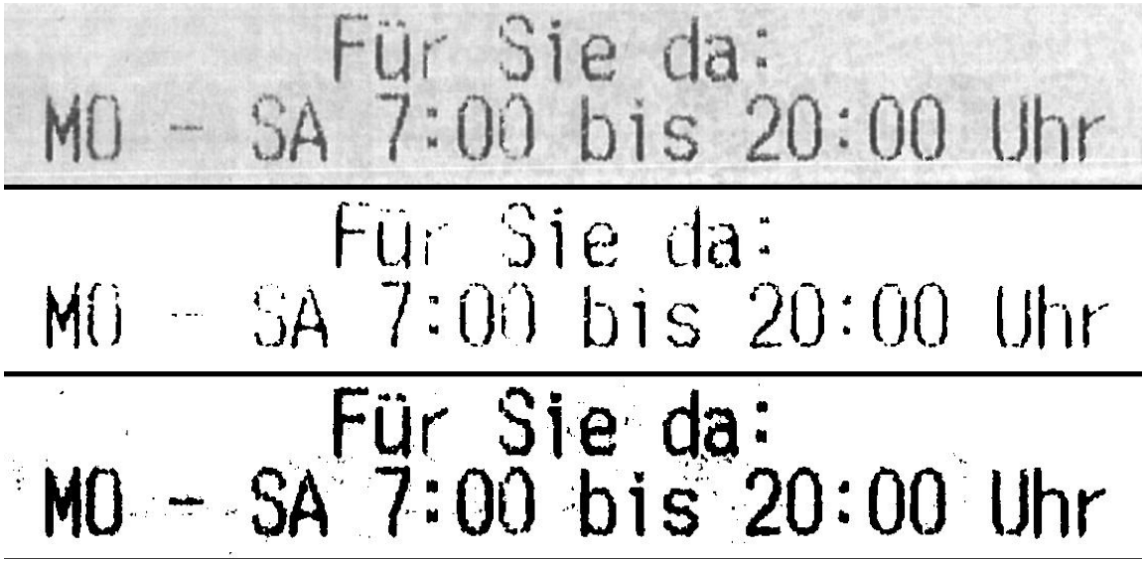

Figure 5.3: A piece of a receipt in grayscale (top), binarized with a threshold of 127 (center) and 200 (bottom). The first threshold leaves black pixels from the script white while the second causes black spots among the characters.

The challenge of finding a good threshold gets even more difficult when processing different receipts which can have different levels of grayscale values for the script and white areas. When using different scanners or cameras with various brightness environment, no predefined threshold will be sufficient for all images. For this reason, a dynamic threshold is required that is calculated for each image.

5.1.1.2 Dynamic threshold with Otsu's method

A dynamic treshold which is caluclated separately for each image can be used when a fixed threshold is not applicable due to a high variance in the brightness of the image set. For the implementation of *OCRAM*, a method proposed by Nobuyuki Otsu in 1979 [4] [5] is used. It relies on optimizing for a minimum variance within the classes (here: black and white pixels) and a maximum variance between the classes.

To achieve this, a histogram of the grayscale values of the document's pixels as in figure 5.4 is generated. Then, a threshold t is sought-after that maximizes the variance between the pixels below and above this threshold.

When comparing figure 5.4 and figure 5.5, it can be seen that receipts are more "fuzzy" in their gray scale values. In figure 5.5, the rising edge for the majority of white (or very light) pixels is steeper and starts on higher (brighter) levels. This means that, where pixels can be separated into black and white quite sharply on ordinary documents, they are rather gray-in-gray for supermarket receipts.

Therefore, determining a good threshold is even more important. Otherwise the next steps (line segmentation, character segmentation and character detection) will be more prone to errors.

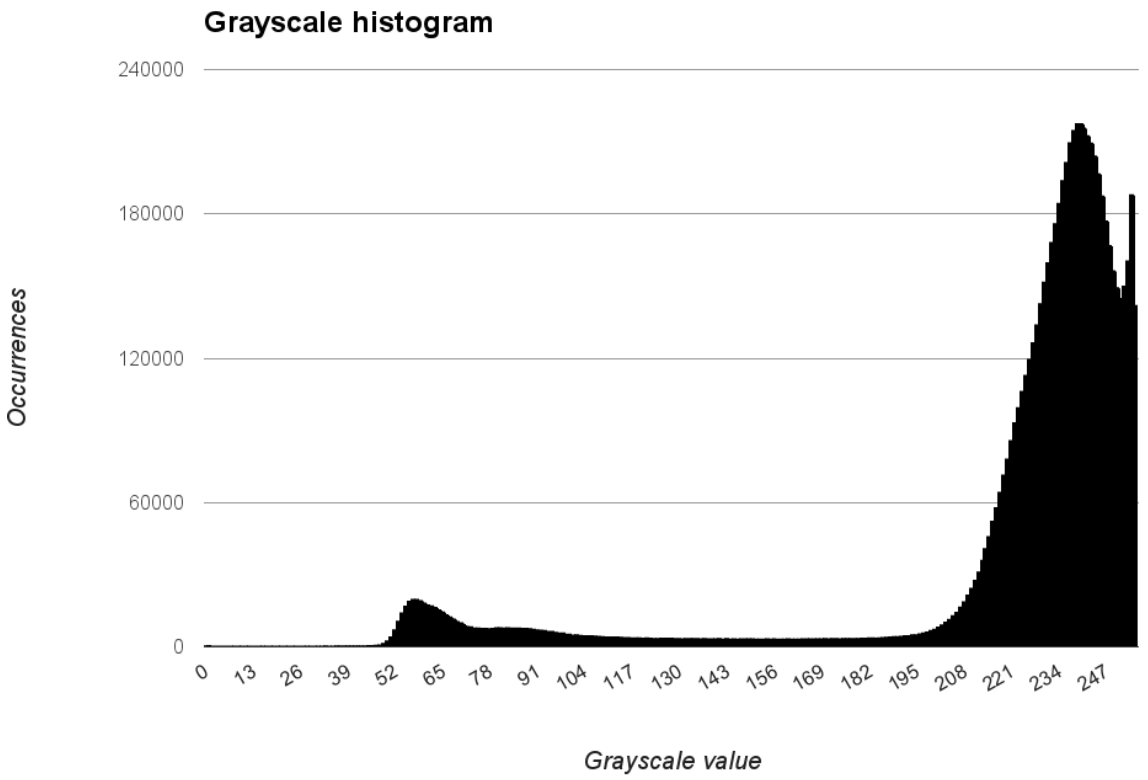

Figure 5.4: Histogram of a receipt's pixels' grayscale values.

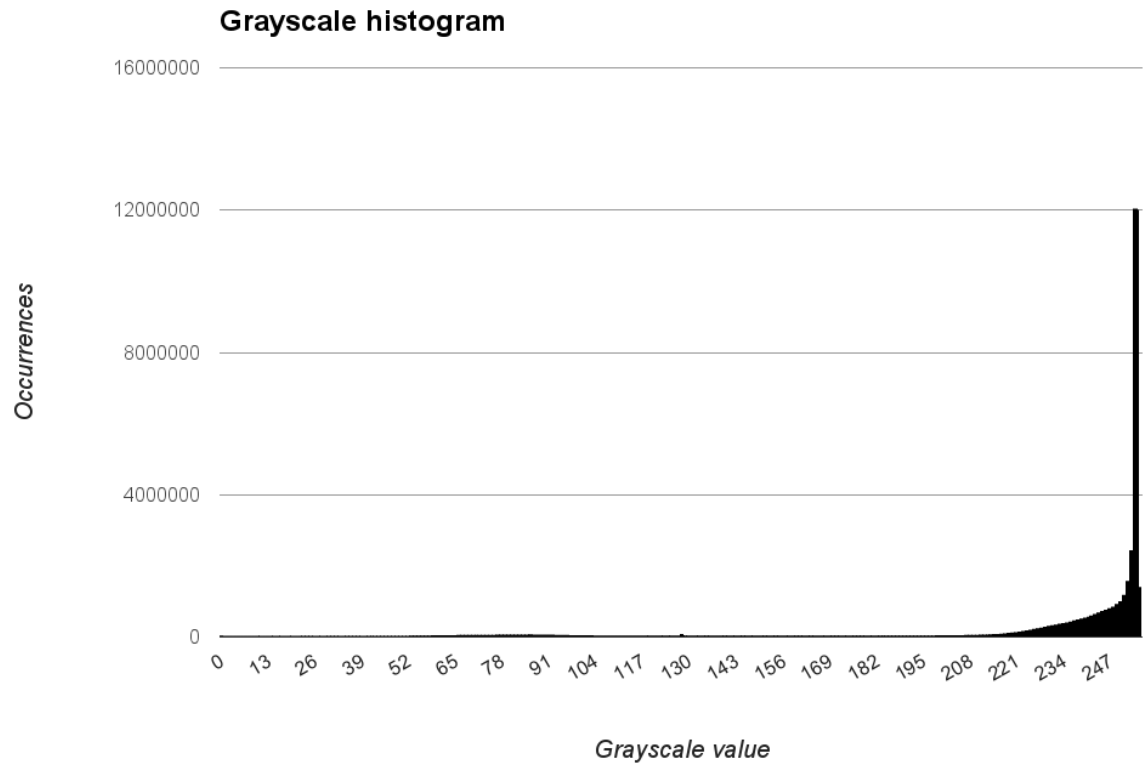

Figure 5.5: Histogram of an ordinary document's pixels' grayscale values.

Figure 5.6: A piece of a receipt in grayscale (top), binarized with a threshold of 127 (center-top), 200 (center-bottom) and 161 (bottom). The threshold of 161 has been calculated by using Otsu's method.

5.1.1.3 Colored images

For colored images, the binarisation is similar to grayscale images. However, instead of reading the luminosity of a pixel directly off the grayscale value, it needs to be calculated based on the color channels. This can be done by weighting all color channels equally or by weighting them differently to simulate human perception or extract particular image features.

5.1.1.4 Further improvements

Further improvements for binarisation include using better thresholding techniques [5] and using local thresholds instead of a global threshold. Different regions on a document may have different distributions of background brightness. This applies especially to thermal printed documents, such as supermarket receipts. Different thresholds could be calculated and applied separately for these regions, for example by extracting the characters first and then calculating a threshold for each character.

In addition a second run of binarisation could be performed. In this second run, all white pixels in a perimeter of e.g. 2 pixel around an black pixel are scanned. If such a pixel is sufficiently dark, it can be considered black, too. This way, the character's shapes are not that faded anymore while keeping dark spots in white areas black.

5.1.2 Black edges

As shown in figure 5.7, scanning can cause dark edges which are converted to black edges during binarization. This leads to problems during line and character segmentation. To avoid these problems, the edges of the input images are overwritten with white pixels. This can be done safely because the receipts are printed with margins.

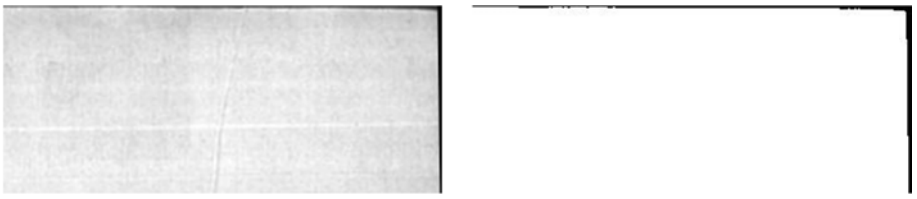

Figure 5.7: Dark/black edges resulting from a scan (left: original grayscale, right: binarized).

5.2 Line segmentation

In *OCRAM*, the workflow differs from the workflow shown in section 2.2 and figure 2.1. As receipts typically do not contain any graphics apart from the market's logo, text area segmentation is skipped. Instead, the step of character segmentation is divided into line segmentation and character segmentation.

During line segmentation, a given document is divided into several lines. The result is displayed in figure 5.8. The separated lines can then be extracted and saved as image files or are further processed for character segmentation.

Figure 5.8: A part of a binarized receipt and the individual, separated lines of text.

5.2.1 White row strategy

A simple strategy to detect lines in a printed image is separating them by white pixel rows. A pixel row is the set of all pixels with the same y-coordinate. If and only if such a row consists only of white pixels it is viewed as whitespace and therefore separates two lines. This approach is utilized by *Java OCR*.

Unfortunately, it shows poor performance when two text lines are connected in one or more places, as depicted in figure 5.9. In this case, no all-white pixel row can be found that separates the lines and they will be recognized as one line. As explained

Figure 5.9: The “g” and the “2” are connected via one black pixel. This results in both lines being detected as one when using the white row strategy.

in section 3.2, receipts often have small line spacing. Therefore, connected lines are a major issue.

In addition, this technique would face problems when confronted with a line that consists solely of x-height characters (such as a, c, e, m, . . .) and characters with a dot above (such as i, j, ä, ö, ü, . . .). In this case, the dots would be cropped out of the line due to the whitespace between the character and the dot.

On account of the issues mentioned above, *OCRAM* does not implement this strategy. There are better choices, which will be introduced in the following sections.

5.2.2 Relative pixel count strategy

Other than the white row stratgey, the relative pixel count strategy relies not on completley white pixel rows, but on predominantly white pixel rows. A fraction parameter *tolerancePortion* can be specified that will act as a threshold to treat a pixel row as mostly white (and therefore whitespace between text lines) or as part of a text line. That is, if the portion of black pixels in a pixel row is below the specified threshold *tolerancePortion*, this pixel row will be considered white.

All pixel rows between two predominatly white rows will be combined to one text line if the resulting line's height is at least 90 % of the typical x-height in the given font (specified via a paramter). For this reason, the horizontal dividers in figure 5.8 between "SUMME" and "Geg. BAR" are not recognized as lines.

In contrast to the white row strategy, this strategy would correctly separate the two text lines in figure 5.9. However, the paramter *tolerancePortion* has to be choosen correctly. If it is too low, lines as in figure 5.9 will not be handled correctly. If *tolerancePortion* is too high, lines might be cropped at the top or the bottom, e.g. when a line consists of mostly x-height characters (such as a, c, e, m, ...) and a few ascender characters (such as b, d, f, h, k, ...). These ascenders would represent a small portion of black pixels on the top of the line. If this portion is smaller than *tolerancePortion*, the characters would only be recognized at their x-height and the ascenders would be cropped.

Furthermore, the appropiate *tolerancePortion* depends on the typical number of black pixels that are contained in a pixel row within a text line. This number increases for bold font lines and for lines with many characters while it decreases for faded script and for lines with few characters, e.g. the line with the text "EUR" in figure 5.10.

Figure 5.10: The line "EUR" has less black pixels per row than the other lines. This can cause problems for an inaccurate threshold in relative pixel count strategy.

5.2.3 Median pixel count strategy

Similar to relative pixel count strategy, this relies on a threshold to determine whether a pixel row is predominantly white or black. But in contrast, this threshold is not configured by a *tolerancePortion* parameter, but it is calculated based on the overall black pixel density on the current receipt.

For this, the number of black pixels is counted for each pixel row. Subsequently, the median is calculated. This median m, together with a factor f, $0 < f \leq 1$ will serve as the threshold t:

$$t = f \cdot m$$

In contrast to the relative pixel count strategy, this respects different font weights which can occur due to faded printing. However, again a factor f needs to be choosen to determine the threshold. Furthermore, it has similar flaws in matters of lines with few characters.

5.2.4 Pixel count gain strategy

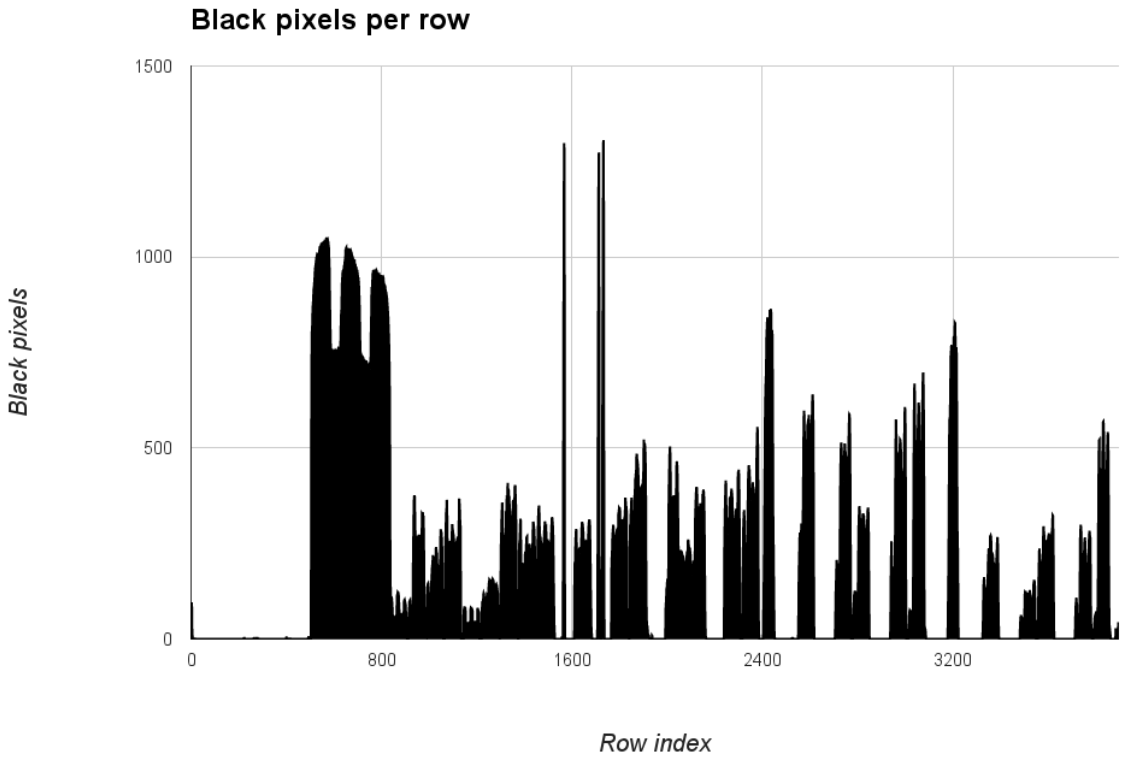

Figure 5.11: The distribution of black pixels per row. The lines of text are easily recognizable. E.g., the thin heights around row index 1600 are the line separators between the lines “FRUCHTQUARK”, “SUMME” and “Geg. BAR” in figure 5.2.

Other than the previous techniques, the pixel count gain strategy does not rely on single pixel rows. Instead, it first calculates the number of black pixels for each row (cf. figure 5.11) and detects rising and falling edges.

If the cumulative gain of an edge exceeds a parameterized minimum, this edge is used as the begin or end of a text line. For rising edges, the start of the edge is considered the begin of the text line. For falling edges, the end of the edge is used as the end of the text line. For an illustration view figure 5.12.

To distinguish the rising and falling edges that indicate the begin and end of a text line from the edges in between the lines, a parameter $minGain$ is used. Experience has shown that a value of $3 \leq minGain \leq 5$ delievers good results. Edges are only recognized if their cumulative gain exceeds this minimum.

The gain $g(y)$ of a pixel row y in comparison to the previous row is calculated as follows:

$$g(y) = \frac{b(y)}{b(y-1)}$$

with $b(y)$ being the number of black pixels in row y.

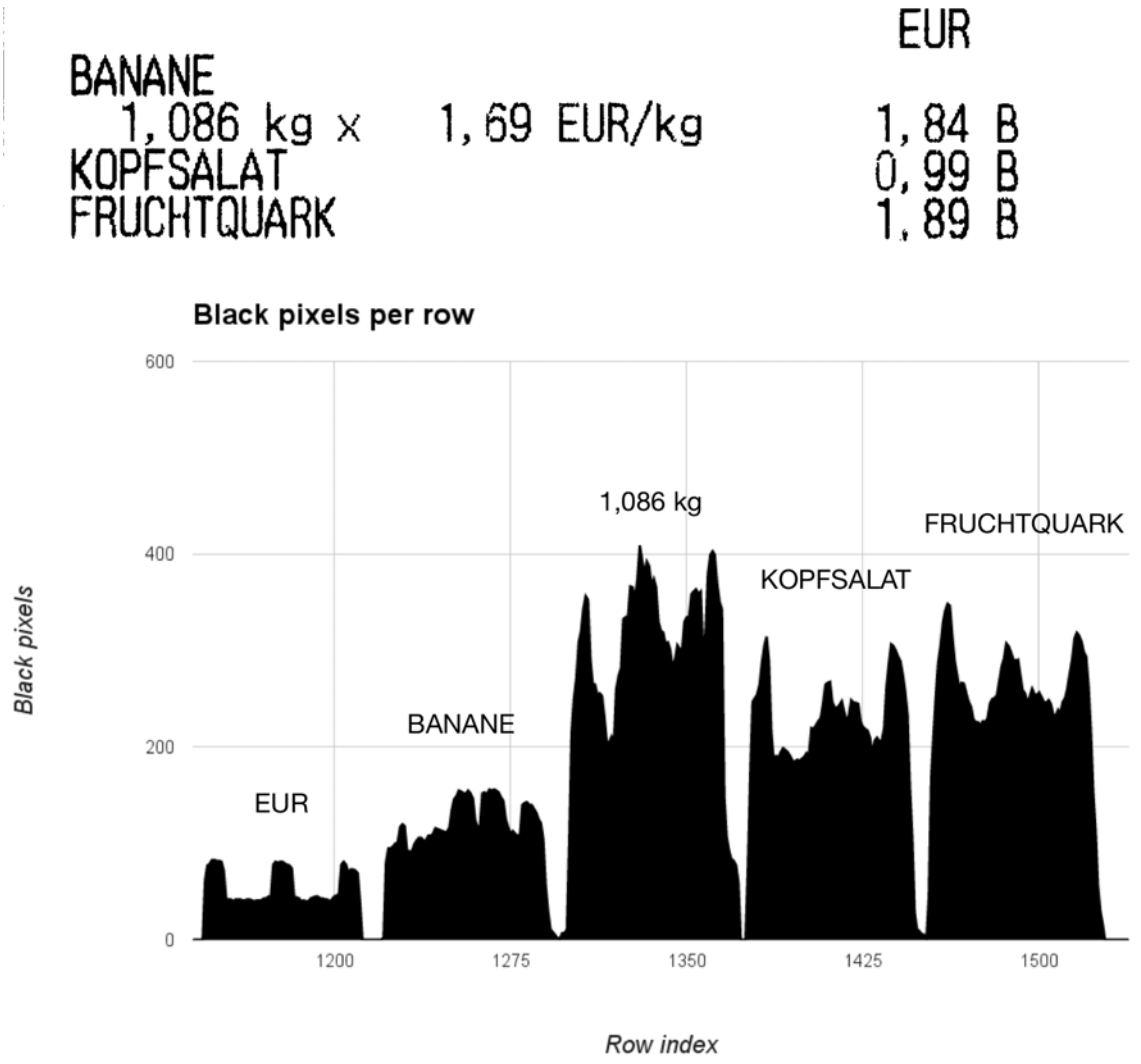

Figure 5.12: A snippet of a receipt and the distribution of black pixels per row. This is bascially a zoom of figures 5.2 and 5.11. The edges on the start and end of lines are easily recognizable.

The cumulative gain g_{cum} of an edge that starts on pixel row y_{start} and ends on row y_{end} is then calculated as follows:

$$\begin{aligned} g_{cum} &= g(y_{start}) \cdot g(y_{start}+1) \cdot g(y_{start}+2) \cdots g(y_{end}-1) \cdot g(y_{end}) \\ &= \prod_{y=y_{start}}^{y_{end}} g(y) \end{aligned}$$

Practice has shown that the pixel count gain strategy works best without critical flaws. It can reliably handle lines that cross over and does not crop top parts of ascenders or bottom parts of descender characters (characters that extend below the baseline, such as y, g, j or p). Improvements can be made by optimizing *minGain* with a large set of test documents and by applying various filters to enhance the differentiation between edges at start and end of a line and edges that occur within a line.

5.3 Character segmentation

Before text can be recognized, the individual characters have to be separated from each other, as depicted in figure 5.13. As for line segmentation, several techniques can be applied, some of them share the basic idea with their line segmentation counterparts.

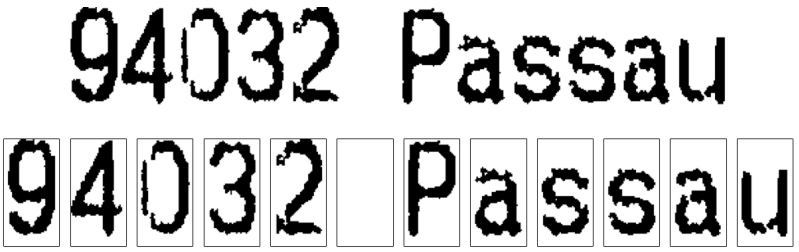

Figure 5.13: A line and the individual extracted characters from this line after character segmentation. The borders around the characters are for visualisation purposes only and are normally omitted.

5.3.1 Monospace strategy

As mentioned in section 4.1, receipts usually have a simple, monospaced font. If the width of a character and the character spacing is known, this can be used by slicing the line in regular intervals.

For the receipts that were used in this thesis (see section 2.1 for details), the font specifications have been investigated manually on 600 dpi scaned images. The exact specifications can be found in table 5.1.

	Plain font	Bold font
Character width	34	37
Character spacing	6	3
Character slot width	**40**	**40**
X-height	47	47
Ascender height	20	20
Descender height	7	7
Line height	**74**	**74**
Cap height	67	67
Line spacing/leading	3	3
Total line height	**77**	**77**

Table 5.1: Font specifications for the examined receipts. All values are in pixels on a 600 dpi scan. Fat values are derived from other values in this table.

The monospace strategy utilizes these font specifications. The general idea is to find the start of the first character in a line (i.e. the leftmost black pixel of a character) and then slice the line into regular parts with a length of $characterSlotWidth$. The result are individual character images as in figure 5.13.

To determine the start of the first character, all pixel columns (cf. pixel rows in section 5.2.1) within the line are analyzed and evaluated based on their portion of black pixels. If a sufficient portion of the pixels in such a column is black, the column is considered a black column, otherwise it is considered a white column. The current implementation uses a 10 % threshold.

However, if the start of the first character is taken as start for the slicing process, the characters are often not horizontally centered and parts of characters are in wrong slices, as depicted in figure 5.14. This behavior induces problems in later steps, especially during character recognition, as equal characters differ from each other due to position shifting and artifacts from neighboring characters.

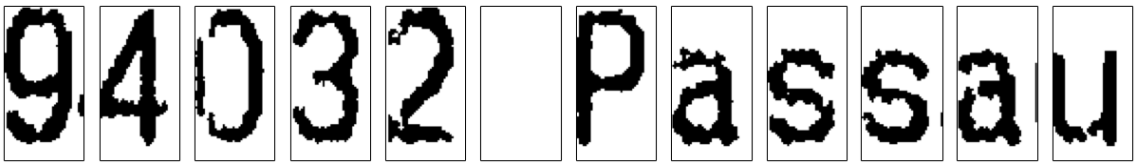

Figure 5.14: A set of characters, sliced in regular intervals, starting at the first character's leftmost pixel. The characters are not horizontally centered and some include parts from other characters, e.g. the image for the character "9" has a piece of the character "4".

To avoid these problems, the character spacing to the left of the first character (left padding) must be considered. This padding can be viewed as half of the spacing between two characters, i.e. 3 pixels on a 600 dpi scan with plain font (cf. table 5.1). Because the exact padding depends on the font, the printing quality of the first letter and other variables, *OCRAM* slices the line in variants with different paddings on the left of the first character and then determines which padding worked best. To do so, the total number of black pixels in the padding areas of each character – the three leftmost and three rightmost pixel columns – is counted for every variant. The variant with the fewest black pixels in the padding areas is then considered the best attempt and its slices are used for further processing. See listing 5.1 for a pseudocode explanation of the process.

5.3.2 Other strategies

Apart from the monospace stratgey, several other strateies have been implemented and evaluated. These represent the vertical counterparts to strategies presented in section 5.2. *Relative pixel count strategy* and *median pixel count strategy* show poor performance due to characters that are crossed over in large parts of their height. For example, the characters "B" and "A" in figure 5.15 are horizontally connected in 18 vertical pixels at the smallest connection. Considering a character height of 64 pixels and a total line height of 77 pixels (cf. table 5.1), this is a portion of 28 % respectively 23 %. On the other hand, the character "U", though being in bold font, only has 5 vertical pixels at the smallest connection. With such circumstances, it is impossible to find a threshold of vertical black pixels that reliably separates two characters. Similar issues arise when using the *pixel count gain strategy*. Because the portion of a connection between two characters can be relatively high while and

```
public List detectCharacters(OCRLine line) {

  List[] sliceVariants = new List[maxLeftPadding];
  int[] blackPixelsInPaddingAreas = new int[maxLeftPadding];

  for(int i = 0; i <= maxLeftPadding; i++) {
    int leftPadding = i;
    List variant = sliceLineWithLeftPadding(line, leftPadding);
    int blackPixels = countBlackPixelsInPaddingAreas(variant);
    sliceVariants[i] = variant;
    blackPixelsInPaddingAreas[i] = blackPixels;
  }

  int bestVariantIndex = findIndexWithLowestValue(
      blackPixelsInPaddingAreas);
  List bestVariant = variants[bestVariantIndex];

  return bestVariant;
}
```

Listing 5.1: Java-style pseudocode to determine the best variant for slicing characters

the portion of a thin horizontal connection inside one character can be low, there is no possibility to distinguish between steep edges between two characters and inside a character. This issue becomes even more critical when considering the flowmarks shown in figure 3.2.

Figure 5.15: The characters “BA” are connected in 18 vertical pixels at the narrowest column. The character “U” is only connected in 5 pixels at the narrowest column.

5.4 Character recognition

5.4.1 Preprocessing

Preprocessing of the entire receipt image has already been discussed in section 5.1. Now, the individual character images are again prepared and enhanced for the character recognition.

At the current state of development, this preprocessing includes only the centering of the character in the image. The center of mass (*COM*) is used as center of the character. The *COM* is calculated by the algorithm displayed in listing 5.2. Once the *COM* has been calculated, the image is shifted so that the *COM* is centered in the image, as shown in figure 5.16.

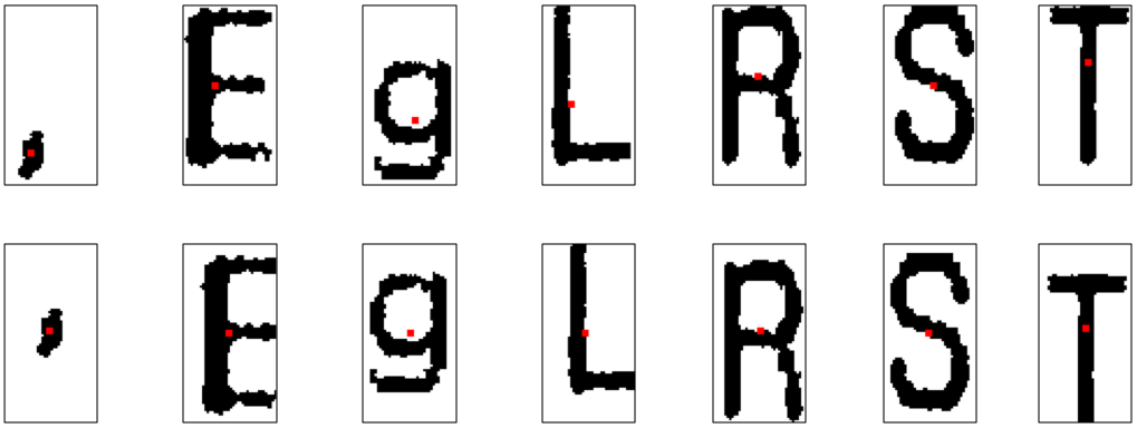

Figure 5.16: **Top:** Original position of the characters.
Bottom: Centered on center of mass.
The red dots indicate position of the center of mass.

This centering increases the robustness against small shifts due to an imperfect line and character segmentation. These small shifts would cause differences between two otherwise similar characters. However, in some cases information is partly lost when centering the character, e.g. for the character "L" in figure 5.16. Furthermore, centering causes small dark spots (which are usually caused by printing defects or dirt on the paper) on the receipt to be centered the same as dots and commas. This sometimes leads to the dark spots being recognized as dots or commas, as both are basically a set of black pixels in the middle of the character image. Despite these drawbacks centering the images has a significant impact on the recognition quality.

Possible improvements during proprocessing are various filters that correct printing defects such as missing pixels or dark spots. First experiments with a *median filter* and *mean filters* showed promising results. Since mean filters blur the images, the characters would also gain resemblance to their templates (see section 5.4.2).

5.4.2 Generation of templates

For each character that may occur within the given text, a template image is needed. While processing the receipts that were used for this thesis, a total of 73 unique characters (cf. table 5.2) were found. For each of these characters, appropiate templates need to be found.

A good template image is one that has a high similarity to all images of the same character, while maintaining a low correlation to all images of different characters. In *OCRAM*, the template for a character is generated by computing the average across all known images of that character (*training images*). As the quality of single character images is rather bad (see section 3.1), no individual image can be used as

```
public static float[] calculateCenterOfBlackMass(IOCRImage image) {
  int sumX = 0;
  int sumY = 0;
  float blackPixels = 0;

  for (int y = 0; y < image.getHeight(); y++) {
      for (int x = 0; x < image.getWidth(); x++) {
          if (image.isPixelBlack(x, y)) {
              sumX = sumX + x;
              sumY = sumY + y;
              blackPixels++;
          }
      }
  }

  float weightX;
  float weightY;

  if (blackPixels == 0) {
      //if there were no black pixels at all, use the middle of the
          image
      weightX = image.getWidth() / 2f;
      weightY = image.getHeight() / 2f;
  } else {
      weightX = sumX / blackPixels;
      weightY = sumY / blackPixels;
  }

  return new float[]{weightX, weightY};
}
```

Listing 5.2: Calculation of the center of mass.

a character prototype. In addition, the exact prototypes (as used by the printer) are not available and *OCRAM* has to deal with different font variations (e.g. plain and bold) Therefore, the average across all *training images* of a character is a reasonable choice and an approximation of the exact character prototype.

Examples for the resulting template images are depicted in figure 5.17. Most template images look like blurred versions of an exact prototype. For some templates, however, the individual images that were combined in that template are still visible due to a small number of *training images*. The percent sign in figure 5.17 is a good example for this. In all receipts that were processed in the context of this thesis, only 6 percent signs occured, while a common character such as an lowercase “e” occured 1467 times (see table 5.3 for details).

5.4.3 Template matching

Matching unknown character images with template images is the core of *OCRAM*. In this step, the unknown image is compared with all templates (in total 73 templates, see table 5.2). For each comparison, the distance respectively the error between image and template is calculated. For each pixel $p_{x,y}$ at position $(x, y) \in P$ in the

	Count	Comment
Lowercase letters	26	including ä, ü, without ö, ß, j, q
Uppercase letters	26	only standard Latin letters
Numberals	10	
Special characters	11	* BLANK ; , - . ! % + ? /
Sum	**73**	

Table 5.2: Characters in the receipts used for this thesis. See table 5.3 for details.

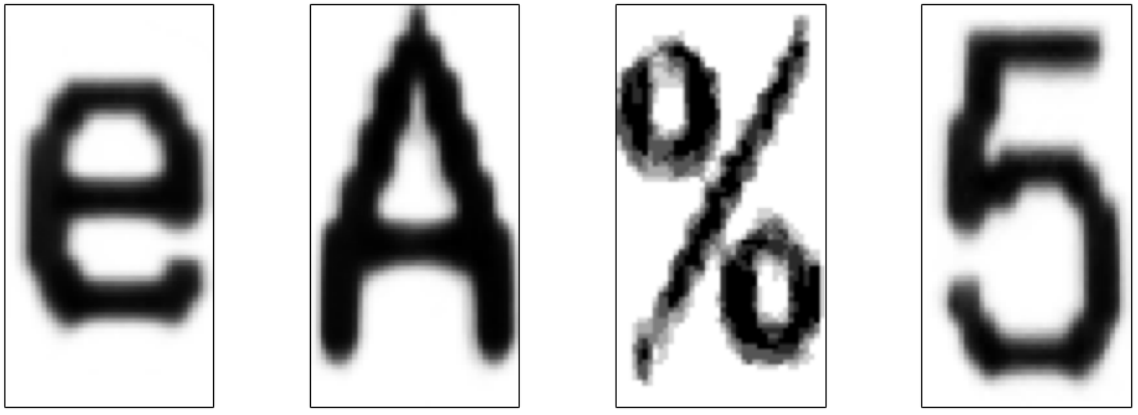

Figure 5.17: Templates for e, A, 5 and %. Due to different frequencies of occurence the templates have varying blur.

set of pixels P, the difference $e(p_{x,y})$ between the grayscale value of the template image $g_t(p_{x,y})$ and the unknown image $g_u(p_{x,y})$ is calculated.

$$e(p_{x,y}) = g_t(p_{x,y}) - g_u(p_{x,y}) \qquad (x,y) \in P$$

Based on these distance values, the *Root Squared Mean Error* (*RMSE*) is calculated.

$$RMSE = \sqrt{\sum^{P} e(p_{x,y})^2}$$

Finally, the *RMSE* is normalized to a value between 0 and 1, which is the *Normalized Root Mean Squared Error* (*NRMSE*), by dividing it by 255, the maximum grayscale value.

$$NRMSE = \frac{RMSE}{255}$$

The *NRMSE* is then used as measure of distance between a template image and an unknown image. A value of 0 means total correlation between both images; basically, the compared images are identical. A value of 1 will be calculated for two images that are exactly inverse, i.e. each black pixel from $image_1$ is black in $image_2$ and vice versa. If one or both of the images has grayscale values other than 0 and 255 (i.e. other than absolute black and absolute white), a *NRMSE* of 1 is not possible.

A	694	a	730	0	890	*	3130
B	557	b	241	1	566	BLANK	11156
C	126	c	181	2	526	:	366
D	127	d	333	3	313	,	597
E	944	e	1467	4	298	-	121
F	201	f	79	5	341	.	450
G	201	g	322	6	190	!	58
H	128	h	283	7	193	%	6
I	193	i	521	8	205	+	3
J	56	j	0	9	459	?	73
K	216	k	539			/	106
L	158	l	248				
M	320	m	161				
N	371	n	717				
O	208	o	185				
P	310	p	89				
Q	11	q	0				
R	691	r	742				
S	423	s	285				
T	203	t	641				
U	436	u	305				
V	12	v	45				
W	135	w	345				
X	1	x	114				
Y	73	y	38				
Z	28	z	49				
Ä	0	ä	31				
Ö	0	ö	0				
Ü	0	ü	108				

Table 5.3: Frequency of occurence in the processed receipts.

After calculating the *NRMSE* between an unknown character image and all available template images, a sorted list of the matches (*MatchList*) is generated. This list contains pairs of a character and the *NRMSE* to the character's template image. As a first attempt, the match with the lowest *NRMSE* will be assigned to the unknown image. In section 5.5, methods to correct errors that are based on this simple choice are presented.

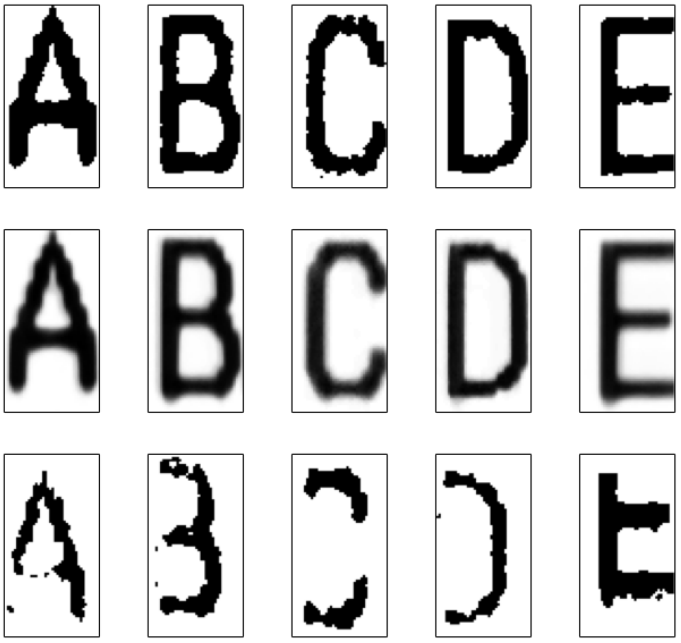

Figure 5.18: Top: Characters with the lowest *NRMSE* when compared to the associated template. **Middle:** Template images. **Bottom:** Characters with the highest *NRMSE* when compared to the associated template. The exact $NRSME$ values can be found in table 5.4.

Character	Minimum *NRMSE*	Maximum *NRMSE*
A	0.1626	0.5021
B	0.1830	0.5899
C	0.1955	0.5048
D	0.1513	0.5892
E	0.1819	0.5316

Table 5.4: Examples for the lowest and highest *NRMSE* values found among all processed receipts for the characters A, B, C, D and E. The values correspond to the images shown in figure 5.18

5.4.4 Reliability prediction

As discussed in the previous section, for each character image a *MatchList* of matches is created and sorted based on the match's *NRMSE*. This *MatchList* can be used to predict the reliability of a match. For this, three reliability classes are introduced:

- HIGH
- MEDIUM
- LOW
- CORRECTED *(used for characters that got reassigned during autocorrection, see section 5.5 for details)*

Each character is assigned to one of the classes HIGH, MEDIUM or LOW after character recognition. The assignment is based on the *NRMSE* difference between the

best match (lowest *NRMSE*) and the second best match (second lowest *NRMSE*). Let A be the *NRMSE* of the best match and B the *NRMSE* of the second best match, then the reliability R is

$$R = B - A$$

The greater R, the more reliable the recognition.

To prodive thresholds for the reliability classes, the mean μ and standard deviation σ across all reliability values have been calculated:

$$\begin{aligned} \mu &= 0,1303696781 \\ \sigma &= 0,05231867861 \\ \sigma^2 &= 0,002737244132 \end{aligned}$$

The reliability class RC is assigned as follows:

$$RC(R) = \begin{cases} HIGH & \text{if } R \geq \mu \\ MEDIUM & \text{if } \mu + \sigma \leq R < \mu \\ LOW & \text{otherwise} \end{cases}$$

This reliability class can be used by applications on a higher level. For example, an application could display the reliability class of a recognized character by providing a green background for characters with the class HIGH, an orange background for characters with the class MEDIUM and a red background for the class LOW. Besides providing the reliability class to higher level applications, the class could also be used to improve autocorrection (cf. section 5.5) by concentrating on characters with low reliability and using context to correct them.

5.4.5 Proposed improvements

5.4.5.1 Improving templates

A possible improvement to enhance the quality of character matching is changing the way templates are generated. Instead of calculating the average across all images depicting a character, the template could be generating by leaving only such pixels black that are most characteristic for that character, i.e. pixels that are black in e.g. 99 % of the images. A related idea is to calculate the average across all images (as described in section 5.4.2) and weight the characteristic pixels higher. Furthermore, pixels that are unique for this character (i.e. pixels that occur only in this character) could be weighted higher. Of course, according to the used technique to generate the templates, the template matching must be adapted.

Apart from different techniques to generate the templates, creating not one but several templates per character can be considered. As images of a character differ among themselves, this could decrease the *NRMSE* between a image and it's associated template. For example, a template for bold uppercase “A” and plain uppercase “A” could be generated rather than averaging over all characters. This would lead to a higher number of templates and therefore result in higher compution time to recognize a character, but would probably increase the recognition quality

5.4.5.2 Filters

Besides that, various filters could be applied to the unknown images to remove white spots within the character strokes and to blur the character's edges, which would result in a version that resembles the template better. Although not evaluated thoroughly, first experiments with *median filters*, *white blot filters*, *binomial mean filters* and *simple mean filter* showed promising results.

Median filter A median filter moves a window across the image and determines the median of the values of this window. This median is then assigned to the pixel in the middle of the window. In an binary image where black is encoded as 0 and white is encoded as 255, such a window with a radius of 1 pixel around the window core (i.e. a 3×3 window) may look like this (the center fixel is bold):

$$\begin{matrix} 0 & 255 & 255 \\ 0 & \mathbf{255} & 255 \\ 0 & 0 & 0 \end{matrix}$$

The values

$$[0, 255, 255, 0, \mathbf{255}, 255, 0, 0, 0]$$

are sorted, which gives

$$[0, 0, 0, 0, \mathbf{0}, 255, 255, 255, 255]$$

The median is the item at the middle of the sorted list. This median (here: 0) is then set as the new value of the pixel in the middle of the windows, resulting in the following matrix:

$$\begin{matrix} 0 & 255 & 255 \\ 0 & \mathbf{255} & 255 \\ 0 & 0 & 0 \end{matrix} \quad \longrightarrow \quad \begin{matrix} 0 & 255 & 255 \\ 0 & \mathbf{0} & 255 \\ 0 & 0 & 0 \end{matrix}$$

As depicted in figure 5.19, this filter removes white spots within the stroke of a character while leaving the edges sharp; only corners are affected negatively by the filter. For a binary image with only two classes of pixel values, the median filter basically sets each pixel to the same value as most of the pixels around it.

Mean filter A mean filter, often referred to as a *moving average*, works similar to the median filter. A window is moved across the image while the mean for each window is calculated. The window's core is then set to that mean. Using the example from the median filter :

$$\begin{matrix} 0 & 255 & 255 \\ 0 & \mathbf{255} & 255 \\ 0 & 0 & 0 \end{matrix}$$

Figure 5.19: A character image (left) and the image after applying a median filter with radius 1 (middle) and radius 2 (right).

The average of the window's values is calculated

$$\frac{0+255+255+0+255+255+0+0+0}{9} = 113.\overline{3} \approx 113$$

and set as the new value of the window's core

$$\begin{array}{ccc} 0 & 255 & 255 \\ 0 & \mathbf{255} & 255 \\ 0 & 0 & 0 \end{array} \longrightarrow \begin{array}{ccc} 0 & 255 & 255 \\ 0 & \mathbf{113} & 255 \\ 0 & 0 & 0 \end{array}$$

Applying a simple mean filter grays out white spots in a character's stroke, but it also blurs the edges (see figure 5.20. This blur can be useful to some extent, as it makes the character resemble the blurred template (cf. figure 5.18). Too much blur, however, will result in an unrecognizable character.

Figure 5.20: A character image (left) and the image after applying a simple mean filter with radius 1 (middle) and radius 2 (right).

Binomial mean filter In contrast to a simple mean filter where each pixel in the moving window has the same weight (i.e. the pixels are unweighted), different weights can be applied to the pixels in the window. A common way is to use the binomial coefficients as weights [6]. The weights on a 3×3 window could look like this:

$$\begin{matrix} 1 & 2 & 1 \\ 2 & \mathbf{4} & 2 \\ 1 & 2 & 1 \end{matrix}$$

The weighted average of the window

$$\begin{matrix} 0 & 255 & 255 \\ 0 & \mathbf{255} & 255 \\ 0 & 0 & 0 \end{matrix}$$

is then calculated as

$$\frac{1 \cdot 0 + 2 \cdot 255 + 1 \cdot 255 + 2 \cdot 0 + 4 \cdot 255 + 2 \cdot 255 + 1 \cdot 0 + 2 \cdot 0 + 1 \cdot 0}{16} = 143.4375 \approx 143$$

A binomial mean filter weights the inner pixel of a windows more than then outer windows. This can lead to a reduced blur on the edges, as shown in figure 5.21

Figure 5.21: A character image (left) and the image after applying a binomial mean filter with radius 1 (middle) and radius 2 (right).

5.4.5.3 Remove white blots

Apart from applying convential filters, a more complex option is to find small blots of white pixels within black areas and remove them by setting them black. Though computationally complex, this method can repair character shapes to some extent without destroying information like most filters do. However, a limit of the maximum size of such a white blot needs to be set to avoid removal of normal white areas. In figure 5.22, the results of such a processing are shown.

Figure 5.22: A character image (left) and the image after removing white blots up to 5 pixels (middle) and blots up to 20 pixels (right).

5.4.5.4 Grayscale character images

After character segmentation, the characters could be extracted from the original grayscale image instead of the binarized image. The grayscale character images contain more information, since their edges are darker, even though not dark enough be identified as black pixels.

5.5 Simple autocorrection

After recognizing all characters on a receipt, the raw text of the receipt (with recognition errors) is available as depicted in figure 5.23. Many recognition errors can be corrected by using context information, e.g. characters that are recognized as letters where numerals are expected. In the following, such errors and their correction mechanism are presented.

5.5.1 Corrected errors

Due to optical similarity, some characters are mistakenly recognized as different characters at times. Some of these errors can be corrected by searching for the errors with regular expressions and replacing them with the correct character:

- mistakenly recognized uppercase O instead of numeral 0
- mistakenly recognized uppercase B instead of numeral 8
- mistakenly recognized lowercase L (l) instead of numeral 1
- mistakenly recognized numeral 0 instead of
  - uppercase O
  - uppercase D
- mistakenly recognized numeral 8 instead of uppercase B

REWE
R E W E
Nibelungenplatz 5
94032 Passau
UID Nr.: DE812706034

		EUR
BANANE		
1,086 kg x	1,69 EUR/kg	1,84 B
KOPFSALAT		0,99 B
FRUCHTQUARK		1,89 B
SUMME	EUR	4,72
Geg. BAR	EUR	10,00
Rückgeld BAR	EUR	5,28

Steuer %	Netto	Steuer	Brutto
B= 7,0%	4,41	0,31	4,72
Gesamtbetrag	4,41	0,31	4,72

30.03.2015 19:06 Bon-Nr.:7460
Markt:0550 Kasse:4 Bed.:282828

Noch keine PAYBACK Karte? Schade.

Für diesen Einkauf hätten Sie
2 Punkte erhalten!

Gleich hier am REWE Service Punkt im Markt
anmelden oder auf www.rewe.de/payback

REWE Markt GmbH

Für Sie da:
MO - SA 7:00 bis 20:00 Uhr

Sie haben Fragen?
Antworten gibt es unter www.rewe.de

```
MMMMMM* MMMMMB!MM 4MM!,MMrMMMMMä
R E W E
Nibelungenplatz 5
94332 Passau
UID Nr.: DE812706034
EUR
BANANE
1,086 kg x   1,69 EUR/kg       1,84 B
KOPFSALAT                       0,99 B
FRUCHTQUARK                     1,89 B
SUMME                  EUR      4,72
Geg. BAR               FUR     13,00
Rückgeld BAh           EUR      5,28
sumer ,       kvnr     sume      rvvnr
4mm*HdE m      1 H      lE!l       ! !
30.03.2015     13:0C    Bon Nr.:7460
Marki:0550     K sse 4  Bed.:282828
*********,,,***,*,**********************
Np h kcin- uAYBACK Karte? Schade.
für dicscn Einkauf hatten Sie
2 Fuakte erhalten!
nici h h:e, am  rwL Sc:vice P nki im Markt
anmelden oder a f www.rewe.de/payback
******,***,****r*,,***********,*********
REWE Markt GmbH
Für 3ie da:
MO - GA 7:0d bis 20:00 Uhr
Sie haben Fragen?
Antwo:icn gibt es unter www rewe de
```

Figure 5.23: A binarized receipt image and the text recognized by *OCRAM*.

- mistakenly recognized numeral 3 instead of
  - uppercase B
  - uppercase S
  - uppercase R
- mistakenly recognized dot instead of
  - blank/whitespace
  - comma
- mistakenly recognized comma instead of dot
- mistakenly recognized lowercase letter instead of uppercase letters (e.g. p instead of P)

In the following, details about the autocorrection are explained with the example of a mistakenly recognized uppercase O instead of the numeral 0. The simple autocorrection for the other errors works similar.

Instead of recognizing the numeral 0 an uppercase O may be recognized, if the character is deformed due to faded print, missing pixels, etc. Fortunately, numerals usually occur in groups with other numerals, dots and commas, e.g. `0,99` or `30.03.2015`. Therefore the following regular expression can be used to find such errors:

```
((,|\.|\d)O)|(O(,|\.|\d))
```

This regular expression finds all occurrences of an uppercase O that are preceded or followed by a comma, an dot or a numeral. If the original recognition results indicate that a character found by such a regular expression might as well be the numeral zero instead of an uppercase O (e.g. if the *NRMSE* values are similar), it is replaced by a zero.

5.5.2 Proposed improvements

Apart from these simple autocorrections, more sophisticated corrections are possible.

5.5.2.1 Keyword dictionary

A dictionary of known keywords can correct recognition errors, if there is a limited number of errors in a word. This dictionary may contain keywords such as “EUR”, “Rückgeld”, or “BAR” as well as common receipt words, that are not contained in the product list, such as “Passau”, “Nibelungenplatz” or “Einkauf”. To correct errors in the product list, a similar yet different approach is described in the next section. To create such a dictionary, the structure of all possible receipts needs to be known. Alternatively, this dictionary can be designed to grow dynamically upon human confirmation of a new entry.

After creating such a dictionary of known words, each recognized word can be tested whether it occurs in the dictionary. If it does not, suggestions for a correction are retrieved, e.g. if the word “Rühkgeld” is recognized, the known word “Rückgeld” could be suggested. Providing such suggestions can work similar to commonly known autocorrect systems.

For example, the known words can first be filtered by the number of characters in the word, then by the most reliably recognized character (based on the *NRMSE* values and the distance to the next best guess). Assuming a database with the following entries, the first filter (number of characters in the word) would leave “anmelden”, “Rückgeld” and “erhalten” in the search scope. If the lowercase e at index 5 of the recognized word “Rühkgeld” is the most reliably recognized character, the second filter would only leave “Rückgeld” as an possible correction.

- Einkauf
- heute
- anmelden
- Rückgeld
- keine

- leider
- erhalten

The more entries such a dictionary contains, the more difficult and computationally expensive this dictionary based correction becomes. In supermarket receipts, however, only a limited set of words occurs in contrast to dictionaries that power the spell checkers of text processors.

5.5.2.2 Product database

Similar to a dictionary, a database of known products (including their prices per piece respectively per weight unit) can be created to correct misrecognized product labels and prices. Other than a regular dictionary, this database needs to know the name and the price of a product. As prices and sometimes even product names may change and new products appear in the stores, the database needs to be able to reflect these changes.

For example, when a product is recognized that can not be found in the database, this product is added to the database if either

- the product has been recognized with high reliability (e.g. low *NRMSE* values or high difference to the second best match)
- a human user confirmed the correctness of the products spelling and price

Moreover, the database may contain a reliability score for each dynamically added or changed entry. This score could increase every time the product is recognized again with high reliablity or confirmed again, eventually leading to an entry that can be considered safe.

5.5.2.3 Plausibility validation

Errors among the prices can often be corrected by validating the plausibility of the bill. This inlcudes checking whether the price per piece respectively the price per weight unit multiplied by the amount equals the total price of a product item. Furthermore, the total sum of the bill must equal the sum of all product entries. Additionally, date and time can be checked for plausibility [6] and the store address can be queried in an online map service.

5.6 Data extraction

After recognizing all characters on a receipt and correcting errors, the raw text of the receipt is available as in the following excerpt:

[6] e.g. 89.71.1919 seems rather unrealistic for a date

```
EUR
BANANE
1,086 kg x   1,69 EUR/kg       1,84 B
KOPFSALAT                       0,99 B
FRUCHTQUARK                     1,89 B
SUMME                  EUR      4,72
Geg. BAR               FUR     13,00
Rückgeld BAh           EUR      5,28
sumer ,       kvnr     sume      rvvnr
4mm*HdE m      1 H      lE!l       ! !
30.03.2015     13:00    Bon Nr.:7460
Marki:0550     K sse 4  Bed.:282828
```

This raw text, however, is not of particular interest. The desired information needs to be extracted by processing the text. In *OCRAM*, a simple keyword and regular expression based text processing is implemented, which will be described in section 5.6.2.

5.6.1 Desired information

Not all characters on a receipt are important in all usecase szenarios. For example, many receipts contain promotional parts or tax information. For an application as mentioned in chapter 1, only the following information is relevant:

- the sum of all bought products
- the given amount of money
- the drawback
- a list of all bought products, including
  - the product name
  - the quantity
  - the price per piece
  - the total price for this product
- the date and time of the receipt
- *optional:* the address of the store
- *optional:* the store's sales tax ID

5.6.2 Text processing

OCRAM uses keyword based and regular expression based data extraction.

5.6.2.1 Keyword based text processing

Each line of the raw text is first tested for keywords which usually occurr at the start of a line. These keywords are:

- SUMME *(sum)*
- Geg. BAR *(total given in cash)*
- Rückgeld BAR *(drawback in cash)*

If such as keyword is detected, the rest of the line is processed accordingly, e.g. when "SUMME" is detected at the start of a line, the characters at the end of the line are converted to a number, which is set as the receipts total sum.

Due to recognition errors, the keyword detection is prone to recognition errors. For example, if a single character is not recognized correctly, leading to "Rückgeld BAh" instead of "Rückgeld BAR", the line would not be detected as the drawback line, even though the desired data (the drawback amount) may be recognized correctly. To reduce the number of missing data extractions, *OCRAM* does not only detect absolute keyword matches, but checks for fuzzy matches, too.

To recognize such a fuzzy match, the Levensthein distance [7] is used. The Levensthein distance is the minimum number of insert, delete or replace actions that is necessary to convert a string to another string. For example, "Rückgeld BAR" and "Rückgeld BAh" have a Levensthein distance of 1, as one replacement is sufficient to convert the latter.

To detect keywords, the Levensthein distance between a recognized line and the keyword is calculated. If the distance is smaller or equal to 2 the keyword is detected. This allows up to two recognition errors in a keyword. A higher threshold would likely work, too, but it was not necessary in the course of this thesis.

5.6.2.2 Regular expression based text processing

Product data as well as date and time cannot be extraced by relying on keywords, as these values are not labeled like the total sum, the drawback and the total given are. They do, however, follow a certain structure which makes it easy to detect the values using regular expressions. For example, the date and time is extracted by searching for the following regular expression:

```
^\s*\d{2}\.\d{2}.\d{4}\s*\d{2}:\d{2}
```

This regular expression matches strings such as "30.03.2015 13:00" (see the excerpt in section 5.6).

Furthermore, product lines need to be detected. Because the name of a product does not have any structure [7], the detection relies on the right part of a product line only. There are three kinds of products, simple products, multi piece products and weight products.

[7] One might believe that all product names are all caps, but unfortunately the are not. There is, for example, a product with the name "Holland", which are tomatoes from the Netherlands.

Simple products Simple products consist of a single line on a receipt, e.g.

```
FRUCHTQUARK                    1,89 B
```

They can be recognized rather simply by detecting the price at the end of the line. The letter after the price ("A" or "B") indicates whether this product is taxed with 7 % VAT ("B") or 19 % VAT ("A"). Furthermore, there might be an asterisk after the tax indication letter. This asterisk marks products that are not viable for discounts. The regular expression to detect the price at the end of a product line therefore is as follows:

```
(\s|-)\d{1,3},\d{2}\s(A|B)(\s\*)?
```

Multi piece products Multi piece products consist of two lines. In the first line, the product name and the total price for all pieces is printed. In the second line, the number of pieces and the price per piece is written.

```
SALATGURKE                     1,78 B
2 Stk x    0,89
```

These products can be detected by recognizing the total price on the right side first and then analyzing the next line.

Weight products Similar to multi piece products, weight products have the name of the product on the first line and information about the quantity on a second line. On the other hand, the total price of the product is not on the first line, but on the second line.

```
BANANE
1,086 kg x   1,69 EUR/kg       1,84 B
```

Therefore, if a line without a price at the end is found, the next line is analysed for weight and price details with appropiate regular expressions.

The detection with regular expressions is prone to fail if a line contains errors from character recognition. In section 5.6.2.1 this issue was handled by using fuzzy matches with the Levensthein distance. A similar approach would be great for the regular expression based text processing, but it is much more complex to implement und computationally costly. This *fuzzy regular expression* would need to match all strings that would normally match the regular expression as well as such strings that meet all conditions of the regular expression except for one or two.

To some extent, this robustness against common errors can be implemented with normal regular expressions, too. A common error during character recognition is recognizing the numeral 8 instead of the letter B (cf. section 5.5.1). A possible resulting recognized line might be:

```
FRUCHTQUARK                    1,89 8
```

The following regular expression which detects the price of a product would not work on a line with such an error:

```
(\s|-)\d{1,3},\d{2}\s(A|B)(\s\*)?
```

But if the regular expression is changed and accepts not only “A” and “B” but also “8” as a tax indicator, the regular expression would match.

```
(\s|-)\d{1,3},\d{2}\s(A|B|8)(\s\*)?
```

Similar modifications to the regular expressions can be found for most common recognition errors. This does, however, only work to some extent.

6 Evaluation

To evaluate the performance of the proposed solutions and their implementation *OCRAM*, an evaluation on two level has been conducted. In the first, the pure character recognition is evaluated. The second evaluation goes one step further and evaluates the quality of the whole OCR system, reaching from preprocessing to error correction and data extraction. It evaluates how much information could be extracted from the image of a receipt.

6.1 Evaluation of character recognition

To evaluate the performance of the character recognition, all character images from a set of 39 receipts have been labeled. This results to a set of 35 659 characters, including 11 156 blanks and 24 503 other, non-whitespace characters. The exact distribution of characters is listed in table 5.3.

This set of labeled characters is then split into ten parts to conduct a 10-fold cross-validation. While doing so, each part is provided with the same amount[8] of samples per character.

After splitting the data into ten parts, one part is used as test part while the nine other parts are used as training data and used to generate templates. The characters in the test part are then recognized using these templates. Afterwards another part is picked for the test data and again, the remaining nine (this time including the test data from the first run) are used to generate template while recognizing the characters in the test part. This processes is repeated ten times, so that each part will have served as test part once. Therefore, each character has been recognized once, but no character has been recognized using this character's image as training data.

[8] With the exception of some parts having one sample more or less

The average recognition rate is 97,36 % (34 715 out of 35 658) including 11 156 blanks. If blanks are not taken into account[9] the average recognition rate is 96,21 %. The recognition rate for each individual character is listed in table 6.1. In the following, the worst and best recognized characters with more than 100 total samples are listed[10].

- O: 87,99 %
- D: 88,10 %
- 8: 89,76 %
- 0: 90,00 %
- ⋮
- 4: 99,66 %
- a: 99,73 %
- i: 99,81 %
- BLANK: 99,89 %

6.1.1 Evaluation of reliability classes

As discussed in section 5.4.4, each recognized character is assigned one of the classes HIGH, MEDIUM or LOW that predicts the recognitions reliability. To evaluate this prediction, the false recognitions within these classes have been counted. The results are shown in table 6.2.

Reliability class	**samples**	**errors**	**recognition rate**
LOW	6 388	888	86,10 %
MEDIUM	6 716	32	99,52 %
HIGH	22 564	23	99,90 %
Total	**35 668**	**943**	**97,36 %**

Table 6.2: Errors and recognition rate per reliability class.

The table shows that the characters that are predicted to be recognized with high reliability actually do have a high recognition rate. There are only few errors within this class although a large portion of all characters falls into this class. That means that even though *OCRAM* does not recognize all characters correctly, it has a good hint about the reliability of the recognitions. This information can be used during automatic error correction or can be displayed to the user to assist manual editing.

[9] Blanks are relatively easy to recognize and account for a large part of the character set.
[10] Characters that occur less than 100 times tend to have a recognition rate of either 0 % or 100 %.

A	99,42 %	a	99,73 %	0	90 %	*	97,38 %
B	95,14 %	b	90,46 %	1	97 %	BLANK	99,88 %
C	94,44 %	c	94,48 %	2	98,29 %	:	98,36 %
D	88,10 %	d	93,71 %	3	97,44 %	,	93,80 %
E	95,97 %	e	95,91 %	4	99,66 %	-	97,52 %
F	96,02 %	f	98,73 %	5	98,24 %	.	98,89 %
G	90,05 %	g	97,82 %	6	93,68 %	!	100 %
H	93,75 %	h	95,76 %	7	98,45 %	%	100 %
I	99,48 %	i	99,81 %	8	89,76 %	+	0 %
J	100 %	j	N/A	9	95 %	?	100 %
K	95,83 %	k	98,33 %			/	99,06 %
L	98,10 %	l	97,58 %				
M	98,13 %	m	96,89 %				
N	97,04 %	n	94,70 %				
O	87,98 %	o	96,76 %				
P	95,48 %	p	95,50 %				
Q	100 %	q	N/A %				
R	93,92 %	r	97,71 %				
S	90,07 %	s	99,30 %				
T	96,55 %	t	98,44 %				
U	92,20 %	u	94,43 %				
V	100 %	v	97,78 %				
W	92.59 %	w	97,97 %				
X	0 %	x	98,25 %				
Y	100 %	y	100 %				
Z	100 %	z	100 %				
Ä	N/A	ä	93,55 %				
Ö	N/A	ö	N/A				
Ü	N/A	ü	91,67 %				

Table 6.1: Recognition rate per individual character.

6.2 Evaluation of data extraction

Other than having a dataset of cropped and centered, manually labelled character images to train and test on, this evaluation relies only receipt images as input and extracted receipt data as stated in section 5.6.

This means that all errors that occur during line and character segmentation or character recognition may propagate and cause following errors during data extraction. On the other hand, this evaluation also includes error correction, i.e. recognition errors can be correction by using context information (see section 5.5). This evaluation is more use-oriented, as the system is viewed as a “black box” with a receipt image as input and extracted information as output.

To measure the performance of the whole system, the information from 39 receipts has been extracted manually (ground truth). Then, the 39 receipt images were processed by *OCRAM* and the extracted information was compared to ground truth.

Every information from ground truth that was not correctly extracted (e.g. spelling error in product name, wrong price or incorrect quantity) or not extracted at all was counted as an error. The total rate of correctly extracted information fields, as described in section 5.6, is 76,37 % with 362 out of 474 correct extractions.

When analyzing the extractions per receipt in table 6.3, the distribution of the extraction rate attracts attention. The extraction rate is not equally distributed around the mean of 76,37 %, it is rather wide spread (the standard derivation is 26,2 %) and split into two groups. The first group contains receipts with relatively high extraction rates (> 80 %), while the second group has low extraction rates (< 70 %). The histogram in figure 6.1 illustrates this by grouping the extraction rates into ranges of 5 %, i.e. a group with all recognition rates from 28,5 to 32,5 % (labelled as 30 %), a group with rates from 32,5 to 37,5 % (labelled as 35 %), etc.

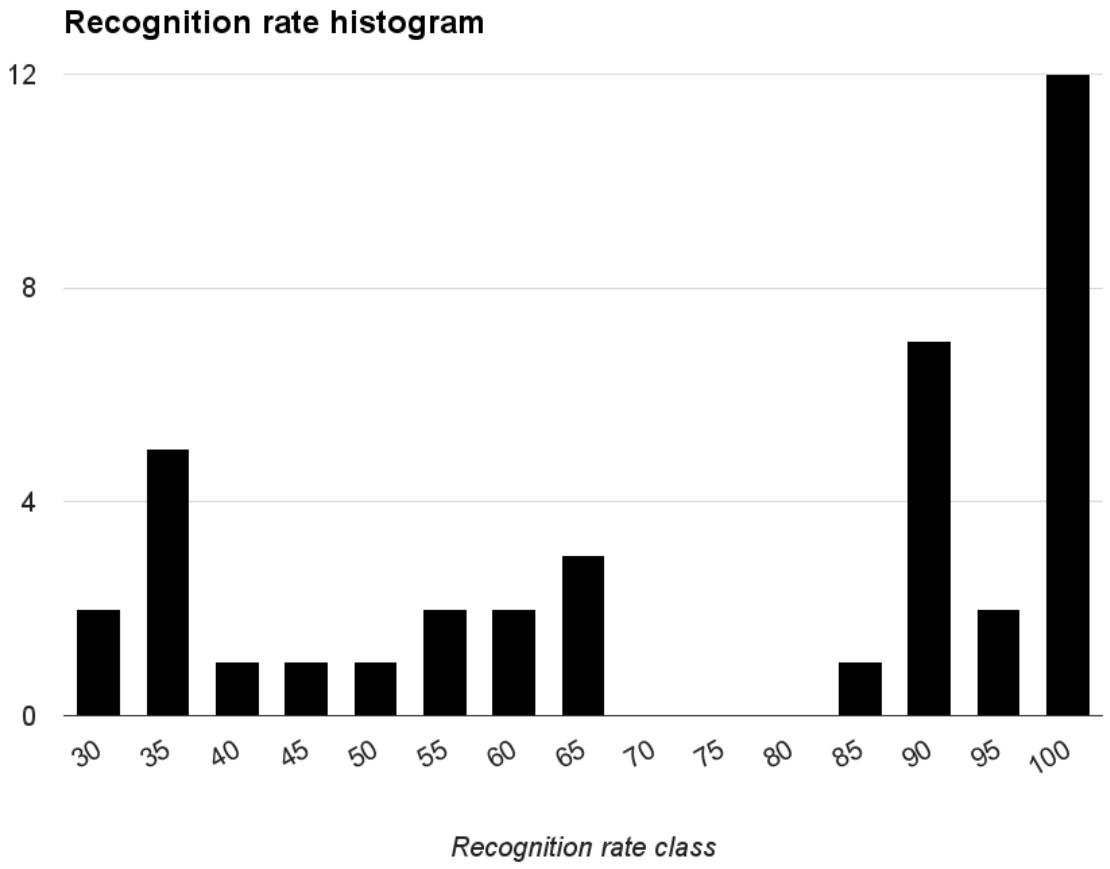

Figure 6.1: A histogram of the frequency of recognition rates. The rates are grouped into 5 % steps, e.g. all rates from 87,5 % to 92,5 % are in the group 90.

The reason for this spread is the varying quality of the receipt images. Some receipts are faded and some have printing defects, for example the receipt with index 35 in figure 6.2, which has defects on the leftmost character. If only receipts with comparatively high quality (extraction rate ≥ 80 %) are taken into account, the data extraction rate reaches an average of 91,3 %. Although this seems like a self-fulfilling prophecy, depending on the use case and quality of the input images, this may be a more realistic rate.

receipt index	**correct extractions**	**extractable data**	**extraction rate**
1	5	8	62,5 %
2	11	12	91,7 %
3	11	12	91,7 %
4	4	13	30,8 %
5	5	11	45,5 %
6	7	8	87,5 %
7	3	9	33,3 %
8	10	18	55,6 %
9	9	9	100 %
10	16	17	94,1 %
11	13	14	92,9 %
12	16	16	100 %
13	9	10	90 %
14	9	10	90 %
15	17	17	100 %
16	18	18	100 %
17	18	18	100 %
18	10	10	100 %
19	12	12	100 %
20	18	20	90 %
21	14	14	100 %
22	9	9	100 %
23	8	13	61,5 %
24	3	9	33,3 %
25	10	12	83,3 %
26	4	11	36,4 %
27	8	9	88,9 %
28	8	12	66,7 %
29	4	11	36,4 %
30	4	11	36,4 %
31	8	12	66,7 %
32	13	13	100 %
33	4	8	50 %
34	14	14	100 %
35	5	17	29,4 %
36	12	12	100 %
37	3	8	37,5 %
38	4	7	57,1 %
39	6	10	60 %
Total/mean	**362**	**474**	**76,4 %**
Variance			**686,9 %**
Standard derivation			**26,2 %**

Table 6.3: Data extraction rate per receipt.

Figure 6.2: Original grayscale and binarized part of the receipt with index 35. The leftmost characters suffer from printing defects, causing errors on the product's names.

7 Future work

To enhance the recognition results, many improvements are possible, some of them have already been discussed in sections 5.4.5 and 5.5.2. Especially a more elaborated autocorrection, based on an dictionary and a product database, is promising. Combined with the recognition reliability class, the autocorrection could focus on the characters with a low reliability and recognize them more from context than from actual recognition. Apart from these suggestions that are specific to template matching and autocorrection, different approaches can be developed. For example, instead of using simple template matching for character recognition, support vector machines, neural networks or other machine learning techniques can be assesed.

Furthermore, the limitations described in section 2.1 need to be tackled. For a real world application, scanning the receipts with flat bed scanners is not practicable. The recognition has to deal with smartphone photographs, which are often distorted and usually have a lower resolution. Receipts will have to be straightened and the line and character segmentation need to be adapted to work with slightly distorted receipt images.

Concentrating on the receipts of a single market is not suitable either, but various markets' receipts could be recognized by training on different fonts and developing individual text processors to extract data from each layout.

Besides these optimizations and improvements, an actual consumer oriented application based on the data extractions needs to be developed. This application could offer a detailed overview of all expenses, categorized in fruits & vegetables, diary products, household goods etc. The application coudld also connect with other services that track expenses and money transactions, such as online bank accounts and budget applications.

8 Conclusion

The objective of this thesis was the development of a proof of concept system that can recognize text on supermarket receipts and extract the information printed on the receipts. This information can be used to offer consumer orientied applications, for example for automatic budgeting and categorizing everyday expenses.

During development of this system, various challenges that are special to receipts such as low quality printing have been discovered and solutions to these challenges have been designed and implemented. The final system, though relatively simple, is capable of recognizing text with fair recognition rate. Even without exhaustive error correction, a data extraction rate of 76,37 % is reached despite dealing with many low quality receipts with printing defects or faded printing.

Many ideas for further development and optimization are yet to be evaluated, but these have great potential and could increase the data extraction rate to a value near 100 %. Especially in the areas of automatically correcting errors based on context information and data extraction a large field for optimizations and improvements is open.

Furthermore, the implementation relies on straight, non distorted scans of the receips. These high-quality scans are only available when scanning with a flatbed scanner but not with photographs. To recognize text on receipt photographs, extensive preprocessing must be performed to straighten the receipts and remove shadows and other artifacts.

Eidesstattliche Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Passau, den 19. Juli 2016

Marco Ziegaus

List of Tables

List of Figures

Source code listing

Bibliography

[1] S. Mori, C. Suen, and K. Yamamoto, “Historical review of OCR research and development,” vol. 80, no. 7, pp. 1029–1058.

[2] L. Eikvil, “OCR - Optical Character Recognition.”

[3] R. Cemer, W. Whitney, and K. Pribluda, “Java OCR.”

[4] N. Otsu, “A Threshold Selection Method from Gray-Level Histograms,” vol. 9, no. 1, pp. 62–66.

[5] M. Sezgin and B. Sankur, “Survey over image thresholding techniques and quantitative performance evaluation,” vol. 13, no. 1, p. 146.

[6] T. Sauer, “Einführung in die Signal- und Bildverarbeitung.”

[7] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions, and reversals,” pp. 707–710.