Evaluierung von schnellen Verfahren zur Zeichenerkennung mit Kameras

Diplomarbeit

---

vorgelegt an der Fachhochschule Köln
Campus Gummersbach

im Studiengang Technische Informatik

ausgearbeitet von: Tobias Hermann
Matrikelnummer: 11032233

Erster Prüfer: Prof. Dr. Erich Ehses
Zweiter Prüfer: Dipl.-Ing. Martin Schulz

Gummersbach, im April 2008

Zusammenfassung

Diese Arbeit beschreibt ein System zur Texterkennung auf Arzneimittelpackungen, welches in ein Kommissioniersystem integriert ist, das auf farbbildgestützter Objektkennung basiert. Es werden grundlegende Methoden der Schrifterkennung erläutert und Verfahren zur Textdetektion sowie zur Aufbereitung von Bildern dargestellt. Ebenso wird in unterschiedliche quelloffene Texterkennungsprogramme und in die Technik eines Datenbankabgleichs für die Zuordnung der von ihnen gelieferten Ergebnisse eingeführt.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Problem

Die Firma AXON Machine Vision GmbH & Co. KG bietet ihren Kunden Lösungen zur Kommissionierung und Sortierung unterschiedlicher Artikel, wie z. B. Zeitungen oder Arzneimittel an. Einen Teil einer solchen Lösung stellt ein Erkennungssystem (im folgenden Axon genannt) dar, was anhand von Bildern unter Zuhilfenahme so genannter Grundmuster, diese Artikel zur Weiterverarbeitung identifizieren kann. Dabei werden Bildmuster rotationsinvariant erkannt und Merkmale des Objektes wie Form, Größe, Farbe, Icons, Barcodes oder Beschriftungen extrahiert. Letztere werden bis jetzt allerdings nicht durch eine Texterkennung gelesen, sondern ebenfalls nur anhand ihrer Optik verglichen. Hier setzt das zu entwickelnde Texterkennungssystem (im folgenden nur noch System genannt) an und erweitert Axon um die Möglichkeit, die Beschriftungen auch inhaltlich zu verarbeiten.
Dies kann Axons Erkennungsrate erhöhen, da ein Teil der Objekte, der bisher auf Grund von Fehlerquellen wie Deformation, Verunreinigung oder zu großer Ähnlichkeit nicht erkannt bzw. voneinander unterschieden werden konnte, durch diese zusätzliche Information nun doch eindeutig zugeordnet werden kann. So wird nicht nur die Menge der Artikel, die nach fehlgeschlagener Erkennung von Hand nachkommissioniert werden müssen, noch weiter verringert, sondern vor allem auch Axons Flexibilität erhöht. Das System wird zunächst nur für den Bereich Arzneimittel entwickelt.

1.2 Anforderungen

Das System muss mit unterschiedlichen Schriftgrößen, -farben und -arten zurechtkommen, ein zuverlässiges Ergebnis liefern, dieses mit einer Liste von gegebenen Zeichenketten vergleichen und bei Unsicherheit diese Axon deutlich melden, damit es dann gegebenenfalls auf andere Kriterien zur Erkennung zurückgreifen bzw. das Objekt als nicht erkannt schleusen kann. Zusätzlich muss das zu entwickelnde System auf PCs mit Hardware des heutigen Standards in unter 0.5 Sekunden ein

Ergebnis liefern, damit Axons Durchsatzrate erhalten bleibt.

1.3 Schnittstelle

Axon übergibt:

- das aufgenommene Bild
- die Information, in welchem Bereich des Bildes sich der Artikel befindet
- einen Winkel, der die Rotation des Artikels auf dem Bild enthält
- ggf. die Information, in welchem Bereich des Artikels sich der zu lesende Text befindet incl. dem Rotationswinkel dieses Bereiches
- eine Liste mit Möglichkeiten (Zeichenketten), was auf dem markierten Bereich im Bild zu lesen sein könnte, welche auch in Form einer Datenbank vorliegen kann

Das System ordnet jeder der übergebenen Zeichenketten einen Wert zu, der besagt, mit welcher Sicherheit es diesen Text auf dem untersuchten Bildbereich erkannt hat. Diese erweiterte Liste gibt es an Axon zurück.
Im System laufen mehrere Prozesse ab. Zu ihnen gehören die Detektion der Position von Text auf dem Bild (falls diese unbekannt ist und nicht übergeben wurde) und die Vorverarbeitung des relevanten Bildbereiches. Diese ist nötig, damit die externen OCR-Programme, an die das Bild übergeben wird, gute Ergebnisse liefern. Nach der Übergabe dieses Bildbereiches an die OCR-Programme folgt die Auswertung der von ihnen gelieferten Ergebnisse (Datenbankabgleich). Der Datenbankabgleich ordnet jedem Element der von Axon übergebenen Liste den Sicherheitswert zu, mit dem Axon weiterarbeiten kann.
Diese Arbeit beschreibt diese Prozesse und ihr Zusammenspiel innerhalb des Systems.

2 Wie funktioniert Schrifterkennung? (OCR)

Die Schrifterkennung beschäftigt sich damit, aus Bilddaten (wie z. B. Scans oder Fotos) Text zu extrahieren und ihn in maschinenlesbaren Code zu überführen. Im Gegensatz zu einem menschlichen Betrachter sind für den Computer alle Zeichen in einem Bild zunächst nur Objekte, die sich nicht von anderen Objekten, wie beispielsweise Farbklecksen, unterscheiden. Das Texterkennungsverfahren muss also auf jedes im Bild gefundene Objekt angewendet werden. Die Schrifterkennung wird auch OCR (Optical Character Recognition) genannt, denn sie erkennt nur einzelne Zeichen und keine ganzen Wörter auf einmal. Deshalb muss jeder Buchstabe einzeln segmentiert werden. Schon hierbei können je nach Schriftart Schwierigkeiten auftreten, weswegen in diesem Kapitel erst die Eigenheiten unterschiedlicher Schriftarten aufgezeigt werden, um danach verschiedene Verfahren zur Zeichenerkennung vorzustellen.

2.1 Schriftarten

Selbst gleiche Buchstaben der selben Schriftart sind ausgedruckt und eingescannt auf der Ebene der Pixel ganz leicht unterschiedlich. Zwischen verschiedenen Schriftarten sind diese Unterschiede natürlich noch wesentlich größer. Wenn eine Schrifterkennung mit vielen Schriftarten klar kommt, wird von Multifont-Fähigkeit gesprochen. Schriftartunabhängigkeit wird als Omnifont bezeichnet.

2.1.1 Proportionalität

Schriftarten werden als proportional bezeichnet, wenn Buchstaben unterschiedliche Breiten haben. Die folgende Tabelle macht dies anhand einer 5 Zeichen langen Buchstabenfolge deutlich.

nicht proportionale Schriftart	proportionale Schriftart
`iiiii`	iiiii
`WWWWW`	WWWWW

In der nicht proportionalen Schriftart haben die Buchstaben “`i`” und “`W`” die gleiche Breite, während sie in der proportionalen Schriftart einen deutlichen Breitenunterschied aufweisen.

2.1.2 Laufweite

Als Laufweite wird der Abstand zweier benachbarter Zeichen im Schriftbild bezeichnet. Dieser variiert nicht nur zwischen den einzelnen Schriftarten, sondern verändert sich auch, wenn bei der Erstellung eines Dokumentes das Blocksatz-Format gewählt wurde und somit die Abstände vergrößert werden, um am Zeilenende einen ebenen Abschluss zu erhalten. Bei einigen Schriftarten ist die Laufweite jedoch so gering, dass es schwer ist, die einzelnen Buchstaben zu segmentieren, da sie ineinander übergehen (siehe Abbildung 2.1).

Abbildung 2.1: Geringe Laufweite

2.1.3 Serifen

Serifen sind kleine Verzierungen am Ende der Linien eines Zeichens, und sie sollen die Lesbarkeit von längeren Texten erhöhen. Das mag für den Menschen zutreffen, kann im Falle der Schrifterkennung durch einen Computer jedoch zu Problemen führen. Abbildung 2.2 zeigt, wie auch hier die Segmentierung durch Verschmelzen der Buchstaben erschwert werden kann.

Abbildung 2.2: Serifen

2.2 Verfahren

Es gibt viele unterschiedliche Ansätze zur Schrifterkennung, die alle je nach Anwendung ihren Nutzen haben. Hier werden die bekanntesten unter ihnen vorgestellt. Sie alle haben gemeinsam, dass das Bild vorher binarisiert (d. h. in ein Schwarzweißbild umgewandelt, siehe Kapitel 5.4) wird.

2.2.1 Mustervergleich

Ein Mustervergleich ist eine einfache Methode, um ein zu lesendes Zeichen einem Buchstaben zuzuordnen.

Bitmuster

Der Bitmustervergleich ist ein Mustervergleichsverfahren. Bei ihm wird das Zeichen Pixel für Pixel mit fertigen Zeichenvorlagen vergleichen und dem Buchstaben mit der größten Übereinstimmung zugeordnet. Dieses Verfahren funktioniert nur, wenn der zu lesende Text in der Schriftart vorliegt, die auch für die Vorlagen verwendet wurde. Es besitzt keine Omnifont-Fähigkeit.

Schablonen

Ein Schablonenvergleich funktioniert ähnlich, nur werden hier nicht alle Pixel verglichen, sondern nur wenige bestimmte, die sich in festgelegten Sichtfenstern befinden. Diese Sichtfenster müssen so gewählt werden, dass signifikante Bereiche von Buchstaben, also Bereiche, in denen sie sich voneinander unterscheiden, abgedeckt sind. Andere Bereiche können vernachlässigt werden. Durch die geringere Zahl an Pixel-Vergleichen ist dieses Verfahren schneller als der Bitmustervergleich, allerdings auch noch auf eine Schriftart (bzw. wenige sehr ähnliche Schriftarten) festgelegt.

2.2.2 Strukturanalyse

Strukturanalyseverfahren funktionieren schriftartunabhängiger als Mustervergleichsverfahren.

Skelettierung

Durch Skelettierung wird ein Zeichen auf sein Skelett reduziert, um so nur die wesentlichen Merkmale zu zeigen. Abbildung 2.3 zeigt eine solche Skelettierung. Die schwarzen Linien innerhalb der Buchstaben stellen das erzeugte Skelett dar.

Abbildung 2.3: Skelettierung

Winkelschnittanalyse

Bei der Winkelschnittanalyse werden Abtastgeraden in unterschiedlichen Winkeln über das Bild gelegt und jeweils die Anzahl der durch sie überlagerten Pixel des Zeichens gezählt.

Topologische Analyse

Die topologische Analyse zerlegt jedes Zeichen in einfache geometrische Elemente wie Geraden und Kreisbögen. Anhand ihrer Position, Größe und Ausrichtung wird erkannt, um welches Zeichen es sich handelt.

3 Konzept

3.1 Framework

Axon bietet ein in C++ implementiertes Framework, welches viele komplexe Klassen und Methoden mit unterschiedlichen Funktionalitäten zur Verfügung stellt. Dazu gehören unter anderem Verfahren zur Erkennung der Lage und Rotation von Objekten sowie Möglichkeiten zur Berechnungen von Richtungsinformationen und Transformationen zwischen unterschiedlichen Koordinatensystemen. Hervorzuheben wäre noch eine auf Geschwindigkeit optimierte Klasse zur Verwaltung von Formen, wie z. B. Umrandungen (IOShapes). Die Optimierung besteht darin, dass bei der Konstruktion einer IOShape (das "IO" steht für "In & Out") das Bild zeilenweise abgetastet wird, und die Punkte, die dabei auf eine Objekt- bzw. Schriftkante fallen, in einer Liste abgelegt werden. Es handelt sich also nur um eine Menge von Punkten, nicht aber um echte Kantendefinitionen in Form von Polygonen.

3.1.1 Bilder

Axon liefert dem System zuerst einmal das Kamerabild. Das Objekt auf dem Bild kann eine beliebige Position und Rotation besitzen, da es von einem menschlichen Kommissionierer aufgelegt wird (siehe Abbildung 3.1).

Abbildung 3.1: Kamerabild

3.1.2 Zusatzinformationen

Falls Bilder von neuen für Axon unbekannten Artikeln aufgenommen werden, erfasst es unterschiedliche optische Merkmale und kann nun unter Umständen schon alle folgenden Artikel dieses Typs eindeutig zuordnen. Wenn dies nicht der Fall ist, wird manuell von einem Mitarbeiter speziell für diesen Artikeltyp ein Grundmuster angelegt. Hier werden unter anderem Bereiche definiert, in denen Axon anhand von festgelegten Farben Erkennungszeichen, wie z. B. Beschriftungen, segmentieren soll. In so einem Fall kann an die genaue Position der Schrift in Form von Umrandungen (implementiert als IOShapes) an das System übergeben werden (siehe Abbildung 3.2).

Abbildung 3.2: Bild mit Umrandungen (IOShapes) um Objekt und Text

Falls für einen Artikel kein Grundmuster angelegt wurde, kann keine IOShape geliefert werden, da der Bildbereich, in dem sie erzeugt werden könnte, nicht bekannt ist, und keine Farbwerte zur Abgrenzung von Schrift und Hintergrund festgelegt wurden. Dann wird die Position der Schrift vom zu entwickelnden System selbst gefunden (Textdetektion).
Grundsätzlich ist in den Zusatzinformationen jedoch immer eine IOShape um das gesamte Objekt, sowie dessen Schwerpunkt und Ausrichtung gegeben.

3.2 Wie ist das System aufgebaut?

Das System soll als eigenes Modul in Axon eingebettet werden. Es bekommt von Axon ein Bild incl. Zusatzinformationen sowie eine Liste mit Namen von Artikeln, die nach Klassifizierung von Größe, Form und Farbe verblieben sind, übergeben. Wenn in den Zusatzinformationen kein Bereich, in dem sich der zu lesende Text auf dem Bild befindet, festgelegt ist, wendet das System sein Verfahren zur Textdetektion an und findet ihn selbstständig.
Dieser Bereich des Bildes wird in der Form verarbeitet, dass ein horizontal ausgerichtetes Schwarzweißbild des zu lesenden Textes resultiert, welches nun an bis

zu drei externe Schrifterkennungsprogramme weitergegeben wird. Die Programme versuchen den Text zu lesen und liefern die Ergebnisse ihrer Versuche an das System zurück.
Die Ergebnisse werden nach einer speziellen Methode mit der Liste mit Namen von Artikeln verglichen, so dass jedem Artikel ein Sicherheitswert zugeordnet wird, der besagt, mit welcher Gewissheit es sich bei dem Objekt auf dem Bild um diesen handelt. Die mit diesen Werten erweiterte Liste wird an Axon zurückgegeben. Dieser Datenfluss ist in Abbildung 3.3 graphisch dargestellt.

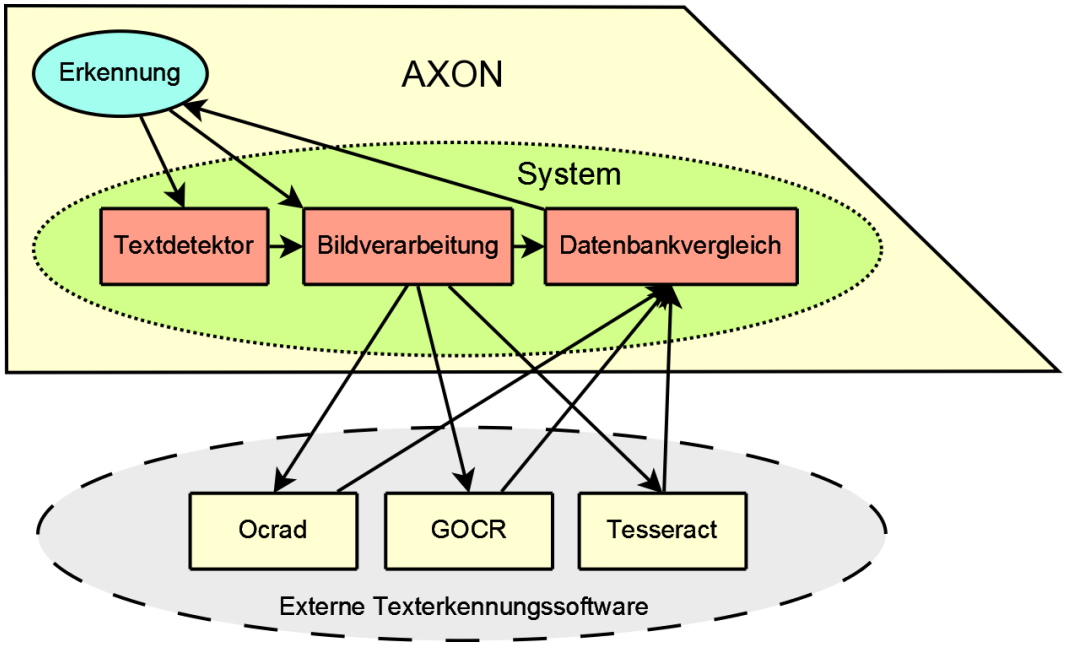

Abbildung 3.3: Datenfluss zwischen Axon, System und den externen Texterkennungsprogrammen

4 Textdetektion

Dem Menschen fällt es leicht, in einer von ihm betrachteten Szene oder auf einem Bild die Bereiche mit lesbarem Text zu finden. Dies funktioniert ähnlich automatisch wie das Finden von Gesichtern. Solche Prozesse auf einen Computer zu übertragen, bringt jedoch einige Schwierigkeiten mit sich. Zuerst einmal müssen Merkmale definiert werden, die solche Bildbereiche eindeutig von anderen, in diesem Fall von Bereichen, die keinen Text beinhalten, unterscheiden. Nachfolgend sind Algorithmen nötig, die die Selektion nach diesen Merkmalen vornehmen.
Hierfür gibt es schon viele interessante Ansätze, von denen im folgenden einige vorgestellt werden.

Rainer Lienhart und Wolfgang Effelsberg stellen in ihrer Publikation "Automatic Text Segmentation and Text Recognition for Video Indexing" ([TDATS]) eine Methode zur Textdetektion dar, bei der es darum geht, Text (z. B. Untertitel) zu finden, der künstlich in ein Video eingeblendet wurde. Dieser Text wird erkannt, und so werden wichtige Informationen aus dem Video extrahiert. Eines des Hauptkriterien zur Findung des Textes ist der Ansatz, dass der eingeblendete Text an einer festen Position verharrt oder sich gleichförmig durch das Bild bewegt (Scroller), während sich die Bildpunkte der Szene von Frame zu Frame ständig verändern. Dadurch, dass sich diese Eigenschaft nicht auf die Erkennung von Text in einzelnen Bildern übertragen lässt, ist der Ansatz für die Textdetektion auf Arzneimittelpackungen ungeeignet. Allerdings werden andere nützliche Merkmale von Text definiert:

- Buchstaben sind einfarbig.
- Die Längen und Breiten von Buchstaben bewegen sich in bestimmten Größenordnungen.
- Buchstaben treten in horizontal angeordneten Gruppen auf und haben einen regelmäßigen Abstand zueinander.

Diese Eigenschaften treffen auch im Falle des zu entwickelnden Systems zu und sind somit übertragbar.

Eine weitere interessante Veröffentlichung steht unter dem Namen “Detection of Text on Road Signs From Video” ([TDRSV]) zur Verfügung. Um die Straßenschilder und den darauf geschriebenen Text auf Frames von einem Video, das aus einem fahrenden Auto aufgenommen wird, in Echtzeit zu erkennen, wird durch Kantendetektion und Binarisierung eine Situation geschaffen, in welcher die einzelnen Objekte auf dem Bild als Buchstaben verifiziert werden.

In “Text Detection from Natural Scene Images: Towards a System for Visually Impaired Persons” ([TDNSI]) werden Vorschläge zur Entwicklung eines Systems gemacht, was es Menschen mit stark eingeschränkter Sehstärke ermöglichen soll, trotzdem Informationen von z. B. Schildern mit Hilfe einer Kamera und eines kleinen Computers zu erhalten. Auch hier ist die Textdetektion auf den Bildern der Kamera mit Hilfe von Kantendetektion beschrieben. Zusätzlich wird ein 8-Farben-Bild erzeugt, indem auf allen drei Farbkanälen (R, G und B) eine Binarisierung durchgeführt wird. So kann bei der Extraktion der verbundenen Komponenten für die Entscheidung, ob es sich um einen Buchstaben handelt, zusätzlich das Kriterium mit aufgenommen werden, dass Buchstaben und die Buchstaben in ihrer Nachbarschaft so gut wie immer die gleiche Farbe haben.
Weitere Bedingungen für die Identifikation einer im Bild gefundenen Komponente als Buchstabe sind:

- Das Seitenverhältnis (Breite/Höhe) liegt zwischen 0.1 und 2.
- Das Produkt aus Höhe und Breite liegt zwischen zwei festgelegten Werten (abhängig von der zu lesenden Schriftgröße).
- Zwei benachbarte Buchstaben unterscheiden sich in ihrer Größe nur um einen bestimmten Faktor.
- Die Abstände der Flächenschwerpunkte von benachbarten Buchstaben variieren in einem festgelegten Bereich.
- Die Flächenschwerpunkte von Buchstaben weichen in ihrer Y-Position nur bis zu einem festen Faktor (in Abhängigkeit ihrer Größe) voneinander ab.

4.1 Verbundene Komponenten

Die Erzeugung von verbundenen Komponenten ([CoCos]) ist ein nützliches Werkzeug für die Textdetektion, denn sie dient der Findung von gleichfarbigen Gebieten im Bild, bei denen es sich um Buchstaben handeln kann. Diese Gebiete

werden anschließend auf die oben beschriebenen Eigenschaften überprüft. So wird entschieden, bei welchen von ihnen es sich wirklich um Buchstaben handelt. In diesem Fall wird das Ausgangsbild zunächst in ein 8-Farben-Bild umgewandelt. Hierfür wird jeder der drei Farbkanäle (R,G,B) binarisiert. Das Ergebnis einer solchen Umwandlung ist in Abbildung 4.1 dargestellt.
Auf der Basis eines solchen Bildes ist es einfacher, verbundene Komponenten via Floodfill zu finden.

Abbildung 4.1: 8-farbige Kanalbinarisierung

4.1.1 Floodfill

Das Floodfill-Verfahren ([WikiFldF]) ist in fast jedem Bildbearbeitungsprogramm integriert. Es erfasst zusammenhängende Pixel gleicher Farbe und füllt diesen Bereich mit einer neuen Farbe auf. In Abbildung 4.4 wird gezeigt, wie ein Bereich des Bildes aus Abbildung 4.1 mittels Floodfill gefüllt wurde.
Die direkte Nachbarschaft eines Pixels kann entweder mit vier oder mit acht Pixeln definiert werden, was beim Floodfilling unterschiedliche Ergebnisse liefert. Bei einer acht Pixel großen Nachbarschaft weitet sich der gefüllte Bereich in Bereiche aus, die mit einer vier Pixel großen Nachbarschaft nicht erfasst würden (vgl. Abbildung 4.2 mit 4.3).
Im Fall des Textdetektors wird eine Nachbarschaft von vier Pixeln benutzt, da möglichst vermieden werden soll, dass zwei Buchstaben in eine gemeinsame Region fallen. Die lässt sich natürlich nicht komplett ausschließen, jedoch lässt sich durch das Verwenden der vier-Pixel-Nachbarschaft die Häufigkeit des Auftretens eines solchen Problems verringern. Abbildung 4.2 zeigt das Ergebnis eines Floodfillings mit einer 4-Pixel-Nachbarschaft. Die Farbe läuft nicht in den linken oberen Teil des Bildes, was beim Verwenden einer 8-Pixel-Nachbarschaft hingegen passiert (siehe Abbildung 4.3).
Es gibt unterschiedliche Methoden, die Floodfill-Idee programmiertechnisch umzusetzen.

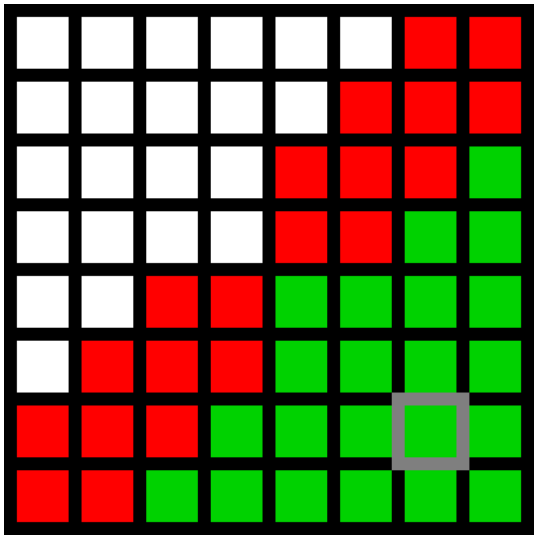

Abbildung 4.2: Floodfilling mit einer 4-Pixel-Nachbarschaft

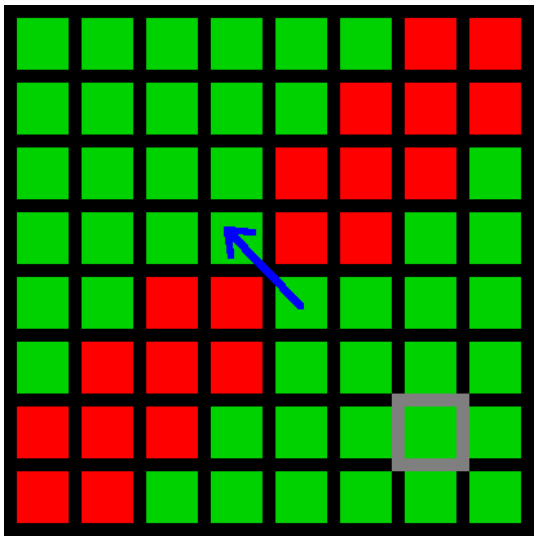

Abbildung 4.3: Floodfilling mit einer 8-Pixel-Nachbarschaft

Abbildung 4.4: Floodfilling am Beispiel “Acic Creme”

Rekursiv

Die rekursive Version des Floodfill-Algorithmus' ist sehr einfach zu implementieren (siehe Abbildung 4.5), hat jedoch den Nachteil, dass bei großen zu füllenden Flächen extrem viele Aufrufe stattfinden, was nicht nur viel Zeit kostet, sondern sogar einen Stack-Überlauf, und somit einen Programmabsturz, verursachen kann.

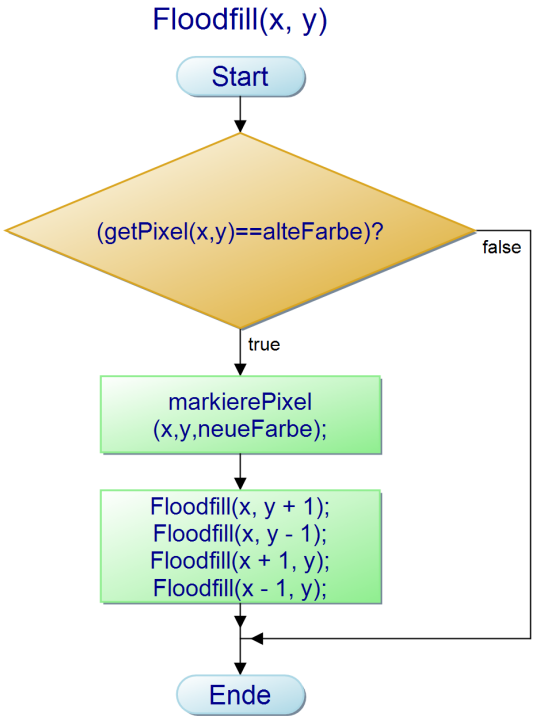

Abbildung 4.5: Programmablaufplan - Floodfill (rekursiv)

Iterativ

Um der Gefahr des Stacküberlaufs aus dem Weg zu gehen, wird eine iterative Implementierung des Floodfill-Algorithmus' verwendet (siehe Abbildung 4.6).
Hier wird ein künstliches Stack-Objekt auf dem Heap erzeugt, in dem lediglich die Bildpunktkoordinaten gestapelt abgelegt werden.

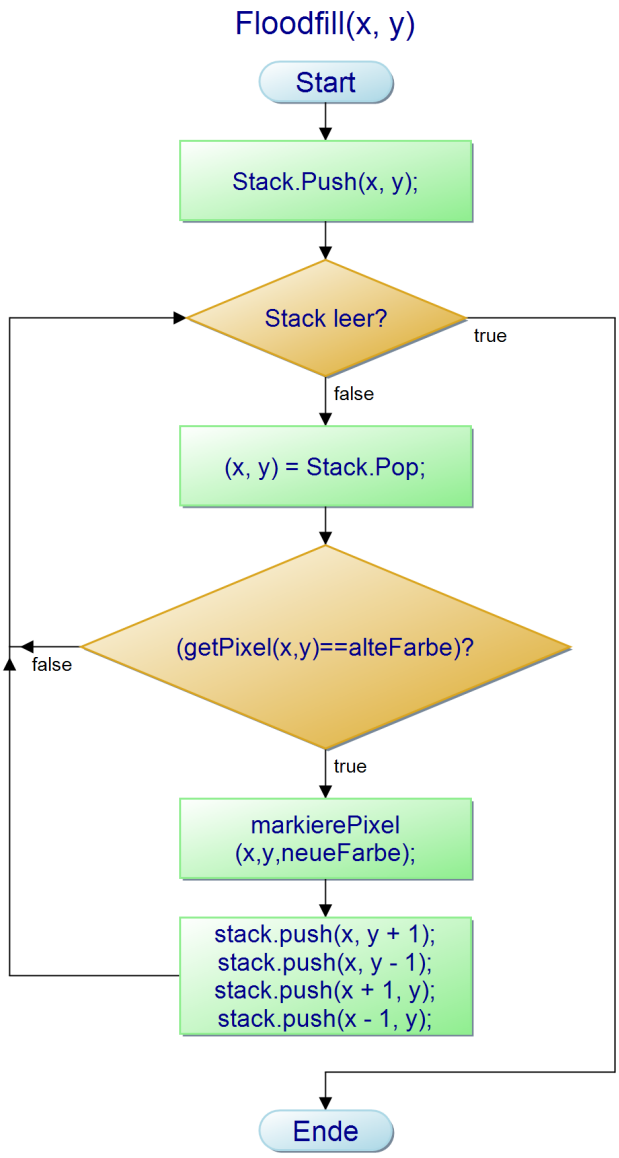

Abbildung 4.6: Programmablaufplan - Floodfill (iterativ)

4.1.2 Regionen auswählen

Nachdem alle Regionen im Bild gefunden wurden (siehe Abbildung 4.7), wird jede Region anhand ihrer Größe und ihres Seitenverhältnisses als Buchstabenkandidat verifiziert.

Abbildung 4.7: Alle detektierten Regionen

Dabei wird die Anzahl der Regionen erheblich reduziert (siehe Abbildung 4.8). Nachdem nun nach sich gegenseitig überlappenden Regionen gesucht wird, und in einem solchen Fall die jeweils kleinere Region entfernt wird, bleibt eine Regionenmenge wie in Abbildung 4.9 übrig.

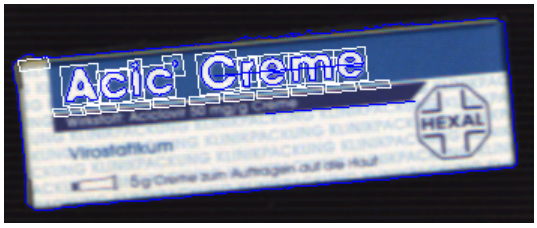

Abbildung 4.8: Detektierte Regionen nach der ersten Reduktion

Des weiteren wird analysiert, bei welchen der verbleibenden Regionen es sich tatsächlich um Buchstaben handeln kann. Hierfür wird das Kriterium, dass Buchstaben links oder rechts neben sich mindestens einen weiteren Buchstaben ähnlicher Schriftgröße haben, herangezogen. Das Ergebnis einer solchen Nachauswahl ist in Abbildung 4.10 dargestellt.

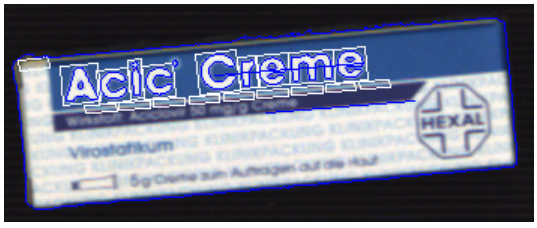

Abbildung 4.9: Detektierte Regionen nach der zweiten Reduktion

4.1.3 Regionen zusammenfassen

Da einzelne Buchstaben wie aus Abbildung 4.10 von den externen OCR-Programmen nur schlecht erkannt werden, werden diese Buchstabenregionen zu Wortregionen

zusammengefasst (siehe Abbildung 4.11).
Die damit markierten Bildbereiche können nun verarbeitet werden.

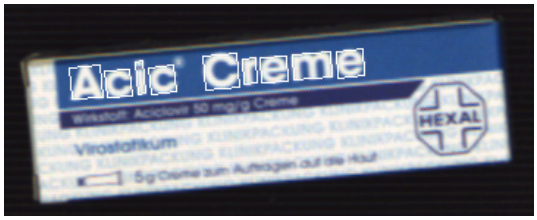

Abbildung 4.10: Detektierte Buchstaben

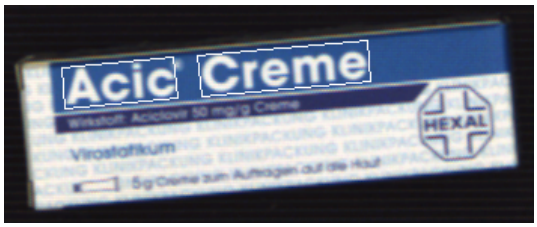

Abbildung 4.11: Detektierte Wörter

4.2 Kantendetektion

Ein völlig anderes Verfahren zur Textdetektion basiert auf Kantendetektion und kann mit Hilfe von Filtern realisiert werden. Hierbei wird davon ausgegangen, dass Textbereiche eine höhere Kantendichte besitzen als der Rest des Bildes.
Die dabei zum Einsatz kommenden Filter werden zunächst allgemein beschrieben, wonach ihre Anwendung beim Aufspüren von Textbereichen im Bild erklärt wird.

4.2.1 Filter

Mit Hilfe eines digitalen Filters werden in der Bildverarbeitung störende Anteile (z. B. vereinzelte Störpixel bzw. Rauschen) aus Bildern entfernt sowie bestimmte Bereiche im Bild (z. B. Kanten) hervorgehoben. Je nach Aufgabe bieten sich dazu unterschiedliche Filterkerne an. Die Filter werden anhand des Beispielbildes aus Abbildung 5.8 demonstriert.

Tiefpass

Mit Hilfe eines Tiefpasses lässt sich ein Bild glätten. Einerseits wird dabei störendes Rauschen entfernt, andererseits verringert sich auch die Schärfe des Bildes, was jedoch, je nach Anwendung, auch gewollt ist.

Die Faltung mit einem 3*3-Tiefpass-Filterkern stellt sich folgendermaßen dar:

$$Ausgabebild = \frac{1}{9} * \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} * Eingabebild$$

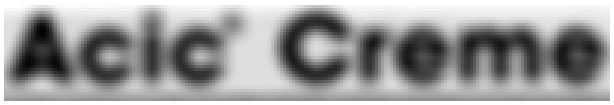

Abbildung 4.12: Ergebnis einer Tiefpass-Filterung

Sobel

Der Sobel-Operator ([Sobel]) ist sehr wirkungsvoll, um Kanten in einem Bild zu finden. Zunächst wird das Ausgangsbild mit zwei unterschiedlichen Kernen gefiltert, wobei einer für die Gradienten in X-Richtung (Abbildung 4.13) zuständig ist, der andere für die Y-Richtung (Abbildung 4.14):

$$S_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}$$

$$S_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

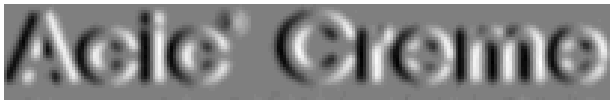

Abbildung 4.13: X-Komponente einer Sobel-Filterung

Abbildung 4.14: Y-Komponente einer Sobel-Filterung

Die beiden Ergebnisse (G_x und G_y) werden nun nach dem Satz des Pythagoras miteinander verknüpft.

$$G = \sqrt{{G_x}^2 + {G_y}^2}$$

So ergibt sich eine richtungsunabhängige Information über die Kanten im Bild (Abbildung 4.15).

Abbildung 4.15: Ergebnis einer Sobel-Filterung

4.2.2 Anwendung der Filter

Die Filter werden benutzt, um Bereiche im Bild mit hoher Kantendichte zu finden. Da diese bei Textregionen erhöht ist, sind diese Gebiete nach der Anwendung eines Kantendetektionsfilters, wie z. B. des Sobel-Filters, besonders auffällig (siehe Abbildung 4.16).

Abbildung 4.16: Anwendung des Sobel-Filters zur Kantendetektion

Dieses Kantenbild durchläuft nun zunächst einen Tiefpassfilter (siehe Abbildung 4.17) und wird danach binarisiert (siehe Abbildung 4.18).
Das Verfahren der Binarisierung wird in Kapitel 5.4 beschrieben.

Abbildung 4.17: Geglättetes Kantenbild

Abbildung 4.18: Binarisiertes geglättetes Kantenbild

Nun sind noch allerhand Linien zu sehen, die durch Kanten der einfarbigen Hintergrundregionen auf der Packung entstanden sind, jedoch nicht zum Text gehören. Diese werden ab einer Länge von Bildbreite/4 für horizontale Linien und Bildhöhe/4 für vertikale Linien entfernt. So bleiben nur noch wenige Linien übrig, der Text jedoch wird nur geringfügig angegriffen (siehe Abbildung 4.19). Die Zahl 4 als Divisor wurde durch Versuche ermittelt. Je größer sie ist, desto mehr Linien werden entfernt, was dazu führen kann, dass wichtige Textanteile wegfallen. Wenn sie kleiner gewählt wird, werden weniger Linien entfernt, und es bleiben mehr Linien, die nicht zum Text gehören, übrig.

Abbildung 4.19: Linienentfernung im Kantenbild

Um die restlichen Linien, die noch entfernt werden müssen, jedoch aufgrund ihrer Länge nicht erfasst wurden, ebenso wie kleine Bereiche, die auch keinen Bezug zum Text haben, zu beseitigen, läuft ein letzter Tiefpassfilter mit anschließender Binarisierung über das Bild (siehe Abbildung 4.20).

Abbildung 4.20: Binarisierung des von Linien bereinigten Bildes

Von den übrig gebliebenen Regionen wird angenommen, dass es sich um Textregionen handelt (siehe Abbildung 4.21).

Abbildung 4.21: Detektierte Textregionen

Dieses Verfahren ist jedoch bei Hintergrundmustern sehr anfällig, da hier überall Kanten gefunden werden. Ein weiteres Problem ist, dass die gefundenen Textregionen nicht so präzise eingegrenzt werden wie bei dem Verfahren der verbundenen Komponenten (vgl. Abbildung 4.11 mit Abbildung 4.21).
Aus diesen Gründen ist die Methode aus Kapitel 4.1, die sich der verbundenen Komponenten bedient, das Mittel der Wahl.

5 Vorverarbeitung der ROI / des Textbereiches

ROI steht für Region of Interest und bezeichnet den Bereich des Bildes, der durch die IOShape bzw. den Textdetektor festgelegt wurde. Zur eigentlichen Texterkennung werden die Bilddaten an externe OCR-Programme, welche die Ergebnisse ihrer Erkennung wieder an das System zurückgeben, weitergegeben. Diese Programme und das Übergabeverfahren werden in Kapitel 6 näher beschrieben. Hier ist zunächst wichtig, was mit dem Kamerabild geschehen muss, damit die Programme die besten Ergebnisse liefern. Sie tun dies, wenn folgende Bedingungen erfüllt sind.

- Die Bilder enthalten nur den zu untersuchenden Text und sind frei von störenden Elementen.
- Der Text auf den Bildern ist horizontal ausgerichtet und liegt nicht schräg.
- Die Bilder sind binarisiert, d. h. sie wurden mit Hilfe einer Helligkeitsschwelle in Schwarzweißbilder umgewandelt, wobei hier der Text schwarz und der Hintergrund weiß zu sein hat, nicht umgekehrt.

Die Bilder müssen also um den Text herum zurechtgeschnitten werden und im korrekten Maße gedreht werden. Unter Umständen muss eine Farbinvertierung stattfinden, worauf die Binarisierung folgt.
Die dafür nötigen Verfahren werden in diesem Kapitel beschrieben.

5.1 Graustufenbild

Die Umwandlung eines Bildes in ein Graustufenbild erfolgt durch Durchschnittsbildung der Werte in den Farbkanälen des Ausgangsbildes (siehe Abbildung 5.1).
$Grauwert = \frac{Rotanteil+Grünanteil+Blauanteil}{3}$
Jeder Bildpunkt wird nun im Speicher von einem einzelnen Byte repräsentiert.

Abbildung 5.1: Graustufenbild

5.2 Rotation

Da die Arzneimittelpackung beliebig auf dem Band und somit auch auf dem aufgenommenen Bild liegen kann, muss der relevante Bildbereich (die ROI) zunächst einmal gedreht werden. Axon liefert zu jedem Objekt die Hauptachse bzw. bei quadratischen und annähernd quadratischen Objekten die beiden Hauptachsen. Aus dieser Information kann der Rotationswinkel der Packung nun eindeutig bestimmt werden. Bei zwei Hauptachsen wird die verwendet, bei der der definierte Textbereich so liegt, dass seine Breite größer als seine Höhe ist. Da im Voraus jedoch nicht erkannt werden kann, ob der Text auf dem Kopf steht, wird mit jeweils zwei Versionen (der normalen und der um 180° gedrehten) weitergearbeitet. Das stellt jedoch kein Problem dar, da die 180°-Rotation durch eine einfache Umkehr der Reihenfolge der Bildpunkte im Bildspeicher realisiert werden kann und somit nicht viel Rechenzeit in Anspruch nimmt. Axon stellt diese Funktion zur Verfügung.
Die OCR-Programme liefern zu den auf dem Kopf stehenden Bildern natürlich unbrauchbare Ergebnisse, welche jedoch danach von den Funktionen der Klasse zum Datenbankvergleich herausgefiltert werden.

5.2.1 Rotationswinkel

Der Rotationswinkel der Packung, der aus der von Axon übergebenen Hauptachse errechnet wurde, ist meist gleich dem Rotationswinkel der IOShape um die Schrift (vgl. Abbildung 3.2).

Abbildung 5.2: Unterschiedliche Rotationswinkel von Packung und Text

Texte können aber auch aufgrund ihrer Form einen falschen Ausrichtungswinkel haben (siehe Abbildung 5.2). Deshalb wird grundsätzlich zuerst der Winkel der Packung zur Rotation verwendet, allerdings nur bis zu einer bestimmen Toleranz, denn falls der zu lesende Text als Aufkleber auf der Packung schräg platziert wurde, muss der Winkel der IOShape benutzt werden (siehe Abbildung 5.3).

Abbildung 5.3: Packung mit Text auf schrägem Aufkleber

Es muss also der Rotationswinkel mit der besten Eignung selektiert werden.

$$Rotationswinkel = \begin{cases} WinkelPackung, & \text{wenn } |WinkelPackung - WinkelIOShape| < 20 \\ WinkelIOShape, & \text{sonst} \end{cases}$$

Falls die Höhe der Packung größer als die Breite ist, ergibt dies für die Packung einen um genau 90° gedrehten Winkel, was jedoch vor der Entscheidung korrigiert wird.

5.2.2 Rotations-Algorithmus

Bildausschnitt

Die Größe des Zielbildes wird vor der Rotation aus den Informationen der IOShape berechnet. Dazu werden alle Punkte der IOShape[1] um das Zentrum der Shape um den gewählten Winkel rotiert. Aus der Ergebnismenge werden die Positionen mit dem größten Abstand zum Zentrum verwendet. Das Rechteck, welches diese umschließt wird als Größenangabe für den Bildausschnitt nach der Rotation verwendet.

Mathematischer Hintergrund

Die neuen Koordinaten eines Punktes nach der Rotation ([KrdTr]) in einer Ebene um den Ursprung des Koordinatensystems werden mit Hilfe der trigonometrischen Funktionen berechnet.
$x' = x * cos\phi + y * sin\phi$
$y' = -x * sin\phi + y * cos\phi$
Axon stellt diese Funktionalität, allerdings nur für einzelne Punkte, schon zur Verfügung. Zur Bildrotation wird jeder Bildpunkt einzeln an seine neue Position gedreht.

Vorwärtsrechnung

Wenn mit Hilfe dieser Formeln die Bildpunktrotation vorwärts berechnet wird, ergeben sich zwei große Nachteile (siehe Abbildung 5.4).

- Das Bild hat Löcher, d. h. in regelmäßigen Abständen tauchen Bildpunkte auf, die keinen Farbwert zugewiesen bekommen haben. Der Grund dafür ist, dass das Ergebnis der Rotationsberechnung gerundet werden muss, um es auf eine Bildpunktkoordinate im Zielbild abzubilden. Wenn nun die neu errechneten Koordinaten zweier benachbarter Bildpunkte in unterschiedliche Richtungen gerundet werden, kann ein Bildpunkt im Zielbild entweder leer bleiben oder doppelt belegt werden.

[1] Wenn dem System keine IOShape übergeben wurde, und der Textdetektor die ROI gefunden hat, wird von ihm eine rechteckige IOShape, welche diese umschließt, generiert und im weiteren benutzt.

- Es befinden sich Kanten im Bild, die den Eindruck entstehen lassen, dass Bildpunkte zeilen- bzw. spaltenweise verschoben seien. Hierfür ist die durch die Rundungsfehler mögliche Doppelbelegung von Bildpunkten im Zielbild verantwortlich.

Beide Probleme lassen sich mit Hilfe einer Rückwärtsrechnung mit zugehöriger Interpolation beseitigen.

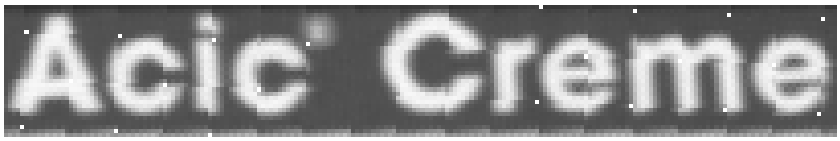

Abbildung 5.4: Ergebnis einer Rotation ohne Rückwärtsrechnung und ohne Interpolation

Rückwärtsrechnung und lineare Interpolation

Das Benutzen der Formeln in Rückwärtsrechnung verhindert bei der Bildrotation das Entstehen von Löchern im Zielbild, indem nicht, wie ursprünglich, jeder Bildpunkt des Ausgangsbildes gedreht und auf das Zielbild gemalt wird, sondern für jeden Bildpunkt des Zielbildes berechnet wird, von welchem Bildpunkt des Ausgangsbildes er seinen Farbwert erhält.

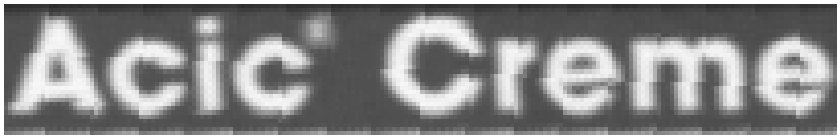

Abbildung 5.5: Ergebnis einer Rotation mit Rückwärtsrechnung ohne Interpolation

Nachdem auf diese Art sichergestellt ist, dass es keine Löcher mehr im Bild gibt, sind jedoch immer noch störende Kanten (siehe Abbildung 5.5) zu erkennen, welche dadurch entstehen, dass bei der Rückrechnung pro Bildpunkt im Zielbild immer nur der Farbwert von einem Bildpunkt des Ausgangsbildes herangezogen wird, obwohl die errechnete Position zwischen vier benachbarten Bildpunkten liegt (siehe Abbildung 5.6).

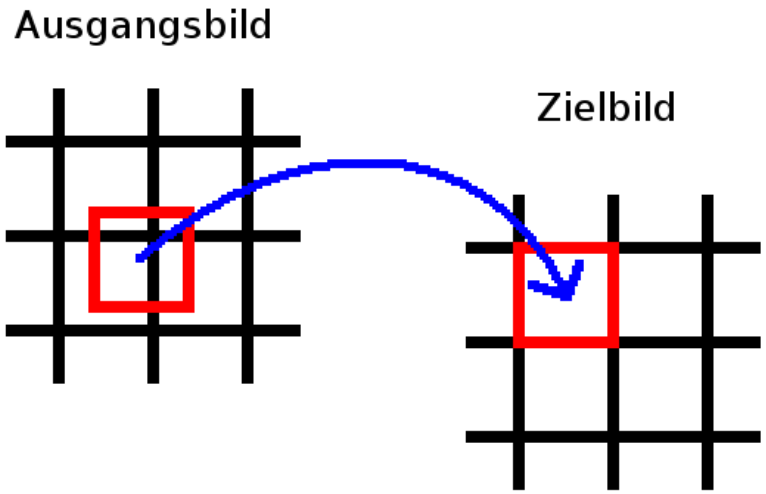

Abbildung 5.6: Interpolation bei der Rotation: Der Farbwert eines Bildpunktes setzt sich aus den Beiträgen von 4 Bildpunkten des Ausgangsbildes zusammen.

Dies hat zur Folge, dass durch die zwangsläufige Rundung Fehler entstehen, und dieses Aliasing-Phänomen auftritt (vgl. [WikiAntiA]).

Mit Interpolation lässt sich dieses Problem beheben, indem der Farbwert für den Bildpunkt im Zielbild anteilig aus den durch die errechnete Position überlagerten Bildpunkten im Ausgangsbild berechnet wird. Je mehr Fläche eines Ausgangsbildpunktes von der Fläche des zurückrotierten Zielbildpunktes überdeckt wird, desto höher ist der Anteil dieses Ausgangsbildpunktes an der Farbgebung des Zielbildpunktes.
Abbildung 5.7 zeigt das zufriedenstellende Ergebnis dieser Rotation mit Rückwärtsrechnung und linearer Interpolation.

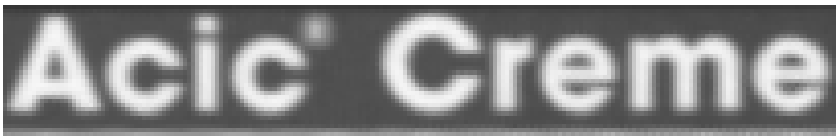

Abbildung 5.7: Ergebnis einer Rotation mit Rückwärtsrechnung und Interpolation

5.3 Invertierung

Da die Texterkennungsprogramme nur richtig funktionieren, wenn die Schrift dunkler als der Hintergrund ist, die Situation auf den Packungen jedoch oft umgekehrt aussieht, muss der ausgewählte Bildausschnitt gegebenenfalls invertiert

werden. Die Schwierigkeit liegt darin, Kriterien festzulegen, wann dies der Fall ist.

5.3.1 Kriterien bei gegebener IOShape

Wenn die vorhandenen Zusatzinformationen eine IOShape beinhalten, die den Text wie in Abbildung 3.2 genau einschließt, wird die Schriftfarbe als Durchschnitt der Farbwerte aller Bildpunkte, die innerhalb dieser IOShape liegen, berechnet. Die Hintergrundfarbe errechnet sich aus allen übrig gebliebenen Bildpunkten. Wenn die Hintergrundfarbe nun dunkler (niedrigerer Grauwert) als die Schriftfarbe ist, wird invertiert.
$InvertierterGrauwert = MaximalerGrauwert - AlterGrauwert$
Der maximale Grauwert beträgt den höchsten Wert, den ein Byte annehmen kann (255).

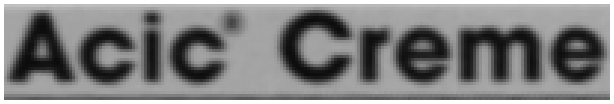

Abbildung 5.8: Invertiertes Graustufenbild

5.3.2 Kriterien ohne IOShape

Es kommt vor, dass nur ein den Text umschließendes Rechteck an das System übergeben wird. In diesem Fall ist unbekannt, welche Bildpunkte den Text und welche den Hintergrund repräsentieren. Zur Lösung dieses Problems gibt es unterschiedliche Ansätze. Zwei von ihnen sind:
1) Die Bildpunkte auf dem Rand des Bildausschnittes werden als Hintergrundfarbe benutzt und mit dem Durchschnittswert der restlichen Bildpunkte verglichen.
2) Zunächst wird der Durchschnitt der Farbwerte aller Bildpunkte gebildet.
Danach wird gezählt, wie viele Bildpunkte heller und wie viele Bildpunkte dunkler als dieser sind. Davon ausgehend, dass der Text weniger Fläche als der Hintergrund beansprucht, muss das Bild invertiert werden, wenn es mehr dunkle als helle Bildpunkte gibt. Methode 1 funktioniert nur sicher, wenn das umschließende Rechteck den Text nicht schon berührt, was jedoch vorkommen kann. Zusätzlich kann es Probleme geben, wenn der Text auf der Packung durch eine Veränderung der Umrandung hervorgehoben wird, und das gegebene Rechteck schon teilweise aus dieser Umrandung herausragt. Methode 2 hingegen verhält sich auch bei diesen Gegebenheiten stabil, weswegen sie vorzuziehen ist.

5.4 Binarisierung

Die Binarisierung wandelt ein Graustufenbild in ein Bild mit nur zwei Farben (in diesem Fall schwarz und weiß) um.

$$b(n) = \begin{cases} 0, & \text{wenn } n < Schwellenwert \\ 1, & \text{sonst} \end{cases}$$

Die Schwierigkeit hierbei liegt in der Wahl des Schwellenwertes, der über die Qualität des Bildes nach der Operation entscheidet. Abbildung 5.9 zeigt, was passiert, wenn der Schwellenwert zu hoch gewählt wurde. Zu viele Bildpunkte des Ausgangsbildes fallen in den Farbbereich, der schwarz wird. Das Bild wird für die Weiterverarbeitung durch eine Texterkennung unbrauchbar.
Abbildung 5.10 zeigt das Gegenteil und somit das Ergebnis einer Binarisierung mit zu niedrigem Schwellenwert. Das Ergebnis einer Binarisierung mit korrekt gewähltem Schwellenwert ist in Abbildung 5.11 zu sehen.

Abbildung 5.9: Zu hoher Schwellenwert für die Binarisierung

Abbildung 5.10: Zu niedriger Schwellenwert für die Binarisierung

Abbildung 5.11: Passender Schwellenwert für die Binarisierung

5.4.1 Histogramm

Ein Histogramm ist ein Flächendiagramm und stellt die Grauwertverteilung eines Bildes graphisch dar. Es liefert Informationen darüber, wie hoch der Anteil jedes Grauwertes am Bild ist. An der X-Achse wird die Helligkeit des Grauwertes aufgetragen und auf der Y-Achse die zugehörige Häufigkeit des Auftretens.

Abbildung 5.12 zeigt das Histogramm des Beispielbildes aus Abbildung 5.8. Das auffällige globale Maximum liegt bei dem Grauwert, den der Text einnimmt. Dieser verläuft dann nach rechts in den helleren Hintergrund.

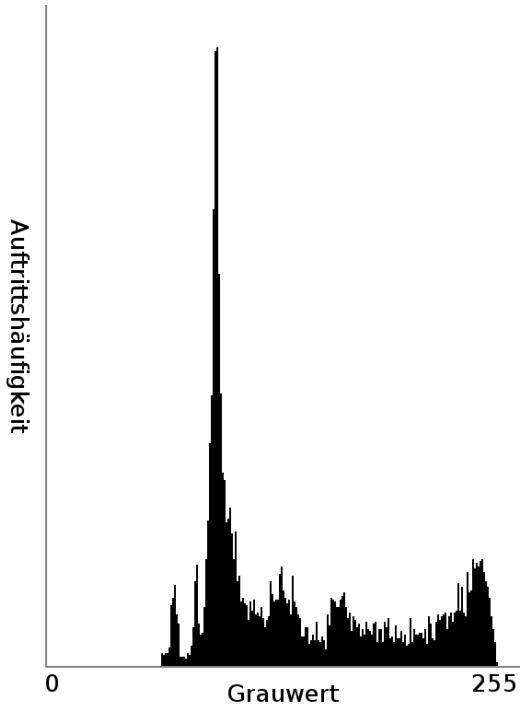

Abbildung 5.12: Grauwert-Histogramm des Beispielbildes "Acic Creme"

5.4.2 Adaptives Binarisieren

Für jede Binarisierung den gleichen Schwellenwert zu benutzen, und dabei das Histogramm des Bildes komplett außer Acht zu lassen, führt zwangsläufig zu schlechten Ergebnissen, da sich Bilder in ihrer Helligkeitsverteilung stark unterscheiden können.
Eines der möglichen Verfahren ist die einfache Adaption, bei der kein global festgelegter Schwellenwert (z. B. 127) herangezogen wird, sondern die Durchschnittshelligkeit aller Farbwerte berechnet und als Schwellenwert benutzt wird. Im Idealfall sollte der Schwellenwert allerdings nicht nur in der Mitte der Verteilungen im Histogramm liegen, sondern sich an der Position befinden, dass er Objekt und Hintergrund optimal voneinander trennt. Häufig liegt diese Position in einem lokalen Minimum des Histrogramms. Weil jedoch oft mehrere lokale Minima vorhanden sind, ist es schwierig, das richtige auszuwählen.
Das Verfahren der einfachen Adaption ist also eine einfache und schnelle Möglichkeit einen guten Schwellenwert zu finden und wird auch im System benutzt, allerdings ist es noch nicht optimal. Es gibt mehrere Ansätze, dieses Problem zu

lösen. Eines davon ist das Verfahren von Otsu, was unter anderem in [LuOtsu] beschrieben wird.

5.4.3 Binarisieren anhand der IOShape

Im Falle des Systems ist es häufig so, dass für das zu erkennende Objekt in Axon schon ein Grundmuster vorhanden ist, was bedeutet, dass ein Bereich im Farbraum für die Textfarbe und einer für die Hintergrundfarbe schon grob festgelegt ist. So kann eine genaue Umrandung um die Schrift in Form einer IOShape übergeben werden (siehe Abbildung 3.2).
Nun kann der durchschnittliche Farbwert der Bildpunkte des Vordergrundes berechnet werden, indem nur der Bereich, der von der IOShape umschlossen ist, betrachtet wird. Aus den restlichen Pixeln des Bildausschnitts wird die durchschnittliche Hintergrundfarbe berechnet. Der Mittelwert dieser beiden Werte wird nun als Schwellenwert benutzt, um die beiden Bereiche bei der Binarisierung optimal voneinander trennen zu können.

6 OCR

Nachdem ein Bild in der Vorverarbeitung rotiert und binarisiert worden ist, muss die eigentliche Texterkennung stattfinden. Hierzu bieten sich einige OCR-Programme aus dem Bereich der freien Software an. Der Vorteil dieser Programme liegt nicht nur darin, dass sie kostenlos zur Verfügung stehen, sondern vor allem darin, dass sie von einer Gemeinschaft vieler Programmierer auf freiwilliger Basis ständig weiterentwickelt werden und der Quelltext zur freien Verfügung steht, was bedeutet, dass er nach belieben weiterentwickelt und für die jeweiligen Bedürfnisse angepasst werden darf.

6.1 PGM-Format

PGM steht für Portable Graymap und ist ein sehr simples Format für Grauwertbilder. Im Header des Beispieles in Abbildung 6.1 steht zunächst “P2”, um das Format festzulegen. Die Raute vor dem nachfolgenden String legt fest, dass es sich um einen Kommentar handelt, der von PGM-Parsern nicht interpretiert wird. Darauf folgen Angaben über die Größe des Bildes in Bildpunkten (hier 7 Zeilen und 13 Spalten) und der maximale Grauwert (4). Nun werden die Grauwerte für jeden Bildpunkt zeilenweise als ASCII-Zeichen abgelegt.
Natürlich können die Bilddaten auch als Binärwerte hinterlegt werden. Hierzu muss lediglich im Header “P5” als Formatangabe stehen.

```
P2
# ASD.pgm
13 7
4
0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 4 0 0 4 4 4 0 4 4 0 0
0 4 0 4 0 4 0 0 0 4 0 4 0
0 4 4 4 0 4 4 4 0 4 0 4 0
0 4 0 4 0 0 0 4 0 4 0 4 0
0 4 0 4 0 4 4 4 0 4 4 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0
```

Abbildung 6.1: Beispielbild im PGM-Format (ASCII)

Ähnliche Formate sind das PBM-Format und das PPM-Format. Sie finden in dem zu entwickelnden System keine Anwendung. Es sei hier jedoch erwähnt, dass PBM für Portable BitMap steht und dieses Format vom Aufbau her dem PGM-Format sehr ähnelt. Dort werden allerdings drei Farbkanäle hinterlegt, wodurch dieses Format 24-Bit-tiefe Farbbilder repräsentieren kann. PPM hingegen steht für Portable PixMap. Hier kann jeder Bildpunkt nur den Wert 0 oder den Wert 1 annehmen. Dieses Format ist somit für binarisierte Bilder konzipiert. Das zu entwickelnde System verwendet trotzdem das PGM-Format, da für zukünftige Anpassungen die Möglichkeit, nicht binarisierte Graustufenbilder zu übergeben, offen gehalten werden soll.

6.2 Die externen Programme

6.2.1 Ocrad

Ocrad (http://www.gnu.org/software/ocrad/ocrad.html) ist eine Kommandozeilensoftware, die Bilder in den Formaten pbm, pgm oder ppm lesen kann, mittels Merkmalsextraktion den enthaltenen Text erkennt, und ihn im Byte-Format (8-bit) oder im UTF-8-Format ausgibt. Die Übergabe des Bildes durch eine Pipe wird unterstützt und über den Parameter "-" aktiviert. Ocrad steht unter der GNU General Public License ([GNUGPL]).

6.2.2 GOCR

Gocr (http://jocr.sourceforge.net) steht ebenfalls unter der GNU General Public License, hat die gleiche Funktionsweise wie Ocrad, hat jedoch einen höheren Be-

kanntheitsgrad und ist unter Linux die Standard-Lösung, wenn es um Texterkennung geht. Die Schnittstelle funktioniert genau wie die von Ocrad. Es werden die selben Formate unterstützt, und es kann ebenfalls eine Pipe verwendet werden.

6.2.3 Tesseract

Tesseract (http://code.google.com/p/tesseract-ocr) basiert auf Quellcode, der Anfang der 90er Jahre von HP entwickelt wurde und unter den Top 3 Engines im 1995 UNLV Accuracy Test war. Ein Jahrzehnt lang wurde das Projekt kaum weiterentwickelt, bis es 2006 unter die Apache-Lizenz gestellt wurde. Seitdem ist das Projekt sehr aktiv und wird ständig verbessert. Tesseract unterstützt ausschließlich das TIFF-Bildformat und bietet auch keine Möglichkeit, das zu erkennende Bild oder den erkannten Text über eine Pipe zu übergeben. Für Ein- und Ausgabe werden externe Dateien benutzt. Dies ist jedoch für die Einbindung in das System nachteilig, da das ständige Schreiben und Lesen von externen Dateien langsam und ineffizient ist. Auch um Tesseract mit Ocrad und GOCR kompatibel zu machen, wurde die Fähigkeit der Pipe-Kommunikation, sowie die Unterstützung für das PGM-Format, zusätzlich implementiert. Die Apache-Lizenz ([OSIApache]), unter der Tesseract steht, erlaubt dies ebenso wie die GPL, solange die Änderungen im Quelltext auch öffentlich gemacht werden. Zusätzlich darf Quelltext, der unter einer dieser Lizenzen veröffentlicht wurde, nicht fest in proprietäre Software hineingelinkt werden. Um die Lizenz nicht zu verletzen, werden diese Programme weiterhin in externen ausführbaren Dateien gehalten.

6.2.4 Vergleich

Geschwindigkeit

Axon muss einen Durchsatz von 3600 Artikeln pro Stunde garantieren, weswegen alle Erkennungsprozesse zeitkritisch sind. Das hat natürlich Auswirkungen auf das System. Ihm bleiben ca. 0.5s Zeit, seine Aufgabe zu erledigen. Deshalb ist es wichtig zu wissen, wie viel Zeit die aufgerufenen externen OCR-Programme für eine Erkennung benötigen.
Ocrad benötigt zur Erkennung aller gegebenen Beispielbilder aus der Testmenge im Schnitt 0,017 Sekunden. GOCR hat einen Bedarf von 0,029 Sekunden, und Tesseract ist mit 0,231 Sekunden mit Abstand am langsamsten. Diese Zeiten wurden auf einem PC ermittelt, der mit 2,8GHz und 512MB RAM dem Standard entspricht, der in den Erkennungstunnels eingesetzt wird. Um Fehlmessungen auszuschließen, die sich bei Zeitmessungen auf Multitasking-Betriebssystemen einschleichen können, wurden die Messungen mit jeweils 1000 Erkennungsaufrufen

durchgeführt und der Mittelwert der gemessenen Zeiten ermittelt.

Genauigkeit

Als Testmenge stehen 77 Beispielbilder von Medikamentenpackungen zur Verfügung. Mit diesen wird ein Genauigkeitstest der externen OCR-Programme vorgenommen.
Die nachfolgende Tabelle stellt dar, wie oft welche OCR-Software die beste Erkennung liefert.

OCR-Software	Ocrad	GOCR	Tesseract
Gewinnhäufigkeit	19	20	44

Dabei kam es 6 mal vor, dass zwei Programme sich den ersten Platz teilten, wobei jedoch kein einziges mal von allen das gleiche Ergebnis geliefert wurde. Die Berechnung des Fehlerwertes einer Erkennung, auf der die Ermittlung des Gewinners unter den OCR-Programmen basiert, wird in Kapitel 7 beschrieben.

Folgerung

Tesseract liefert bei über der Hälfte der Beispiele das beste Erkennungsergebnis, bei ca. einem Viertel der Beispiele ist es sogar unverzichtbar, da Ocrad und GOCR bei Schrifttypen mit Serifen und geringen Buchstabenabstand sehr unzuverlässig arbeiten und eine hohe Fehlerrate haben. Da Tesseract jedoch sehr viel Zeit in Anspruch nimmt, wurde im entwickelten System die Möglichkeit implementiert, jede einzelne OCR-Software deaktivieren zu können. So können gegebenenfalls zuerst nur die schnellen Programme zur Erkennung herangezogen werden, und wenn diese kein gutes Ergebnis geliefert haben, kann je nach noch zur Verfügung stehender Zeit, zusätzlich das langsamere aber zuverlässigere Tesseract gestartet werden.

7 Datenbankabgleich

Nachdem beispielsweise das binarisierte Bild des Produktes “Acic Creme” aus Abbildung 5.10, wie in Kapitel 5 beschrieben, an alle drei Texterkennungsprogramme übergeben wurde, erhält das System folgende Ausgaben:

Ocrad: `Acic Cieme`
GOCR: `Acio  Oleme`
Tesseract: `Acic’ Creme`

Zusätzlich wird von Axon eine Liste von Medikamentennamen übergeben. Diese kann sehr stark variieren.
Wenn anhand der Größe und des Aussehens des Artikels schon viele Präparate aus der Datenbank ausgeschlossen werden konnten, besteht die Liste der übrig gebliebenen Möglichkeiten vielleicht nur noch aus drei Namen, und die Aufgabe des Systems ist es, zu bestimmen, um welches dieser drei Medikamente es sich handelt, und mit welcher Sicherheit diese Aussage getätigt werden kann.
Wenn dieser Artikel Axon jedoch noch unbekannt ist, und die Region, in der der Text steht, nicht übergeben wurde, sondern von der Textdetektion selbstständig gefunden wurde, kann die Liste der Möglichkeiten nahezu die komplette Menge aus der Datenbank umfassen. Der Datenbankabgleich muss also flexibel genug sein, um mit diesen Aufrufvarianten zuverlässig zu funktionieren. Zusätzlich muss er auch noch bei großen Datenmengen schnell genug sein, um den vorgegebenen Laufzeitrahmen nicht zu sprengen.

7.1 Sonderzeichen entfernen

Viele Medikamente haben ein hochgestelltes Urheberrechtszeichen (“®”) im Namen. Dieses wird von den OCR-Programmen oft als Hochkomma (“’”) interpretiert. Zusätzlich erzeugen diese Programme bei etwas längeren Wortabständen häufig mehrere Leerzeichen. Bevor der eigentliche Datenbankabgleich stattfindet, müssen solche Zeichen erst einmal entfernt werden.
Die Zeichen “‘”, “´”, “,” und “’” werden genau wie Leerzeichen am Anfang und am Ende des Strings komplett entfernt. Mehrfach hintereinander auftretende Leer-

zeichen werden zu einem einzigen zusammengefasst.

“`Acic Cieme`” bleibt somit “`Acic Cieme`”,
“`Acio  Oleme`” wird zu “`Acio Oleme`”,
und “`Acic’ Creme`” wird zu “`Acic Creme`”.

7.2 Stringvergleich

Um die erkannten Strings mit denen aus der Liste der übergebenen Möglichkeiten zu vergleichen, muss eine besondere Art des Stringvergleiches herangezogen werden. Eine einfache Prüfung auf Gleichheit gibt schon bei einer kleinen Abweichung `false` zurück. So könnte z. B. ein “`Acic Cieme`” niemals dem Produkt “`Acic Creme`” zugeordnet werden, obwohl es für den menschlichen Betrachter eindeutig wäre.
Es gibt verschiedene Ansätze um ähnliche Strings, die nicht unbedingt genau gleich sein müssen, einander zuzuordnen.
Einer davon ist der Soundex-Vergleich ([Sndex]). Dabei handelt es sich um einen phonetischen Algorithmus, der für jedes Wort, in Abhängigkeit seines Klangs, eine Zeichenfolge erzeugt. Somit können zwei Wörter, die komplett unterschiedlich geschrieben sind, einander trotzdem zugeordnet werden, wenn ihr Klang gleich ist. Allerdings sind die Fehler, die die OCR-Programme machen, nicht unbedingt der Art, dass die Wörter ihren Klang behalten. Deshalb ist die Soundex-Methode für das Texterkennungssytem ungeeignet.
Ganz anders präsentiert sich die Levenshtein-Distanz.

7.2.1 Levenshtein-Distanz

Die Levenshtein-Distanz[1] wird auch Edit-Distanz genannt, denn es handelt sich um einen Algorithmus zur Bestimmung der minimal nötigen Anzahl an Editierungsoperationen (Einfügen, Löschen und Ersetzen) um eine Zeichenkette in eine andere zu überführen. Somit ist beispielsweise die Levenshtein-Distanz zwischen “`Acio Oeme`” und “`Acic Creme`” gleich 3.
Das erste “`o`” wird durch ein “`c`” ersetzt, das zweite“`O`” durch ein “`C`” und ein “`r`” wird eingefügt.
Natürlich könnte auch jede beliebige höhere Anzahl an Operationen benötigt werden, um diese Umwandlung durchzuführen. Beispielsweise könnten einfach alle

[1] Die Levenshtein-Distanz wurde nach ihrem Erfinder, dem russischer Mathematiker Wladimir Iossifowitsch Lewenshtein, benannt.

Zeichen aus dem ersten String gelöscht und dann alle Zeichen des zweiten Strings eingefügt werden. Der Weg, auf dem jedoch die kürzeste Distanz gefunden wurde, ist in der folgenden Tabelle markiert.

		A	**c**	**i**	**c**		**C**	**r**	**e**	**m**	**e**
	0	1	2	3	4	5	6	7	8	9	10
A	1	**0**	1	2	3	4	5	6	7	8	9
c	2	1	**0**	1	2	3	4	5	6	7	8
i	3	2	1	**0**	1	2	3	4	5	6	7
o	4	3	2	1	**1**	2	3	4	5	6	7
	5	4	3	2	2	**1**	**2**	3	4	5	6
O	6	5	4	3	3	2	2	**3**	4	5	6
e	7	6	5	4	4	3	3	3	**3**	4	5
m	8	7	6	5	5	4	4	4	4	**3**	4
e	9	8	7	6	6	5	5	5	4	4	**3**

Diese Tabelle kommt zustande, indem zuerst einmal die erste Zeile und die erste Spalte mit der jeweiligen Buchstabenposition vorinitialisiert werden, worauf sich die Berechnung der restlichen Werte in der Tabelle auf folgende Rekursionsgleichung stützt:

$$T_{x,y} = min \begin{cases} T_{x-1,y-1} + 0 & \text{gleicher Buchstabe} \\ T_{x-1,y-1} + 1 & \text{einfügen} \\ T_{x-1,y} + 1 & \text{ersetzen} \\ T_{x,y-1} + 1 & \text{löschen} \end{cases}$$

In diesem Modell sind die Kosten für Einfügen, Ersetzen und Löschen gleich eins, und die Kosten für das Beibehalten eines Buchstabens gleich null. Dank der rekursiven Gleichung findet sich die Kostensumme automatisch im letzten Tabellenfeld wieder und kann von dort ausgelesen und weiterverarbeitet werden.

7.2.2 Anpassung mit Gewichten

Damit der Levenshtein-Algorithmus für das System wirklich von Nutzen ist, wird eine Anpassung an ihm vorgenommen, die darin besteht, dass das Ersetzen bestimmter Zeichen mit unterschiedlichen Kosten gewertet wird. Zur Identifikation einer Arzneipackung ist häufig die Dosierung der entscheidende Faktor. Aus diesem Grund bekommt der Austausch einer Zahl im erkannten String einen höheren Kostenwert zugewiesen, als der Austausch eines Buchstabens.

7.2.3 Verwechslungen -> Zeichengruppen

Die externen OCR-Programme verwechseln auf Grund der optischen Ähnlichkeit einige Zeichen bei der Erkennung häufiger miteinander als andere. Diese Zeichen werden in Gruppen zusammengefasst:

i I l L | 1
e c (C <
O o 0 D
X K k x
f t r
2 Z z
5 s S
8 B
. ,

Bei der Zusammenstellung der Gruppen ist zu beachten, dass kein Zeichen in zwei Gruppen auftaucht, denn so wäre ein Brücke zwischen ihnen geschlagen, und es könnten (wenn auch mit mehr als nur einem Schritt) Zeichen, die keine optische Ähnlichkeit besitzen, kostengünstig ineinander überführt werden, was es unbedingt zu vermeiden gilt.
Wenn nun während der Ausführung des Levenshtein-Algorithmus' ein Austausch innerhalb einer Zeichengruppe stattfindet, wird dieser mit einem niedrigeren Wert belegt als andere Austausche.

Da der Algorithmus den Weg mit den geringsten Werten in der Tabelle zur Berechnung des Ergebnisses wählt, funktioniert dieses Verfahren der angepassten Gewichte ohne weitere Veränderungen problemlos.

7.2.4 Parametrierung

Um die Laufzeit des Algorithmus' zu verringern werden statt Fließkommazahlen ganzzahlige Werte verwendet. Diese werden, um eine maximale Flexibilität zu erhalten, aus einer hinterlegten Ini-Datei ausgelesen, in der folgende Werte zu finden sind:

Name	Bedeutung	Wert
`similar_chars`	Wert zum Beibehalten eines Buchstabens	0
`similar_digits`	Wert zum Beibehalten einer Ziffer	0
`different_chars`	Austausch von zwei unterschiedlichen Buchstaben	100
`different_digits`	Austausch von zwei unterschiedlichen Ziffern	170
`char_case_change`	Austausch Groß- und Kleinschreibung	10
`different_digit_to_char`	Ziffer durch Buchstaben ersetzen	100
`different_char_to_digit`	Buchstaben durch Ziffer ersetzen	170
`group_char_to_char`	Aust. e. B. durch e. anderen innerhalb e. Gruppe	15
`group_digit_to_char`	Aust. e. Z. durch e. B. innerhalb e. Gruppe	40
`group_char_to_digit`	Aust. e. B. durch e. Z. innerhalb e. Gruppe	40

Die Werte sind während der Entwicklung händisch angepasst worden und werden im Laufe der Zeit sicherlich noch weiter feinjustiert werden.

7.3 Anwendung

Beim Vergleich der von den externen OCR-Programmen erkannten Texten mit der durch die Datenbank gegebenen Möglichkeiten wird der Levenshtein-Algorithmus mit den angepassten Gewichten bei jedem Stringvergleich angewendet.

7.3.1 Komplette Strings

Wenn der Text in einem durch ein Grundmuster vorgegebenen Bereich auf der Packung erkannt wurde, gibt es pro OCR-Programm zwei Strings, die mit den Datenbankeinträgen abgeglichen werden. Es sind zwei Strings, da zwar immer die Hauptrichtung der Packung sowie der zu lesende Bereich durch das Grundmuster gegeben ist, es jedoch immer sein kann, dass die Packung um 180° gedreht ist, also auf dem Kopf steht. Folglich sind es, wenn alle drei OCR-Programme aktiviert sind, sechs erkannte Zeichenfolgen.
Am Beispiel “Acic Creme” könnten diese so aussehen:

```
Acic Cieme
Acio Oleme
Acic Creme
awal_ ._ _O
awa . .
SLLISIQ ,3 OV
```

Falls es in diesem Fall ein Medikament in der Datenbank gäbe, das eine der unteren drei Bezeichnungen trüge, wäre dieser Satz von erkannten Zeichenketten nicht mehr eindeutig zuzuordnen. Dieses Szenario lässt sich theoretisch nicht vermeiden, allerdings ist in der Praxis während der Tests keine falsche Zuordnung auf Grund dieser Eigenheit aufgetreten, da die in auf dem Kopf stehenden Bildern erkannten Strings nahezu immer komplett sinnlose Zeichenfolgen sind, und beim Datenbankabgleich automatisch ausgeschlossen werden.

Bei diesem Datenbankabgleich wird jede Zeichenkette aus der Menge der durch die Datenbank gegebenen Möglichkeiten mit jeder Zeichenkette aus der Menge der erkannten Strings mit Hilfe des angepassten Levenshtein-Algorithmus' verglichen. Jedes Element aus der Datenbank bekommt den passendsten Kandidaten aus den erkannten Strings, sowie den zugehörigen Ähnlichkeitswert des Vergleichs, zugeordnet. Dieser Wert wird nun auf den Bereich zwischen `0` und `1` normalisiert.

Folgenden Tabelle stellt das Ergebnis (Plätze 1-10) anhand des Beispieles "Acic Creme" dar:

Wert	Datenbank	erkannt
`1.00`	Acic Creme	Acic Creme
`0.76`	Algesal Creme	Acic Creme
`0.66`	Actilyse 10mg	Acic Cieme
`0.65`	Actilyse 20mg	Acio Cieme
`0.65`	Actilyse 50mg	Acic Cieme
`0.64`	Aldactone 10 ml	Acic Creme
`0.64`	Antifungol HEXAL Creme	Acic Creme
`0.64`	Akineton Tabletten	Acio Oleme
`0.63`	Arelix 6-Tabletten	Acio Oleme
`0.63`	ACC injekt	Acic Cieme

Diese Tabelle wird in Form einer Variablen vom Typ
`std::vector<std::pair<double,std::pair<std::string,std::string > > >`
an Axon zurückgegeben, das die in ihr enthaltenen Informationen zur Klassifikation des Objektes benutzt.

7.3.2 Einzelne Wörter

Wenn für das zu erkennende Objekt keine IOShape mit übergeben werden konnte, und der Textdetektor deshalb erst einmal die zu lesenden Bereiche finden musste (siehe Abbildung 4.21), wird der zu erkennende Text in mehrere Bereiche unterteilt. Darunter finden sich Bereiche, die nicht zum Namen gehören, jedoch Strukturen aufweisen, die den Textdetektor dazu gebracht haben, auch sie an die

OCR-Programme weiterzugeben.
So ergeben sich im Beispiel “Acic Creme”, wenn vom Textdetektorergebnis aus Abbildung 4.21 ausgegangen wird, die Zeichenketten
“`|Acic`”, “`Acio`”, “`IAcic`”,
“`_ _o |`”, “`.. l`”, “`_:> VI`”,
“`Cieme`”, “`Oleme`”, “`Creme`”,
“`awa__`”, “`awaJ`”, “`GUJGJQ`”,
“`.. | . . - -`”, “`t`”, “`V`”, “`- - . . | ..`” und “`A`”, sowie zwei leere Strings.
Wenn Axon anhand der Größe und der Farbe des Artikels die Menge aller Zeichenketten aus der Datenbank schon einschränken kann, jedoch kein passendes Grundmuster gefunden wird, werden je nach Geläufigkeit der erkannten optischen Eigenschaften ca. 20 Zeichenketten an das System übergeben. Nun muss jedes erkannte Wort (in diesem Fall 18 mit einer Länge von ca. sechs Buchstaben) mit allen Wörtern aus diesen 20 Zeichenketten verglichen werden. Ausgehend von einem Schnitt von zwei jeweils sechs Zeichen langen Wörtern pro Zeichenkette ergibt sich eine Menge von 25920 einzelnen zu gewichtenden Zeichenvergleichen. Anzahl der Namen in der Datenbank kann den Wert 20 jedoch je nach Axons Konfiguration auch überschreiten.

7.4 Komplexität

Die Komplexität des Datenbankvergleichalgorithmus lässt sich unabhängig davon bestimmen, ob komplette Strings oder einzelne Wörter verglichen werden. Im Endeffekt wird jedes von den OCR-Programmen erkannte Zeichen einmal mit jedem Zeichen aus der übergebenen Liste verglichen. Wird die Anzahl der erkannten Zeichen als x bezeichnet und die Anzahl aller Zeichen in der Liste als y , ergibt sich in der Groß-O-Notation:
$f \in \Theta(x * y)$

7.5 Optimierung

In Anbetracht der Menge von ca. 25.000 einzelnen zu gewichtenden Zeichenvergleichen ist eine Optimierung sinnvoll. Hierbei kommen unterschiedliche Techniken zum Einsatz:

- Kleine Low-Level-Funktionen, die in der inneren Schleife des Levenshtein-Algorithmus' aufgerufen werden, werden als `inline` deklariert. Somit wird die Zeit, die für die Übergabe der Parameter über den Stack benötigt wird,

eingespart.

- Die Tabelle, mit welcher der Algorithmus arbeitet (siehe Kapitel 7.2.1) wird in der Größe 256*256 angelegt. So kann die Multiplikation bei der Indizierung zur Adressierung der zweiten Dimension durch eine bitweise Shift-Operation («8), ersetzt werden.

- Obwohl die Buchstaben innerhalb der zu vergleichenden Wörter eigentlich vom Typen `unsigned char` sind, werden trotzdem Variablen vom Typ `int` verwendet. Diese benötigen im Normalfall (in einer 32-Bit-Umgebung) zwar den vierfachen Speicherplatz (vier Byte), da ein `char` jedoch vor den meisten Operationen sowieso in ein `int` konvertiert werden muss, kann so Zeit eingespart werden, weil diese Konvertierung wegfällt.

- Zur Berechnung der Kosten für das Ersetzen eines Buchstabens durch einen anderen werden im Normalfall die zuvor aus der Ini-Datei geladenen Zeichengruppen nach Treffern durchsucht. Um diesen Vorgang zu beschleunigen, wird ein 65536 (256^2) int großes Array vorberechnet, in welchem die Kosten für das Ersetzen jedes Buchstabens mit jedem anderen, sowie das Einfügen und Entfernen, enthalten sind und nur noch ausgelesen werden müssen.

8 Tests

Das System wurde während der Entwicklung ständig getestet, um die integrierten Verfahren auf Korrektheit zu prüfen. In diesem Kapitel werden jedoch lediglich die abschließenden Tests beschrieben.

8.1 Beispielbilder

Es stehen 77 Beispielbilder zur Verfügung, die in Axons Betrieb beim Kunden entstanden sind. Dabei handelt es sich um Artikel, für die schon Grundmuster eingerichtet sind. Somit kann für Tests ohne den Textdetektor eine IOShape übergeben werden.

8.2 Testumgebung

Als Testumgebung wurde ein Programm entwickelt, dass das System in der selben Art wie Axon aufruft. Es läd die Beispielbilder und die zugehörigen IOShapes von einem Datenträger in den Hauptspeicher, liest Artikelnamen zur Übergabe in der Liste aus einer Datenbank und bietet Schalter zur Konfiguration unterschiedlicher Testverfahren. Diese Schalter ermöglichen folgende Entscheidungen:

- Soll das System incl. Textdetektor getestet werden oder soll dieser übergangen werden und dafür zusätzlich eine IOShape übergeben werden?
- Welche der drei externen OCR-Programme sollen aufgerufen werden?
- Wie hoch ist die Sicherheitsgrenze? D. h. unterhalb welchen Sicherheitswertes werden die Erkennungen verworfen? Die Sicherheitsgrenze ist ein wichtiger Parameter, denn sie entscheidet darüber, ob ein unsicher erkannter Artikel noch als erkannt gewertet werden soll oder nicht.

Das Programm bietet zusätzlich Möglichkeiten zur graphischen Ausgabe der Bilder, der IOShapes, der vom Textdetektor gefundenen Textregionen, der unterschiedlichen Ergebnisse der externen OCR-Programme und der durch den Datenbankabgleich erweiterten Liste. Diese Ausgabe können in Textdateien umgeleitet

werden, um eine Analyse zu vereinfachen.
Die Testumgebung kann zusätzlich automatisiert werden, sodass die komplette Testmenge verarbeitet wird, und die daraus resultierende Statistik am Ende ausgewertet werden kann.

8.3 Ergebnis

8.3.1 Geschwindigkeit

Vor dem Test der Geschwindigkeit des Systems, wurde zunächst die Zeit gemessen, die die automatisierte Testumgebung benötigt, um die zu übergebenden Bilder zu laden. Diese Zeit wurde im folgenden von den Testzeiten abgezogen, um die Netto-Zeiten der einzelnen Teile des Systems zu erhalten.
Die durchschnittlichen Laufzeiten der Komponenten betragen:

- Textdetektor: 0,23s
- Bildverarbeitung: 0,03s
- OCR (alle drei zusammen): 0,28s
- Datenbankabgleich: 0,02s

Die Bildverarbeitung zeigt sich erwartungsgemäß schnell, wobei hingegen der Textdetektor mehr Zeit benötigt. Er ist zusätzlich in seiner Laufzeit sehr variabel, was daran liegt, dass seine Komplexität $f \in \mathcal{O}(x^2)$ beträgt (x ist die Anzahl der durch Floodfilling auf der Packung gefundenen Regionen/Buchstabenkandidaten). In ihm steckt noch Potential zur Optimierung, vor allem was seine Laufzeitzuverlässigkeit betrifft. Die Laufzeit der externen OCR-Programme verhält sich innerhalb des Systems natürlich genauso, wie sie es schon bei den einzelnen Tests aus Kapitel 6 tat. Sie ist stabil. Der Datenbankabgleich ist dank der Optimierungen im Falle einer übergebenen Liste mit 77 Artikelnamen sehr schnell und somit auch für längere Übergabelisten gewappnet. Die Länge dieser Liste ist von Axons Konfiguration abhängig.
Die vorgegebene maximale Gesamtlaufzeit von 0.5s wird nur sicher eingehalten, wenn der Textdetektor nicht aufgerufen wird. Wenn er benutzt werden soll, wird durch Verzicht auf Tesseract bei der Schrifterkennung die Gesamtlaufzeit im Schnitt auch unter 0.5s bleiben, allerdings wird in Fällen, wo der Textdetektor sehr viele Regionen auf der Packung findet und miteinander vergleichen muss, diese Zeit überschritten.

8.3.2 Sicherheitsgrenze

Zu der Liste der Zeichenketten, mit der der Datenbankabgleich die Ergebnisse der OCR-Programme vergleichen soll, wird eine Sicherheitsgrenze mit übergeben. Sie entscheidet, wie hoch die Ähnlichkeit des gelesenen Textes mit einem Eintrag aus der Datenbank zu sein hat, damit noch eine Zuordnung vorgenommen wird. Kapitel 7 hat gezeigt, dass im Normalfall immer mehrere Listeneinträge zugeordnet werden. Dies ist auch so gewünscht, denn Axon nutzt mehrere unterschiedliche Erkennungsmöglichkeiten und möchte vom System nur die Information haben, welche der übergebenen Vorschläge wie gut zum Bild passen. Die Zeichenkette mit dem höchsten Sicherheitswert ist meistens die richtige, und die restlichen können ignoriert werden. Wenn jedoch dieser Sicherheitswert niedrig ist, was bedeutet, dass keine Zeichenkette wirklich gut gepasst hat, weil z. B. der Text für die OCR-Programme unleserlich war, kann es vorkommen, dass die beste der vorhandenen Zuordnungen falsch ist. Axon übergibt die Sicherheitsgrenze, um einen Wert festzulegen, der mindestens erreicht werden muss, um das Ergebnis als glaubhaft einstufen zu können. Eine Nichterkennung ist im Zweifelsfall immer besser als eine Falscherkennung.
Deshalb muss eine Sicherheitsgrenze gefunden werden, die solche Falscherkennungen so gut wie möglich ausschließt und sie zu Nichterkennungen macht. Die Zahl der richtig erkannten Artikel lässt sich jedoch mit der Anpassung der Sicherheitsgrenze natürlich nicht verbessern, sondern nur verschlechtern. Das Erhöhen der Sicherheitsgrenze macht nämlich nicht nur aus Falscherkennungen Nichterkennungen, sondern auch aus richtigen Erkennungen. Folglich darf die gewählte Sicherheitsgrenze auch nicht zu hoch sein.
Die folgenden vier Diagramme zeigen die Anzahl sowie den prozentualen Anteil der richtigen Erkennungen, der Nichterkennungen und der Falscherkennungen in Abhängigkeit der Sicherheitsgrenze mit Übergabe von IOShapes, also ohne Benutzung des Textdetektors.

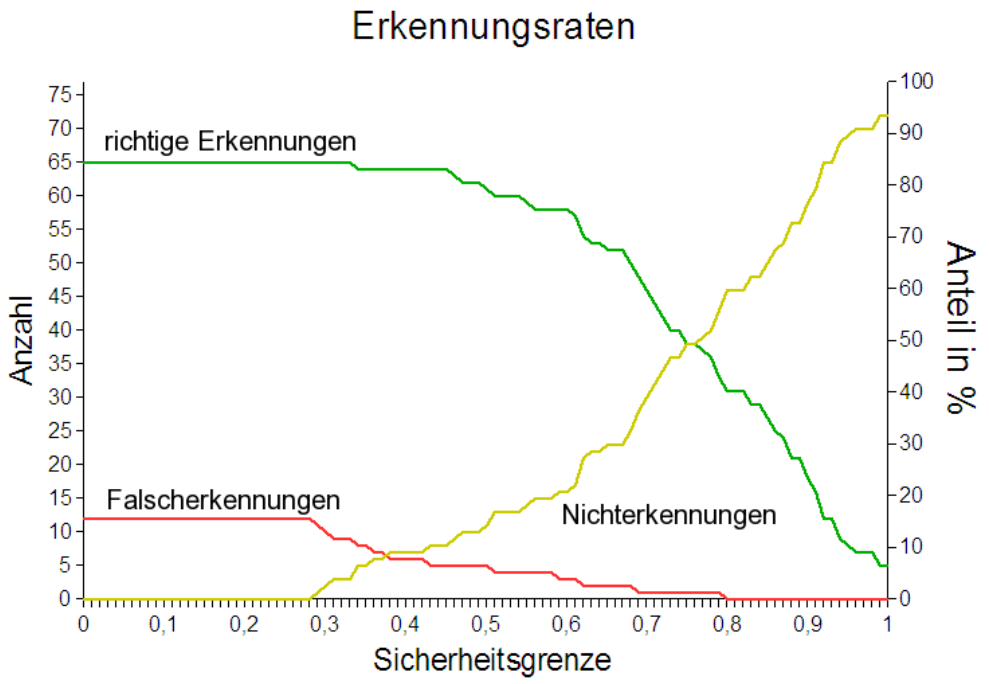

Abbildung 8.1: Erkennungsraten von Ocrad mit IOShapes

Hier wird direkt deutlich, was für einen großen Einfluss die Sicherheitsgrenze auf die Zahl der richtigen Erkennungen hat. Wenn jedoch alle Falscherkennungen eliminiert werden sollten (Sicherheitsgrenze >= 0.80), müsste auch in Kauf genommen werden, dass über die Hälfte der richtigen Erkennungen als Nichterkennung gewertet werden würde, was selbstverständlich nicht der Sinn der Sache ist. Dabei muss bedacht werden, dass im späteren Einsatz meist nicht so viele Namen übergeben werden, und je kürzer die Liste der Namen ist, desto weniger ähnliche Namen kommen vor, was zur Folge hat, dass weniger Falscherkennungen vorkommen. Alle Werte in diesem Kapitel beziehen sich also nur auf die hier verwendete Testmenge. Genaue Werte für den Einsatz in Axon, sowie eine dafür geeignete Sicherheitsgrenze können hier noch nicht geliefert werden.

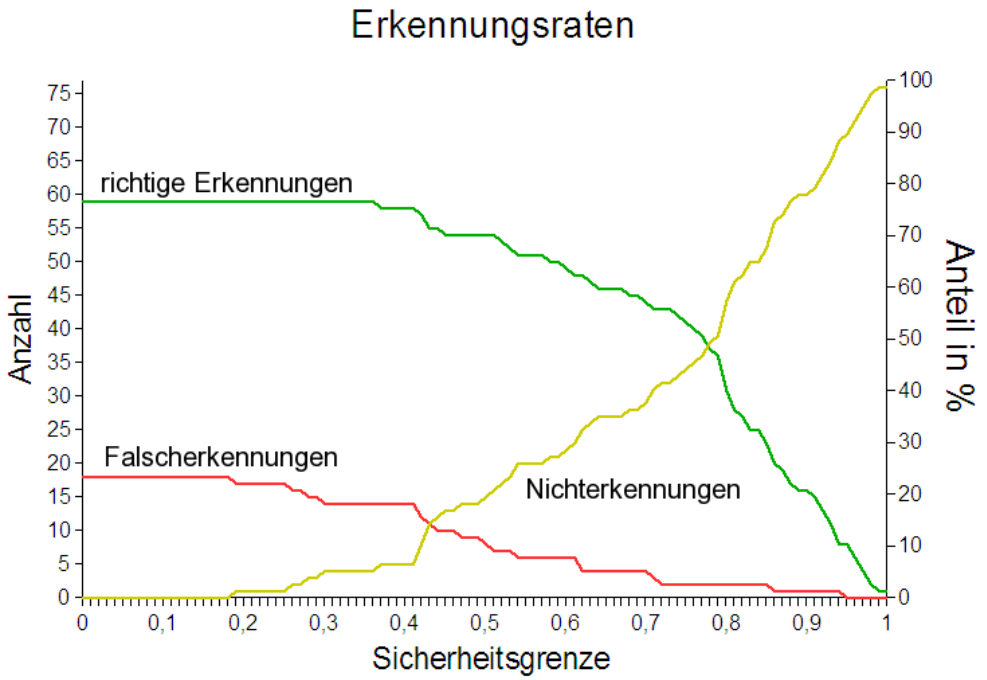

Abbildung 8.2: Erkennungsraten von GOCR mit IOShapes

GOCR hat eine schlechtere Erkennungsrate als Ocrad, mehr Falscherkennungen und fordert eine höhere Sicherheitsgrenze zum Ausschluss aller Falscherkennungen.

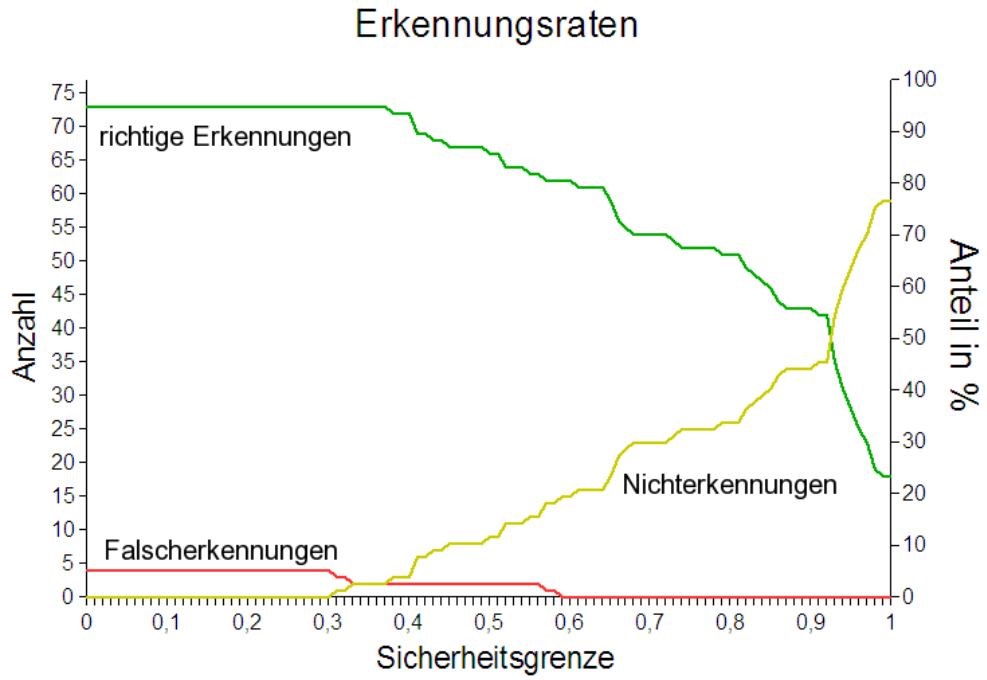

Abbildung 8.3: Erkennungsraten von Tesseract mit IOShapes

Tesseract schneidet wie erwartet mit Abstand am besten ab. Besonders auffällig ist, dass schon ab einer Sicherheitsgrenze von 0.6 alle Falscherkennungen eliminiert sind, jedoch noch über 60 richtige Erkennungen übrig bleiben. Je höher die Grenze wandert, desto mehr von ihnen fallen dann natürlich in den Bereich der Nichterkennungen.

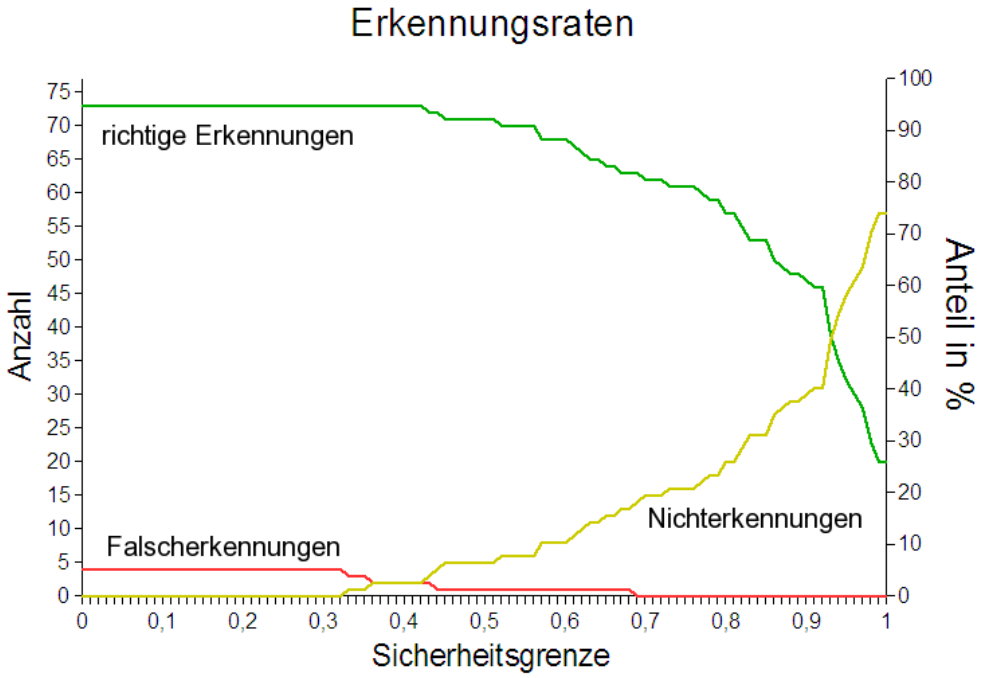

Abbildung 8.4: Erkennungsraten von Ocrad + GOCR + Tesseract mit IOShapes

Das beste Ergebnis liefert natürlich die Kombination aus allen drei OCR-Programmen. Hier fällt auf, dass sich eine Falscherkennung mit einer Sicherheit von 0.69 unter den Erkennungen befindet und die anderen einen Wert von unter 0.45 haben. Wenn also diese Falscherkennung nicht wäre, könnte mit einer Sicherheitsgrenze von 0.46 eine Erkennungsrate von 71 richtigen Erkennungen und nur sieben Nichterkennungen erreicht werden, was 92% entspricht. Diese eine für den Test fatale Falscherkennung ergibt sich daraus, dass in der Testmenge die Medikamente “Afonilum retard” und “Afonilum retard forte” vorkommen. Hier wurde auf der Packung für “retard forte” eine Schriftart verwendet, die allen drei OCR-Programmen starke Probleme bereitet. Zusätzlich ist das Wort “forte” sehr klein geschrieben und kann überhaupt nicht gelesen werden. Deshalb werden die drei Leseergebnisse des Bildes dieses Artikels, welche “Atonilum la r” , “AloniIum a rB” und “Afonilum mam mm” lauten, mit einem Wert von 0.69 dem Artikel “Afonilum retard” zugeordnet. Auch hier muss bedacht werden, wie das System später in Axon eingesetzt wird, denn die Packungen der Artikel “Afonilum retard”

und “Afonilum retard forte” unterscheiden sich in Größe und Seitenverhältnis so stark, dass schon lange bevor das System aufgerufen wird, einer der beiden Artikel ausgeschlossen werden kann, und somit beide Namen nicht in der Liste der möglichen Namen an das System übergeben werden.

Das folgende Diagramm zeigt die Erkennungsraten, wenn keine IOShape mit übergeben wird, sondern der Textdetektor die zu untersuchenden Regionen selber finden muss und sie von allen drei externen Programmen gelesen werden.

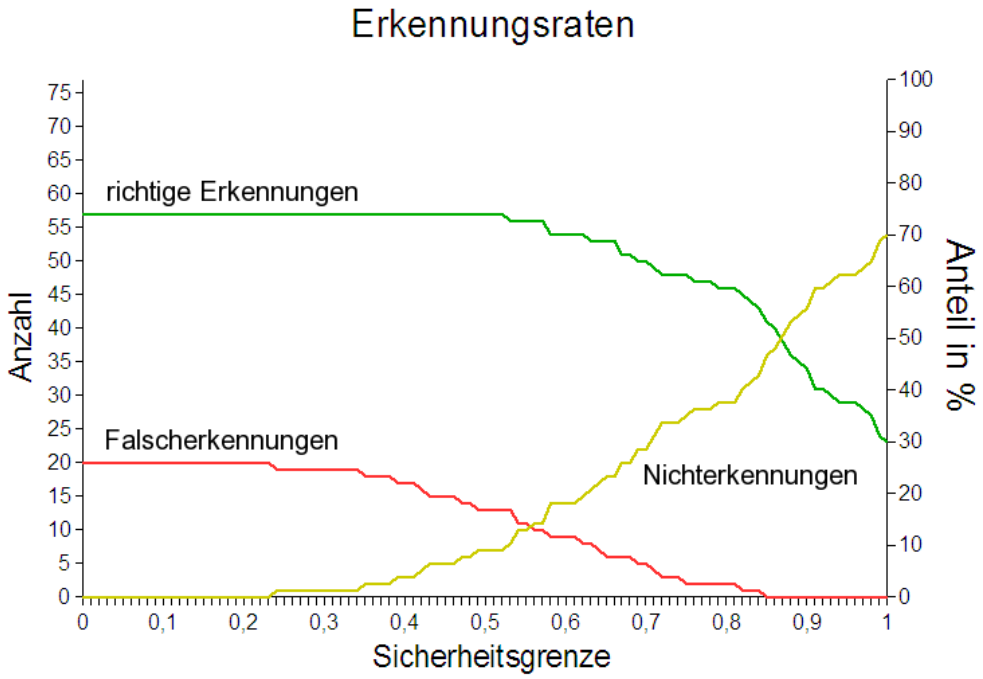

Abbildung 8.5: Erkennungsraten von Ocrad + GOCR + Tesseract mit Textdetektor

Hier ist das Ergebnis deutlich schlechter als im Versuch mit Übergabe der IOShapes, denn der Textdetektor findet oft Regionen, die zwar Text beinhalten, jedoch nicht zum Artikelnamen gehören, wie z. B. Text im Herstellerlogo. Es kommt ebenfalls vor, dass Worte nicht komplett zusammenhängend segmentiert werden, was vom Datenbankabgleich nicht abgefangen werden kann. Hier bietet der Textdetektor noch Möglichkeiten zur Verbesserung.

8.4 Bewertung

Das System erreicht im Schnitt gute Erkennungsraten, hat jedoch seine Stärken und Schwächen. Es funktioniert sehr gut, wenn eine IOShape übergeben wird, und

die Buchstaben der Schrift im zu untersuchenden Bereich durch die Binarisierung gut voneinander trennbar sind und die Schriftgröße nicht zu gering ist. Bei kleinen undeutlichen Schriftarten ist die Erkennungsrate gering. Der Textdetektor hat seine Hauptschwäche in der unberechenbaren Laufzeit, die bei Packungen, auf denen viele kleine Regionen segmentiert werden, steil nach oben geht. Generell liefert er jedoch bei einem Großteil der Bilder in der Testmenge gute Ergebnisse. Der Datenbankabgleich stellt eine große Stärke des Systems dar. Er ist nicht nur schnell und macht Fehler der Texterkennung wett, sondern er ist auch flexibel genug, um zur Lösung von anderen Problemen, die einen solchen Stringvergleich benötigen, eingesetzt zu werden.

9 Ausblick

9.1 Einsatz in Axon

Bei Axons Konfiguration kann das hier entwickelte System als Klassifikator oder als Verifikator eingebunden werden. Wenn bei der Erkennung eines Artikels anhand von Form, Farbe und Größe schon viele Bezeichner ausgeschlossen werden konnten, muss beispielsweise nur noch zwischen zwei unterschiedlichen Dosierungen des gleichen Medikaments unterschieden werden. Abbildung 9.1 und 9.2 zeigen einen solchen Fall.

Abbildung 9.1: Beispielbild “ACIC 250 PI”

Abbildung 9.2: Beispielbild “ACIC 500 PI”

Axon übergibt dem System das zu erkennende Bild, eine IOShape um die zu lesende Zahl und eine Stringliste mit den Einträgen “250” und “500”. Das System bekommt von den externen OCR-Programmen die Strings “250”, “2S0” und “25O”

geliefert, worauf der Datenbankabgleich folgendes Ergebnis an Axon zurück gibt:

Wert	Datenbank	erkannt
1.00	250	250
0.43	500	250

Axon kann nun eine eindeutige Zuordnung vornehmen.

Ein weiteres Beispiel sind die Packungen von Ratiopharm, welche sich teilweise sehr ähnlich sehen, jedoch fast alle eine für den Computer gut lesbare Schriftart aufweisen. Hier kann Axon eine längere Stringliste mit allen Ratiopharm-Produkten, die zu dieser Packungsgröße und Farbe passen, übergeben.

Als Verifikator kann das System zum Einsatz kommen, wenn Axon bis auf einen einzigen Bezeichner alle anderen schon ausschließen konnte, und noch eine zusätzliche Absicherung benötigt wird.
Natürlich sind für beide Fälle (Verifikation und Klassifikation) auch schon andere Funktionen (ohne Texterkennung) implementiert, jedoch stellt das System eine flexibel einsetzbare Erweiterung dar. Wie häufig und in welchen Situationen es genutzt wird, liegt in der Hand des Bearbeiters.

Axon eignet sich nicht nur zur Erkennung von Pharmapackungen, sondern wird in ganz unterschiedlichen Bereichen eingesetzt. Einer davon ist die Erkennung von Zeitungen. Diese funktioniert hauptsächlich über das Lesen von Barcodes aus den aufgenommenen Bildern. Wenn diese jedoch verschmutzt, und damit unleserlich geworden sind, kann das System eingesetzt werden, um die unter dem Barcode gedruckte inhaltlich äquivalente Ziffernfolge zu lesen.
Ebenso wäre es möglich, die Überschriften der Zeitungen zu lesen. Mit einem angepassten Textdetektor müsste nicht einmal bekannt sein, wo auf dem Bild die Überschrift steht, es würde genügen dem Textdetektor mitzuteilen, auf welche Schriftgröße er sich beschränken soll.

9.2 Mögliche Verbesserungen

9.2.1 Bessere OCR-Programme

Eine offensichtliche Verbesserungsmöglichkeit für das System würde die Einführung besserer externer OCR-Programme bringen. Da die drei bis jetzt verwendeten Programme aktive Opensource-Projekte sind, wird davon ausgegangen, dass diese in Zukunft weiterentwickelt werden, und so noch bessere Erkennungsergebnisse zu Stande kommen. Es besteht auch die Möglichkeit, sich an der Entwicklung eines der Programme aktiv zu beteiligen. Es können jedoch ebenfalls komplett neue (auch proprietäre) Programme eingebunden werden.

9.2.2 Textdetektor

Die Zuverlässigkeit des Textdetektors kann durch weitere Eingrenzung der Eigenschaften der gefundenen verbundenen Komponenten erhöht werden. Ziel weiterer Forschungen ist es, die Erkennungsraten, die mit Textdetektor entstehen, denen, die mit Übergabe einer IOShape auftreten, anzunähern.

9.2.3 Geschwindigkeit weiter optimieren

Der Textdetektor benötigt, wie in Kapitel 8.3.1 beschrieben, besonders in Bezug auf seine Laufzeitstabilität noch eine Geschwindigkeitsoptimierung.
Die Bildverarbeitung läuft mit durchschnittlichen 0,03s sehr schnell, und muss momentan nicht optimiert werden. Unter Umständen könnte dies jedoch nötig werden, wenn beispielsweise bei der Zeitungserkennung wesentlich größere Bildausschnitte gedreht werden müssen. Zusätzlich kann die zu verarbeitende Pixelmenge ansteigen, wenn hochauflösende Kameras zum Einsatz kommen. Von Inline-Assembler ist jedoch abzusehen, da der Code portierbar bleiben soll.
Die externen OCR-Programme Ocrad und GOCR sind sehr schnell. Tesseract hingegen benötigt mehr Zeit. Der Aufwand, es zu optimieren, wäre jedoch auf Grund des Umfangs dieses Programms sehr groß. Allerdings kann unter Umständen mit vorherigem Laden des Programms in den Speicher Zeit für den Aufruf eingespart und die Gesamtlaufzeit verkürzt werden, wenn es in einem dauerhaften Thread gehalten wird. Dies gilt es zu prüfen.
Der Datenbankabgleich hat schon einige Optimierungen erfahren und ist mit einer Durchschnittslaufzeit von 0,02s bei Übergabe einer Stringliste mit 77 Elementen schon sehr schnell. Wenn es nötig wird, gegen wesentlich längere Listen abzugleichen, können noch zusätzliche vorzeitige Abbruchbedingungen integriert werden, um weitere Zeichenvergleiche zwischen ohnehin schon weit voneinander entfernten Strings zu unterbinden.

Literaturverzeichnis

[BeJä05] Bernd Jähne, Digitale Bildverarbeitung, Springer, Berlin, 2005

[KrdTr] Hochschule Aalen, Koordinatentransformation bei Drehung des Koordinatensystems
http://www.fbm.fh-aalen.de/profumit/Fachinhalte/Lernhypertext/mathematikthemen/Koordinatentransformation/koordinatentransformation.html
title=Koordinatentransformation\&oldid=40244136
18.02.2008

[LuOtsu] Jürgen Luhr, Methoden und Algorithmen zur Bildsegmentierung
http://lmb.informatik.uni-freiburg.de/lectures/segmentierung/Segmentierung_Schwellwert.pdf
27.02.2008

[WikiAntiA] Wikipedia.org, Antialiasing
http://de.wikipedia.org/w/index.php?title=Antialiasing\&oldid=40725659
21.01.2008

[CoCos] R. Fisher, S. Perkins, A. Walker and E. Wolfart, Connected Components Labeling
http://homepages.inf.ed.ac.uk/rbf/HIPR2/label.htm
25.01.2008

[Sobel] R. Fisher, S. Perkins, A. Walker and E. Wolfart, Sobel Edge Detector
http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm
05.03.2008

[Unsharp] R. Fisher, S. Perkins, A. Walker and E. Wolfart, Spatial Filters
http://homepages.inf.ed.ac.uk/rbf/HIPR2/unsharp.htm
25.01.2008

[GNUGPL] gnu.de, GNU General Public License
http://www.gnu.de/documents/gpl.de.html
28.01.2008

[OSIApache] Open Source Initiative OSI - Apache License, Version 2.0
http://www.opensource.org/licenses/apache2.0.php
28.01.2008

[Sndex] JewishGen® Inc., Soundex Coding
http://www.jewishgen.org/infofiles/soundex.html
18.02.2008

[WikiFldF] Wikipedia.org, Floodfill
http://de.wikipedia.org/w/index.php?
title=Floodfill&oldid=37734352
18.02.2008

[Lvnshtn] Carsten Sulzberger, Levenshtein-Algorithmus
http://www.levenshtein.de/
30.01.2008

[TDNSI] Nobuo Ezaki, Marius Bulacu, Lambert Schomaker, Text Detection from Natural Scene Images: Towards a System for Visually Impaired Persons, Proc. of 17th Int. Conf. on Pattern Recognition (ICPR 2004), IEEE Computer Society, 2004, pp. 683-686, vol. II, 23-26 August, Cambridge, UK

[TDRSV] Wen Wu, Xilin Chen and Jie Yang, Detection of Text on Road Signs From Video, Transactions of intelligent Transportation Systems, Vol. 6, No. 4, Dezember 2005

[TDATS] Rainer Lienhart und Wolfgang Effelsberg, Automatic Text Segmentation and Text Recognition for Video Indexing, ACM/Springer Multimedia Systems, Vol. 8. pp.69-81, Januar 2000

Erklärung über die selbstständige Abfassung der Arbeit

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

(Datum, Ort, Unterschrift)