Bayerische Julius-Maximilians Universität Würzburg
Institut für Informatik
Lehrstuhl für Informatik II

Diplomarbeit

Mustererkennung in Frühdrucken

Winfried Höhn

14. September 2006

Betreuer:
Prof. Dr. Jürgen Albert
Dr. Hans-Guenter Schmidt
Dipl.-Inform. Stefan Selbach

Erklärung

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

Würzburg, den 14. September 2006

_____

(Winfried Höhn)

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation und Zielsetzung

Schon seit weit über 2000 Jahren existieren Bibliotheken, um das Wissen zu bewahren und weiter zu verbreiten. Die berühmteste unter ihnen war wohl die Bibliothek von Alexandria. Die Erhaltung und Zugänglichmachung dieses Wissens erfolgt hierbei schon länger nicht mehr nur durch den physischen Erhalt der Bücher, durch Konservierung und Restauration, sondern zunehmend auch durch die Digitalisierung. Also durch scannen der Buchseiten mit hoher Auflösung.

Durch die digitale Verfügbarkeit werden neue Nutzungsmöglichkeiten eröffnet, wie das zugänglich machen über das Internet. Außerdem kann hierdurch auch die Verfügbarkeit von älteren Büchern, welche Aufgrund ihres Zustandes oder Wertes der Allgemeinheit nicht direkt zugänglich gemacht werden können, erhöht werden. Ein Problem stellt hierbei aber schon alleine die Größe dar. So können einzelne Buchseiten leicht über 100MB erreichen, was dem online zur Verfügung stellen der Scans in bester Qualität klare Grenzen setzt. Zusätzlich zu den technischen Problemen ist auch zu Bedenken, dass bei schlechter Druckqualität oder einem für heutige Nutzer ungewohntem Schriftbild das Lesen erheblich erschwert wird. Durch ein Umwandeln der Bilddaten in Text können diese Probleme behoben werden. Hierdurch wird nicht nur eine erhebliche Verkleinerung der Datenmenge und Wählbarkeit einer modernen Schriftart erreicht, sondern die Textdaten können nun zum schnelleren Auffinden von Informationen auch elektronisch durchsucht werden.

Ziel dieser Arbeit ist es, die Möglichkeiten der Methoden der Texterkennung in folgenden Optical Character Recognition (OCR) genannt, im Hinblich auf Frühdrucke zu untersuchen. Diese unterscheiden sich von heutigen Schriftstücken in Schrift und Sprache, so dass übliche Verfahren nicht ohne weiteres eingesetzt werden können.

1.2 Aufbau der Arbeit

In Kapitel 2 folgt eine Erklärung von Frühdrucken und Inkunabeln, sowie eine Erklärung der einzelnen Teilschritte von OCR Systemen und ein Überblick über die zur

Realisierung der Teilschritte verwendeten Methoden. Am Ende des Kapitels folgt ein Überblick über erhältliche OCR Software.

In Kapitel 3 werden die OCR-Programme OmniPage [22] und FineReader [1] hinsichtlich ihrer Eignung zum Erkennen von Frühdrucken untersucht.

In Kapitel 4 wird der Aufbau der eigenen Implementierung beschrieben und die Auswahl der Algorithmen für die einzelnen Schritte erläutert.

In Kapitel 5 wird auf die Implementierung und Funktionsweise der ausgewählten Algorithmen eingegangen.

In Kapitel 6 werden die erreichten Ergebnisse bewertet und Möglichkeiten zur Weiterarbeit aufgezeigt.

2 Grundlagen

2.1 Frühdrucke

Dieser Abschnitt orientiert sich an [33].

Die ersten gedruckten Schriften werden als Inkunabeln oder Wiegendrucke bezeichnet. Im 20. Jahrhundert legte man sich darauf fest, nur Schriften welche bis zum 31. Dezember 1500 entstanden sind, als Inkunabeln zu bezeichnen. Diese Festlegung wird dadurch bestärkt, dass sich die ersten Bücher noch stark an handgeschriebenen Werken orientierten, was sich Anfang des 16. Jahrhunderts langsam änderte.

Als Postinkunabeln oder Frühdrucke werden gedruckte Schriften, welche zwischen 1501 und ca. 1520 entstanden, bezeichnet. Manchmal werden auch noch Schriften bis 1530 oder 1550 zu den Frühdrucken gerechnet.

Durch die schon erwähnte Orientierung an den mittelalterlichen Handschriften, weisen diese Drucke einige Besonderheiten auf. Nach dem Drucken wurden von Hand gemalte reichlich verzierte Initialen eingefügt. Auch wurden für Ligaturen eigene Lettern gefertigt, welche sich in der Form deutlich von den Einzelbuchstaben unterscheiden, siehe Abbildung 2.1. Zudem gibt es auch Lettern für häufige Worte, Wortanfänge und Endungen, siehe Abbildung 2.2 und Abbreviaturen, das sind Buchstaben, welche z.B. mit einem Strich, einem Punkt oder einem Kreis markiert sind und ausgelassene Buchstaben vor oder nach der Abbreviatur anzeigen, einige Beispiele sind in Abbildung. 2.3 zu sehen. Einen Überblick über die von Gutenberg zur Erstellung seiner 42 zeiligen Bibel genutzten Lettern gibt Abbildung 2.4.

Abbildung 2.1: Ligaturen aus der Gutenbergbibel für p a, p e, langes-s e und langes-s d

Als Schriftarten wurden hier zuerst Textura verwendet, welche sich durch ihr dunkles Schriftbild und die kleinen Wort- und Zeilenzwischenräume schwer lesen lässt, ein Beispiel ist in Abbildung 2.5 zu sehen. Um 1472 kam die Schwabacher hinzu und Anfang

Abbildung 2.2: In der Gutenbergbibel verwendete Abkürzungen für con-, et, -ue, -us

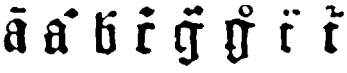

Abbildung 2.3: Abbreviaturen aus der Gutenbergbibel

des 16. Jahrhunderts setzte sich die Fraktur immer mehr durch, welche schlankere Lettern besitzt und so auch bei kleineren Schriftgrößen lesbar bleibt.

Abbildung 2.4: Übersicht über die in der 42 zeiligen Gutenbergbibel verwendeten Typen *(Quelle: Universitätsbibliothek Würzburg)*

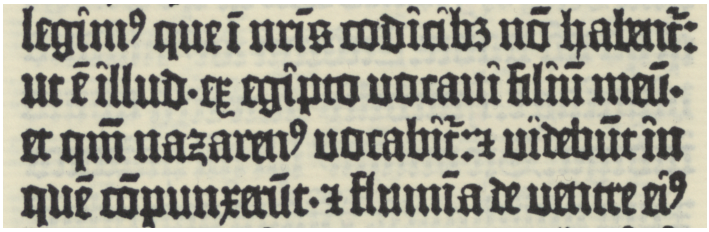

Abbildung 2.5: Ausschnitt aus der Gutenbergbibel *(Quelle: Universitätsbibliothek Würzburg)*

2.2 Aufgaben und Verwendung der OCR

OCR Software wird eingesetzt, um aus Bildern am Computer weiterverarbeitbare Strukturinformationen und Textdaten zu extrahieren. Dies stellt eine Alternative zur manuellen Neuerfassung der Dokumente dar. Die erzeugten Textdaten können im Gegensatz zum Bildmaterial leichter verändert und auch durchsucht werden. Außerdem können sie durch ihre geringere Größe leichter zugänglich gemacht werden, z.B. über das Internet. Wie beim Projekt Gutenberg [26], welches digitalisierte Texte enthält, deren Copyright abgelaufen ist.

Andere Anwendungsbereiche sind die automatische Briefsortierung, Erfassung von Nummernschildern und das Ablesen von Barcodes.

2.3 Teilschritte der OCR

Die OCR lässt sich grob in folgende Schritte aufteilen:

- Vorverarbeitung
- Layout Analyse
- Merkmalsextraktion
- Klassifikation
- Nachverarbeitung

Diese Einteilung ist nicht zwingend und die Grenzen zwischen den einzelnen Schritten sind oftmals fließend. So wurde in [15] ein Verfahren vorgestellt, welches die Merkmalsextraktion, Klassifikation und Nachverarbeitung zu einer einheitlichen Operation verbindet.

Jeder der Schritte wird im folgenden kurz erläutert und Methoden zu seiner Realisierung angegeben.

2.3.1 Vorverarbeitung

Bei der Vorverarbeitung werden von den nachfolgenden Schritten nicht benötigte Daten und Störungen, wie z.B. Flecken und Rauschen aus den Bildern entfernt und diese für die nächsten Schritte vorbereitet. Dieser Schritt besteht meistens aus einer Umwandlung des Bildes in ein Binärbild und einer Korrektur von Verdrehungen, wenn die Vorlage nicht gerade eingescannt wurde.

Die Algorithmen zum Binarisieren lassen sich in globale und lokale Methoden unterteilen. Globale Methoden berechnen meist aus dem Histogramm einen Schwellenwert für das gesamte Bild und setzen alle Punkte mit kleineren Helligkeitswerten auf Schwarz und alle restlichen Punkte auf Weiß. Bei dunkler Schrift auf hellem Hintergrund sollte nach diesem Schritt die Schrift schwarz und der Hintergrund weiß sein. Durch ungleichmäßige Beleuchtung und Flecken auf dem Papier können aber auch Teile des Hintergrunds nach der Binarisierung schwarz sein. Dies versuchen lokale Verfahren zu vermeiden, indem sie für jeden Bildpunkt einen eigenen Schwellenwert berechnen. Hierzu wird beim Ermitteln des Schwellenwerts meist die Umgebung des Punktes verwendet. Das Ergebnis ist dann von der Wahl der Umgebungsgröße abhängig, welche bei den meisten Verfahren nicht automatisch bestimmt wird, sondern als Parameter übergeben werden muss.

Der Binarisierungsalgorithmus von Niblack berechnet für jeden Punkt (x, y) den Schwellenwert durch $T(x, y) = m(x, y) + ks(x, y)$, wobei $m(x, y)$ und $s(x, y)$ der Mittelwert und die Standartabweichung in einer lokalen Umgebung um den Punkt (x, y) sind. Die Größe der Nachbarschaft und k sind Parameter des Algorithmus. Vergleiche dieses und weiterer Binarisierungsalgorithmen finden sich in [31, 21]. Ein weiterer parameterfreier Binarisierungsalgorithmus wird in [4] beschrieben.

Für manche Verfahren zur Layout Analyse müssen die Zeilen parallel zur x-Achse verlaufen und daher kann in diesem Schritt auch eine Skew-Korrektur nötig sein. Verdrehungen entstehen, wenn das einzuscannende Dokument nicht exakt im Scanner platziert wird oder durch die Wölbung von Buchseiten. Manchmal wird schräg verlaufender Text auch als Gestaltungsmerkmal verwendet, hauptsächlich in Werbetexten. Bei einer Projektion auf die x-Achse verschmelzen verschiedene Spalten schon bei unter 4°Verdrehung und Zeilen schon bei weniger als 1°, siehe Tabelle 2.6. Verschiede Verfahren zur Skew-Detection werden in [25, 5, 16] beschrieben, welche auch Referenzen zu weiteren Verfahren enthalten.

2.3.2 Layout Analyse

Die Layout Analyse hat die Aufgabe, die Struktur des Dokuments zu erfassen. Hierzu zählt einerseits die Erkennung von Textblöcken, Bildern und Verschmutzungen, welche nicht durch die Vorverarbeitung ausgefiltert wurden. Des Weiteren zählt auch die Zerlegung der Textblöcke in Spalten, Absätze, Zeilen, Worte und Buchstaben zur Layout Analyse. Da auf dieser Stufe noch kein konkretes Wissen über das Aussehen der Buchstaben und die verwendete Schriftart vorliegt, kann die Erkennung von Buchstabengrenzen schwierig sein, wenn sich z.B. Buchstaben berühren. Daher kann die Zerlegung in Buchstaben auch erst in späteren Schritten erfolgen oder man lässt eine Übersegmentierung der Buchstaben zu, d.h. ein Buchstabe kann in mehrere Segmente zerlegt werden. An jeder Buchstabengrenze muss aber eine Zerlegung stattfinden. In

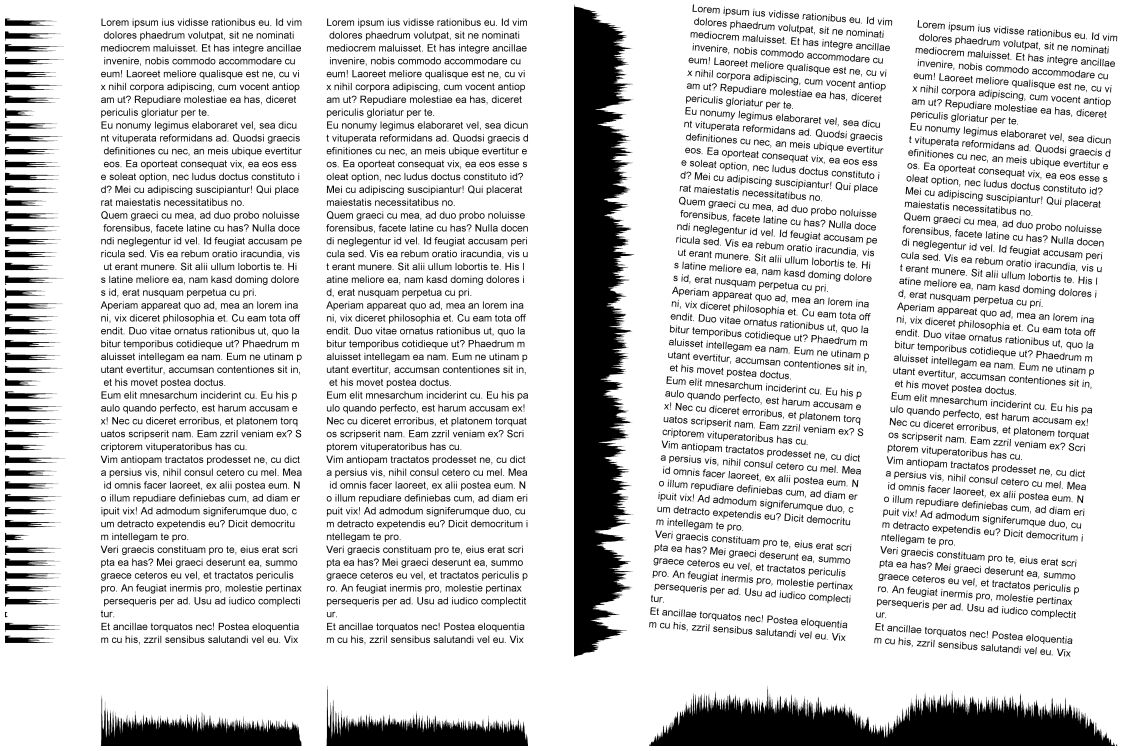

Abbildung 2.6: Projektion einer Seite und der um 4°rotierten Seite auf die Achsen

den nachfolgenden Stufen müssen dann diese Segmente wieder zu Buchstaben zusammengefügt werden.

Die Layout Analyse kann sowohl Top-Down, als auch Bottom-Up erfolgen. Bei Bottom-Up Strategien wird von den zusammenhängend schwarzen Segmenten im Bild ausgegangen, von welchen angenommen wird, dass sie Buchstaben oder sich berührende Buchstaben darstellen. Diese werden erst zu Worten, dann weiter zu Zeilen und Absätzen zusammengefasst. Das Vorgehen beim Zusammenfassen von Segmenten einer Zeile zu Worten wird in [32] beschrieben. So könnten auch Top-Down und Bottom-Up Methoden gemeinsam verwendet werden, indem bis zur Ermittlung der Zeilen Top-Down vorgegangen wird und die übrige Analyse Bottom-Up erfolgt.

Bei Top-Down Strategien wird das komplette Ausgangsbild in immer kleinere Strukturen zerlegt, dazu kann z.B. ein recursives X-Y-cut Verfahren benutzt werden. Bei diesem wird für den untersuchten Bereich – anfangs das ganze Bild – eine Projektion auf die x- und y-Achse ausgeführt, wie in der linken Abbildung in Tabelle 2.6 zu sehen, und an relevanten Minima wird der Bereich dann unterteilt, hier wären das die beiden Spalten. So sind für die Unterteilung in Spalten breite absolute Minima interessant, für Zeilen die Minima des geglätteten Histogramms. Für die neu gefundenen Bereiche wird das Verfahren nun rekursiv wiederholt, bis keine Minima, an denen das Bild weiter unterteilt werden kann, mehr vorhanden sind [siehe 18, 19].

Die in [11, 12] beschriebenen Verfahren benutzen zur Zerlegung in Bild und Textblöcke die Texturanalyse [siehe 29, S. 213ff], hierbei werden die Blöcke anhand ihrer verschiedenen Pixelverteilungen unterschieden.

Eine weitere Möglichkeit der Layout Analyse besteht darin, im Bild maximale leere Rechtecke zu suchen. Als Maximalitätskriterium kommt nicht nur die Fläche in Frage. Stattdessen kann man auch längliche Rechtecke, welche möglichst viele Segmente berühren bevorzugen. Dadurch erhält man Rechtecke welche Textspalten trennen. Lässt man nicht nur achsenparallele Rechtecke zu, so kann auf eine Skew-Korrektur verzichtet werden [siehe 7].

Vergleiche und Bewertungen mehrerer Layout Analyse Verfahren sind in [20, 14] zu finden.

2.3.3 Merkmalsextraktion

Ziel ist es, Merkmale aus den segmentierten Muster zu gewinnen, mit welchen eine möglichst eindeutige Zuordnung zu einem Buchstaben erfolgen kann. Die Merkmale sollten je nach Verwendungszweck invariant unter Rotation, Translation und Skalierung sein, wenn die Richtung der Zeile bestimmt wurde und die Zeichen nicht innerhalb einer Zeile verdreht sind, ist eine Invarianz unter Rotation nicht nötig. Die Merkmale

können z.B. aus dem segmentierten Zeichen, dessen Umriss oder dessen Skelett berechnet werden. In [30] werden als Möglichkeiten hierzu angegeben:

- Template matching
- Deformable templates
- Unitary transforms
- Zoning
- Geometric moments
- Zerenike moments
- Projection histograms
- Contour profiles
- Spline curve
- Fourier descriptors
- Graph description
- Discrete features

Aus manchen dieser Merkmale lassen sich zumindest annähernd die Zeichen aus denen sie erstellt wurden zurückgewinnen, etwa bei den Zerenike moments oder den Fourier descriptors. Dadurch ist sichergestellt, dass sie alle wichtigen Strukturinformationen enthalten. Beim Template Matching existiert für jedes Zeichen eine Vorlage und das segmentierte Zeichen wird pixelweise mit den Vorlagen verglichen, um die mit der geringsten Abweichung zu finden. Das Zoning teilt ein Zeichen in verschiedene Kacheln auf und berechnet für jede einzelne Kachel ein oder mehrere Merkmale, wie der Anteil an schwarzen Pixels oder der Schwerpunkt [vgl. 30].

Aus den mit einem oder mehreren dieser Verfahren berechneten Werten wird ein Merkmalsvektor gebildet. Bei Zoning hätte er die Länge berechnete Merkmale pro Kachel mal Anzahl der Kacheln. Merkmalsvektoren können sowohl eine feste Länge besitzen, als auch variabler Länge sein. Beim Template Matching hängt die Länge des Merkmalsvektors von der Größe der Vorlagen ab. Wenn alle Vorlagen auf eine einheitliche Größe normiert werden, so ist auch die Länge des Merkmalsvektors konstant, ansonsten ist sie variabel.

2.3.4 Klassifikation

Bei der Klassifikation soll einem Merkmalsvektor der Buchstabe zugeordnet werden, welcher den Zeichendaten aus denen der Merkmalsvektor erstellt wurde entspricht. Statt einer eindeutigen Zuordnung können die Klassifikatoren den verschiedenen Zeichen auch Wahrscheinlichkeiten, mit denen diese dem Merkmalsvektor entsprechen, zuordnen. Dann muss im Nachverarbeitungsschritt anhand des Kontextes die sinnvollste Möglichkeit ausgewählt werden.

Als Klassifikatoren können beispielsweise Next-Neighbor Klassifikatoren verwendet werden, diese berechen anhand einer Metrik den Abstand des Merkmalsvektors zu den Merkmalsvektoren aller Zeichenklassen und ordnen ihn der Klasse zu, zu der er den geringsten Abstand besitzt. Als Metrik kann z.B. die Euklidische-Distanz verwendet werden.

Des Weiteren kann die Klassifikation auch durch ein Hidden Markov Model [siehe 6, 8], eine Support Vector Machine [siehe 9], ein Neuronales Netz oder einen Bayesschen Klassifikator erfolgen.

2.3.5 Nachverarbeitung

In diesem Schritt sollen Fehler oder Unsicherheiten der vorhergegangenen Schritte, dem Segmentieren und Klassifizieren der Zeichen, erkannt und wenn möglich behoben werden. Die erkannten Worte werden auf ihre Plausibilität getestet und gegebenenfalls zu einem plausiblen Wort korrigiert. Hierbei sollte aber auch darauf geachtet werden, dass die Korrektur sich an den Zeichendaten orientiert und keine automatische Rechtschreibkontrolle darstellt, sonst steigt die Gefahr unbekannte Worte fälschlicherweise zu einem ähnlichen Wort im Lexikon zu korrigieren. Die Plausibilitätsprüfung kann z.B. mit einem Lexikon oder einer Trigrammstatistik erfolgen. Wenn das Wort nicht im Lexikon vorkommt, wird versucht im Lexikon ein möglichst ähnliches Wort zu finden. Hierzu können, wenn verfügbar, auch die Informationen über Zeichenwahrscheinlichkeiten aus dem Klassifikationsschritt verwendet werden.

Um den Abstand zweier Worte zu berechnen, verwendet man bei der Rechtschreibkontrolle oft die Levensthein-Distanz, diese gibt die minimale Anzahl von Einfügungen, Löschungen und Ersetzungen an, um vom Ausgangswort zum Zielwort zu gelangen. Für die OCR-Nachkorrektur muss die Levensthein-Distanz angepasst werden, die Ersetzung von Buchstaben sollte umgekehrt proportional zu ihrer Verwechslungsgefahr gewichtet sein. Zudem muss das Verbinden zweier aufeinanderfolgender Buchstaben und das Aufteilen eines Buchstabens hinzugefügt werden. Dies ist nötig, falls die Zeichensegmentierung ein Zeichen falsch getrennt hat und etwa ein **m** als **in** erkannt

wurde oder zwei Zeichen zu einem verschmolzen wurden und **vv** als **w** erkannt wurde [vgl. 28].

2.4 Kommerzielle OCR Systeme

Heute ist eine Vielzahl kommerzieller OCR-Programme im Preisbereich von 50 € bis 500 € verfügbar. Die bekanntesten sind FineReader [1], OmniPage [22], PaperPort [23], TextBridge [24] und ReadIris [3]. Diese Programme unterstützen meist über 100 Sprachen und können das Layout einer Seite inklusive Bilder und Tabellen erkennen und übernehmen. Für die gebräuchlichsten Sprachen sind bei den Programmen meist auch Fachwörterbücher für Bereiche wie Jura oder Medizin enthalten. Auch werden die Attribute der Schrift wie Fett und Kursiv meist automatisch übernommen. Die meisten Anbieter werben für ihre Programme mit Erkennungsraten von über 99% oder nahezu 100% bei guten Vorlagen. Die meisten Programmen besitzen auch eine Funktion um schlecht erkannte und neue Zeichen zu trainieren. Bei schlechter Qualität der Vorlage oder ungewöhnlichen Schriftarten sinken die Erkennungsraten aber schnell, so auch beim Erkennen von Fraktur, welche sich aus den in Frühdrucken verwendeten Schriften entwickelt hat, aber vom Schriftbild den modernen Schriftarten ähnelt.

Eine Ausnahme stellt FineReader XIX dar, diese Version basiert auf FineReader 7.0 und wurde erweitert, um mit den Besonderheiten der Frakturschriften von 1800 bis 1938 umgehen zu können. So mussten einerseits besondere Buchstabenformen, wie das lange-s und die Ligaturen, andererseits auch Änderungen in der Sprache selbst beachtet werden. Hierzu wurden auch für fünf europäische Sprachen Sprachmodelle dieser Zeit erstellt [vgl. 2]. Durch die Unterstützung einiger Eigenheiten der Fraktur, welche auch in den in Frühdrucken verwendeten Schriftarten auftreten, soll auch die Möglichkeit untersucht werden mit kommerziellen OCR-Programmen die Frühdrucke zu erkennen.

Es sind auch einige freie OCR Programme verfügbar, welche aber bei Weitem nicht die Leistungsfähigkeit der kommerziellen Programme erreichen. Von Google wurde auch ein freies OCR Programm veröffentlicht, welches auf dem zwischen 1985 und 1995 von Hewlett Packard entwickelten und 2005 freigegebenen Tesseract basiert. Obwohl Tesseract seit 1995 nicht mehr weiterentwickelt wurde, soll es momentan das beste freie OCR Programm sein [vgl. 13].

3 Analyse vorhandener OCR Lösungen

Vor der Implementierung einer eigenen Lösung sollten die am Lehrstuhl vorhandenen OCR-Programme, OmniPage und FineReader, auf ihre Eignung untersucht werden, Frühdrucke zu erkennen.

Es war nicht zu erwarten, dass die Programme von sich aus in der Lage sind die Texte zu erkennen. Beide bieten jedoch die Möglichkeit Zeichen zu trainieren.

3.1 OmniPage

Zuerst wurde die Erkennung mit OmniPage getestet. Wenn man versucht, den Text mit den Standarteinstellungen und Latein als Sprache ohne Training zu erkennen, werden nur ca. 30% der Buchstaben richtig erkannt. Die Textblöcke im Bild wurden richtig erkannt, wie in Abbildung 3.3 zu sehen ist, ist der erkannte Text aber falsch ausgerichtet.

Beim Trainieren muss die Seite auch zuerst normal eingelesen werden, dann können einzelne Buchstaben im erkannten Text ausgewählt und trainiert werden. Hierzu wird für ein Zeichen im Kontextmenü **Zeichen trainieren...** aufgerufen, wie auch in Abbildung 3.3 zu sehen. Daraufhin öffnet sich ein neues Fenster, siehe Abbildung 3.1, in welchem ein Ausschnitt aus der aktuellen Zeile gezeigt wird und darunter ein vergrößertes Ausschnittsfenster, in welchem das von OmniPage vermutete Zeichen blau markiert ist und die angrenzenden Zeichen gelb markiert sind. Durch Klicken in die gelben Bereiche kann man diese zur Markierung hinzufügen, wobei OmniPage die Größe des hinzuzunehmenden Bereichs automatisch und manchmal intuitiv nicht nachvollziehbar festlegt. Genauso können die schon blau markierten Bereiche durch Anklicken wieder entfernt werden, hierbei verkleinert sich der angezeigte Ausschnitt automatisch und falls einmal zu viel entfernt wurde, kann es passieren, dass keine gelben Bereiche mehr im Bild sind um die Auswahl wieder zu vergrößern. In diesen Fällen muss der Dialog für dieses Zeichen erneut aufgerufen werden. Manchmal werden auch schon trainierte Zeichen zu der markierten Auswahl hinzugenommen, in Abbildung 3.2 sollte das zweite v in provocavit, welches als U erkannt wurde, ausgewählt und trainiert

werden. Im anfänglichen Dialog waren av gemeinsam markiert, obwohl das a direkt vorher schon trainiert wurde. Die Markierung des a ließ sich auch nicht entfernen, es konnte nur das v aus der Markierung entfernt werden. Durch diese Probleme können sich berührende Zeichen nur schlecht trainiert werden.

Abbildung 3.1: In OmniPage einen Buchstaben trainieren

Abbildung 3.2: Probleme in Omnipage ein Zeichen zu trainieren

Nachdem ein Zeichen trainiert wurde, korrigiert OmniPage dieses Zeichen im gesamten Text und listet alle Änderungen zur Kontrolle auf. Für die fette Schrift in den Frühdrucken ist die Kontrollansicht, siehe Abbildung 3.4, zu klein und es lassen sich kaum Zeichen erkennen. Für die leicht verwechselbaren Zeichen ist es unmöglich sie

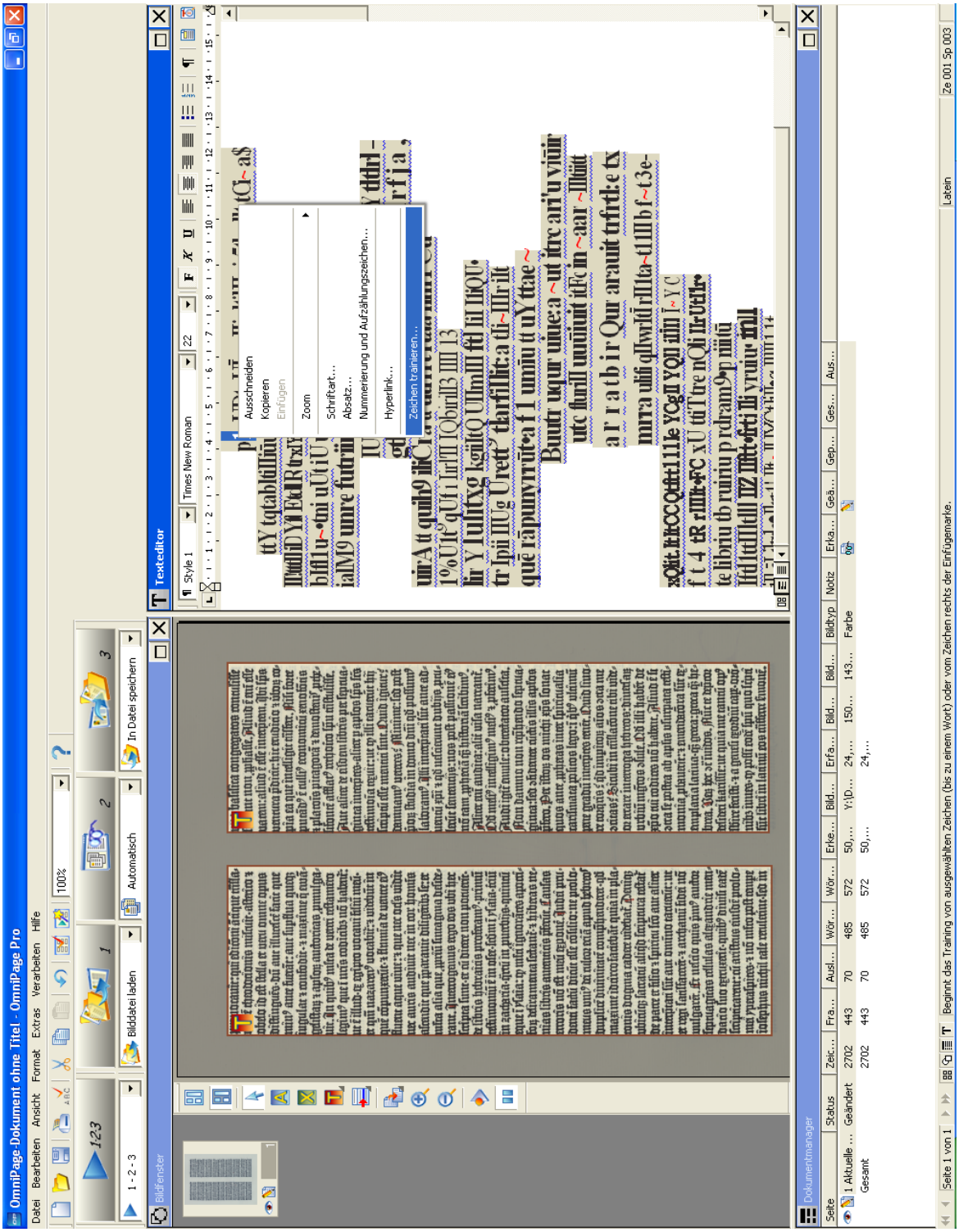

Abbildung 3.3: Oberfläche von OmniPage

noch zu unterscheiden. Als Hilfe wird für jedes Wort auch noch der Kontext angezeigt, durch die geringe Erkennungsrate von ca. 30% ist dieser aber auch nicht hilfreich.

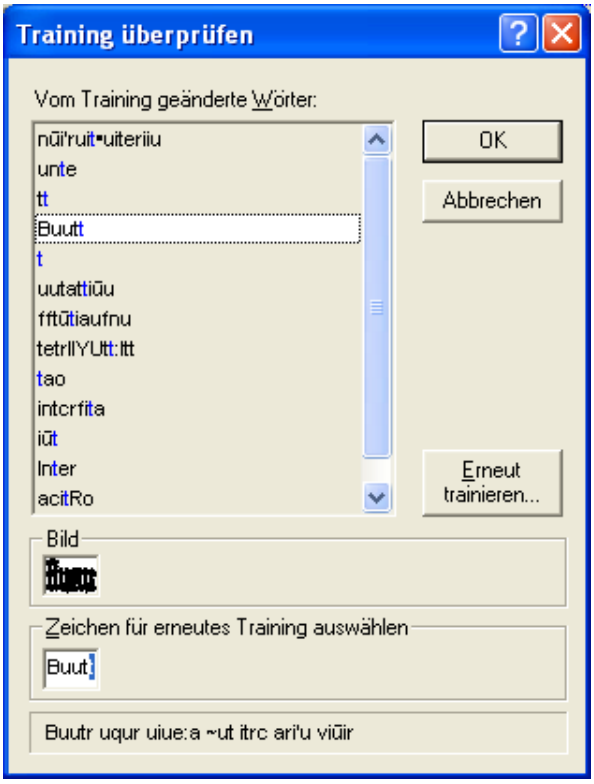

Abbildung 3.4: Kontrollansicht von OmniPage beim trainieren

Das Training von OmniPage scheint darauf ausgelegt zu sein, das Erkennungsalphabet um einzelne zusätzliche Zeichen zu erweitern, z.B. mathematische Sonderzeichen, oder schwer erkennbare Varianten von Zeichen hinzuzufügen nicht aber um eine unbekannte Schriftart zu trainieren. Dafür spricht zum einen, dass die Zeichen nicht systematisch trainiert werden, sondern jedes zu trainierende Zeichen ausgewählt werden muss und dass die eingebauten Muster nicht deaktiviert werden können und so immer mit zur Erkennung verwendet werden. Hierdurch und durch die Probleme beim Markieren zusammenhängender Zeichen und dass sich manche Zeichen nicht zum Trainieren auswählen lassen, ist OmniPage nicht für die Erkennung von Frühdrucken geeignet.

3.2 FineReader

Auch mit FineReader wurde zuerst die Zeichenerkennungsrate ohne Training getestet, die Erkennungssprache wurde auf Latein gestellt und für **Print type** wurde **Automatic** ausgewählt. FineReader hatte hierbei Probleme die Textblöcke richtig zu erkennen, siehe Abbildung 3.5, sie wurden in mehrere Teile zerlegt und Teile des Textes wurden als Bild eingeordnet. In den erkannten Textteilen oder wenn die Textblöcke vor dem Erkennen markiert wurden, betrug die Zeichenerkennungsrate ca. 50%. Wenn der **Print type** auf **Gothic** umgestellt wurde, ergab sich das gleiche Bild.

3.2.1 Training

Das Trainieren bei FineReader gestaltet sich komplett anders als bei OmniPage, hier muss zuerst bei den Erkennungsoptionen, siehe Abbildung 3.6, welche unter **Read** → **Options...** erreicht werden können oder unter **Tools** → **Options...** und dem Reiter **Recognition Train user pattern** ausgewählt werden. Zusätzlich muss man auch noch **Use built in patterns** deaktivieren, da FineReader sonst beim Training und der Zeichenerkennung auch nach Übereinstimmungen mit seinen eingebauten Mustern sucht, dadurch werden Zeichen welche mit den eingebauten Mustern übereinstimmen beim Trainieren übersprungen. Danach lässt man die erste Seite von FineReader erkennen. Durch die Umstellung auf **Train user pattern** wird nun der Text nicht automatisch erkannt, sondern ein Fenster mit der ersten Zeile angezeigt, siehe Abbildung 3.7, in welchem man nacheinander alle Zeichen markieren und ihnen den passenden Buchstaben zuordnen muss, dies wiederholt sich für alle Zeilen der Seite. Für schon trainierte Zeichen schlägt FineReader die passenden Buchstaben vor und überspringt sie nach ausreichendem Training. Durch die Ligaturen und die zusammenhängenden Zeichen ist es nicht immer möglich ein einzelnes Zeichen auszuwählen, in diesen Fällen werden die nicht einzeln auswählbaren Zeichen als Ligatur trainiert. Die in den Frühdrucken vorkommenden Abbreviaturen wurden als {<Grundzeichen>;<Abbreviaturzeichen>} trainiert, da keine geschweiften Klammern in den Texten vorkommen, können diese so später leicht weiterverarbeitet werden. Fehler beim Trainieren können, solange man sich noch im selben Wort befindet, korrigiert werden. Man geht mit dem Back-Button zurück bis zur fehlerhaften Stelle und trainiert von da ab die Zeichen neu, an Leerzeichen wird diese Funktion deaktiviert.

Das Trainieren einer Seite mit 3000 Zeichen benötigt gut 90 Minuten, daher ist dieses Vorgehen nur sinnvoll wenn eine größere Anzahl von Seiten erkannt werden soll. Die trainierten Musterdaten werden automatisch im aktiven Batch gespeichert, falls man diesen nicht speichert, wird er beim Beenden von FineReader mitsamt der Musterdaten gelöscht. Über die Schaltfläche **Pattern Editor...** im **Recognition** Reiter erhält man einen Überblick über die trainierten Muster und kann deren Eigenschaften bearbeiten,

Abbildung 3.5: FineReader

siehe Abbildung 3.8 und 3.9. Man kann hier noch nachträglich der einem Muster zugeordnete Buchstabe und die Formatierungen ändern oder ein Muster löschen.

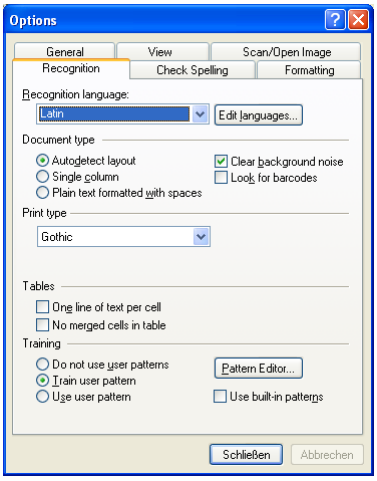

Abbildung 3.6: Erkennungsoptionen

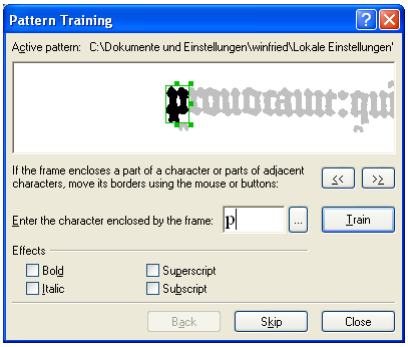

Abbildung 3.7: Trainingsfenster

Um die trainierten Muster zu verwenden, muss man im Recognition Reiter Use user pattern auswählen und die restlichen Seiten erkennen lassen. Bei den hier verwendeten Dokumenten hatte FineReader Probleme die Textblöcke richtig zu erkennen. Wie auch schon im untrainierten Zustand mussten diese manuell ausgewählt werden. Die Zeichenerkennungsrate lag nun bei ca. 75%.

Dokumente mit einer anderen Auflösung oder Schriftgröße als der zum Trainieren verwendeten, können mit den erstellten Mustern nicht erkannt werden. Die Schriftart muss natürlich auch übereinstimmen.

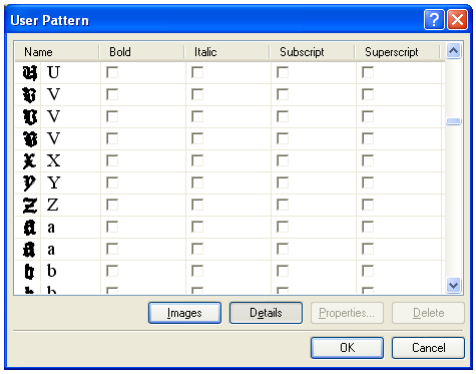

Abbildung 3.8: User Pattern

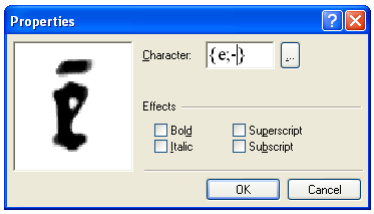

Abbildung 3.9: Pattern Properties

3.2.2 Auswirkungen des Wörterbuches

Da FineReader auch die Möglichkeit bietet ein Benutzerwörterbuch anzulegen, wurde auch versucht hiermit die Erkennungsrate zu steigern. Um die maximal mögliche Verbesserung zu erreichen wurde für eine Seite ein Wörterbuch zusammengestellt, welches genau die auf der Seite verwendeten Wörter enthielt. Die Erkennungsrate blieb aber auch mit dem Wörterbuch annähernd gleich, es wurden zwar einige Fehler gegenüber der Erkennung ohne Wörterbuch korrigiert, dafür traten aber auch neue Fehler auf. Bei diesen Korrekturen und neuen Fehlern ließ sich kein System feststellen. Die einzige Auffälligkeit war, dass sie sich größtenteils auf als unsicher markierte Buchstaben beschränkten, wobei die unsicheren Buchstaben in den beiden Versionen nicht an allen Stellen übereinstimmten.

Da auf den getesteten Seiten nur wenige Zeichen, ca. 3%, als unsicher markiert und von diesen über die Hälfte richtig erkannte Zeichen waren, wurde die Verwendung des Wörterbuchs nicht weiterverfolgt.

3.2.3 Auswertung

Durch das Trainieren konnte die Zeichenerkennungsrate deutlich von ca. 50% auf ca. 75% erhöht werden. Es fällt schnell auf, dass sich die Fehler in wenige Klassen einteilen lassen.

Die größten Probleme bereitet, bedingt durch die kleinen Wortzwischenräume in dem Text, das Erkennen der Leerzeichen, dies macht fast ein Drittel der Gesamtfehler aus.

Gut ein Fünftel der Fehler besteht darin, dass FineReader keine Übereinstimmung mit den trainierten Mustern feststellen kann und ein ^, für ein nicht erkanntes Muster, ausgibt. Dieser Fehler entsteht meist durch das falsche Segmentieren der Zeichen. So werden manchmal i-Punkte oder Überstriche aus der darunter liegenden Zeile zu einem Zeichen hinzugenommen. Manchmal werden nur Teile des Zeichens oder mehrere Zeichen zusammen versucht zu erkennen. Auffallend häufig wurde der Bindestrich am Zeilenende und das Abbreviaturzeichen für -ue, zu sehen in 2.2 auf Seite 8, nicht erkannt, dies konnte auch durch zusätzliches Trainieren nicht verbessert werden. Einige Beispiele hierzu sind in Abbildung 3.10 zu sehen.

Abbildung 3.10: Zeichen die FineReader nicht erkennen konnte

Fast genauso häufig sind Fehlsegmentierungen von i, m, n, v, w und Kombinationen dieser Buchstaben. Es kann vorkommen, dass ein Buchstabe in mehrere Zeichen zerlegt wird, so kann ein m als ni erkannt werden oder der umgekehrte Fall tritt ein und mehrere Buchstaben werden zu einem Zeichen kombiniert, z.B. vi zu m. Zudem kann auch ein Teil aus dem darauffolgenden Zeichen fälschlicherweise zu dem vorherigen Zeichen hinzugenommen werden, so kann nn als mi erkannt werden. Dieser Fehler tritt so häufig auf, da alle diese Zeichen aus kräftigen senkrechten Strichen bestehen, welche gegebenenfalls oben oder unten leicht verbunden sind, manchmal sind diese Verbindungen auch nicht zu sehen. Die Anzahl der senkrechten Striche wird von FineReader richtig erkannt aber oft nicht, welche der Striche wie verbunden sind. Zu dieser Gruppe wurden auch Fehlsegmentierungen gezählt, bei denen der Teil eines m, n, v oder w als c, r oder t erkannt oder daraus gebildet wurde, z.B. m als tn oder rn als m erkannt wurde. Beispiele hierzu sind in Abbildung 3.11 zu sehen.

Des Weiteren wurden auch Zeichen komplett übersehen, besonders häufig betraf dies den Punkt in der Zeilenmitte. Wenn ein Zeichen übersehen wurde, wurde an dieser Stelle auch kein Leerzeichen eingefügt, sondern die Zeichen davor und danach folgten direkt aufeinander. Häufiger als das komplette Übersehen von Zeichen wurden die

Abbildung 3.11: Fehlsegmentierungen, durch die vi als m und m als ni erkannt wurden

Abbreviaturzeichen übersehen und nur das Grundzeichen erkannt. Also für ē oder n̈ werden nur e oder n erkannt.

Eine weitere Fehlerquelle ist die Verwechslung von ähnlichen Zeichen. Wie in Abbildung 3.12 zu sehen, sind sich c, r und t sehr ähnlich, noch schwieriger wird die Unterscheidung, wenn diese das vorausgehende oder nachfolgende Zeichen berühren. Weitere Gruppen schwer zu unterscheidender Zeichen bilden f, ī, l und das lange-s sowie h und b. Vereinzelt traten auch Verwechslungen zwischen e und c auf, wenn der Strich beim e kaum oder gar nicht zu erkennen war, eine Übersicht über alle in der Gutenbergbibel verwendeten Drucktypen ist in Abbildung 2.4 zu sehen.

Abbildung 3.12: Die in der Gutenbergbibel verwendeten Typen für c, r und t

Eine genaue Auflistung der Fehlerhäufigkeiten der einzelnen Klassen, wie sie auf fünf Seiten der Gutenbergbibel auftraten, findet sich in Tabelle 3.1.

Nicht erkannte Leerzeichen	32,66%
Nicht erkannte Zeichen	21,06%
Fehlsegmentierungen von i, m, n, v, w und Kombinationen dieser Buchstaben	17,97%
Übersehene Abbreviaturen	8,02%
Verwechslung von f, ī, l, langes-s	5,70%
Verwechslungen von c, r und t	3,19%
Übersehene Zeichen	3,09%
Verwechslung von h und b	1,16%
Sonstiges	7,15%

Tabelle 3.1: Aufteilung der Erkennungsfehler von FineReader

Es hat sich gezeigt, dass die Zeichenerkennungsfehler in einzelnen Worten gehäuft auftreten, z.B. führt die Erkennung von ni als m zu zwei Zeichenfehlern. So wurden trotz einer Zeichenerkennungsrate von nur ca. 75% etwa 65% der Worte richtig erkannt. Bei einer zufälligen Verteilung der Fehler, einer Fehlerrate von 75% und einer Wortlänge von 4 Buchstaben wäre nur eine Worterkennungsrate von 31% zu erwarten. Durch die

Häufung von Fehlern wurde hier trotz einer längeren durchschnittlichen Wortlänge von 5,87 Buchstaben eine bessere Worterkennungsrate erreicht.

4 Entwurf

Wegen der hohen Erkennungsrate von FineReader sollte die Arbeit so angelegt werden, dass sie auch die Erkennungsroutinen von FineReader nutzen kann und dessen Erkennungsrate durch eine Vorverarbeitung des Ausgangsbildes und eine Nachverarbeitung des erkannten Textes steigert. Es können auch die Erkennungsergebnisse von FineReader und dem neu entwickelten Programm verglichen werden, um daraus auf den eigentlichen Text zu schließen. In Abbildung 4.1 ist der Ablauf der einzelnen Schritte der Implementierung dargestellt.

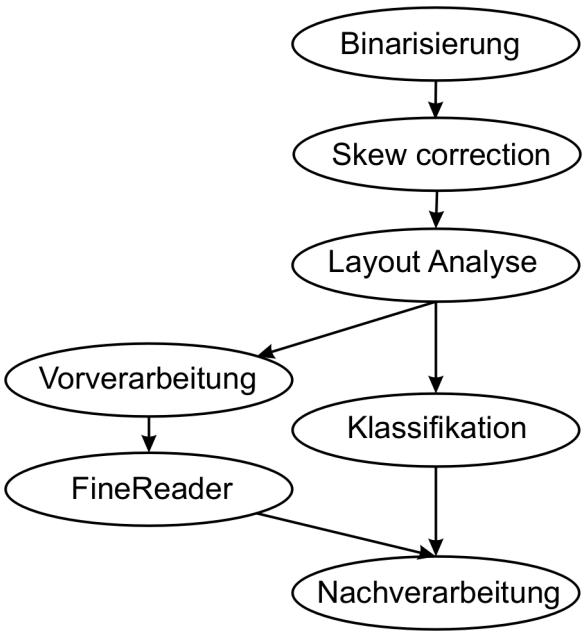

Abbildung 4.1: Ablauf

Die Implementierung erfolgt in Java 5 mit ImageJ [27]. ImageJ ist ein Bildverarbeitungsprogramm, welches schon viele elementare Bildverarbeitungsoperationen, wie Mittelwert-, Minimum-, Median-, Maximum-Filter und auch fortgeschrittenere Operationen wie das Skelettieren[1] eines Binärbildes, implementiert hat. Das Programm kann außerdem durch Plugins erweitert werden, wovon hier Gebrauch gemacht wurde.

[1] Ausdünnen von Segmenten auf eine ein Pixel breite Zentrallinie

Ein weiterer Vorteil von ImageJ ist, das sich seine Bildverarbeitungsbibliothek auch ohne die Oberfläche nutzen lässt und das Programm gemeinfrei ist, so lassen sich die entwickelten Routinen später in jedes Programm integrieren.

4.1 OCR Erkennungsroutine

Durch die Komplexität des Themas stand nicht die Zeit zur Verfügung, für die einzelnen Schritte mehrere Verfahren auszuprobieren und diese genauer zu untersuchen, die Schritte zur Vorverarbeitung, Layout Analyse, Klassifikation und Nachverarbeitung würden jeder für sich alleine genug Stoff für eine eigene Arbeit bieten. Daher wurde bei der Auswahl der Verfahren für die einzelnen Schritte auch auf die möglichst reibungslose Implementierbarkeit geachtet. Zusätzlich musste auch noch die Zusammenarbeit mit FineReader beachtet werden.

4.1.1 Voraussetzungen

Die implementierten Algorithmen setzen bei dem zu erkennenden Text einige Eigenschaften voraus. Die Buchstaben sollten sich nicht berühren, insbesondere sollten sich keine Buchstaben aus verschiedenen Worten berühren. Mit mehreren kleinen, sich berührenden Buchstabengruppen und einigen größeren kommen die Algorithmen aber dennoch problemlos zurecht. Des Weiteren werden parallele Zeilen mit einheitlicher Schriftgröße erwartet. Auf diese und einige weitere Annahmen wird genauer in Kapitel 5, bei den Algorithmen die davon betroffen sind, eingegangen. Die Frühdrucke erfüllen die hier vorausgesetzten Eigenschaften und solange nicht mit besonderen gestalterischen Mitteln oder ausgefallenen Schriftarten gearbeitet wird, erfüllen auch heutige Texte diese Voraussetzungen.

Die hier vorgestellte Methode zur Schrifterkennung lässt sich nicht für Handschriften verwenden, auch Blockschrift kann hiermit nicht erkannt werden. Die Erkennung von Handschriften, insbesondere historischen Handschriften, stellt ein weitaus schwierigeres Problem dar, als die Erkennung gedruckter Texte. Bei historischen Handschriften wird z.B. versucht eine Transkription Wort für Wort dem Dokument zuzuordnen. Die in [17] beschriebene Methode erreicht hier eine Treffergenauigkeit von 74,5%, wenn die Transkription Zeile für Zeile zugeordnet wird, bei der seitenweisen Zuordnung wird eine Treffergenauigkeit von 60,5% erreicht.

4.1.2 Binarisierung

Zur Binarisierung stehen, wie schon in Abschnitt 2.3.1 erwähnt, viele Verfahren zur Verfügung. Da die Verfahren hier zur Binarisierung von eingescanntem Text verwendet werden und diese typischerweise gleichmäßig ausgeleuchtet sind und ein Bimodales Histogramm besitzen, wie in Abbildung 4.2 zu sehen, eignen sich hier globale Verfahren wie das von Otsu oder ein iteratives Verfahren zur Suche eines Minimums zwischen den beiden Maxima. Da das Verfahren von Niblack in [31] am besten bei der Binarisierung von Handschriften abgeschnitten hat und man erst an der Erkennungsrate des Gesamtsystems erkennen kann welches Verfahren am besten geeignet ist, wurden alle drei Verfahren implementiert.

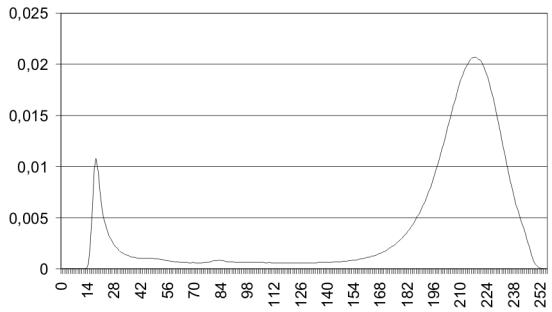

Abbildung 4.2: Bimodales Histogramm

4.1.3 Skew-Detection

Als nächstes wird aus dem segmentierten Schwarz-Weiß Bild die Verdrehung berechnet, dazu wird berechnet welche Segmente benachbarte Buchstaben innerhalb einer Zeile sein könnten. Aus den Richtungen der Verbindungslinien dieser Segmente wird ein Histogramm gebildet. Das Maximum im Histogramm gibt den Winkel der Verdrehung an.

Da durch eine Rotation zur Korrektur der Verdrehung Artefakte entstehen würden, siehe Tabelle 4.1, wird die Verdrehung nicht korrigiert, sondern der Winkel um den das Bild verdreht ist wird an die nächsten Stufen weitergegeben. Dies ist insbesondere für die weitere Verarbeitung mit FineReader wichtig, da dieser selbst auch eine Skew-Correction durchführt. Falls dessen Algorithmus für den hier geradegerichteten Text noch eine Verdrehung feststellt, würde das Bild noch einmal rotiert und es würden unnötige Artefakte hinzukommen

Falls eine Stufe das richtig ausgerichtete Bild benötigt, muss sie mit dem bekannten Verdrehungswinkel die Rotation rückgängig machen und falls nötig für eine Beseitigung der Artefakte sorgen. Wenn die Artefakte, deren Größe 1 Pixel beträgt, im

Um 30°rotiertes B Das um 30°rotierte B geradegerichtet

Tabelle 4.1: Artefakte

Vergleich zur Strichdicke klein sind, können diese beispielsweise durch einen Median-Filter entfernt werden.

4.1.4 Layout Analyse

Zur Layout Analyse wurden in Abschnitt 2.3.2 drei Verfahren erläutert. Die Verfahren basierend auf der Texturanalyse könnten bei den Frühdrucken durch die breiten Striche der Zeichen und das dunkle Schriftbild Probleme bei der Unterscheidung von Text und Zeichnungen bereiten. Eine Untersuchung dieses Sachverhalts und gegebenenfalls die Suche nach anwendbaren Texturanalyseverfahren und passenden Parametern könnten den Rahmen dieser Arbeit sprengen, daher kommen nur die anderen beiden Verfahren für eine Implementierung in Frage.

Das rekursive X-Y-Cut Verfahren und das Verfahren zur Suche nach maximalen leeren Rechtecken sind beide gut geeignet zum zerlegen von Seiten im Manhattan-Layout[2], welches auch in den Frühdrucken verwendet wird. Bei dieser Implementierung wird das rekursive X-Y-Cut Verfahren zur Erkennung der Textblöcke eingesetzt, da mit ihm auch die Zeilen ohne größeren Aufwand aus den Textblöcken extrahiert werden können. Beim Verfahren zur Suche nach maximalen leeren Rechtecken, müsste noch ein extra Verfahren zum Finden der Zeilen implementiert werden. Bei ausreichendem Zeilenzwischenraum, welcher bei Frühdrucken meist nicht vorhanden ist, könnten auch die Zeilen mit dieser Methode getrennt werden.

4.1.5 Zeichensegmentierung

Durch sich berührende Buchstaben und Buchstaben bei denen Verbindungslinen fehlen, welche also in mehrere Segmente zerfallen, ist das auffinden von Buchstabengrenzen schwierig. Deswegen wird eine Übersegmentierung der Buchstaben vorgenommen. Ein in drei Segmente zerfallenes **m**,wobei dessen vorderes Segment mit dem vorhergehenden **e** verbunden ist, ist in Abbildung 4.3 zu sehen. Die Buchstaben bestehen meist aus kräftigen vertikalen Strichen und filigranen horizontalen Verbindungen. Hierdurch

[2]Beim Manhattan-Layout besteht eine Seite nur aus parallel angeordneten rechteckigen Textblöcken.

eignet sich zum Segmentieren eine Projektion auf die x-Achse. An den Minima der Projektion wird eine Buchstabengrenze angenommen. Durch dieses Verfahren werden so z.B. zwei Buchstabengrenzen innerhalb eines m und eine Grenze innerhalb von n, v und o vermutet.

Abbildung 4.3: Das Wort eisdem mit einem in drei Segmente zerfallenenem m

4.1.6 Klassifikation

Durch die große Bandbreite der in Frühdrucken verwendeten Schriftarten ist es unmöglich eine kleine Gruppe von Merkmalen zu bestimmen anhand derer sich die Zeichen erkennen lassen. Es müssten viele Merkmale berechnet werden und beim Trainieren die für diese Schriftart aussagekräftigen bestimmt werden. Daher wird hier das Template Matching verwendet, bei welchem für jeden Buchstaben eine oder mehrere Vorlagen gespeichert werden und das zu erkennende Zeichen mit allen Vorlagen verglichen wird um die mit der größten Übereinstimmung zu finden. Um die Anzahl der Vergleiche zu reduzieren, werden die Zeichen nur mit Vorlagen ähnlicher Breite verglichen und zum Weiteren reduzieren der Vergleiche wird Zoning verwendet.

4.1.7 Nachverarbeitung

Die Nachverarbeitung erfolgt Lexikonbasiert mithilfe der erweiterten Levenstheln-Distanz. Denn bei dieser kann man auch die Wahrscheinlichkeiten der verschiedenen möglichen Buchstaben, welche bei der Klassifikation ermittelt wurden, mit einfließen lassen. Da diese Informationen bei FineReader nicht zur Verfügung stehen, kann man nur aus den Fehlern die gemacht wurden statische Wahrscheinlichkeiten für die Verwechslungshäufigkeit von Zeichen gewinnen, was aber aufwendig ist. Daher wird als Alternative versucht, sowohl das erkannte Wort als auch die Lexikoneinträge zu normieren und diese dann zu vergleichen. Zur Normierung werden für eine Gruppe leicht verwechselbarer Buchstaben alle Vorkommen dieser Buchstaben in einem Wort durch ein Zeichen ersetzt. Alle Lexikoneinträge, deren normierte Form mit der des erkannten Wortes übereinstimmen, sind mögliche Korrekturen. Dieses Verfahren kann auch genutzt werden, um in den von FineReader erkannten Texten zu suchen. Solange in dem gesuchten Wort nur die in Abschnitt 3.2.3 erwähnten Fehlsegmentierungen, vergessenen Abbreviaturzeichen und Gruppen von Verwechslungen vorkommen, wird jedes Auftreten des Wortes gefunden.

4.2 Kombination mit FineReader

Von den in Tabelle 3.1 aufgeführten Fehlern lassen sich die nicht erkannten Leerzeichen und ein Teil der nicht erkannten Zeichen durch eine Vorverarbeitung des Bildes vermeiden. Die Leerzeichen werden aufgrund des geringen Wortabstandes nicht erkannt und daher wird dieser vergrößert. Ein Teil der nicht erkannten Zeichen kommt dadurch zustande, dass FineReader i-Punkte und Überstriche aus darrunterliegenden Zeilen mit zur Zeile hinzunimmt, um dies zu verhindern wird auch der Zeilenabstand vergrößert. Die hierzu nötigen Informationen werden in der Layout Analyse ermittelt, daher wird nach diesem Schritt die Vorverarbeitung für FineReader eingeschoben.

Das vorverarbeitete Bild wird dann durch FineReader erkannt und die Ergebnisse von FineReader werden nachverarbeitet um Fehlsegmentierungen, übersehene Abbreviaturen und Zeichenverwechslungen zu korrigieren. Hierzu werden die erkannten Worte, falls sie nicht im Wörterbuch vorkommen, unter Berücksichtigung der typischen Fehler mit einem Wörterbuch verglichen um mögliche Kandidaten für eine Korrektur zu finden. Werden mehrere Kandidaten gefunden, so können diese noch über den Kontext selektiert werden. Es kann auch über die Auftrittswahrscheinlichkeiten der einzelnen Fehler die wahrscheinlichste Korrektur ermittelt werden.

5 Implementierung

In diesem Kapitel werden zwei verschiedene Implementierungen beschrieben, da bei der ersten Implementierung einige Probleme aufgetreten sind, welche sich nicht einfach umgehen ließen. Da die Zeilen auch in verschiedenen Spalten auf der gleichen Höhe liegen, sollten diese ohne vorherige Layout Analyse gemeinsam gefunden werden und erst bei der Erkennung des Textes, anhand der Lücke zwischen den Spalten getrennt werden. Bei der Erklärung der vorherigen Implementierung in Abschnitt 5.3 wird genauer auf die Probleme eingegangen. Es war am Ende der Arbeit jedoch nicht mehr genügend Zeit übrig, um die nachfolgenden Schritte an die neu entwickelte Layout Analyse anzupassen.

5.1 Vorverarbeitung

5.1.1 Binarisierung

Zur Binarisierung wurden drei Algorithmen implementiert. Zwei globale Verfahren, welche für das ganze Bild einen Schwellenwert berechnen und ein lokales Verfahren, welches für jeden Punk im Bild anhand seiner Umgebung einen eigenen Schwellenwert berechnet. Zusätzlich kann auch der in ImageJ vorhandene Binarisierungsalgorithmus verwendet werden.

Iteratives Verfahren

Bei dem Iterativen Verfahren muss zuerst ein Schwellenwert geschätzt werden um Vordergrund und Hintergrund zu trennen. Dies geschieht hier indem der Histogrammwert berechnet wird, welcher die dunkelsten 5% der Pixel enthält und der Histogrammwert, welcher die hellsten 5% der Pixel enthält. Der Mittelwert aus diesen beiden Werten ist die anfängliche Schätzung des Schwellenwertes. Dann werden die Mittelwerte für den Vordergrund und Hintergrund berechnet und der Mittelwert dieser beiden Mittelwerte ist der neue Schwellenwert. Dies wird iterativ so lange wiederholt, bis die Schwellenwerte in zwei aufeinanderfolgenden Schritten sich um maximal eins unterscheiden. Mit diesem Verfahren binarisierte Bilder sind in Abbildung 5.8 und 5.9 zu sehen.

Verfahren von Otsu

Das Verfahren von Otsu sucht den Schwellenwert, welcher die Varianzen im Vordergrund und Hintergrund minimiert, gleichzeitig aber die Varianz zwischen Vordergrund und Hintergrund maximiert. Sei $hist[0...255]$ das Histogramm und μ der Mittelwert des Histogramms, $\mu_V(s)$ und $\sigma_V^2(s)$ der Mittelwert und die Varianz des Vordergrunds, $\mu_H(s)$ und $\sigma_H^2(s)$ der Mittelwert und die Varianz des Hintergrunds und $P_V(s)$ und $P_H(s)$ die Pixelanzahl von Vordergrund bzw. Hintergrund bei gegebenem Schwellenwert s.

Die Varianz zwischen den Klassen ist nun $\sigma_{zw}^2 = P_V(s)(\mu_V(s)-\mu)^2+P_H(s)(\mu_H(s)-\mu)^2$ und die innerhalb der Klassen $\sigma_{in}^2 = P_V(s)\sigma_V(s)^2 + P_H(s)\sigma_H(s)^2$. Man berechnet nun für jedes s den Quotient $\frac{\sigma_{zw}^2}{\sigma_{in}^2}$ und wählt das s, für welches der Quotient maximal wird, als Schwellenwert. Mit diesem Verfahren binarisierte Bilder sind in Abbildung 5.10 und 5.11 zu sehen.

Verfahren von Niblack

Das Verfahren von Niblack berechnet für jeden Punkt (x, y) den Schwellenwert durch $T(x, y) = m(x, y) + ks(x, y)$, wobei $m(x, y)$ und $s(x, y)$ der Mittelwert und die Standartabweichung in einer lokalen Umgebung um den Punkt (x, y) sind. Die Größe der Nachbarschaft N und k sind Parameter des Verfahrens und müssen an die jeweils verwendeten Bilder angepasst werden.

In dieser Implementierung wird ein quadratisches Fenster um den Punkt (x, y) verwendet. Über alle Pixel in diesem Fenster wird ein Histogramm erstellt und aus diesem werden Mittelwert und Standartabweichung berechnet. Da sich bei aufeinanderfolgenden Punkten die Fenster größtenteils überlappen, bis auf die erste Spalte des vorderen und die letzte Spalte des hinteren Fensters, siehe Abbildung 5.1, werden zusätzlich zu dem Histogramm über die gesamte Nachbarschaft auch noch N Histogramme, eines für jede Spalte, verwendet. Wenn das Fenster nun um ein Pixel verschoben wird, wird vom Gesamthistogramm das Histogramm der ersten Spalte des alten Fensters abgezogen und gelöscht. Für die im neuen Fenster hinzugekommene Spalte wird ein neues Histogramm erstellt, zum Gesammthistogramm addiert und zu den Spaltenhistogrammen hinzugefügt. Hierdurch müssen nicht in jedem Schritt N^2 Punkte aus dem Bild gelesen werden, sondern immer nur N.

Wie in den Abbildungen 5.12 und 5.13 zu sehen ist, erzeugt der Algorithmus Artefakte wenn nur Vordergrund oder Hintergrund im Fenster sind. Die Fenstergröße sollte so gewählt werden, dass nie nur Vordergrund oder Hintergrund im Fenster liegen. Sollte es sich nicht vermeiden lassen, dass manchmal nur Hintergrund im Fenster liegt,

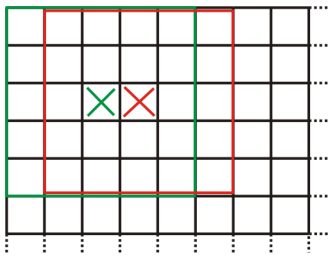

Abbildung 5.1: 5x5 Fenster um zwei aufeinanderfolgende Punkte

dann können die Artefakte im Hintergrund mit einer Nachverarbeitung entfernt werden und man erhält ein sauber binarisiertes Bild. Falls man nicht vermeiden kann, dass manchmal nur Vordergrund im Fenster liegt, weil man die Größen der Vordergrundsegmente nicht kennt und diese auch nicht vom Benutzer abfragen kann, so lässt sich dieses Verfahren nicht verwenden. Ein Algorithmus zum Entfernen von Artefakten im Hintergrund wurde auch implementiert.

Nachverarbeitung

Hier wurde ein Algorithmus implementiert um „Geister"-Segmente, welche in binarisierten Bildern auftauchen, zu entfernen. Der Algorithmus beruht darauf, dass an den Rändern von Segmenten die Helligkeit beim Übergang vom Hintergrund zum Vordergrund schnell signifikant abfällt. Bei Flecken hingegen fällt die Helligkeit nur langsam ab und wenn beim Verfahren von Niblack Strukturen im Hintergrund verstärkt wurden, ändert sich an diesen Stellen die Helligkeit kaum.

Dazu wird der durchschnittliche Gradient am Rand des Objektes mit dem Sobel-Operator [siehe 29, S. 176ff] berechnet und falls dieser eine vorgegebene Schwelle unterschreitet wird das Segment aus dem binarisierten Bild entfernt. Zusätzlich können hier auch Segmente welche eine vorgegebene Größe unterschreiten entfernt werden. Mit dem Verfahren von Niblack binarisierte und mit diesem Verfahren nachbearbeitete Bilder sind in Abbildung 5.16 und 5.17 zu sehen. Die Gradienten der binarisierten Bilder in Abbildung 5.14 und 5.15.

Dieser Schritt ist hauptsächlich dazu gedacht, die Artefakte des Verfahrens von Niblack zu entfernen. Bei dunklen Flecken im Bild kann er auch bei den anderen Verfahren die Ergebnisse verbessern.

Vergleich

Hier sind zum Vergleich zwei Beispielbilder mit den hier vorgestellten Algorithmen und dem in ImageJ eingebautem Verfahren binarisiert. Die Verfahren liefern alle ähn-

liche Ergebnisse, zu sehen in Tabelle 5.1, wobei aber auffällt, dass das Verfahren von Otsu den Schwellenwert höher ansetzt als die anderen Verfahren. Dadurch ist es anfälliger für Verschmutzung und zusammenhängende Buchstaben, kann aber verblassende Buchstabenteile besser erfassen. Die Histogramme zu den Beispielbildern, mit den eingezeichneten Schwellenwerten, sind in Abbildung 5.4 und 5.5 zu sehen.

Verfahren	Abb. 5.2	Abb. 5.3
ImageJ	141	140
Iterativ	117	121
Otsu	149	172

Tabelle 5.1: Schwellenwerte der Binarisierungsverfahren

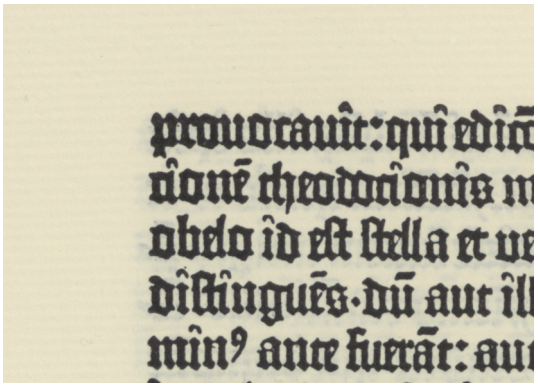

Abbildung 5.2: Ausschnitt aus der 42 zeiligen Gutenbegbibel *(Quelle: Universitätsbibliothek Würzburg)*

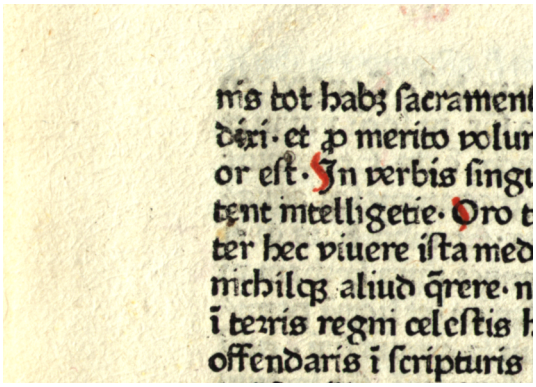

Abbildung 5.3: Ausschnitt aus einem weitern Bibeldruck *(Quelle: Universitätsbibliothek Würzburg)*

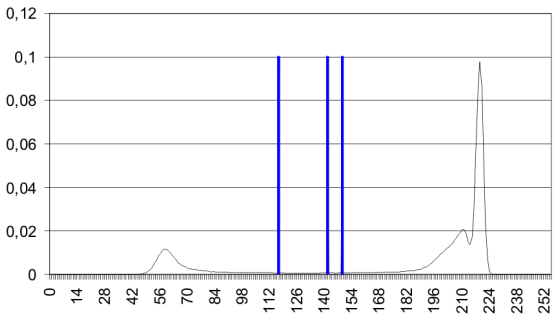

Abbildung 5.4: Histogramm von Abbildung 5.2 mit eingezeichneten Schwellenwerten

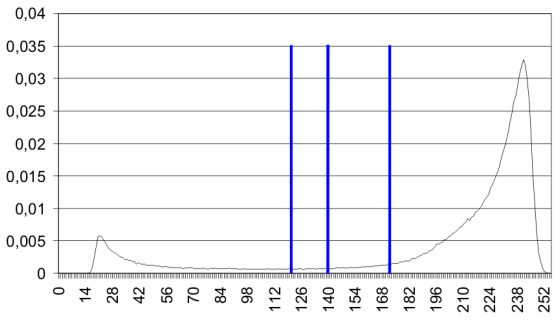

Abbildung 5.5: Histogramm von Abbildung 5.3 mit eingezeichneten Schwellenwerten

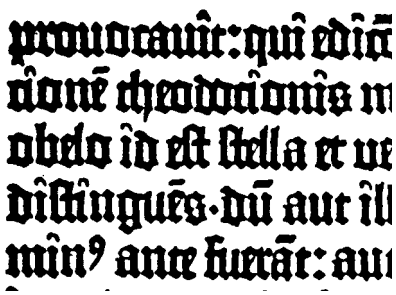

Abbildung 5.6: Mit ImageJ binarisierte Abbildung 5.2

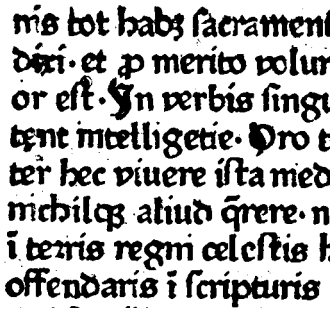

Abbildung 5.7: Mit ImageJ binarisierte Abbildung 5.3

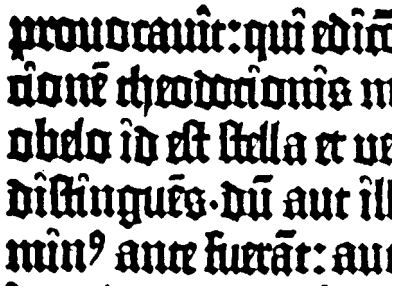

Abbildung 5.8: Mit dem iterativen Verfahren binarisierte Abbildung 5.2

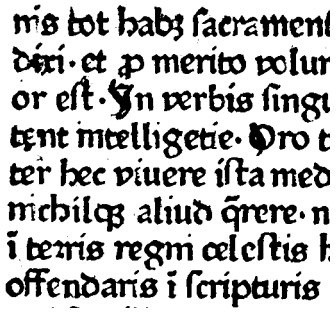

Abbildung 5.9: Mit dem iterativen Verfahren binarisierte Abbildung 5.3

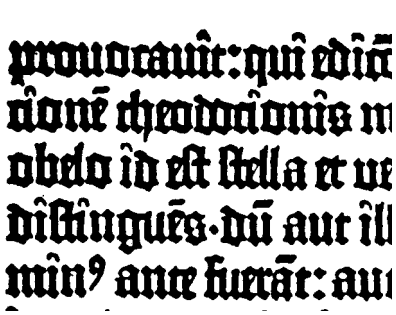

Abbildung 5.10: Mit dem Verfahren von Otsu binarisierte Abbildung 5.2

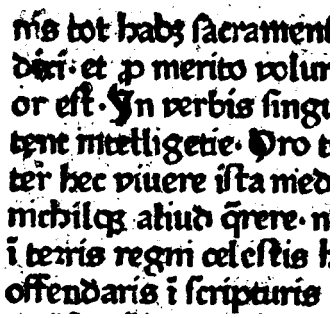

Abbildung 5.11: Mit dem Verfahren von Otsu binarisierte Abbildung 5.3

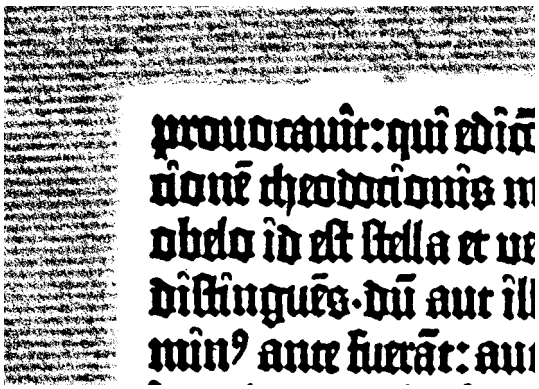

Abbildung 5.12: Mit dem Verfahren von Niblack binarisierte Abbildung 5.2, mit einem Fenster der Größe 101 und $k = -0.2$

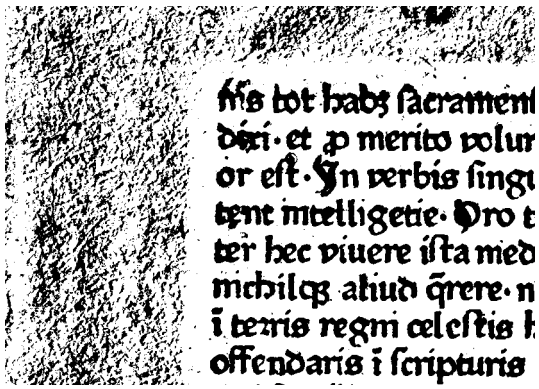

Abbildung 5.13: Mit dem Verfahren von Niblack binarisierte Abbildung 5.3, mit einem Fenster der Größe 101 und $k = -0.2$

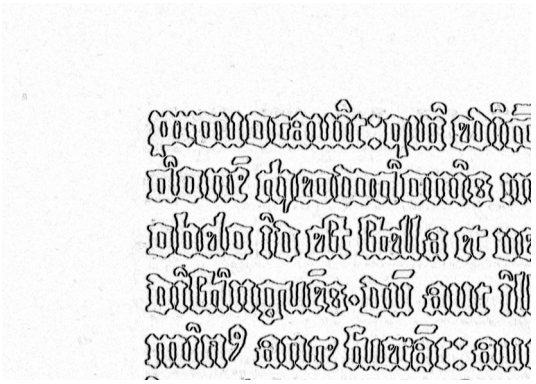

Abbildung 5.14: Gradient von Abbildung 5.2

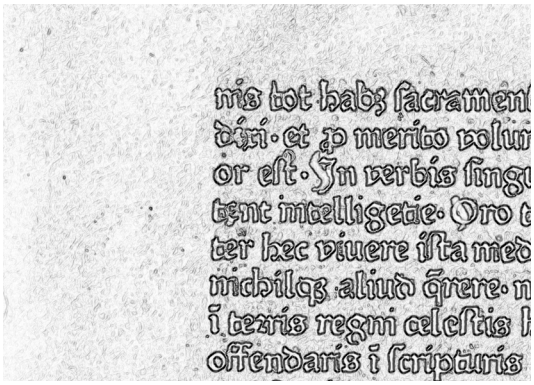

Abbildung 5.15: Gradient von Abbildung 5.3

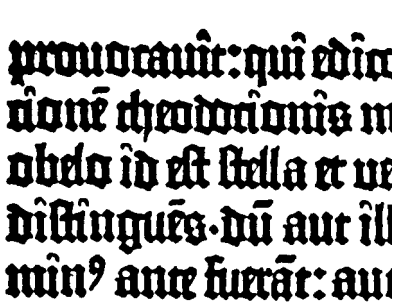

Abbildung 5.16: Mit dem Verfahren von Niblack binarisierte und nachbearbeitete Abbildung 5.2

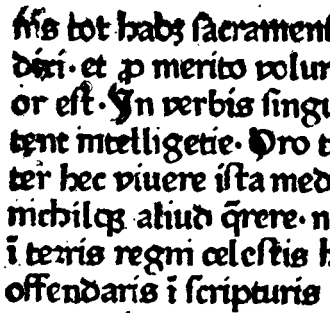

Abbildung 5.17: Mit dem Verfahren von Niblack binarisierte und nachbearbeitete Abbildung 5.3

5.1.2 Segmentierung

Die Nachfolgenden Schritte arbeiten nicht mehr direkt mit dem Bild, sondern nur noch mit den im Bild enthaltenen Segmenten, den Bounding-Boxes oder den konvexen Hüllen der Segmente. Dazu müssen die einzelnen Segmente des Bildes bestimmt werden. Hierzu wird das Bild zeilenweise verarbeitet und in jeder Zeile werden aufeinanderfolgende schwarze Pixel zu einem Run zusammengefasst, d.h. es wird gespeichert wo eine Folge von schwarzen Pixel beginnt und wie lange sie ist. Wenn der Run ein Segment aus der vorherigen Zeile berührt, wird er zu dem Segment aus der vorherigen Zeile hinzugefügt. Sollte ein Run mehr als ein Segment aus der vorherigen Zeile berühren, dann wird er zu einem der Segmente hinzugefügt und die anderen Segmente die er berührt werden auch zu diesem Segment hinzugefügt. Wenn der Run kein Segment berührt, wird er zu einem neuen Segment hinzugefügt. Nach einem Durchlauf sind alle Segmente des Bildes bestimmt.

5.1.3 Skew Detection

Zur Skew-Detection wird zuerst aus den Segmenten ein Graph aufgebaut, welcher für alle benachbarten Segmente eine Kante beinhaltet. Die benachbarten Segmente werden mittels einer Distanztransformation [siehe 29, S. 282f] bestimmt. Bei der Distanztransformation wird für jeden Punkt im Bild der kürzeste Abstand zu einem Segment bestimmt, hier wird zusätzlich noch gespeichert, welches Segment das nächste ist. Sind zwei Punkte benachbart, deren näheste Segmente sich unterscheiden, so sind diese Segmente benachbart.

Für die Kanten im Graphen wird nun bestimmt, ob sie zwei Buchstaben innerhalb einer Zeile verbinden. Für jede dieser Kanten wird der Vektor zwischen den Schwerpunkten der Segmente, welche die Kante verbindet, gebildet und seine Richtung bestimmt. Sollte die Richtung nicht zwischen $-90°$ und $90°$ liegen, wird die Richtung des negierten Vektors verwendet. Über die Richtungen wird ein Histogramm gebildet, da die Schwerpunkte der Segmente, wegen Ober- und Unterlängen, innerhalb einer Zeile nicht immer auf der gleichen Höhe liegen, wird das Histogramm noch geglättet. Das Maximum im geglätteten Histogramm gibt die wahrscheinlichste Verdrehung an. So können Verdrehungen zwischen $-90°$ und $90°$ erkannt werden. Für Verdrehungen außerhalb dieses Bereiches wird die Verdrehung um $180°$ falsch berechnet und das Bild auf den Kopf gestellt. In Abbildung 5.18 ist ein Ausschnitt aus einer Seite zu sehen, in der die Verbindungslinien zwischen benachbarten Segmenten eingezeichnet sind. Die Linien die wahrscheinlich aufeinanderfolgende Buchstaben verbinden sind grün eingezeichnet. Histogramme über die Linienrichtung und die geglättete Linienrichtung sind in den Abbildungen 5.19 und 5.20 dargestellt. Für diese Seite wurde eine Verdrehung von $-0,6°$ berechnet.

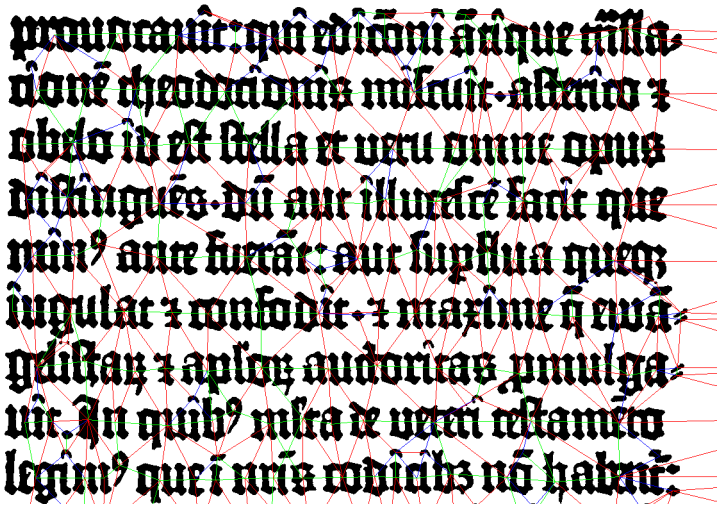

Abbildung 5.18: Verbindungen zwischen benachbarten Segmenten. Bei blauen Linien stimmt das Größenverhältnis zwischen den Segmenten nicht und bei roten Linien sind sie zu weit voneinander entfernt. Die grünen Linien sind die vermuteten Verbindungen zwischen Buchstaben.

Wie bestimmt wird, wann zwei Segmente als benachbarte Buchstaben angesehen werden, lässt sich über ein übergebenes Objekt steuern, welches eine Methode implementiert die für benachbarte Buchstaben true und ansonsten false zurückgibt. Falls nichts übergeben wird, werden Segmente von unter 10 Pixel Höhe als Buchstaben ausgeschlossen, da von gescannten Bildern ausgegangen wird und auf diesen kleinere Buchstaben nicht mehr zu erkennen wären. Die Höhe der benachbarten Segmente

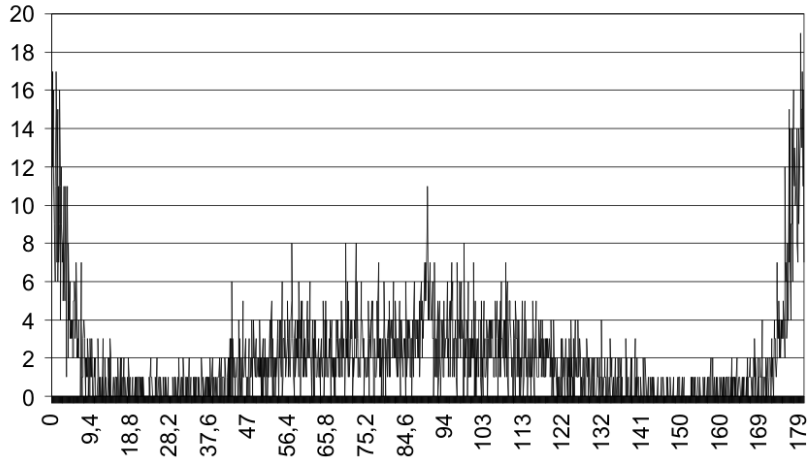

Abbildung 5.19: Histogramm der Verbindungsrichtungen der vermuteten benachbarten Buchstaben

kann auch genutzt werden, um diese als benachbarte Buchstaben auszuschließen. Die kleinsten Buchstaben besitzen weder Ober- noch Unterlängen und Buchstaben mit Ober- oder Unterlängen sind maximal doppelt so groß wie diese. Da bei Frakturschriften und Textura Ober- und Unterlängen häufiger sind und manche Buchstaben beides besitzen, wie das h in Abbildung 2.4 auf Seite 9, darf das größere Segment maximal um den Faktor $2,8$ größer sein. Als drittes wird der Abstand zwischen den Segmenten genutzt, um diese als benachbarte Buchstaben auszuschließen. Wenn Segmente mehr als die dreifache Breite des schmaleren oder mehr als die Höhe des kleineren Segments voneinander entfernt sind, werden diese ausgeschlossen.

Da sich bei eingescannten Dokumenten die Zeilenrichtung nicht exakt bestimmen lässt und die Zeilen nicht immer gerade verlaufen, z.B. bei eingescannten Büchern, wurden Bilder mit Text erzeugt und anschließend gedreht, um die Genauigkeit des Algorithmus zu testen. Für diese Bilder wurde dann der Skew berechnet, wobei sich über den gesamten Bereich von $-90°$ bis $90°$ eine durchschnittliche Abweichung von $0,1°$ und eine maximale Abweichung von $0,3°$ ergab. Die kumulierte Verteilung der Abweichungen ist in Abbildung 5.21 zu sehen.

Für die Skew-Detection ist es wichtig, dass sich nicht zu viele Buchstaben berühren, da sonst zu wenige Verbindungslinien zum Berechnen des Histogramms zur Verfügung stehen. Wenn sich Buchstaben berühren, werden dadurch aber auch die Einflüsse von Ober- und Unterlängen auf den Schwerpunkt das Segments gemindert, was zu einer höheren Genauigkeit führt.

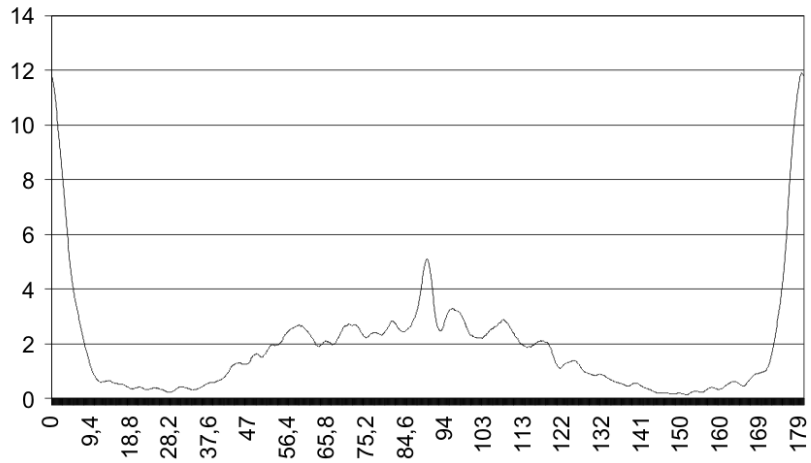

Abbildung 5.20: Geglättetes Histogramm der Verbindungsrichtungen der vermuteten benachbarten Buchstaben

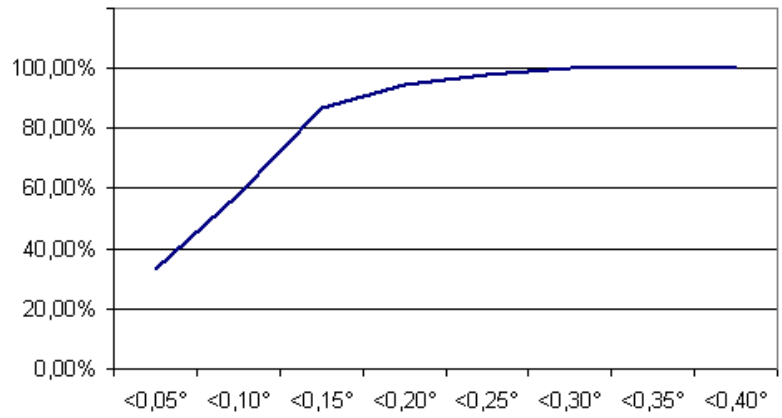

Abbildung 5.21: Kumulierte Verteilung der Abweichungen

5.2 Layout Analyse

Um das Bild in einzelne Textblöcke zu zerlegen wird um das Bild eine zum Text parallele Bounding-Box gelegt und die zur äußeren Bounding-Box parallelen Bounding-Boxen der einzelnen Segmente berechnet. Dann wird ein Histogramm über die auf die äußere Bounding-Box projizierten Bounding-Boxen der Segmente berechnet. Zu kleine und zu große Segmente werden ausgelassen. Hierbei wird nur die Höhe der Segmente betrachtet, da bei der Breite von einem einzelnen i bis zu den sich möglicherweise berührenden Buchstaben eines ganzen Wortes alles zugelassen werden muss. Um die minimale und maximale zugelassene Höhe der Segmente zu bestimmen, wird der Median der Segmenthöhen verwendet. Die Segmente müssen mindestens die halbe Höhe des Medians besitzen und dürfen maximal die zweieinhalbfache Höhe des Medians haben. Es werden hier die Bounding-Boxen verwendet, da dadurch der Einfluss von Bildern minimiert wird. In Bildern kommen mehr schwarze Pixel vor und daher würden diese zu Peaks im Histogramm führen, falls es über die projizierten Pixel berechnet werden würde.

Die Histogramme werden geglättet, dann wird deren maximaler Wert bestimmt und die Projektion in Segmente eingeteilt, welche weniger als 20% des Maximalwertes haben und in Segmente die mehr als 20% des Maximalwertes haben. Segmente die weniger als die durchschnittliche Buchstabenhöhe/breite haben werden mit den angrenzenden Segmenten verbunden. Anschließend wird für die einzelnen Segmente überprüft, ob diese keinen Text enthalten und benachbarte Segmente mit Text werden verbunden. Die Einteilung in Segmente funktioniert nur, wenn die Zeilen alle ungefähr parallel zur Bounding-Box verlaufen, ansonsten würde das Histogramm verschmieren. Solange die Projektionen von verschiedenen Zeilen sich nicht überlappen werden sie noch erkannt.

Um zu überprüfen, ob ein Segment Text enthält, wird der Graph mit den benachbarten Segmenten verwendet. Es wird kontrolliert, ob im zu prüfenden Bereich des Graphen eine Verbindung zwischen 2 Segmenten besteht, welche in Zeilenrichtung verläuft, die Größe der Segmente zwischen der halben und der zweieinhalbfache Höhe des Medians, der Höhe der Segmente, liegt und ob der Abstand zwischen den Segmenten nicht größer als der Median ist.

Die vertikalen und horizontalen Segmente mit Text werden geschnitten und jede Schnittmenge ergibt einen neuen Textblock, für welchen das Verfahren rekursiv wiederholt wird.

Um die Zeilen zu erhalten werden die Pixel des Blocks vertikal projiziert und die Projektion mit der halben Buchstabenhöhe geglättet. Von diesem Histogramm wird nun die Ableitung berechnet und die Nulldurchgänge der Ableitung gesucht. Die Nulldurchgänge markieren abwechselnd die Maxima und Minima und zwischen zwei Minima liegt

eine Zeile. In Abbildung 5.22 und 5.22 die erkannten Zeilen blau eingezeichnet und die geglätteten Histogramme der projizierten Bounding-Boxen sind am Rand eingezeichnet.

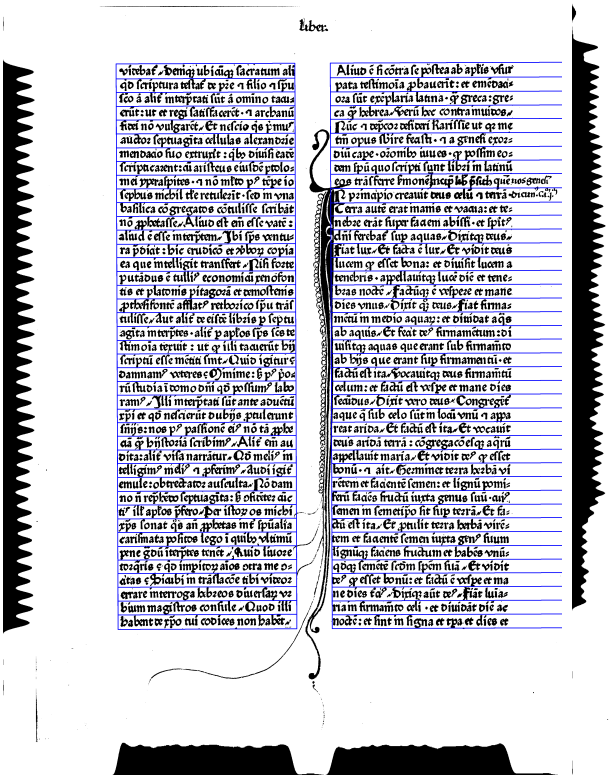

Abbildung 5.22: Von der Layout Analyse erkannte Zeilen

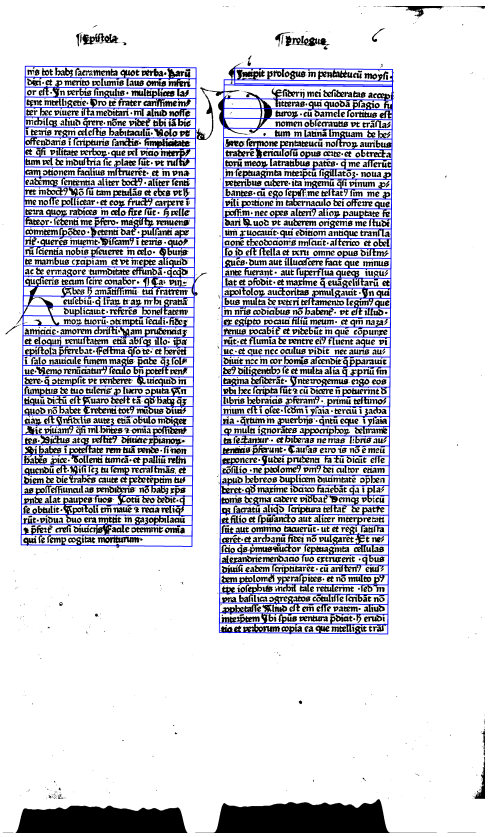

Abbildung 5.23: Von der Layout Analyse erkannte Zeilen

Da die Layout Analyse von einem Manhattan Layout ausgeht, werden Initialen nicht erkannt sondern mit in die Zeilen aufgenommen, wie in Abbildung 5.23. Die Überschriften in den Abbildungen 5.22 und 5.23 werden nicht als Zeilen erkannt, weil sich alle Buchstaben berühren oder durch die Unterstreichung verbunden sind, daher werden sie gar nicht als Text erkannt.

5.3 Vorherige Implementierung

Auch hier wurde ein Graph benutzt, welcher die benachbarten Segmente enthält, wie in Abschnitt 5.1.3 beschrieben. Da in den Frühdrucken und auch den meisten heutigen Texten, die Zeilen in verschiedenen Spalten auf der gleichen Höhe liegen, wurden

die Zeilen direkt aus den wahrscheinlichen Verbindungslinien von aufeinaderfolgenden Buchstaben durch eine Hough-Transformation ermittelt.

Mit der Hough-Transformation können Geraden oder beliebige andere parametrisierbare Objekte in Bildern gefunden werden. Eine Gerade lässt sich im Zweidimensionalen durch $y = mx + t$ darstellen, mit den beiden Parametern m und t. Um alle Geraden durch den Punkt (x_0, y_0) zu bestimmen, löst man die Geradengleichung nach t auf und erhält mit $t(m) = y_0 - mx_0$ eine Kurve im Parameterraum (t, m). Dieser wird Hough-Raum genannt und an jedem Punkt durch den die Kurve verläuft um eins inkrementiert. Führt man dies nun für weitere Punkte der gleichen Geraden aus, so ergibt sich ein Maximum an der Stelle, an der die Parameter mit den Parametern der Geraden übereinstimmen. Da mit der obigen Geradengleichung keine zur y-Achse parallelen Geraden dargestellt werden können, verwendet man die alternative Gleichung $\cos(\alpha)x + \sin(\alpha)y - d = 0$ mit den Parametern (α, d), wobei α der Winkel zwischen x-Achse und dem Normalenvektor der Geraden ist und d der Abstand vom Ursprung zur Geraden ist. Dieses Verfahren ist unempfindlich gegen Rauschen [vgl. 29, S. 255ff]. Drei Geraden und ihre Hough-Transformation sind in Abbildung 5.24 dargestellt.

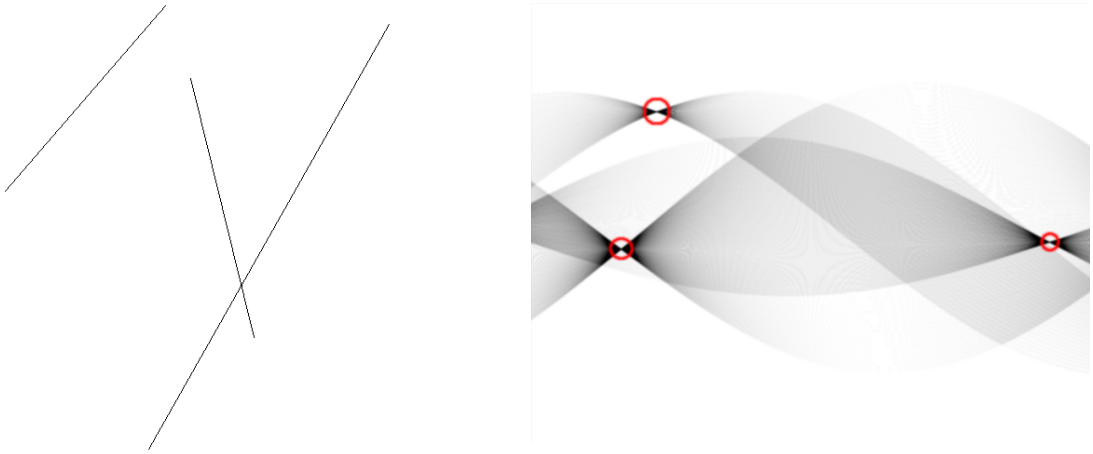

Abbildung 5.24: Links ist ein Bild mit drei Geraden und rechts ist ihre Hough-Transformation zu sehen

Aus dem Graphen wurden wie in Abschnitt 5.1.3 die möglichen Verbindungen zwischen Buchstaben bestimmt. Für die gefundenen Verbindungen wird eine Hough-Transformation ausgeführt, wobei der Parameter α nicht alle Werte annehmen muss, da schon die Richtung der Verbindungslinie ungefähr mit der Zeilenlinie übereinstimmt. In den untersuchten Frühdrucken war die Abweichung meist nicht größer als 15°.

Als Problem stellte sich jedoch heraus, dass auf manchen Seiten die Wölbung etwas

zu stark ist und die Zeilen aus verschiedenen Spalten nicht exakt genug durch eine Gerade erfasst werden können. Dies ist in Abbildung 5.25 zu sehen. Die schwarze Linie sollte durch die Mitte der Zeile laufen, am rechten Rand tut sie dies auch, aber am linken Rand ist sie fast am oberen Rand der Buchstaben ohne Oberlänge. Die weiteren eingezeichneten Linien sollen die Grenzen der Zeile, sowie die Grund- und x-Linie markieren.

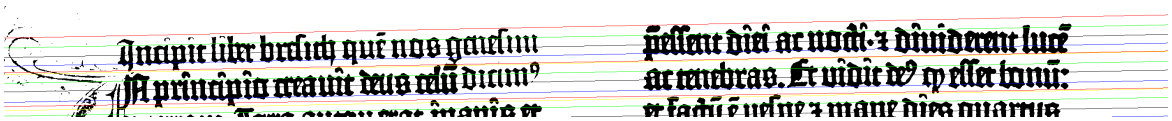

Abbildung 5.25: Markierte Zeilen auf einer etwas gewölbten Seite. Die schwarze Linie markiert die Mitte der Zeile, die rote den oberen und die gelbe den unteren Rand. Durch die blaue und grüne Linie wird die Grund- und x-Linie markiert.

Um die Grund- und x-Linie einer Zeile zu finden, wird ein Histogramm über die auf eine Senkrechte zur Zeilenrichtung projizierten Pixel berechnet. Das gößte Maximum und das kleinste Minimum in der Ableitung des Histogramms geben die Position von x-Linie und Grundlinie an, wie in Abbildung 5.26 dargestellt.

Abbildung 5.26: Die zu einer Zeile berechnete Grund- und x-Linie. Links ist das Histogramm und rechts die Ableitung des Histogramms dargestellt.

Die weiteren Probleme neben der Wölbung, welche zum Ersetzen der Hough-Transformation zum Finden der Zeilen führte waren, die manchmal auftretenden Schwierigkeiten beim nachträglichen Trennen der Spalten, besonders wenn im Spaltenzwischenraum viele Bilder waren. Der große Unterschied in der Intensität der Maxima im Hough-Raum, wenn eine Spalte weniger Zeilen hat, führte manchmal dazu, dass Zeilen ohne „Partnerzeile“ in der anderen Spalte nicht gefunden wurde.

5.4 Segmentierung

Um innerhalb der Zeilen die Worte und Buchstaben zu finden, wird das Histogramm der auf die Mittellinie der Zeile projizierten Pixel berechnet. Wenn das Histogramm auf einer Länge von mehr als 6 Pixel den Wert Null hat, wird ein Leerzeichen angenommen. Durch die eng geschriebenen Buchstaben ist es schwierig einen Wert automatisch, z.B. aus der Verteilung der Lückengröße, zu berechnen. Daher muss der Wert manuell festgelegt werden.

Bei Buchstabengrenzen hat das Histogramm ein Minimum. Es können aber auch innerhalb von Buchstaben und durch Rauschen Minima auftreten. Da in diesem Schritt noch nichts über die Buchstaben bekannt ist, können diese Minima nicht unterschieden werden. Die Minima an Buchstabengrenzen fallen jedoch zu den benachbarten Maxima auf mindestens die Hälfte ab, wogegen die durch Rauschen verursachten Minima meist nur um wenige Prozent unter den benachbarten Maxima liegen. Daher werden alle Minima die nicht auf mindestens 75% der benachbarten Maxima abfallen ignoriert. Maxima die nicht mindestens um 133,33% größer als das vorhergehende Minimum sind, werden auch ignoriert. Ansonsten könnte ein kleines Maximum in einem Tal zwischen zwei großen Maxima dazu führen, dass ein Minimum ignoriert wird. So werden die durch Rauschen verursachten Minima fast vollständig eliminiert und es bleiben alle Minima an Buchstabengrenzen enthalten.

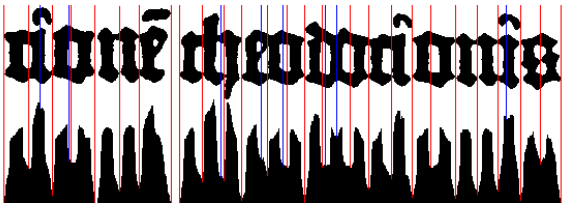

Abbildung 5.27: Segmentierung von Worten. In rot sind die möglichen Buchstabengrenzen eingezeichnet und mit blau sind ignorierte Minima markiert.

5.5 Vorverarbeitung für FineReader

Wie schon in Abschnitt 4.2 erwähnt, lassen sich die Probleme beim Erkennen der Leerzeichen und ein Teil der nicht erkannten Zeichen durch eine Vorverarbeitung der Bilddaten vermeiden. Um das Erkennen von Leerzeichen zu erleichtern, wird an jeder bei der Segmentierung erkannten Wortgrenze ein Abstand eingefügt. Um an dieser Stelle nicht aus Versehen ein Segment zu zerreißen, werden alle Segmente, deren Schwerpunkt hinter der Wortgrenze liegt, nach hinten verschoben. Dies betrifft hier hauptsächlich das p mit Anstrich. An den Zeilengrenzen wird das Bild auseinandergeschnitten und mit zusätzlichem Abstand wieder zusammengefügt. Eine Sonderbehandlung erfahren kleine Segmente wie i-Punkte und Überstriche, durch welche die Zeilengrenze verläuft. An der oberen Zeilengrenze werden sie zur Zeile hinzugenommen, sobald sie zu mindestens 20% innerhalb der Zeile liegen und an der unteren Grenze werden sie hinzugenommen, wenn sie zu mindestens 90% innerhalb der Zeile liegen. Durch den fehlenden Zeilenzwischenraum liegen die i-Punkte und Überstriche nah an oder erstrecken sich über die Zeilengrenzen. Da bei den hier verwendeten Schriftarten keine Punkte und Striche unter den Zeichen vorkommen, werden durch die obige Festlegung diese Segmente zur unteren Zeile hinzugenommen. In Abbildung 5.29 ist der in Abbildung 5.28

zu sehende Ausschnitt mit vergrößerten Zwischenräumen abgebildet.

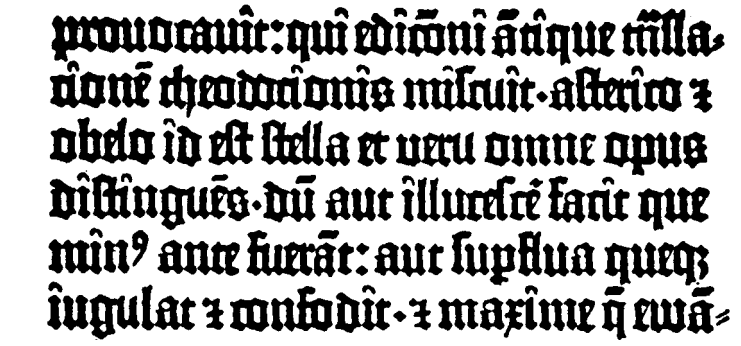

Abbildung 5.28: Ausschnitt aus der Gutenbegbibel

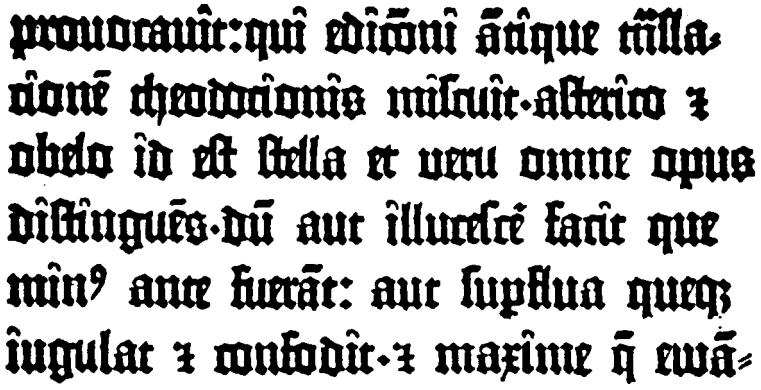

Abbildung 5.29: Ausschnitt aus der Gutenbegbibel mit vergrößerten Zwischenräumen

Durch die Vorverarbeitung konnten die Fehler bei der Leerzeichenerkennung fast vollständig behoben werden. In den Fällen, in denen die Zwischenräume nicht durch die Vorverarbeitung erkannt wurden, weil z.B. ein p mit Anstrich den Zwischenraum ausfüllte, blieb das Problem bestehen. Die nicht erkannten Zeichen konnten um 15% reduziert werden.

5.6 Klassifikation

Die Klassifikation der Zeichen erfolgt mittels Template-Matching. Hierzu wird das aktuelle Muster mit allen Templates, welche in der Breite nicht um mehr als 20% von diesem Abweichen verglichen. Um auch verschiedene Schriftgrößen erkennen zu können, werden die Größen des Musters und der Templates angeglichen. Hierfür ist für jedes Template der Abstand zwischen Grundlinie und x-Linie gespeichert. Da dieser

Abstand auch für das aktuelle Muster bekannt ist, kann das Template auf die Größe des Musters skaliert werden. Nachdem Muster und Template auf die gleiche Größe gebracht wurden, müssen die beiden vor dem Vergleich noch möglichst genau übereinander gelegt werden. Man könnte zwar das Template über das Muster schieben und die Position mit der größten Übereinstimmung speichern, dies währe aber sehr rechenintensiv.

Um die Muster übereinander zu legen wird für das Skelett des Templates eine Distanztransformation berechnet, dadurch erhält man eine Matrix in welcher an jedem Punkt die Entfernung zum nächsten Punkt auf dem Skelett gespeichert ist. Für alle Translationen in einem festgelegten Bereich wird mit Hilfe der Matrix der mittlere quadratische Abstand ermittelt, den das Musterskeletts zum Templateskelett hat. Der Translationsvektor, welcher den geringsten Abstand erzeugt, wird zum aufeinander ausrichten von Template und Muster verwendet. Die Ausrichtung könnte auch mit Hilfe des Umrisses ermittelt werden, dies wäre aber schon für nur geringe Größenabweichungen anfällig.

Um die Übereinstimmung zwischen Muster und Template zu bestimmen, kann man die Differenz zwischen übereinstimmenden und nicht übereinstimmenden Pixel bilden und dieses durch die Fläche des Templates teilen. Wenn beide exakt gleich sind ergibt dies den Wert eins, ansonsten kleinere Werte. Schon geringe Abweichungen können diesen Wert negativ beeinflussen. Wenn das Template am Rand nur um ein Pixel kleiner ist, würde der komplette Rand als unterschiedlich gezählt. Da die Muster durch den Druckprozess und das Binarisieren anfällig für kleine Änderungen am Rand sind, sind solche kleinen Abweichungen sehr Wahrscheinlich. Um diesen Effekt zu mindern werden die übereinstimmenden und nicht übereinstimmenden Pixel jeweils mit ihrem Abstand zum Rand gewichtet, dadurch wird der Einfluss von kleinen Abweichungen am Rand reduziert.

Um die zum Vergleichen nötigen Templates zu erhalten, werden die Zeichen auf einer Seite trainiert. Hierbei können nur rechteckige Bereiche ausgewählt werden, daher müssen manche Zeichen als Ligaturen trainiert werden. Da ein Muster mit allen Templates verglichen wird, sollte die Menge der Templates gering gehalten werden. Daher wird von ähnlichen Templates für den selben Buchstaben nur eines behalten.

Um die Anzahl an Vergleichen noch weiter zu verringern, werden die Muster nur mit Templates verglichen, welche in der Breite um maximal 15% davon abweichen und welche die gleiche Anzahl an Segmenten besitzen. Es kann zum weiteren reduzieren der Vergleiche auch noch Zoning verwendet werden. Hierbei werden alle Segmente des Musters in drei Teile zerlegt, die Ober- und Unterlänge und den restliche Mittelteil. Für jeden Bereich wird der relative Anteil an schwarzen Pixel ermittelt. Die ermittelten Werte werden nun mit denen des Templates verglichen und nur falls die Abweichungen einen gewissen Schwellenwert nicht überschreiten wird eine Ausrichtung vorgenommen und der eigentliche Vergleich durchgeführt. Durch die ungenauen Werte für Grund-

und x-Linie, wegen der Ermittlung der Zeilenausrichtung über Spaltengrenzen hinweg, müsste der Schwellenwert so groß gewählt werden, dass er keinen Effekt zeigen würde. Durch die mit der neue Layout Analyse ermöglichte genauere Bestimmung von Grund- und x-Linie könnte dieses Problem behoben werden.

5.7 Nachverarbeitung

Zur Nachverarbeitung sollten die Levensthein-Distanz und eine Normalisierung verwendet werden um Korrekturkandidaten in einem Lexikon zu finden. Vor dem Ende der Arbeit konnte nur noch die Normalisierung ansatzweise Implementiert werden.

Bei dem Normalisierungsansatz werden alle leicht verwechselbaren Zeichen durch ein festes Zeichen ersetzt. Die verwendeten Ersetzungsregeln decken den größten Teil der Fehler von FineReader ab. So werden zuerst die Abbreviaturzeichen durch ihre Grundzeichen ersetzt, dann alle t, c und r durch i, sowie v und n durch ii und alle m und w durch iii ersetzt. Zusätzlich werden noch das lange-s und f durch l ersetzt. Zum Programmstart wird ein Lexikon nach diesem Verfahren normalisiert und in einer Hashtabelle die normalisierten Worte als Schlüssel zusammen mit allen Worten aus denen sie gebildet werden können gespeichert. Wird später ein Wort gesucht, so muss es nur normalisiert werden und die an dieser Stelle in der Hashtabelle gespeicherten Worte abgerufen werden. Zusätzlich kann auch, falls ein Zeichen nicht erkannt wurde, mithilfe eines regulären Ausdrucks, in dem das nicht erkannte Zeichen durch „.“ ersetzt wurde, die gesamte Liste von normalisierten Wörtern durchsucht werden.

Um die prinzipielle Tauglichkeit dieses Ansatzes zu testen wurde, da kein Wörterbuch mit den hier verwendeten Bibeltexten zur Verfügung stand, aus den lateinischen Bibeltexten des alten Testaments auf [10] eine Liste mit 7011 Worten generiert. Es wurden alle u durch v ersetzt und für die Worte wurden auch ihre durch Abbreviaturen verkürzten Versionen zum Wörterbuch hinzugenommen, um eine größere Ähnlichkeit zu der in den Frühdrucken verwendeten Sprache herzustellen. Hierdurch wurde das Wörterbuch auf 17669 Einträge vergrößert. Durch die Normalisierung entstanden 14080 Klassen, auf welche sich 17642 Worte, nach Elimination von Dopplungen, verteilten.

In 83,9% der Klassen befindet sich nur ein Wort, was eine automatische Korrektur ermöglicht. Zwei Worte befinden sich in 11,93% der Klassen. Die restlich 4,07% verteilen sich auf die anderen Klassen, wobei sich maximal 24 Worte in eine Klasse befanden. Sollte sich dies so auch auf die größeren Real genutzten Wörterbucher übertragen lassen, würde daher jedesmal eine überschauliche Anzahl an Korrekturvorschlägen geliefert.

6 Leistungsbewertung und Ausblick

Durch das Fehlen des Nachverarbeitungsschrittes und die noch nicht erfolgte Einbindung der Layout Analyse ist es schwierig die Leistung des Gesamtsystems zu bewerten.

Die Layout Analyse erkennt, abgesehen von den Überschriften die Textblöcke der Frühdrucke korrekt, nimmt aber Initiale innerhalb der Textblöcke zu den Zeilen hinzu. Diese könnte man noch mit Texturanalyseverfahren identifizieren und so ihre Übername als Bilder ermöglichen.

Um den Klassifikationsalgorithmus ohne Nachverarbeitung zu testen, wurde an dem ersten erkannten Zeichensegment begonnen, das Template mit der besten Übereinstimmung gesucht und der ihm zugeordnete Buchstabe ausgegeben. Danach wird die Erkennung hinter dem erkannten Template fortgesetzt. Das Problem dieses Ansatztes ist, dass sobald ein Erkennungsfehler auftritt, bei dem die Segmentlänge nicht mit dem eigentlichen Zeichen übereinstimmt, die nächste Erkennung mitten in einem Zeichen beginnt und daher keine sinnvollen Ergebnisse liefern kann. Dies setzt sich fort, bis die Erkennung wieder an einer Buchstabengrenze endet. Um nun die Erkennungsrate zu testen wurden die scans der Frühdrucke in einer Auflösung von 300dpi verwendet. Die so erreichte Erkennungsrate liegt bei ca. 70%. Die hierbei auftretenden Fehlerarten sind denen von FineReader sehr ähnlich, bei den Fehlsegmentierungen treten hier zusätzlich Fälle auf, in denen z.B. das erste Segment eines q mit einem c erkannt wird oder das erste Segment eines h als l. Eine Übersicht über die Häufigkeit der Fehlergruppen wird in 6.1 geben. Diese Fehler entstehen dadurch, dass die Entscheidung für ein Zeichen getroffen wird ohne zu berücksichtigen, dass dadurch möglicherweise negativen Auswirkungen auf das Erkennen von Folgezeichen entstehen. Die Gruppen der leicht verwechselbaren Zeichen sind im Gegensatz zu denen von FineReader leicht vergrößert. So können c, r und t auch noch mit einem e verwechselt werden oder f, ī, l und das lange-s auch noch zusätzlich mit einem t. Die niedrige Anzahl an übersehnen Abbreviaturen kommt dadurch zustande, dass bei diesen Zeichen im Gegensatz zu FineReader auch häufig andere Fehler auftreten. So wurde des öfteren ein ū als ii erkannt und als Fehlsegmentierung gezählt. Durch vergrößern oder verkleindern der zu erkennenden Vorlage um bis zu 30%, unter Beibehaltung der vorher trainierten Muster, treten kaum Änderungen in der Erkennungsrate auf.

Nicht erkannte Leerzeichen	2,63%
Nicht erkannte Zeichen	21,92%
Fehlsegmentierungen	26,03%
Übersehene Abbreviaturen	2,37%
Verwechslung von f, ī, l, langes-s, t	15,07%
Verwechslungen von c, r, t und e	17,81%
Verwechslung von h und b	1,41%
Sonstiges	12,76%

Tabelle 6.1: Aufteilung der Erkennungsfehler

Durch einen Nachverarbeitungsschritt sollte sich die Erkennungsrate steigen lassen, da ein Teil der Fehlsegmentierungen vermieden werden kann und anhand des Kontextes ein Buchstabe auch dann ausgewählt werden kann, wenn er nicht am besten mit dem Muster übereinstimmt.

Das Template Matching ist durch die aufwendigen Vergleiche langsam und seine Erkennungsrate bei Ähnlichen Zeichen wird sich nicht stark verbessern lassen. Daher sollten auch die anderen in Abschnitt 2.3.3 erwähnten Verfahren auf ihre Verwendbarkeit für die Erkennung von Frühdrucken getestet werden.

A Literaturverzeichnis

[1] ABBYY. ABBYY FineReader OCR Produktlinie. http://www.abbyy.de/finereader_ocr/, 2006.

[2] ABBYY. Fraktur ABBYY FineReader XIX. http://www.frakturschrift.de/, 2006.

[3] abitz. ReadIris Pro 11 für Windows OCR Texterkennung scannen Scanner. http://www.abitz.com/iris/readiris.php3, 2006.

[4] S. W. Adnan. Automatic thresholding of gray-level using multi-stage approach, 2003.

[5] A. Amin, S. Fischer, T. Parkinson, and R. Shiu. Fast algorithm for skew detection, 1996.

[6] J. Anigbogu and A. Belaid. Hidden markov models in text recognition. *IJPRAI*, 9:925–958, 1995.

[7] T. M. Breuel. High performance document layout analysis, 2003.

[8] H. Bunke, M. Roth, and E. Schukat-Talamazzini. Off-line Cursive Handwriting Recognition Using Hidden Markov Models. *Pattern Recognition*, 28(9):1399–1413, 1995.

[9] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[10] Carey. The latin library. www.thelatinlibrary.com, 2006.

[11] S. B. Ching. Logical block labeling for diverse types of document images.

[12] K. Etemad, D. S. Doermann, and R. Chellappa. Multiscale segmentation of unstructured document pages using soft decision integration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(1):92–96, 1997.

[13] Google. Google code - updates: Announcing tesseract ocr. http://google-code-updates.blogspot.com/2006/08/announcing-tesseract-ocr.html, 2006.

[14] R. M. Haralick. Document image understanding: Geometric and logical layout. In *CVPR94: IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 385–390, 1994.

[15] T. Hong, J. Hull, and S. Srihari. A unified approach towards text recognition, 1996.

[16] E. Kavallieratou, N. Fakotakis, and G. Kokkinakis. Skew angle estimation for printed and handwritten documents using the wigner-ville distribution. *Image and Vision Computing*, 20:813–824, 2002.

[17] E. M. Kornfield, R. Manmatha, and J. Allan. Text alignment with handwritten documents. In *DIAL*, pages 195–211. IEEE Computer Society, 2003.

[18] S. Lee and B. Ryu. Parameter-free geometric document layout analysis. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 23:1240–1256, 2001.

[19] J. Liang, J. Ha, R. Haralick, and I. Phillips. Document layout structure extraction using bounding boxes of different entities. In *Workshop on Applications of Computer Vision*, pages 278–283, 1996.

[20] J. Liang, I. Phillips, and R. Haralick. Performance evaluation of document layout analysis on the uw data set, 1997.

[21] A. Niemistö. A comparison of nonparametric histogram-based thresholding algorithms, 2004.

[22] NUANCE. Nuance - OmniPage. http://www.nuance.de/omnipage/, 2006.

[23] NUANCE. Nuance - TextBridge Pro 11. http://www.nuance.de/paperport/, 2006.

[24] NUANCE. Nuance - TextBridge Pro 11. http://www.nuance.de/textbridge/, 2006.

[25] O. Okun, M. Pietikainen, and J. J. Sauvola. Document skew estimation without angle range restriction. *IJDAR*, 2(2-3):132–144, 1999.

[26] Projekt Gutenberg. Projekt gutenberg-de. http://projekt.gutenberg.de, 2006.

[27] W. Rasband. Imagej. U. S. National Institutes of Health, Bethesda, Maryland, USA, http://rsb.info.nih.gov/ij, 1997-2006.

[28] G. Seni, V. Kripasundar, and R. K. Srihari. Generalizing edit distance to incorporate domain information: handwritten text recognition as a case study. *Pattern Recognition*, 29(3):405–414, 1996.

[29] K. D. Tönnies. *Grundlagen der Bildverarbeitung*. Pearson Studium, 2005.

[30] O. Trier, A. Jain, and T. Taxt. Feature extraction methods for character recognition - a survey, 1996.

[31] O. D. Trier and A. K. Jain. Goal-directed evaluation of binarization methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(12):1191–1201, 1995.

[32] Y. Wang, I. Phillips, and R. Haralick. Statistical-based approach to word segmentation. In *ICPR*, pages 4555–4558, 2000.

[33] Wikipedia. Inkunabel — Wikipedia, Die freie Enzyklopädie. http://de.wikipedia.org/w/index.php?title=Inkunabel&oldid=21244558, 2006. [Online; Stand 29. August 2006].

Alle URLs waren im September 2006 erreichbar.