



Fachhochschul-Bachelorstudiengang  
**SOFTWARE ENGINEERING**  
A-4232 Hagenberg, Austria

# **Kategorisierung von lokalisierten Bilddateien mittels OCR**

Bachelorarbeit

zur Erlangung des akademischen Grades  
Bachelor of Science in Engineering

Eingereicht von

**Simon Gruber**

Begutachtet von Barbara Traxler MSc

Hagenberg, Februar 2024

## Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Die vorliegende, gedruckte Bachelorarbeit ist identisch zu dem elektronisch übermittelten Textdokument.

19.02.2024  
Datum

  
Unterschrift

# Kurzfassung

Diese Bachelorarbeit konzentriert sich auf eines der vielen Anwendungsgebiete der optischen Texterkennung: Extraktion von Textdaten aus digitalen Screenshots. Das Ziel ist es, die Menge und Qualität der gewonnenen Daten zu maximieren, um die Verwaltung von grafischen Ressourcen für die Produktdokumentation von COPA-DATA zu vereinfachen. Gleichzeitig wird ein Beitrag zur Forschung im Bereich der Texterkennung in grafischen Oberflächen geleistet.

Dazu wird eine Auswahl von Algorithmen für die Aufbereitung der Bilder bzw. Filterung der Ergebnisdaten der Texterkennung getroffen, welche in ihrer Grundfunktion erklärt werden. Der Einfluss der gewählten Algorithmen auf die Ergebnisse der Texterkennung wird anschließend anhand gängiger Metriken für die Sprach- und Texterkennung objektiv miteinander verglichen.

Durch den Vergleich der Ergebnisdaten in dem automatisch generierten Bericht wird deutlich, dass die Analyse verschiedener Bilder mit nur einem Verfahren nicht zu optimalen Ergebnissen führt. Besonders bei der Binarisierung, konkret bei Anwendung der unterschiedlichen Schwellenwertverfahren, müssen die Parameter individuell auf die unterschiedlichen Bildmerkmale angepasst werden, um keine wichtigen Details zu verlieren. Für die automatisierte Texterkennung ist es also sinnvoll, den verwendeten Algorithmus manuell auf Basis der Eingangsdaten zu bestimmen oder die Ergebnisdaten mehrerer Algorithmen automatisch zu kombinieren. So können möglichst viele Details eingefangen und das Endergebnis der Texterkennung innerhalb des Bildes optimiert werden.

Für weitere Forschung oder Anpassung an spezifische Anforderungen kann die prototypische Implementierung bzw. deren Komponenten wiederverwendet werden. Durch den modularen Aufbau ist es möglich, neue Funktionalität hinzuzufügen oder bestehende zu verändern. Somit kann selbst nach Änderung der Anzeigesprache oder einer farblichen Neugestaltung der grafischen Oberfläche stets mit wenig Aufwand die ideale Vorgehensweise zur Texterkennung ermittelt werden.

# Abstract

This bachelor's thesis focuses on one of the many fields of optical character recognition: the extraction of textual data from digital screenshots. The goal is to maximize the amount and quality of the resulting data, hence simplifying the management of graphical resources for COPA-DATA's product documentation. Additionally, a contribution to research in the field of text recognition in graphical user interfaces is made.

For this purpose, a selection of algorithms for image processing and filtering of the resulting data is made. After explaining the selected algorithms in their basic function, they are objectively compared using common metrics for speech and text recognition.

By comparing the resulting data in an automatically generated report, it becomes apparent that analyzing different pictures using just one method does not lead to satisfying results. In order to not lose any important details, specifically when working with binarization techniques such as thresholding, the chosen parameters must match the individual characteristics of the picture. Therefore, for automated text recognition, it is sensible to either manually determine the algorithm to be used based on the input data or to automatically combine the results of multiple algorithms. This way, as many details as possible can be captured and the end results of text recognition within the image can be optimized.

The prototypical implementation and its respective components are designed to be reusable for further research or adaptation to specific requirements. Due to the modular structure of the automated comparison system, it is possible to add new functionality and to edit existing features. As a result, a satisfying text recognition approach can always be determined with little effort, even after changing the display language or after a complete redesign of the graphical user interface.

# Inhaltsverzeichnis

<b>Kurzfassung</b>	<b>iii</b>
<b>Abstract</b>	<b>iv</b>
<b>1 Einleitung</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Zielsetzung . . . . .	2
1.3 Aufbau . . . . .	3
<b>2 Optische Texterkennung</b>	<b>4</b>
2.1 Überblick Texterkennungssysteme . . . . .	4
2.2 Vorverarbeitung . . . . .	5
2.2.1 Eigenschaften von Screenshots . . . . .	5
2.2.2 Optimieren von Daten für Tesseract . . . . .	5
2.3 Nachbearbeitung . . . . .	6
2.3.1 Schlagworte . . . . .	6
2.3.2 Mehrsprachigkeit . . . . .	6
2.4 Metriken . . . . .	7
2.4.1 Word Error Rate . . . . .	7
2.4.2 Character Error Rate . . . . .	8
<b>3 Konzept</b>	<b>10</b>
3.1 Texterkennungssystem . . . . .	10
3.2 Bildbearbeitungswerkzeug . . . . .	10
3.3 Algorithmen zur Vorverarbeitung . . . . .	10
3.3.1 Resampling . . . . .	11
3.3.2 Rahmen . . . . .	12
3.3.3 Binarisierung . . . . .	12
3.3.4 Feste Schwellenwertmethode . . . . .	12
3.3.5 Adaptive Schwellenwertmethode . . . . .	14
3.3.6 Dreiecks-Schwellenwertmethode . . . . .	14
3.3.7 Schwellenwertmethode nach Otsu . . . . .	14
3.3.8 Schwellenwertmethode nach Kapur . . . . .	16
3.4 Algorithmen zur Nachbearbeitung . . . . .	17
3.4.1 Filterung anhand der Genauigkeit . . . . .	17

3.4.2	Normalisierung . . . . .	18
3.4.3	Entfernung von Duplikaten . . . . .	19
3.4.4	Filterung anhand der Wortlänge . . . . .	20
3.4.5	Sprachabhängige Filterung mittels Regular Expressions . . . . .	21
3.5	Testaufbau . . . . .	21
<b>4</b>	<b>Prototypische Implementierung</b>	<b>22</b>
4.1	Komponenten und Bibliotheken . . . . .	22
4.1.1	Fremdbibliotheken . . . . .	22
4.1.2	Verarbeitungsketten . . . . .	23
4.1.3	Lookup . . . . .	26
4.1.4	Optische Texterkennung . . . . .	26
4.1.5	Automatische Berichterstellung . . . . .	27
4.2	Vergleichsdaten . . . . .	27
4.3	Programmablauf . . . . .	28
4.3.1	Texterkennung . . . . .	28
4.3.2	Vergleich mit Soll-Daten . . . . .	30
<b>5</b>	<b>Evaluierung</b>	<b>32</b>
5.1	Verarbeitungsstatistik . . . . .	32
5.2	Ergebnisse . . . . .	32
5.3	Prozessoren im Überblick . . . . .	32
<b>6</b>	<b>Zusammenfassung</b>	<b>38</b>
	<b>Quellenverzeichnis</b>	<b>42</b>
	Literatur . . . . .	42
	Medien . . . . .	45

# Kapitel 1

## Einleitung

### 1.1 Motivation

Die in Salzburg ansässige COPA-DATA GmbH bietet die Softwareplattform zenon an, die als umfassende Gesamtlösung Unternehmen in zahlreichen Anwendungsgebieten bei der Automatisierung ihrer Herstellungsprozesse unterstützt [35].

Die zenon-Plattform kann sowohl vom Kunden selbst, als auch durch das Professional Services Team individuell auf Kundenanforderungen zugeschnitten und in bestehende Prozesse eingebunden werden. Den Grundstein für die hohe Anpassungsfähigkeit bildet die Produktdokumentation, in der Schnittstellendokumentation, Anleitungen und Beispiele in verschiedensten Sprachen, Formaten und mit kundenspezifischen Erweiterungen umfassend sowohl für Mitarbeiter, als auch für Kunden festgehalten sind.

In der Produktdokumentation werden, besonders in Hinblick auf die grafischen Werkzeuge wie die zenon Engineering Studio Entwicklungsumgebung oder die zenon Service Engine, zahlreiche Grafiken wie Abbildung 1.1 verwendet, um Beispiele verständlicher zu machen und Anleitungen übersichtlicher zu gestalten. Um bei dem großen Funktionsumfang der zenon-Produktpalette, den vielen Sprachen, Anpassungen und den unterschiedlichen Themengebieten innerhalb der Dokumentation den Überblick zu behalten, benötigt das interne „Technical Content and Translation“-Team unterstützend zu dem intern verwendeten Inhaltsverwaltungssystem eine dedizierte Anwendung zur Verwaltung von sprachabhängigen Bilddateien.

Während das Programm auch die Basisfunktionalität, das Speichern, Bearbeiten, Löschen, Abrufen beziehungsweise das generelle Verwalten von Screenshots und der zugehörigen Metainformation abdecken soll, konzentriert sich diese Bachelorarbeit primär auf die Kategorisierungsfunktionalität der Bildinhalte.

Mithilfe von optischer Texterkennung (engl. *Optical Character Recognition*, kurz: *OCR*) sollen Mitarbeiter hochgeladene Screenshots und Grafiken aufgrund ihrer Inhalte verschlagworten können, um sie später anhand dieser zu finden.

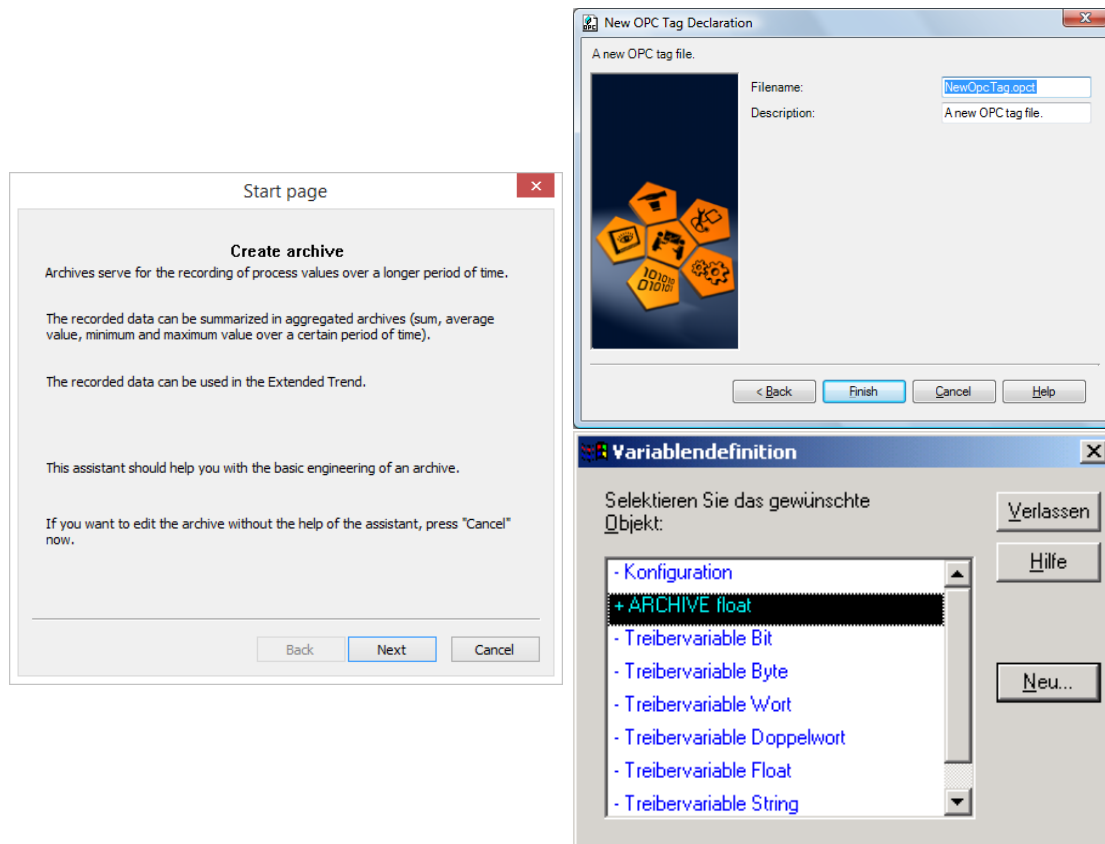


Abbildung 1.1: Beispielhafte Auswahl typischer Dialogscreenshots.

## 1.2 Zielsetzung

Das Ziel dieser Bachelorarbeit ist das Ermitteln einer Vorgehensweise für Texterkennung in Screenshots von grafischen Oberflächen. Verschiedene Algorithmen zur Bildbearbeitung vor der Texterkennung sowie Nachbearbeitung bzw. Filterung der Ergebnisdaten sollen evaluiert werden. Die Ergebnisdaten sollen anschließend anhand von festgelegten Qualitätskriterien aus der programmatischen Sprach- und Schrifterkennung, beispielsweise der Zeichenfehler- oder Wortfehlerrate, analysiert und miteinander verglichen [12] werden.

Eine prototypische Implementierung dient als Basis für die nachfolgenden Tests und Analysen, anhand derer die Algorithmen automatisch evaluiert werden sollen. Alle entwickelten Komponenten sollen als Bibliotheken zur Verfügung gestellt werden, um die Texterkennung inklusive automatischer Bildverarbeitung und Filterung der erkannten Inhalte bzw. Schlagworte später in anderen Anwendungen weiterverwenden zu können.



### 1.3 Aufbau

In Kapitel 2 wird der technische Grundstein für die Bachelorarbeit gelegt. Informationen zur Vorgehensweise und zu den wichtigsten Punkten bei der Vorverarbeitung und Nachbearbeitung werden erarbeitet und dienen als Basis für die spätere Entscheidungsfindung in Kapitel 3. Weiters werden die Metriken erklärt, die für den Vergleich der Qualität der Texterkennungsergebnisse notwendig sind.

Sind die Rahmenbedingungen geklärt, ist gemäß Kapitel 3 eine begründete Auswahl an verwendeten Algorithmen und Technologien zu treffen. Verschiedene Algorithmen werden verglichen und auf ihre Stärken und Schwächen in Bezug auf Screenshotdaten untersucht. Die geeigneten Algorithmen werden anschließend in Kapitel 4 für den Vergleich verwendet.

Basierend auf dem Konzept wird in der prototypischen Implementierung ein Programm geschaffen, welches verschiedene Algorithmen anhand einer Stichprobe, bestehend aus beispielhaft ausgewählten Screenshots, vergleicht. Mittels der flexiblen Komponenten soll eine spätere Anpassung des Programmablaufs bzw. die Implementierung weiterer Algorithmen einfach sein, was schnelles Prototyping und Testen fördern soll. Das Ergebnis ist ein automatisch generierter Bericht, anhand dessen die Ergebnisdaten untersucht werden können.

Schlussendlich müssen die in Implementierung generierten Daten evaluiert werden. Dazu werden die jeweiligen Sektionen des Berichts in Kapitel 5 einzeln betrachtet und die Ergebnisse verglichen.

## Kapitel 2

# Optische Texterkennung

Für einen technischen Überblick und als Grundlage für die Auswahl der innerhalb dieser Bachelorarbeit verwendeten Algorithmen wird im folgenden Kapitel näher auf Texterkennungssysteme, Grundlagen zur Vorverarbeitung, Nachbearbeitung sowie die verwendeten Metriken zum Vergleich eingegangen.

### 2.1 Überblick Texterkennungssysteme

Optische Texterkennung wird in der Informationstechnik eingesetzt, um Textinhalte aus gedruckten oder digital rasterisierten Medien zu extrahieren. Dieses Verfahren kann für diverse Anwendungsgebiete genutzt werden, wie beispielsweise für Handschrifterkennung [19] oder für das Ablesen von Nummernschildern eines Autos [1, 34]. Auf dem Markt gibt es dafür kommerzielle Komplettlösungen wie „Anyline“ [34], „IronOCR“ [39], „Google Cloud Vision“ [36], „Amazon Textract“ [30] oder „Microsoft Azure Computer Vision“ [32], die oftmals gute Ergebnisse mit geringen Fehlerraten erzielen und sich in bestehende Prozesse oder Anwendungen integrieren lassen [8, 25].

Ein Beispiel für Open-Source-Software zur Texterkennung ist die „Tesseract Open Source OCR Engine“ (kurz: Tesseract). Das zugehörige Repository hat verfügt neben über 56000 Sternen auf GitHub auch über eine aktive Community, die das Projekt ständig weiterentwickelt [43]. Tesseract ist seit 2005 unter der Freie-Software-Lizenz „Apache 2.0“ lizenziert [22] und basiert seit der Major-Version 4 auf einem neuronalen Netz, durch welches mithilfe von sprachspezifischen Trainingsdaten Texte in Bildern erkannt werden können [42].

Aktuelle Texterkennungssysteme arbeiten oft mit einer Kombination aus neuronalen Netzwerken und fortgeschrittenen Bildverarbeitungsalgorithmen, um Texte zu erkennen. Zahlreiche wissenschaftliche Werke wie beispielsweise [6] oder [9] erklären die grundlegende Funktionsweise von optischen Texterkennungswerkzeugen. Die genauen Schritte zur richtigen Vorbereitung der Bilddaten – besonders in Bezug auf Screenshots – werden jedoch oftmals nur oberflächlich behandelt.

## 2.2 Vorverarbeitung

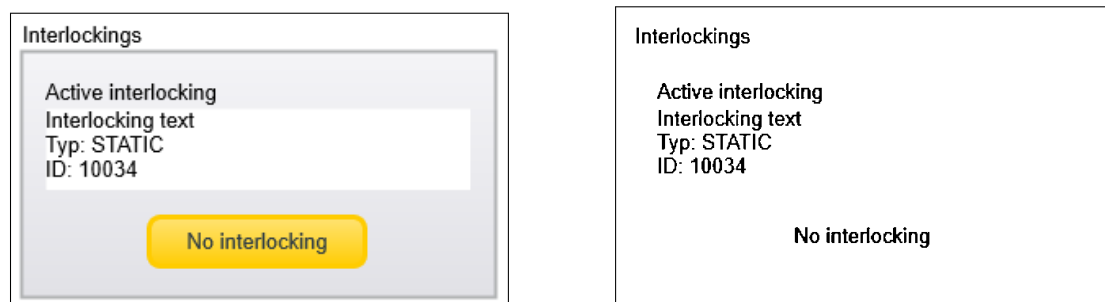
Im Folgenden wird beschrieben, wie die zu verarbeitenden Bilddaten für optimale OCR-Ergebnisse vorzubereiten sind.

### 2.2.1 Eigenschaften von Screenshots

Die zu verarbeitenden Bilder im Kontext dieser Bachelorarbeit sind ausschließlich digitale Bildschirmaufnahmen von grafischen Benutzeroberflächen. Es kann daher angenommen werden, dass die Screenshots keine Transparenz aufweisen, die Perspektive der Aufnahme nicht verzerrt ist und im Normalfall nicht mit Bildrauschen zu rechnen ist, weshalb Rauschfilterung [13] nicht notwendig ist. Auch wird angenommen, dass der Kontrast in den meisten Fällen ausreicht, die relevanten Inhalte zu erkennen. Weiters ist bei der Bildverarbeitung auf farbige Hintergrundflächen zu achten, mit deren Unterstützung Bildelemente oft markiert, gruppiert oder getrennt werden. Nach Sichtung des zu verarbeitenden Bilddatensatzes fällt auf, dass manche Screenshots durch das Selektieren mit der Maus sehr eng abgeschnitten wurden. Das ist bei der Datenverarbeitung ebenfalls zu berücksichtigen. Eine beispielhafte Auswahl typischer Screenshots findet sich in Abbildung 1.1.

### 2.2.2 Optimieren von Daten für Tesseract

Für die Verwendung von Tesseract ist es wichtig, unabhängig von der Diversität der Ausgangsdaten möglichst einheitliche Bilder zu erzeugen [42]. Während störende Elemente wie Bildrauschen aus dem Bild entfernt werden, sollen Texte ohne Einfluss der eigentlichen Hinter- bzw. Vordergrundfarbe gut zu erkennen sein. Auch eine deutliche Abgrenzung von Formen oder grafischen Symbolen ist von großer Wichtigkeit [23] [16]. Wie in Abbildung 2.1 gezeigt, sollen farbige Hintergrundflächen und grafische Dekorationselemente verschwinden, während nur der gut lesbare textuelle Inhalt des Bildes übrig bleiben soll.



**Abbildung 2.1:** Ein optimales Ergebnisbild. Jegliche farblichen Flächen wurden durch die Bildverarbeitung entfernt. Übrig bleibt klar lesbarer Text mit einem hohen Kontrast zum Hintergrund.

## 2.3 Nachbearbeitung

Auch die Nachbearbeitung der erkannten Textdaten spielt eine wesentliche Rolle. Hier werden Konzepte aus dem Themengebiet des Natural Language Processings angewandt, welches sich mit der Interaktion zwischen menschlicher Sprache und Computern beschäftigt. Durch die Kombination von Techniken aus der Informatik, Linguistik und dem maschinellen Lernen werden beispielsweise Textanalyse, Übersetzungen, Spracherkennung oder Dialogsysteme möglich [4]. Durch die große Aufmerksamkeit und die vielseitige Nutzung der Technologien sowie dem Aufkommen von neuronalen Netzwerken wurden in diesem Forschungsgebiet immer wieder Fortschritte erzielt [5, 10]

Dadurch gibt es zahlreiche wissenschaftliche Ressourcen, welche als Grundlage für die Vorgehensweise zur Interpretation und Extraktion relevanter Schlagworte aus den erkannten Freitextdaten dienen.

### 2.3.1 Schlagworte

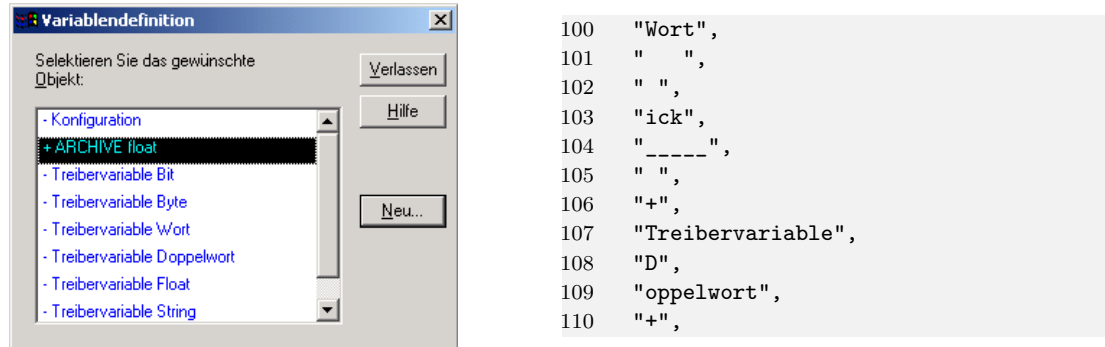
Für die spätere Suche von Screenshots sollen relevante Schlagworte aus den erkannten Textdaten extrahiert werden. Ein Wort eignet sich dann als Schlagwort, wenn es in relevantem Bezug zum jeweiligen Bild steht und dabei eine wichtige Aktion oder Information widerspiegelt. Inhalte, die direkt in der grafischen Benutzeroberfläche ersichtlich sind, haben daher einen hohen Informationsgehalt und eignen sich besonders gut als Suchworte. Um die Schlagwortmenge so aussagekräftig wie möglich zu halten, müssen Wörter mit geringer Bedeutung entfernt werden. Beispielsweise haben sogenannte Stoppwörter (engl. *Stop words*) wie *und* oder *oder* keine besondere Semantik und fördern aufgrund ihrer Häufigkeit das Auftreten von Verwechslungen [28].

### 2.3.2 Mehrsprachigkeit

Das Textverarbeitungssystem muss in der Lage sein, mehrsprachige Bilddateien einlesen und interpretieren zu können. So sollen beispielsweise Bilder mit englischen, deutschen oder italienischen Inhalten zugeführt und die Ergebnisdaten richtig verarbeitet werden können. Um eine Filterung für verschiedene Zeichensätze zu ermöglichen und eine Unterstützung für Sprachen mit nicht-lateinischen Schriften zu gewährleisten, werden dynamische Sprachfilter verwendet, die individuell an die jeweilige Sprache angepasst werden können. Für die weitere Beschreibung der generellen Vorgehensweise und Tests wird sich in den folgenden Schritten jedoch auf deutsche und englische Inhalte beschränkt.

#### 2.3.2.1 Filtern von Symbolen

Bei der Texterkennung kann es vorkommen, dass grafische Elemente als diverse Unicode-Symbole erkannt werden. Wie in Abbildung 2.2 versuchen Texterkennungssysteme oftmals, grafische Dekorationselemente textuell nachzubilden. Auch finden sich in den ungefilterten Ergebnisdaten Aufzählungszeichen wie „•“ oder unterschiedliche Varianten von Bindestrichen „–“. Diese Zeichen sind gemäß Anwendungsanforderungen nicht relevant für die Schlagwortsuche und können somit entfernt bzw. ignoriert werden.



**Abbildung 2.2:** Auszug aus den ungefilterten Ergebnisdaten bei Durchführung der Texterkennung in dem gezeigten Screenshot.

## 2.4 Metriken

Um die erkannten Ergebnisse unter Verwendung der verschiedenen Pre- und Postprocessing Schritte mittels eines einheitlichen Systems vergleichen zu können, wird auf die in der optischen Texterkennung gängigen Metriken „Character Metric“, auch bekannt als „Character Error Rate“ und „Word metric“ bzw. „Word Error Rate“ [12], basierend auf der Levenshtein-Distanz [14] zurückgegriffen.

Sowohl die Character- als auch die Word Error Rate sind häufig genutzte Vergleichswerte, die ihren Ursprung in der computergestützten Sprachverarbeitung bzw. automatischen Spracherkennung haben [27]. Da die optische Texterkennung und die automatische Spracherkennung jeweils darauf abzielen, maschinenlesbaren Text aus nicht-strukturierten Daten zu extrahieren, sind die Prinzipien dieser Metriken auch auf die optische Texterkennung anwendbar [24].

### 2.4.1 Word Error Rate

Die Wortfehlerrate (engl. *Word Error Rate*, kurz: *WER*) beschreibt den prozentualen Anteil der falsch erkannten oder fehlenden Wörter eines Textes im Vergleich zu einer Referenz, welche im Falle der folgenden Vergleiche immer alle sichtbaren Texte im Bild repräsentiert. Je niedriger die WER, desto genauer ist der OCR-Vorgang. Um die WER zu berechnen, bildet man die Summe aller notwendigen Ersetzungen, Entfernungen und Einfügungen, um aus dem erkannten Text den Referenztext bilden zu können und setzt sie mit der Gesamtwortanzahl im Referenztext in Verhältnis. [12, 14, 18]

#### 2.4.1.1 Berechnung

Die mathematische Formel für die Word Error Rate lautet wie folgt [12]:

$$\text{WER} = \frac{S + D + I}{N}$$

wobei die einzelnen Komponenten folgende Größen darstellen:

- $S$  beschreibt die Anzahl der falsch erkannten Wörter (engl. *Substitutions*)
- $D$  beschreibt die Anzahl der im Resultat fehlenden Wörter (engl. *Deletions*)
- $I$  beschreibt die Anzahl der im Resultat fälschlicherweise eingefügte Wörter (engl. *Insertions*)
- $N$  beschreibt die Gesamtanzahl der Wörter in der Referenz

#### 2.4.1.2 Vorteile und Nachteile

Die WER spiegelt ohne großen Rechenaufwand direkt wider, wie stark die erkannten Texte der Referenz gleichen. Hierbei werden fehlerhafte Einsetzungen, Löschungen und falsch erkannte Wörter bzw. Teilwörter gleichermaßen gewichtet. Es ist jedoch nicht möglich, die korrekte Reihenfolge der erkannten Wörter darzustellen oder bestimmte wichtige Stellen im Text höher zu gewichten als andere. Zudem werden fehlerhaft erkannte Wörter als vollwertige Ersetzung wahrgenommen, auch wenn nur ein einzelnes Zeichen falsch ist. Dadurch wird das Ergebnis stark beeinflusst.

Um die Verfälschung der Ergebniswerte durch die WER möglichst gering zu halten, muss mindestens eine weitere Fehlermetrik, beispielsweise die Character Error Rate, zum Vergleich verwendet werden.

#### 2.4.2 Character Error Rate

Die Zeichenfehlerrate (engl. *Character Error Rate*, kurz: *CER*) beschreibt die Anzahl der falsch erkannten oder fehlenden Zeichen im Vergleich zu einem Referenzwort und basiert wie die Word Error Rate auf der Levenshtein-Distanz [14]. Je niedriger die CER, desto genauer ist der OCR-Vorgang. Ähnlich wie die WER wird die CER aus der Summe aller Ersetzungen, Entfernungen und Einfügungen, notwendig um aus dem erkannten Wort die Referenz bilden zu können, gebildet [14]. Diese Summe wird anschließend durch die Zeichenanzahl des Referenzwortes geteilt [12, 18].

##### 2.4.2.1 Berechnung

Das Verfahren zur Ermittlung der CER gleicht im Wesentlichen dem der WER, bezieht sich allerdings auf die einzelnen Zeichen eines Wortes. Die mathematische Formel lautet wie folgt [12]:

$$\text{CER} = \frac{S + D + I}{N}$$

wobei die einzelnen Komponenten folgende Größen darstellen:

- $S$  beschreibt die Anzahl der falsch erkannten Zeichen (Substitutions)
- $D$  beschreibt die Anzahl der im Resultat fehlenden Zeichen (Deletions)

- $I$  beschreibt die Anzahl der im Resultat fälschlicherweise eingefügte Zeichen (Insertions)
- $N$  beschreibt die Gesamtanzahl der Zeichen in der Referenz

#### 2.4.2.2 Vorteile und Nachteile

Die CER fasst in einem Wert zusammen, wie viele Änderungen auf Zeichenebene notwendig sind, um aus dem erkannten Wort das Referenzwort zu bilden. Es ist dabei wie bei der WER nicht relevant, in welcher Reihenfolge diese Zeichen auftreten. Ebenso gibt es keine gesonderte Gewichtung für Ersetzungen, Löschungen oder Einfügungen, wodurch besonders bei kurzen Wörtern auch kleinere Abweichungen bereits zu einer hohen CER führen können.

Durch den detaillierten Vergleich der einzelnen Wörter auf Zeichenebene stellt die CER ein ausreichend gutes Komplement zur WER dar und wird in den folgenden Vergleichen ebenfalls verwendet werden.

## Kapitel 3

# Konzept

Im folgenden Kapitel wird eine Auswahl an verwendeten Technologien und Algorithmen getroffen. Auch wird der Grundstein für den Testaufbau gelegt, anhand dessen später die prototypische Implementierung erfolgt.

### 3.1 Texterkennungssystem

Wie bereits in Abschnitt 2.1 beschrieben, ist die Nutzung vieler OCR-Anwendungen bzw. Dienstleistungen kostenpflichtig und die genaue innere Vorgehensweise dieser Programme ist nicht öffentlich bekannt [31, 33, 37]. Aufgrund dieser Tatsachen wird als Texterkennungssystem für die prototypische Implementierung dieser Bachelorarbeit die quelloffene und kostenlose Tesseract Open Source OCR Engine verwendet.

### 3.2 Bildbearbeitungswerkzeug

Als Werkzeug für die Durchführung der notwendigen Bildbearbeitungsschritte wurde aufgrund persönlicher Erfahrungen die Softwarebibliothek „ImageMagick“ [38] gewählt. Sie ist umfassend dokumentiert, flexibel und kann aufgrund der Unterstützung für eine Vielzahl von Programmiersprachen direkt in Programme eingebunden werden. Viele für die Bildverarbeitung notwendige Operationen sind bereits implementiert, was schnelles Prototyping vereinfacht. Deshalb wird die Bibliothek für die Realisierung von Bildbearbeitungsschritten in der prototypischen Implementierung verwendet, kann jedoch auch durch eine andere Implementierung ersetzt werden.

### 3.3 Algorithmen zur Vorverarbeitung

Wie bereits in Abschnitt 2.2 beschrieben, werden die zu verarbeitenden Bilder bei der Vorverarbeitung (engl. *Preprocessing*) für die Texterkennung vorbereitet, um die Qualität der erkannten Textdaten zu verbessern.

In einigen Tesseract-Implementierungen sind bereits rudimentäre Bildverarbeitungswerkzeuge verfügbar [42]. Mit diesen Werkzeugen werden die eingespeisten Bilder –



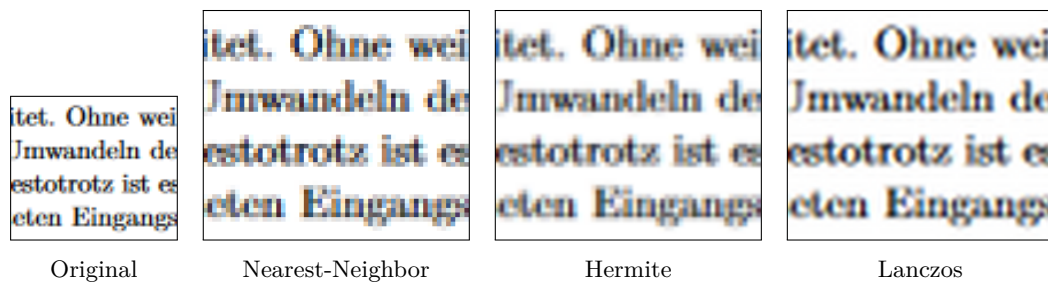
sofern nicht bereits im richtigen Format vorhanden – automatisch für die Texterkennung vorbereitet. Ohne weitere Einstellungen zu treffen, bewirkt diese Bildverarbeitung ein Umwandeln der Eingangsgrafiken in ein meist gut für Tesseract geeignetes Bild. Es ist jedoch zu beachten, dass die Bildverarbeitungsschritte individuell auf die erwarteten Eingangsdaten anzupassen sind. So können die Bilddaten den in Unterabschnitt 2.2.2 definierten optimalen Tesseract-Eingangsdaten angenähert werden.

Die folgenden Vorverarbeitungsschritte basieren auf der empfohlenen Vorgehensweise zur Verbesserung der Output-Qualität laut Tesseract-Dokumentation [42]. Gemäß den Annahmen in Kapitel 2 werden jedoch weder perspektivische Fehler, noch ein eventuelles Rauschen korrigiert.

### 3.3.1 Resampling

Bei Resampling wird die Bildauflösung durch Interpolation verändert. Interpolation beschreibt die Methode, fehlende Pixelwerte zwischen bekannten Punkten mittels eines festgelegten Verfahrens zu ergänzen. Abhängig vom gewählten Verfahren ist das Ergebnis meist ein glattes und kontinuierliches Bild. Um die für Tesseract optimale Mindestauflösung von 300 dpi [42] zu gewährleisten, muss das Eingangsbild, sofern es die Mindestauflösung unterschreitet, zunächst entsprechend vergrößert werden.

Da Tesseract auf klare und scharfe Kontraste angewiesen ist, um Text korrekt zu identifizieren, eignen sich nicht alle von ImageMagick zur Verfügung gestellten Skalierungsmethoden für die Weiterverarbeitung. Wie in Abbildung 3.1 zu sehen ist, neigen einige Filter besonders beim Hochskalieren dazu, Unschärfen oder Artefakte zu erzeugen, die die Genauigkeit der Texterkennung negativ beeinflussen können.

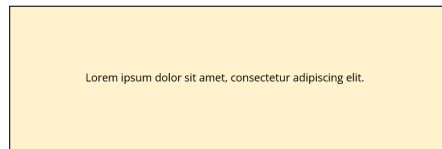


**Abbildung 3.1:** Ein Vergleich unterschiedlicher Resampling-Filter. Durch die Aufteilung der Fehler auf mehrere Pixel bleiben Details und Konturen bei Anwendung des Lanczos-Filters vergleichsweise gut erhalten und der Text ist gut lesbar.

Nach einigen Tests fällt auf, dass Bilder, die mittels des Spline-Verfahrens [2, 26] oder der Hermite-Interpolation [21] skaliert wurden, weiche Konturen ohne harte Farbübergänge aufweisen. Tesseract profitiert jedoch stark von klaren Texten und hohen Kontrasten, weswegen diese Art des Resamplings keine ideale Basis für das Preprocessing bietet. Deswegen wird für die weiteren Schritte die Interpolation nach Lanczos [7] für das Resampling verwendet.

### 3.3.2 Rahmen

Befindet sich Text zu nah am Rand des Bildes, wird dieser nicht immer richtig erkannt. Ebenso kann auch ein zu großer einfärbiger Rahmen am Rand des Bildes die Texterkennung stören. Bei Rahmengrößen wie in Abbildung 3.2 kommt es vor, dass Bildsektionen fälschlicherweise als „leer“ erkannt und übersprungen werden, wodurch der zu erkennende Text nicht in die Ergebnisdaten mit aufgenommen wird.



**Abbildung 3.2:** Ein im Verhältnis zur Bildgröße zu großer einfärbiger Rahmen

Für die weitere Vorgehensweise ist es sinnvoll, alle Bilder mit einem schmalen Rahmen zu umgeben. So wird sichergestellt, dass sich die Texte nicht zu nah am Bildrand befinden und Fehler in der Texterkennung werden reduziert.

### 3.3.3 Binarisierung

Das Erzeugen eines Binärbildes ist durch Anwendung von Segmentierungsverfahren möglich. Schwellenwertverfahren (engl. *Thresholding*) bilden eine Untergruppe der Segmentierungsverfahren und werden genutzt, um Graustufenbilder Pixel für Pixel in binarisierte Ergebnisbilder mit zwei Segmenten, also einem Vordergrund und einem Hintergrund umzuwandeln. Der dazu notwendige Schwellenwert kann entweder fest definiert oder anhand von verschiedensten Algorithmen automatisch bzw. halbautomatisch ermittelt werden. Ziel ist es, durch die Binarisierung textuelle Bildinhalte unabhängig von der eigentlichen Vorder- und Hintergrundfarbe mit ausreichendem Kontrast darzustellen. Dadurch ist das Texterkennungssystem in der Lage, die einzelnen Textelemente und deren Inhalte besser zu identifizieren und zu verarbeiten. ImageMagick bietet eine Vielzahl an Schwellenwert-Algorithmen, von denen nachfolgend jene genauer beschrieben werden:

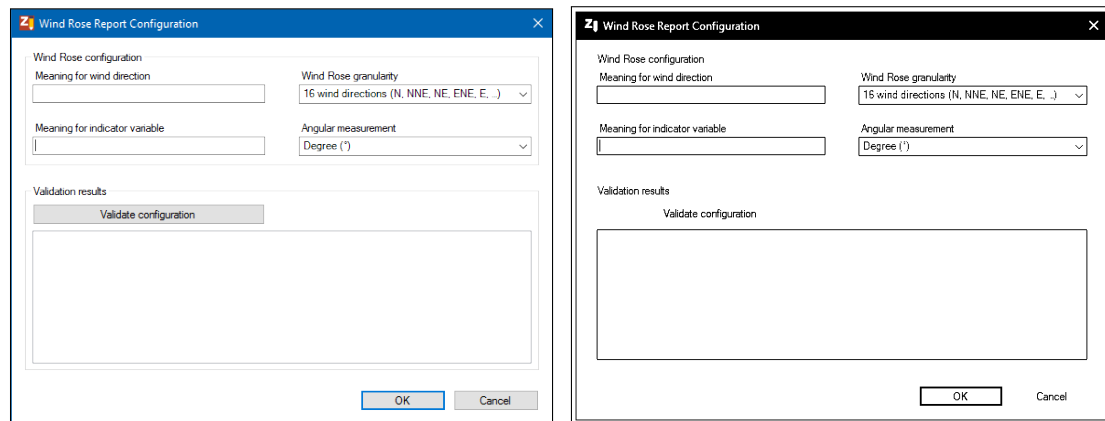
- Feste Schwellenwertmethode
- Adaptive Schwellenwertmethode
- Dreiecks Schwellenwertmethode
- Schwellenwertmethode nach Otsu
- Schwellenwertmethode nach Kapur

### 3.3.4 Feste Schwellenwertmethode

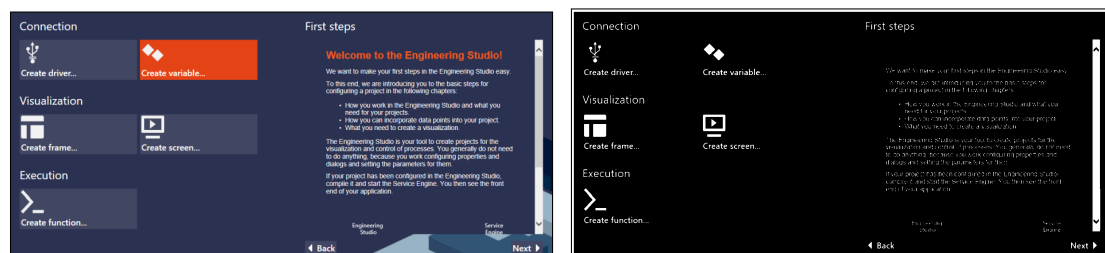
Ein häufig für die Bildsegmentierung genutztes Verfahren ist die feste Schwellenwertmethode, im Englischen auch „Fixed Thresholding“ genannt. Bei diesem Bildverarbeitungsverfahren wird ein manuell vordefinierter Grenzwert auf das gesamte Bild angewandt. Liegt ein Pixelwert über dem festgelegten Schwellenwert, gilt dieser als Teil des Vor-

dergrunds, andernfalls als Hintergrund [20]. Somit können Objekte, also die einzelnen Buchstaben in den Grafikdateien, von ihrem Hintergrund getrennt werden.

Durch den globalen, fixen Schwellenwert ohne Berücksichtigung von Pixelnachbarschaften weist das fixe Schwellenwertverfahren einen geringen Berechnungsaufwand und daher eine hohe Performance auf [20]. Besonders bei Screenshotdateien kann es vorkommen, dass die eigentlich bunten grafischen Elemente der Benutzeroberfläche aufgrund ihrer Helligkeit über dem Schwellenwert liegen. Dadurch werden sie, genau wie der Text, als Vordergrund wahrgenommen und die gesamte Fläche wird einfarbig. Somit können jegliche Texte innerhalb dieser Fläche nicht vom Texterkennungssystem erkannt werden und die Qualität und Menge der erkannten Daten wird stark reduziert. Der Unterschied der Ergebnisdaten ist besonders im Vergleich von Abbildung 3.3 und Abbildung 3.4 ersichtlich.



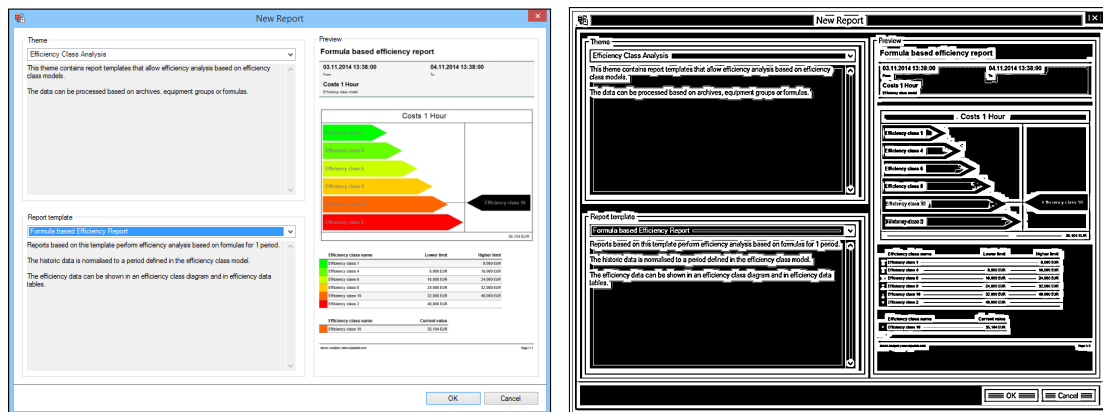
**Abbildung 3.3:** Anwendung des festen Schwellenwertverfahrens auf einen Beispielscreenshot. Bei einem passenden Schwellenwert und nur geringfügig verschiedenen Farbflächen ist der Textinhalt gut vom Hintergrund abgrenzbar. Der Schwellenwert im gezeigten Bild beträgt 60 %.



**Abbildung 3.4:** Anwendung des festen Schwellenwertverfahrens auf einen Beispielscreenshot. Bei einem falsch gewählten Schwellenwert oder komplexen UI-Elementstrukturen ist der Text nicht erkennbar. Der Schwellenwert im gezeigten Bild beträgt 80 %.

### 3.3.5 Adaptive Schwellenwertmethode

Die adaptive Schwellenwertmethode gehört zu den halbautomatischen Schwellenwertalgorithmen. Bei diesem Verfahren wird der Schwellenwert auf Basis der lokalen Eigenschaften eines Bildbereichs angepasst, der durch die manuell festgelegte sogenannte „Blockgröße“ definiert wird. Diese bestimmt die Seitenlänge des Rechtecks, innerhalb dessen ein fester Schwellenwert ermittelt wird. Durch diese dynamische Berechnung können im Gegensatz zur festen Schwellenwertmethode verschiedenfarbige Texte auf Hintergründen unterschiedlicher Helligkeit besser abgegrenzt werden und die Menge an erkanntem Text wird erhöht, wie in Abbildung 3.5 ersichtlich. Wird die Blockgröße falsch gewählt, können jedoch Artefakte auftreten, welche bei entsprechender Menge, wie im Falle von Abbildung 3.6, die Texterkennung negativ beeinflussen [20].



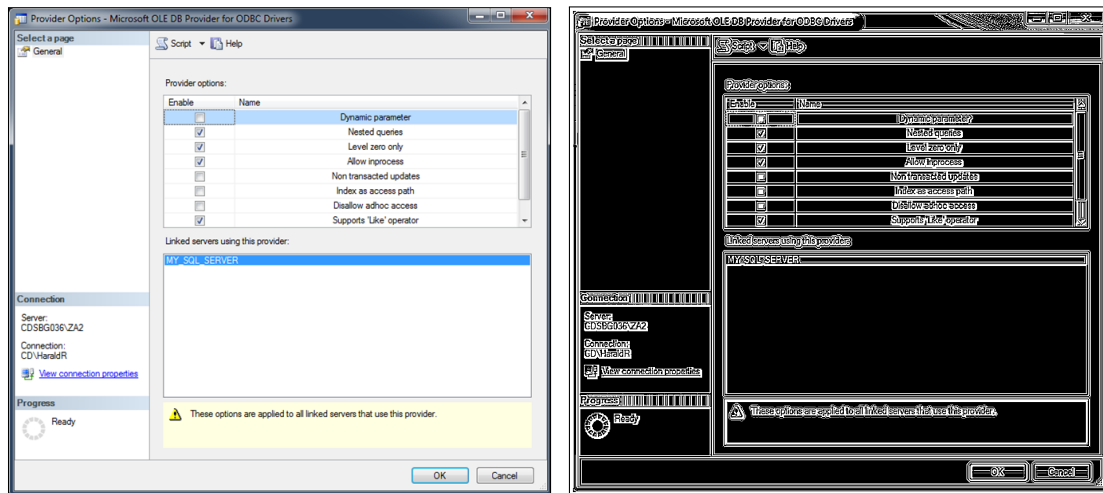
**Abbildung 3.5:** Anwendung der adaptiven Schwellenwertmethode auf einen Beispielscreenshot. Die Blockgröße ist gut an den Bildinhalt angepasst und alle Details bleiben erhalten. Dieses Verfahren punktet hier besonders bei den farbigen „Energy Labels“, deren Textinhalte sonst mittels keinem anderen Verfahren komplett erkannt wurden.

### 3.3.6 Dreiecks-Schwellenwertmethode

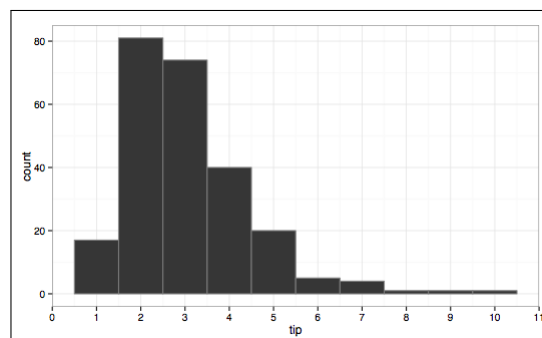
Das Dreiecks-Schwellenwertverfahren verwendet die Häufigkeitsverteilung der Helligkeitswerte eines Bildes, um einen globalen Schwellenwert zu ermitteln [29]. Werden diese Helligkeitswerte in einem Diagramm dargestellt, spricht man von einem Histogramm, wie in Abbildung 3.7 zu sehen. Für dieses Schwellenwertverfahren wird innerhalb des Histogramms eine Linie vom Höchstwert (engl. *Peak*) zum Minimum gezeichnet und die Normale mit der maximalen Länge ermittelt. Dieses Verfahren erzielt die besten Ergebnisse, wenn die zu extrahierenden Elemente Intensitätswerte aufweisen, die an der Basis des ermittelten Peaks liegen. Für Screenshots von UI-Elementen mit komplexer Struktur und farblich stark variierenden Komponenten ist es eher nicht geeignet.

### 3.3.7 Schwellenwertmethode nach Otsu

Das Schwellenwertverfahren nach Otsu ermittelt einen globalen Schwellenwert durch Einteilung des Bildes in zwei Klassen (Vordergrund und Hintergrund). Dazu wird für



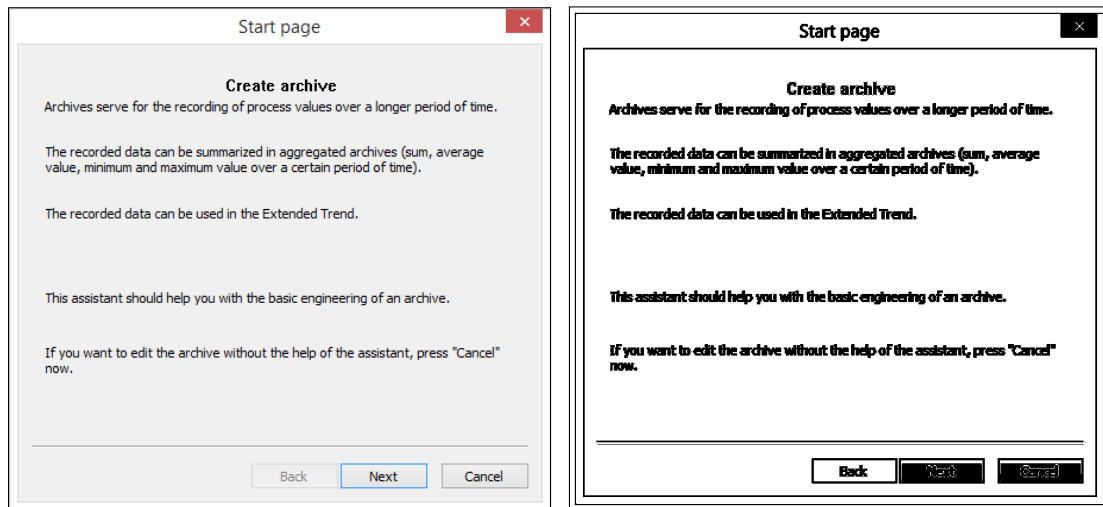
**Abbildung 3.6:** Anwendung der adaptiven Schwellenwertmethode auf einen Beispielscreenshot. Die bei unangepasster Blockgröße entstehenden Artefakte schränken die Funktionsweise des Texterkennungssystems deutlich ein.



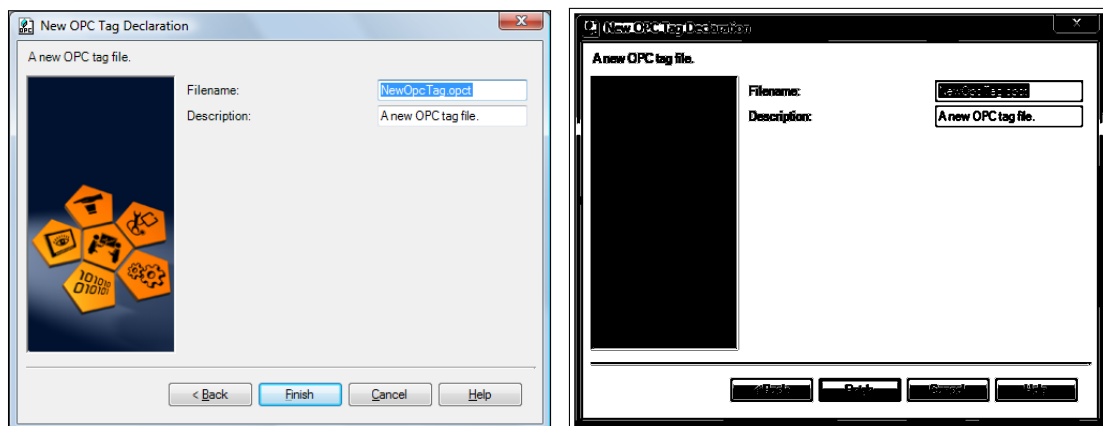
**Abbildung 3.7:** Beispiel eines unimodalen Histogramms mit einem deutlichen Spitzenwert [45]

jede Position des Schwellenwerts im Histogramm die Varianz der beiden dadurch entstehenden Klassen ermittelt. Der Schwellenwert ist dann optimal, wenn die Varianz der jeweiligen Klassen minimal ist [17]. Aufgrund dieser Eigenschaften funktioniert das Verfahren am besten, wenn das Histogramm des Bildes wie in Abbildung 3.10 eine bimodale Verteilung aufweist, also zwei klare Spitzen hat.

Enthält ein Bild jedoch starkes Hintergrundrauschen oder weist es lokale Helligkeitsunterschiede auf, wie es bei grafischen Oberflächen mit ihren unterschiedlich eingefärbten Oberflächensektionen oft der Fall ist, wird ein Schwellenwert ermittelt, der nicht immer für alle Inhalte des Bildes optimal ist. Dank der Bestimmung eines einzelnen globalen Wertes für das gesamte Bild entstehen ähnliche Probleme wie bei der fixen Schwellenwertmethode. Die Texterkennung liefert Ergebnisse mit geringem Informationsgehalt. Die Unterschiede zwischen den Ergebnissen für gut geeignete Bilder im Vergleich zu Bildern mit großen lokalen Unterschieden sind in Abbildung 3.11 bzw. Abbildung 3.12



**Abbildung 3.8:** Anwendung der Dreiecks-Schwellenwertmethode auf einen Beispielscreenshot. Gleichen sich die Inhalte farblich, werden einige Details extrahiert. Durch kleinste Farbvariationen im Bild weicht der Schwellenwert jedoch vom Optimum ab und die Texte sind nur schwer zu erkennen.



**Abbildung 3.9:** Anwendung der Dreiecks-Schwellenwertmethode auf einen Beispielscreenshot. Bereits bei mäßiger Variation der Helligkeit verschwinden die ersten Details.

ersichtlich.

### 3.3.8 Schwellenwertmethode nach Kapur

Die Schwellenwertmethode nach Kapur [11] zielt darauf ab, einen Schwellenwert zu finden, der die Entropie zwischen den Vorder- und Hintergrundregionen maximiert. Wie in Abbildung 3.13 und Abbildung 3.14 zu sehen, liefert die Verwendung dieses Schwellenwertverfahrens gute Ergebnisse bei Bildern mit starker Varianz der Vorder- und Hintergrundkontraste bzw. einer breiten Helligkeitsverteilung.

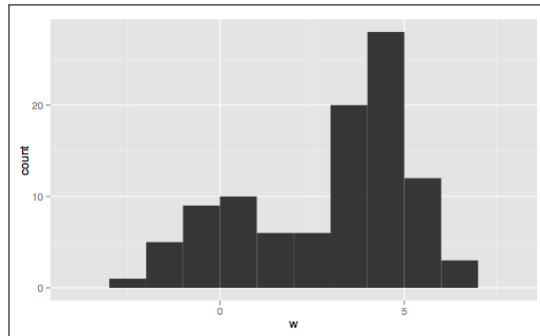


Abbildung 3.10: Beispiel eines bimodalen Histogramms [44]

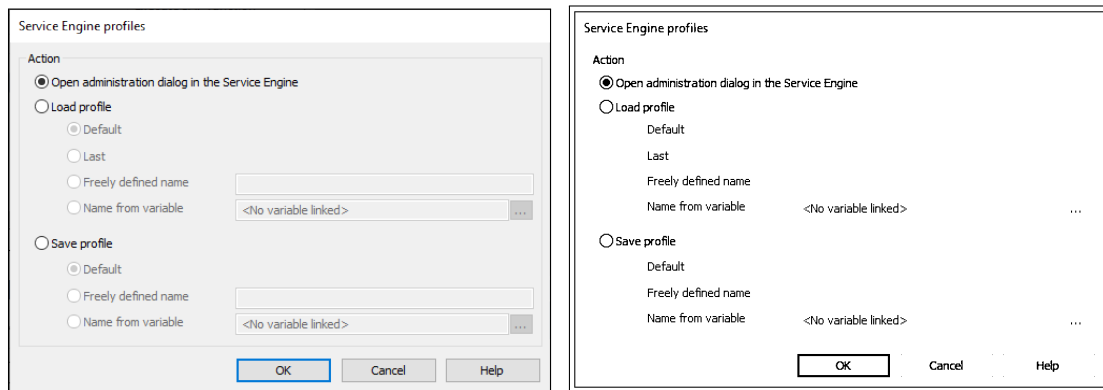


Abbildung 3.11: Anwendung der Schwellenwertmethode nach Otsu [17] auf einen Beispielscreenshot. Wird ein passender Schwellenwert ermittelt, lässt sich der Text gut vom Hintergrund trennen.

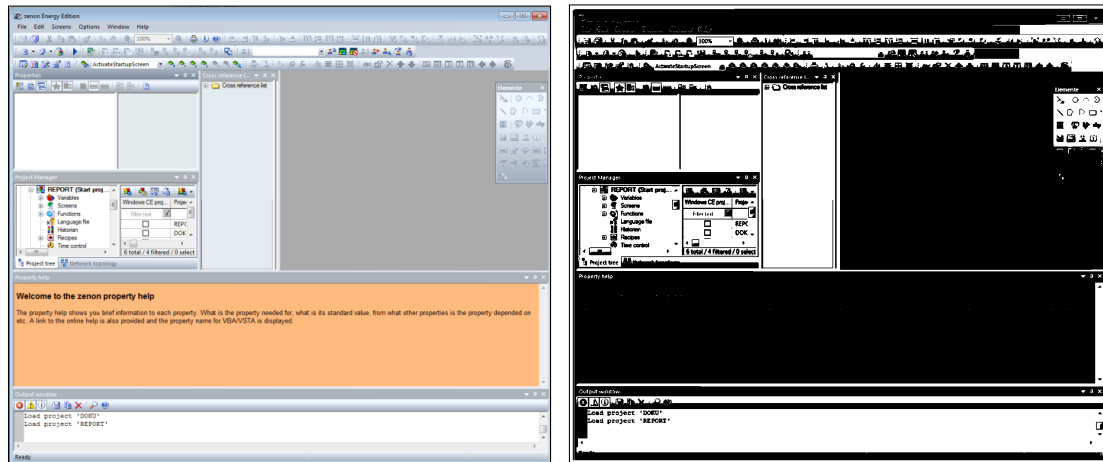
### 3.4 Algorithmen zur Nachbearbeitung

Die extrahierten Textdaten aus den verarbeiteten Bilddaten werden in einer schlagwortbasierten Suchfunktion verwendet. Um Redundanz innerhalb des Datensets zu reduzieren und falsch erkannte Ergebnisdaten zu verhindern, müssen die Ergebnisdaten der Texterkennung im Rahmen der Nachbearbeitung (engl. *Postprocessing*) gemäß Abschnitt 2.3 weiterverarbeitet werden.

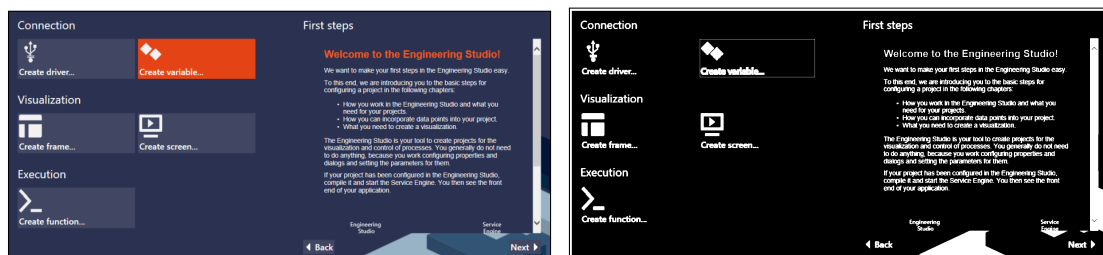
#### 3.4.1 Filterung anhand der Genauigkeit

Tesseract stellt im Rahmen der Texterkennung neben den erkannten Texten die jeweiligen Metadaten zur Verfügung. Auch die geschätzte Genauigkeit (engl. *Confidence*) wird für jedes erkannte Wort mit angegeben. Sie bestimmt, mit welcher Sicherheit ein Texterkennungssystem das jeweilige Wort erkannt hat, wobei Wörter mit hoher Confidence eher richtig, mit niedriger Confidence eher falsch erkannt wurden.

Wie in Abbildung 3.15 gezeigt, prüft der Confidence-Filter die jeweiligen Wörter auf ihre Metadaten und verwirft jene, deren Confidence unter einem fixen Schwellenwert liegt. Je



**Abbildung 3.12:** Anwendung der Schwellenwertmethode nach Otsu [17] auf einen Beispielscreenshot. Bei komplexen Strukturen im User-Interface gehen aufgrund des globalen Schwellenwerts Details verloren.



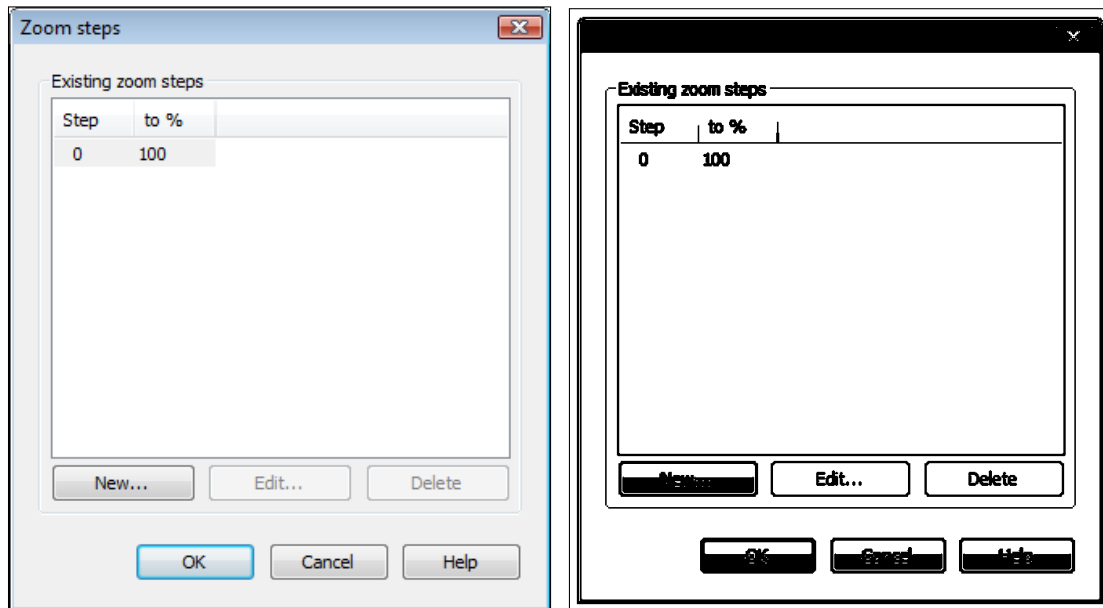
**Abbildung 3.13:** Anwendung der Schwellenwertmethode nach Kapur auf einen Beispielscreenshot. Trotz der vielen verwendeten Farben wird der Inhalt gut dargestellt.

nach Einstellung bzw. „Härte“ des Filters wird die Anzahl der falsch erkannten Inhalte innerhalb der Schlagwortmenge drastisch reduziert. Ist der Filter zu streng eingestellt, werden jedoch insgesamt weniger Worte in die Ergebnisse mit aufgenommen und es kann vorkommen, dass auch ursprünglich korrekt erkannte Wörter aufgrund eines niedrigen Confidence-Wertes verworfen werden.

### 3.4.2 Normalisierung

Um die aus der Texterkennung gewonnenen Daten zunächst für die weitere Filterung vorzubereiten, ist es sinnvoll, die Redundanz der Daten möglichst zu reduzieren und die einzelnen Wörter zu normalisieren. Beispielsweise kann durch das Umwandeln aller Textdaten in Kleinbuchstaben die Variation der Daten eingeschränkt werden, ohne jedoch für die Suche relevante Information zu verlieren. Diese Methode wurde auch in Abbildung 3.16 angewandt.





**Abbildung 3.14:** Anwendung der Schwellenwertmethode nach Kapur auf einen Beispiel-screenshot. Trotz der eigentlich einfachen Oberfläche erzeugt das Verfahren Artefakte, die die Texterkennung deutlich erschweren.

```

82  {
83    "Text": "das",
84    "Confidence": 96.24165344238281
85  },
86  {
87    "Text": "gewünschte",
88    "Confidence": 96.33263397216797
89  },
90  {
91    "Text": "u",
92    "Confidence": 10.228431701660156
93  },

```

```

58  {
59    "Text": "das",
60    "Confidence": 96.24165344238281
61  },
62  {
63    "Text": "gewünschte",
64    "Confidence": 96.33263397216797
65  },
66  {
67    "Text": " ",
68    "Confidence": 95
69  },

```

**Abbildung 3.15:** Auszug aus den Ergebnisdaten der Texterkennung aus 2.2 nach der Confidence-Filterung. Alle Wörter unter dem Schwellenwert werden entfernt.

### 3.4.3 Entfernung von Duplikaten

Nach der Normalisierung werden Duplikate innerhalb der erkannten Textdaten entfernt, um die Effizienz der nachfolgenden Filterverfahren zu erhöhen. In den Daten aus Abbildung 3.17 wird die Liste an erkannten Wörtern stark gekürzt und die Redundanz damit verringert.

```

14 "Selektieren",
15 "Sie",
16 "das",
17 "gewünschte",
18 " ",
19 "ARCHIVE",

```

```

14 "selektieren",
15 "sie",
16 "das",
17 "gewünschte",
18 " ",
19 "archive",

```

**Abbildung 3.16:** Auszug aus den Ergebnisdaten der Texterkennung aus 2.2 nach der Normalisierung. Alle Wörter beinhalten nun ausschließlich Kleinbuchstaben.

```

33 "+",
34 "treibervariable",
35 "doppelwort",
36 "+",
37 "treibervariable",
38 "float",
39 "+",
40 "treibervariable",

```

```

14 "doppelwort",
15 "treibervariable",
16 "variablendefinition",
17 "+",
18 "selektieren",

```

**Abbildung 3.17:** Auszug aus den Ergebnisdaten der Texterkennung aus 2.2 nach der Duplikatentfernung.

#### 3.4.4 Filterung anhand der Wortlänge

Verarbeitet das Texterkennungssystem Texte mit unregelmäßigen Abständen oder grafischen Artefakten in der Schrift, werden statt des eigentlichen Wortes fälschlicherweise oft kurze Symbolkombinationen erkannt. Um diese Kombinationen aus den Ergebnisdaten zu entfernen, können Zeichenketten, wie in Abbildung 3.18 gezeigt, mithilfe des Wortlängenfilters ungeachtet ihres Inhaltes verworfen werden.

Zusätzlich kann dieser Filter an die Anforderung des Zielsystems angepasst werden. So haben beispielsweise Wörter, die weniger als zwei Zeichen beinhalten, einen für die Schlagwortsuche zu geringen Informationsgehalt und werden daher entfernt.

```

16 "variablendefinition",
17 "+",
18 "selektieren",
19 "#:",
20 "archive",

```

```

15 "variablendefinition",
16 "selektieren",
17 "#:",
18 "archive",

```

**Abbildung 3.18:** Auszug aus den Ergebnisdaten der Texterkennung aus 2.2 nach der Wortlängenfilterung. Alle Wörter, die kürzer sind als der Schwellenwert, werden aus den Ergebnisdaten entfernt.

### 3.4.5 Sprachabhängige Filterung mittels Regular Expressions

Nachdem die zu filternden Textdaten durch vorherige Schritte vorverarbeitet wurden, werden die Ergebnisdaten ein letztes Mal mithilfe von regulären Ausdrücken (engl. *Regular Expressions*) durchsucht. Aufgrund der dynamischen Erweiterbarkeit der Regular Expressions kann für jede Sprache ein individueller Filter angelegt werden, der den jeweiligen Zeichensatz beschriftet und unbekannte Sonderzeichen oder Symbole entfernt. So sind beispielsweise im Deutschen Umlaute erlaubt, während häufig auftretende, jedoch unerwünschte Symbole wie das phonetische Zeichen „æ“ oder mehrere hintereinandergereihte Leerzeichen explizit entfernt werden können. Die Anwendung eines simplen deutschen Sprachfilters (`([\w'\-äöüÄÖÜß]{2,})`) wird in Abbildung 3.19 gezeigt.

```
14 "doppelwort",
15 "treibervariable",
16 "variablendefinition",
17 "+",
18 "selektieren",
19 "#:",
```

```
10 "doppelwort",
11 "treibervariable",
12 "variablendefinition",
13 "selektieren",
14 "archive",
15 "en"
```

**Abbildung 3.19:** Auszug aus den Ergebnisdaten der Texterkennung aus 2.2 nach der Filterung mit Regular Expressions. Findet der sprachabhängige Filter keine Treffer, wird das Wort aus den Ergebnisdaten entfernt.

## 3.5 Testaufbau

Der Testaufbau soll ein dynamisches Verketteten von unterschiedlichen Bildverarbeitungs- und Textfilterungsschritten erlauben.

Für einen objektiven Vergleich zwischen den verschiedenen Vorgehensweisen und Algorithmen wird eine Grundabfolge der jeweiligen Schritte in einer „Processing-Pipeline“ definiert. Diese Grundabfolge kann anschließend mit den jeweiligen zu vergleichenden Algorithmen erweitert werden. Die Ergebnisse können schließlich anhand der in Abschnitt 2.4 beschriebenen Fehlermetriken mit einer durch den Menschen verschlagworteten Vergleichsmenge abgeglichen werden.

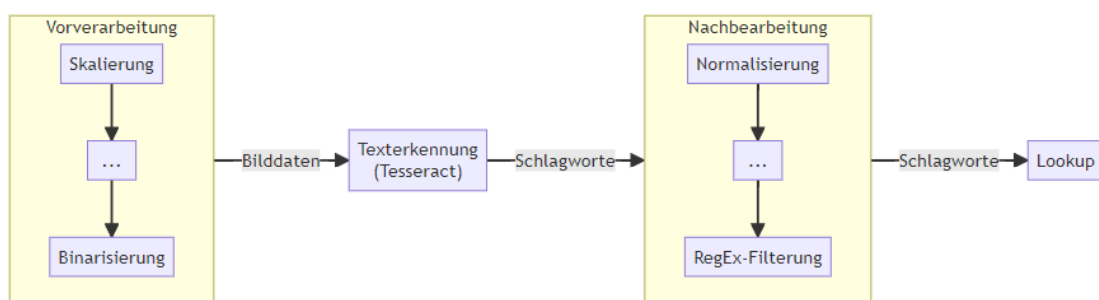
## Kapitel 4

# Prototypische Implementierung

In der prototypischen Implementierung werden die in Kapitel 2 bzw. Kapitel 3 angeführten Algorithmen zu Verarbeitungsketten zusammengefügt. Anhand einer Stichprobe wird ein automatisierter Bericht generiert, welcher detaillierte Informationen über Gesamtergebnisse der Texterkennung mit den jeweiligen Bildern und den verwendeten Algorithmen in Bezug setzt.

### 4.1 Komponenten und Bibliotheken

Die für die prototypische Implementierung verwendeten Bibliotheken stellen Komponenten zur Verarbeitung von Screenshotdaten zur Verfügung. Um die Funktionalität flexibel auf unterschiedliche Anforderungen anpassen zu können, sind die Komponenten möglichst lose miteinander gekoppelt. Ein beispielhafter Ablauf eines Texterkennungsprogramms unter Nutzung der Bibliotheken ist in Abbildung 4.1 abgebildet.



**Abbildung 4.1:** Beispielhafter Ablauf eines Texterkennungsprogramms

#### 4.1.1 Fremdbibliotheken

Die prototypische Implementierung erfolgt in C# .NET. Entsprechend der in Kapitel 2 und Kapitel 3 vorgestellten Technologien und Werkzeuge werden folgende NuGet-Bibliotheken als Basis für die Implementierung verwendet:

- **Magick.NET** [40]  
Version: 13.1.0  
Lizenz: Apache-2.0
- **Tesseract** [41]  
Version: 5.2.0  
Lizenz: Apache-2.0

#### 4.1.2 Verarbeitungsketten

Beim Entwurf des Verarbeitungssystems für die unterschiedlichen Bild- und Textverarbeitungsschritte wurde bewusst auf Flexibilität geachtet. Mithilfe von Interfaces und Builder-Methoden ist es möglich, Verarbeitungsschritte gemäß Programm 4.1 als Prozessoren (engl. *Processors*) zu definieren. So stellt beispielsweise das Normalisieren eines durch Tesseract erkannten Wortes einen Verarbeitungsschritt dar, wie in Programm 4.2 gezeigt.

```
public interface IProcessor
{
    IEnumerable Process(IEnumerable inputs);
}

public interface IProcessor<in TInput, out TOutput> : IProcessor
{
    IEnumerable<TOutput> Process(IEnumerable<TInput> inputs);

    new IEnumerable Process(IEnumerable inputs)
    {
        return Process((IEnumerable<TInput>)inputs);
    }
}
```

**Programm 4.1:** Auszug aus Datei „IProcessor.cs“: Schnittstelle eines Prozessors.

Sollen mehrere Schritte verbunden werden, bietet das Processing-Framework die Möglichkeit, eine Verarbeitungskette aufzubauen. Gemäß der Schnittstellendefinition in Programm 4.3 können Delegates oder komplette Prozessoren dynamisch als einzelne Schritte aneinandergereiht werden. Die Typensicherheit wird durch das generische Typensystem von C# stets gewahrt.

Ist die Aufbauphase abgeschlossen, kann die Verarbeitungskette, konfiguriert in Programm 4.4, gestartet werden.

Abhängig von den verwendeten Prozessoren können also Eingangsdaten jeglichen Typs, in diesem Fall Bildobjekte der Magick.NET Bibliothek oder Ergebnisdaten des Texterkennungsvorgangs dynamisch verarbeitet werden.

```

public class ToLowerProcessor
    : Processor<ScanResult, ScanResult>
{
    public override IEnumerable<ScanResult> Process(
        IEnumerable<ScanResult> inputs
    )
    {
        foreach (var kv in inputs)
        {
            kv.Word.Text = kv.Word.Text.ToLower();
            yield return kv;
        }
    }
}

```

**Programm 4.2:** Auszug aus Datei „ToLowerProcessor.cs“: Normalisieren als einzelner Verarbeitungsschritt.

```

public interface IProcessorChainConfiguration<TInput, TOutput>
    : IProcessorChain<TInput, TOutput>
{
    IProcessorChainConfiguration<T, TOutput, TInput, TOutput> Use<T>(
        IProcessor<TInput, T> processor);

    IProcessorChainConfiguration<T, TOutput, TInput, TOutput> Use<T>(
        Func<IEnumerable<TInput>, IEnumerable<T>> processorFunc);

    IProcessorChainConfiguration<TInput, TOutput> Complete(
        IProcessor<TInput, TOutput> processor);

    IProcessorChainConfiguration<TInput, TOutput> Complete(
        Func<IEnumerable<TInput>, IEnumerable<TOutput>> processorFunc);
}

public interface IProcessorChainConfiguration<TInput, TOutput, TInChain, TOutChain>
{
    IProcessorChainConfiguration<T, TOutput, TInChain, TOutChain> Use<T>(
        IProcessor<TInput, T> processor);

    IProcessorChainConfiguration<T, TOutput, TInChain, TOutChain> Use<T>(
        Func<IEnumerable<TInput>, IEnumerable<T>> processorFunc);

    IProcessorChain<TInChain, TOutChain> Complete(
        IProcessor<TInput, TOutput> processor);

    IProcessorChain<TInChain, TOutChain> Complete(
        Func<IEnumerable<TInput>, IEnumerable<TOutput>> processorFunc);
}

```

**Programm 4.3:** Auszug aus Datei „IProcessorChainConfiguration.cs“: Schnittstelle zur Konfiguration einer Verarbeitungskette

```
var postProcessor = new ProcessorChainConfiguration<ScanResult, ScanResult>()
    .Use(new ConfidenceFilter(50))
    .Use(new ToLowerProcessor())
    .Use(new DuplicateFilter())
    .Complete(new RegexFilter(WordRegex));

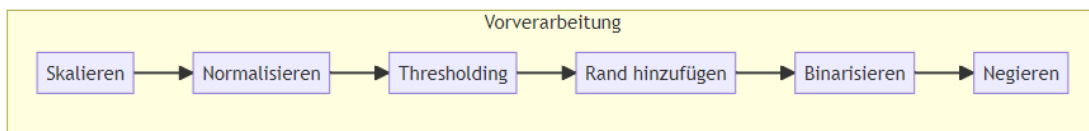
// ...

postProcessor.Process(data);
```

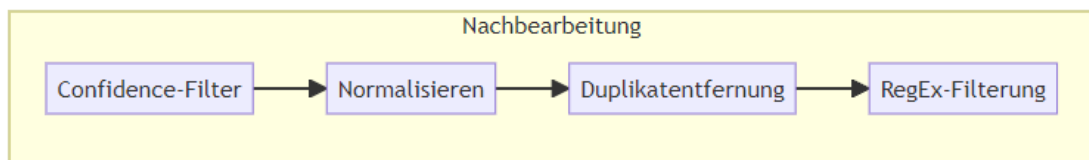
**Programm 4.4:** Auszug aus Datei „ImageViewModel.cs“: Konfiguration und Starten einer Verarbeitungskette.

#### 4.1.2.1 Bildverarbeitungskette

Für den Ablauf der Bildverarbeitung und der anschließenden Ergebnisfilterung werden die Erkenntnisse aus Kapitel 3 mithilfe des in Unterabschnitt 4.1.2 beschriebenen Processing-Frameworks angewandt. Die resultierende Konfiguration folgt dem Ablauf aus Abbildung 4.2 und Abbildung 4.3 und ist in Programm 4.5 bzw. 4.6 definiert.



**Abbildung 4.2:** Beispielhafter Ablauf der Vorverarbeitungskette



**Abbildung 4.3:** Beispielhafter Ablauf der Nachbearbeitungskette

Angefangen mit einem Ausgangsbild, welches über die Softwarebibliothek Magick.NET geladen wurde, beginnt die Bildverarbeitung zunächst mit dem Resampling. Falls der geladene Screenshot die Mindestauflösung von 300 dpi unterschreitet, wird es mittels Lanczos2-Verfahren, eine von Magick.NET mitgelieferte Implementierung des Lanczos2-Algorithmus mit leichter Schärfung [38], auf die Mindestauflösung vergrößert. Anschließend wird das Bild normalisiert, in Graustufen umgewandelt und jegliche Transparenz durch einen weißen Hintergrund ersetzt. Danach wird es mittels Schwellwertverfahren binarisiert. Rund um das Bild wird ein Rahmen mit einer Dicke von 10px eingefügt. Um Texterkennungsfehler durch falsche Vorder- bzw. Hintergrundfarben auszuschließen, wird das Bild gemeinsam mit einer farblich invertierten Version an das Texterkennungssystem weitergegeben.

```
var preprocessing = new ProcessorChainConfiguration<MagickImage, MagickImage>()
    .Use(new CloneImageProcessor())
    .Use(new ResizeProcessor(FilterType.Lanczos2Sharp, PixelInterpolateMethod.Mesh))
    .Use(new NormalizeProcessor())
    .Use(_thresholdProcessor)
    .Use(new AddBorderProcessor(10))
    .Use(new BinarizeProcessor())
    .Use(new NegateCloneProcessor())
    .Complete(OnPreprocessed);
```

**Programm 4.5:** Auszug aus Datei „EvaluationProcessor.cs“: Definition der Preprocessing-Kette.

Nach Verarbeitung des übergebenen Screenshots durch das Texterkennungssystem werden die Ergebnisse gefiltert. Dazu werden zunächst die Metadaten der einzelnen Wörter betrachtet und alle Elemente mit einer Confidence unter einem Schwellenwert von 50% verworfen. Danach werden die erkannten Texte mittels der C# -Funktion `ToLower()` normalisiert und anschließend auf Duplikate untersucht. Wurden alle Duplikate verworfen, werden die Wörter mittels sprachabhängigen Regular Expressions untersucht. Gemäß den Rahmenbedingungen in Unterabschnitt 2.3.2 wird ein simpler englischer (`[\\w'\\-]{2,}`) sowie deutscher (`[\\w'\\-äöüÄÖÜß]{2,}`) Sprachfilter festgelegt.

```
var postprocessing = new ProcessorChainConfiguration<ScanResult, ScanResult>()
    .Use(new ConfidenceFilter(50))
    .Use(new ToLowerProcessor())
    .Use(new DuplicateFilter())
    .Complete(new RegexFilter(wordRegex));
```

**Programm 4.6:** Auszug aus Datei „EvaluationProcessor.cs“: Definition der Postprocessing-Kette.

#### 4.1.3 Lookup

Die „Lookup“ Bibliothek abstrahiert das Speichern von Schlüssel-Wert-Paaren. Dabei kann flexibel zwischen verschiedenen Speicherimplementierungen gewechselt werden. So ist es beispielsweise möglich, die Werte in einer Listenstruktur im Arbeitsspeicher, in einer Datei oder – mittels der .NET EntityFramework-Bibliothek – in einer Datenbank persistent abzulegen. Unabhängig von der Ablagestruktur im Hintergrund können Lookups mittels einer gemeinsamen Schnittstelle, abgebildet in Programm 4.7, manipuliert werden.

#### 4.1.4 Optische Texterkennung

Die OCR-Bibliothek beinhaltet elementare Funktionen für die Texterkennung. Sie enthält Funktionen zur Bearbeitung von Bildern mittels Magick.NET inklusive anschließender Verarbeitung mittels Tesseract. Kernkomponenten wie das Texterkennungssystem werden automatisch auf Basis der Eingabeparameter konfiguriert und die Verwendung



```
public interface ILookup<TKey, TValue>
: ILookup,
IDictionary<TKey, ICollection<TValue>>,
IDisposable
{
    ICollection<TValue> Add(TKey key);

    public void Add(TKey key, TValue value);

    public void AddRange(TKey key, IEnumerable<TValue> values);

    public bool Remove(TKey key, TValue value);

    public ICollection<TValue> GetOrAdd(TKey key);
}
```

**Programm 4.7:** Auszug aus Datei „ILookup.cs“: Definition der gemeinsamen Schnittstelle für Lookups

der Ergebnisdaten in externen Programmteilen wird durch die zur Verfügung gestellten Modelle für die Ergebnisdaten vereinfacht. Außerdem enthält die Bibliothek eine Reihe von vordefinierten Verarbeitungsketten bzw. Prozessoren für die Bild- und Textverarbeitung.

#### 4.1.5 Automatische Berichterstellung

Mithilfe des ReportGenerator-Frameworks wird die automatische Berichterstellung für unterschiedlichste Ausgabeformate abstrahiert. Durch die mitgelieferten Schnittstellendefinitionen ist es möglich, eigene Ausgabeformate zu definieren. Der gesamte Funktionsumfang des ReportGenerators, beispielsweise das Erstellen von Tabellen oder das Anlegen von Überschriften, kann durch die Implementierung von Interfaces wie in Programm 4.8 an die jeweilige Syntax und Dokumentstruktur angepasst werden.

## 4.2 Vergleichsdaten

Als Ausgangsdaten für die Durchführung wurde eine zufällige Auswahl an Dokumentationsscreenshots getroffen, wie in Abbildung 4.4 abgebildet. Zusätzlich wurden auch Bilder wie in Abbildung 4.5 in die Stichprobe mit aufgenommen, die beispielsweise aufgrund ihrer Auflösung oder Kontrastverhältnisse schwer lesbar sind.

Die textuellen Inhalte aller ausgewählten Bilder wurden manuell extrahiert und für den programmatischen Vergleich in einer Datei im selben Verzeichnis wie die Quellbilddatei festgehalten. Ein Beispiel dafür ist der Screenshot „zrs\_ZAMS\_filter-alarmgroup\_001“ in Abbildung 4.6, welcher insgesamt 15 verschiedene Wörter beinhaltet.

```

public interface IDocumentGenerator : IStreamWriter
{
    IDocumentGenerator Append(string? text = default);

    IDocumentGenerator AppendLine(string? text = default);

    IDocumentGenerator AppendParagraph(string? text = default);

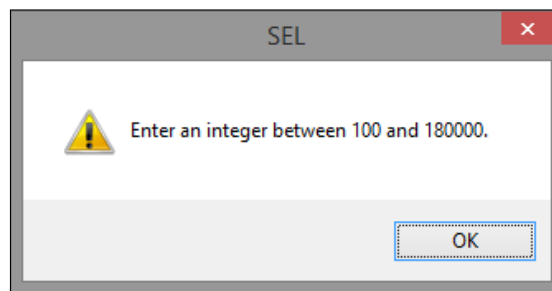
    IDocumentGenerator AppendHeading(int level, string text);

    IDocumentGenerator AppendTable(int columns, Action<ITableGenerator> table);

    string FormatImage(string path, IBounds? bounds = default);
}

```

**Programm 4.8:** Auszug aus Datei „IDocumentGenerator.cs“: Hauptschnittstelle für den ReportGenerator



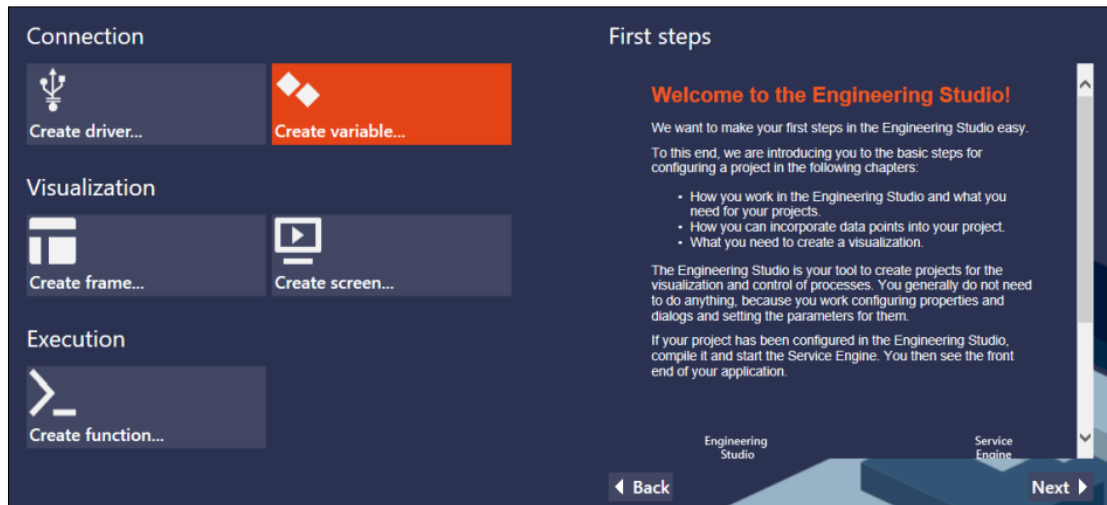
**Abbildung 4.4:** Ein gut für die Texterkennung geeigneter Screenshot. Die wesentlichen Inhalte weisen einen guten Kontrast zum Hintergrund auf und befinden sich in Bereichen mit gleichmäßiger Helligkeit.

## 4.3 Programmablauf

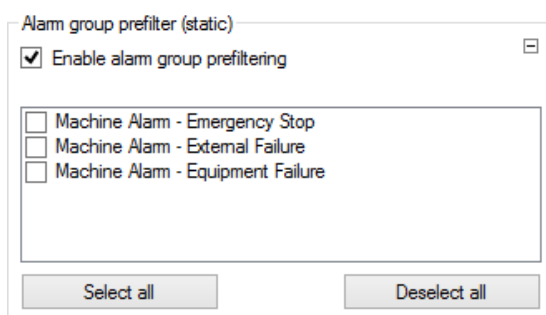
Die prototypische Implementierung besteht neben den oben genannten Komponenten aus einem ausführbaren Kommandozeilenprogramm zur Texterkennung und einem Programm zum Vergleich der Ergebnisse mit den manuell verschlagworteten Soll-Daten. Alle relevanten Daten werden in entsprechenden Ausgabeverzeichnissen festgehalten und dadurch für den jeweiligen nächsten Schritt verfügbar gemacht.

### 4.3.1 Texterkennung

Zu Beginn der Ausführung des Kommandozeilenprogramms wird für jedes zu verarbeitende Bild abhängig von den definierten Schwellenwertverfahren eine Reihe von Prozessoren angelegt. Dazu wird der statische Teil der Bildverarbeitungskette gemäß Unterunterabschnitt 4.1.2.1 innerhalb der „EvaluationProcessor“ Klasse definiert, wie in Programm 4.9 auszugsweise dargestellt. Lediglich die zu evaluierenden Prozessoren für die jeweiligen Schwellenwertverfahren können außerhalb der Klasse dynamisch definiert werden. Der EvaluationProcessor legt die erzeugten Ergebnisdaten, bestehend aus den



**Abbildung 4.5:** Ein schlecht lesbarer Screenshot. Aufgrund der vielen Symbole und der bunten Flächen stellt dieses Bild eine Herausforderung für das Texterkennungssystem dar.



```
{
  "words": [
    "group",
    "prefilter",
    "static",
    "enable",
    "alarm",
    "prefiltering",
    "machine",
    "emergency",
    "stop",
    "external",
    "failure",
    "equipment",
    "select",
    "all",
    "deselect"
  ]
}
```

**Abbildung 4.6:** Ein Screenshot und die daraus manuell extrahierten Schlagworte.

gefundenen Wörtern und zugehörigen Metadaten wie die Confidence, auf Dateiebene ab. Um überprüfen zu können, welches Bild an das Texterkennungssystem übergeben wird, werden auch die verarbeiteten Bilder nach der Binarisierung gespeichert.

Ist die Erstellung der Bildbearbeitungsprozessoren abgeschlossen, wird jeder einzelne Prozessor über die Systembibliothek „System.Threading.Tasks“ als eigene Ausführungseinheit (engl. *Task*) gestartet. In der Kommandozeile wird anschließend der aktuelle Status jedes Tasks angezeigt. Wurden alle Tasks abgeschlossen, wird das Programm

```
private static IEnumerable<EvaluationProcessor> MakeThresholdVariations()
{
    for (int i = 4; i <= 24; i += 4)
    {
        yield return new(new ThresholdAdaptiveProcessor(i));
    }

    for (int i = 20; i <= 80; i += 10)
    {
        yield return new(new ThresholdProcessor(i));
    }

    yield return new(new AutoThresholdProcessor(AutoThresholdMethod.Kapur));
    yield return new(new AutoThresholdProcessor(AutoThresholdMethod.OTSU));
    yield return new(new AutoThresholdProcessor(AutoThresholdMethod.Triangle));
}
```

**Programm 4.9:** Auszug aus Datei „Program.cs“: Definition der Thresholding Prozessoren.

beendet.

#### 4.3.2 Vergleich mit Soll-Daten

Wurden die in den jeweiligen Screenshots erkannten Textdaten abgelegt, werden diese Daten im zweiten Kommandozeilenprogramm „ReportGenerator“ mit den manuell verschlagworteten Daten verglichen und die Ergebnisse in einen Bericht (engl. *Report*) gespeichert.

Als zentrale Komponente für den Vergleich spielt die Berechnung der in Abschnitt 2.4 erklärten Metriken eine wesentliche Rolle. Wie in Programm 4.10 ersichtlich, wird die Distanz zwischen zwei C# -Enumerables, seien es zwei Strings oder zwei Listen, über das Verfahren nach Levenshtein berechnet.

Nach der Ermittlung der jeweiligen Distanzen auf Wort- bzw. Bildbasis werden sie mit den jeweiligen Ursprungsbildern, Prozessoren und den verwendeten Algorithmen in Bezug gesetzt. Die so aufbereiteten Ergebnisse werden anschließend an den ReportGenerator übergeben und in einem Bericht zusammengefasst.

```
public static double GetDistance<T>(T reference, T? hypothesis)
    where T : IEnumerable
{
    var refArr = reference.Cast<object>().ToArray();
    var hypArr = hypothesis?.Cast<object>().ToArray() ?? Array.Empty<object>();

    var distance = new int[refArr.Length + 1, hypArr.Length + 1];

    for (var x = 0; x <= refArr.Length; x++)
    {
        distance[x, 0] = x;
    }

    for (var y = 0; y <= hypArr.Length; y++)
    {
        distance[0, y] = y;
    }

    for (var x = 0; x < refArr.Length; x++)
    {
        for (var y = 0; y < hypArr.Length; y++)
        {
            var cost = Equals(refArr[x], hypArr[y]) ? 0 : 1;

            var c1 = distance[x, y] + cost; // Bottom left

            var c2 = distance[x, y + 1] + 1; // Top left
            var c3 = distance[x + 1, y] + 1; // Bottom right

            distance[x + 1, y + 1] = Min(c1, c2, c3); // Top right
        }
    }

    return distance[refArr.Length, hypArr.Length];
}
```

**Programm 4.10:** Auszug aus Datei „Calculator.cs“: Berechnung der Levenshtein-Distanz.

## Kapitel 5

# Evaluierung

Nachdem die vorbereiteten Bilddaten an das Texterkennungssystem gemäß Kapitel 4 übergeben und die Ergebnisse ermittelt wurden, werden die extrahierten Textdaten mit den manuell erstellten „Soll-Daten“ verglichen. Anhand der Statistik kann festgestellt werden, welche Vorgehensweise zu der besten Qualität führt. Um die Ergebnisse zu visualisieren, erstellt der in Unterabschnitt 4.1.5 beschriebene „ReportGenerator“ auf Basis der Bilddateinamen automatisch einen Bericht mit den Vergleichsdaten in Tabellenform. Der erstellte Bericht wird in verschiedene Kategorien unterteilt.

### 5.1 Verarbeitungsstatistik

In der Sektion „Processor Stats“ (siehe Tabelle 5.1), welche die Verarbeitungsstatistik enthält, wird die Gesamtwortfehlerrate pro Bild mit dem jeweiligen Prozessor in Verhältnis gesetzt. Anhand der Metriken ist zu erkennen, dass gewisse Bilder aufgrund ihrer Eigenschaften (niedrige Auflösung, schwierige Farbgebung) von allen Prozessoren gleichermaßen schlecht erkannt werden. Jedoch fallen auch Spezialfälle auf: So werden beispielsweise die Textdaten in den Dialog-Knöpfen des Bilds „worldview\_zoom\_steps\_001“ aufgrund eines falsch gewählten Schwellenwerts unkenntlich gemacht. Die Texterkennung schlägt fehl.

### 5.2 Ergebnisse

Die Sektion „Scan Results“ (siehe Tabelle 5.2 bzw. Tabelle 5.3) zeigt die Verarbeitungsergebnisse auf Metrik- bzw. Wortbasis und bildet den Abschluss des Detailvergleichs. Hier werden alle Verfahrenskombinationen einzeln und mit allen verfügbaren Daten aufgeführt.

### 5.3 Prozessoren im Überblick

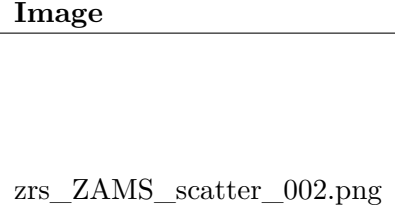
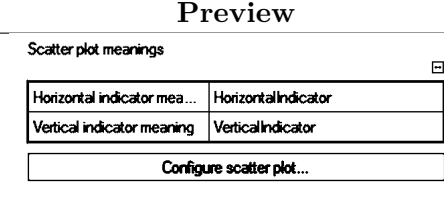
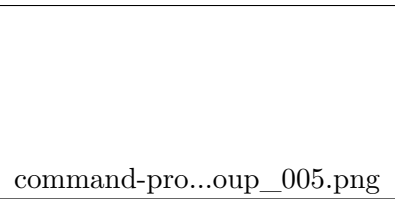
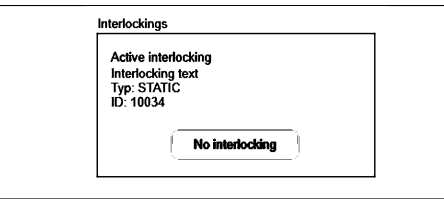
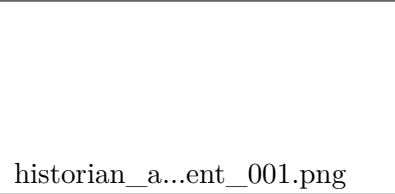
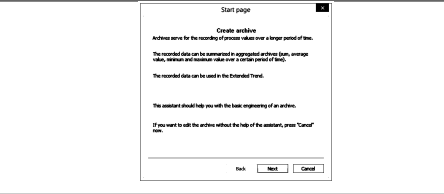
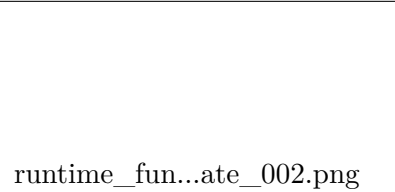
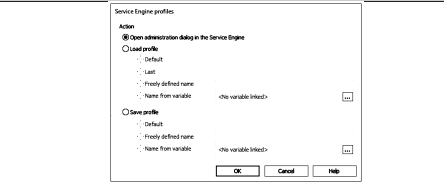
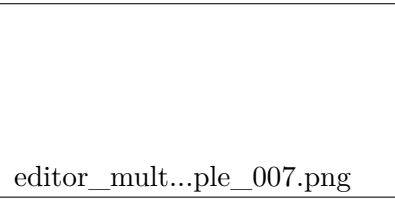
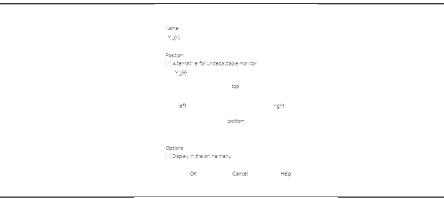
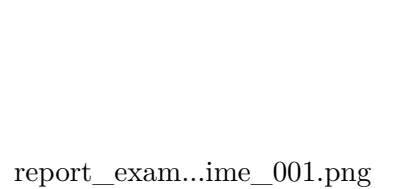
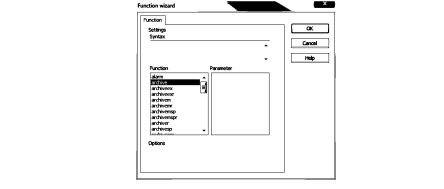
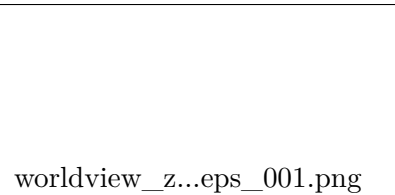
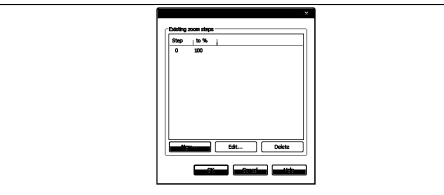

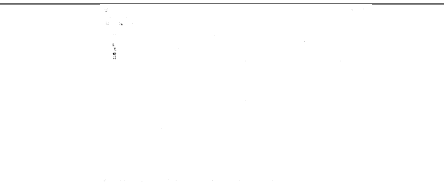
Neben dem Detailvergleich beinhaltet der generierte Bericht auch die „Processing Summary“. Diese Kategorie zeigt eine kurze Übersicht aller Ergebnisse. Je nach Rubrik wird

jeweils der Median bzw. Durchschnitt der Character Error Rate und der Word Error Rate berechnet.

Auf Basis der Daten in Tabelle 5.4, Tabelle 5.5 und Tabelle 5.6 lässt sich der Gesamterfolg der Bildvorbereitung bzw. der darauf folgenden Filterung feststellen.

Die Dreiecks-Schwellenwertmethode, wie in Unterabschnitt 3.3.6 vermutet, eignet sich beispielsweise nicht für die Texterkennung. In der Detailübersicht zeigt sich, dass für die Bilder oft ein Schwellenwert gewählt wird, der die Texterkennung unmöglich macht. Bei Anwendung des fixen Schwellenwertverfahrens werden mit dem richtigen Schwellenwert durchschnittlich sehr gute Ergebnisse erzielt. Beim Verfahren nach Otsu wird oft ein geeigneter Schwellenwert gewählt, wodurch in den meisten Fällen ein gutes Ausgangsbild für die Texterkennung entsteht.

Obwohl die Fehlerquoten der Texterkennung mit Vorbereitung der Daten niedriger sind als die der Texterkennung ohne Vorbereitung, ist das Ergebnis insgesamt unzufriedenstellend. Selbst bei Verwendung des fixen Schwellenwertverfahrens mit einem Wert von 40 % werden durchschnittlich Ergebnisse mit einer Wortfehlerrate von mindestens 46 % bzw. 1,5 falsch erkannten Zeichen pro Wort erreicht. Die relativ hohe Standardabweichung von 26 % lässt auf eine hohe Streuung der Ergebnisdaten, also unregelmäßig gute Erfolge schließen.

Image	Preview	Distance
		0,09
		0,11
		0,19
		0,21
		1,00
		1,00
		1,00
		1,00

**Tabelle 5.1:** Auszug aus der „Processor Stats“ Tabelle im generierten Bericht. Die Eigenschaften der Originalbilder im Vergleich zu den verarbeiteten Bildern geben Aufschluss über die Arbeitsweise und Effektivität des Prozessors. Im obersten Bild sind die Inhalte gut zu erkennen, wogegen in der untersten Reihe nach der Filterung nur wenige Artefakte zurückbleiben.



Processor	Elapsed	WER	CER (avg)	Matches	Image
ThresholdProcessor(30%)	0,6ms	0,0%	0,00	9 / 9	<div>Interlockings</div> <div>Active interlocking Interlocking text Typ: STATIC ID: 10034</div> <div>No interlocking</div>
ThresholdProcessor(40%)	0,6ms	0,0%	0,00	9 / 9	<div>Interlockings</div> <div>Active interlocking Interlocking text Typ: STATIC ID: 10034</div> <div>No interlocking</div>
ThresholdProcessor(70%)	0,7ms	0,0%	0,00	9 / 9	<div>Interlockings</div> <div>Active interlocking Interlocking text Typ: STATIC ID: 10034</div> <div>No interlocking</div>
ThresholdProcessor(60%)	1,1ms	0,0%	0,00	9 / 9	<div>Interlockings</div> <div>Active interlocking Interlocking text Typ: STATIC ID: 10034</div> <div>No interlocking</div>
ThresholdAd...ssor(12_12)	0,5ms	77,8%	3,00	2 / 9	<div>Interlockings</div> <div>Active interlocking Interlocking text Typ: STATIC ID: 10034</div> <div>No interlocking</div>
ThresholdAd...ssor(04_04)	1,0ms	88,9%	3,22	1 / 9	<div>Interlockings</div> <div>Active interlocking Interlocking text Typ: STATIC ID: 10034</div> <div>No interlocking</div>
ThresholdAd...ssor(08_08)	0,6ms	100,0%	5,89	0 / 9	<div>Interlockings</div> <div>Active interlocking Interlocking text Typ: STATIC ID: 10034</div> <div>No interlocking</div>
AutoThresho...r(Triangle)	12,8ms	100,0%	5,89	0 / 9	<div>Interlockings</div> <div>Active interlocking Interlocking text Typ: STATIC ID: 10034</div> <div>No interlocking</div>

Tabelle 5.2: Auszug aus der „Scan Results“ Tabelle im generierten Bericht. Für jede Ausgabedatei werden sämtliche Statistiken aufgelistet.

Processor	10034	active	id	interlocking	...	text	typ
ThresholdProcessor(30%)	10034	active	id	interlocking	...	text	typ
ThresholdProcessor(40%)	10034	active	id	interlocking	...	text	typ
ThresholdProcessor(70%)	10034	active	id	interlocking	...	text	typ
ThresholdProcessor(60%)	10034	active	id	interlocking	...	text	typ
ThresholdAd...ssor(12_12)	no	no	no	interlocking	...	no	no
ThresholdAd...ssor(04_04)	text	text	text	interlockings'	...	text	text
ThresholdAd...ssor(08_08)	-	-	-	-	...	-	-
AutoThresho...r(Triangle)	-	-	-	-	...	-	-

**Tabelle 5.3:** Auszug aus der „Scan Results“ Tabelle im generierten Bericht. Neben der numerischen Statistik erfolgt zudem ein Vergleich der tatsächlichen Inhalte mit den erkannten Wörtern. „-“ steht hierbei für ein nicht erkanntes Wort.

Processor	Error	Deviation
ThresholdProcessor(40%)	46,04%	26,36%
ThresholdProcessor(30%)	46,84%	28,66%
AutoThresholdProcessor(OTSU)	48,76%	33,26%
ThresholdProcessor(50%)	49,95%	31,09%
ThresholdProcessor(70%)	49,96%	29,32%
...	...	...
ThresholdAdaptiveProcessor(12_12)	70,85%	24,37%
ThresholdAdaptiveProcessor(16_16)	71,32%	26,39%
ThresholdAdaptiveProcessor(08_08)	73,94%	22,37%
AutoThresholdProcessor(Triangle)	91,24%	15,16%
ThresholdAdaptiveProcessor(04_04)	94,55%	6,38%

**Tabelle 5.4:** Auszug aus der „Processing Summary“ Tabelle im generierten Bericht: Auflistung der Verfahren mit den durchschnittlich besten und schlechtesten Ergebnissen auf Basis der Word Error Rate. Die jeweilige Verarbeitungsmethode ist in der Spalte „Processor“ zu finden, die Wortfehlerrate und die Standardabweichung in „Error“ und „Deviation“.

Processor	Changes	Deviation
ThresholdProcessor(40%)	1,55	2,39
ThresholdProcessor(30%)	1,74	2,75
ThresholdProcessor(50%)	1,93	2,72
ThresholdProcessor(70%)	2,15	2,97
AutoThresholdProcessor(OTSU)	2,15	3,06
...	...	...
ThresholdAdaptiveProcessor(24_24)	3,00	3,18
ThresholdAdaptiveProcessor(08_08)	3,17	3,07
ThresholdAdaptiveProcessor(16_16)	3,22	3,15
AutoThresholdProcessor(Triangle)	4,71	3,22
ThresholdAdaptiveProcessor(04_04)	4,87	2,88

**Tabelle 5.5:** Auszug aus der „Processing Summary“ Tabelle im generierten Bericht: Auflistung der Verfahren mit den durchschnittlich besten und schlechtesten Ergebnissen auf Basis der Character Error Rate. Die jeweilige Verarbeitungsmethode ist in der Spalte „Processor“ zu finden, die Zeichenfehlerrate und die Standardabweichung in „Changes“ und „Deviation“.

Processor	Time	Deviation
ThresholdAdaptiveProcessor(16_16)	0,49ms	0,28ms
ThresholdAdaptiveProcessor(20_20)	0,59ms	0,36ms
ThresholdAdaptiveProcessor(12_12)	0,59ms	0,28ms
ThresholdAdaptiveProcessor(24_24)	0,62ms	0,19ms
AutoThresholdProcessor(Kapur)	0,62ms	0,06ms
...	...	...
ThresholdAdaptiveProcessor(04_04)	0,88ms	0,11ms
ThresholdProcessor(50%)	0,95ms	0,64ms
AutoThresholdProcessor(Triangle)	1,09ms	2,50ms
ThresholdProcessor(20%)	1,58ms	4,22ms
ThresholdProcessor(40%)	1,70ms	0,47ms

**Tabelle 5.6:** Auszug aus der „Processing Summary“ Tabelle im generierten Bericht: Auflistung der Verfahren mit den durchschnittlich besten und schlechtesten Ergebnissen auf Basis der Laufzeit. Die jeweilige Verarbeitungsmethode ist in der Spalte „Processor“ zu finden, die Laufzeit und die Standardabweichung in „Time“ und „Deviation“.

## Kapitel 6

# Zusammenfassung

Im Rahmen dieser Bachelorarbeit setzt sich mit der Anwendung von optischer Texterkennung für die Verschlagwortung von Oberflächenscreenshots auseinander. Das Ziel, eine Vorgehensweise für die automatisierte Verschlagwortung von Bilddateien zu entwickeln und die Anwendbarkeit verschiedener Algorithmen zur Bildbearbeitung und Texterkennung zu evaluieren, wurde erreicht. Insgesamt wurden die Grundlagen moderner Bildverarbeitung bzw. der Funktionsweise von Texterkennungssystemen aufgearbeitet. Das Ergebnis ist ein universelles System zur Analyse der Vergleichswerte, das auch die Integration weiterer Algorithmen ermöglicht. Dies erlaubt eine kontinuierliche Verbesserung und Anpassung des Texterkennungssystems an neue Anforderungen.

Aufgrund der Auswahl und Parametrisierung der ersten Tests beträgt die resultierende Fehlerrate in etwa 40%. Nach Abschluss der Bachelorarbeit wird es weitere Optimierungsversuche geben. Das nächste Ziel ist es, eine Wortfehlerrate von 20% nicht zu überschreiten, um eine zuverlässige Erkennung der Texte zu gewährleisten. Dazu werden weitere Bildbearbeitungsmethoden, beispielsweise Kantendetektoren [15] in Verbindung mit sogenanntem „Fuzzy Matching“ [3] für die Erkennung der Schlagworte verwendet. Ebenso gibt es unter Verwendung der bereits implementierten Algorithmen noch Verbesserungspotential. Beispielsweise könnten mehrfach gefundene Wörter, die in der aktuellen Version des Prototyps einfach verworfen werden, zur Ermittlung der Wichtigkeit eines Schlagwortes genutzt werden.

Nach Abschluss der Optimierungsversuche wird das gewonnene Wissen weiter in den Screenshot-Manager einfließen und die Texterkennungsfunktionalität in das Programm integriert. Die gefundenen mehrsprachigen Schlagworte werden in Zukunft für jedes Bild ermittelt und in einer Datenbank abgelegt, um ein einfaches und effizientes Suchen innerhalb der Anwendung zu ermöglichen.

# Abbildungsverzeichnis

1.1	Beispielhafte Auswahl typischer Dialogscreenshots. . . . .	2
2.1	Ein optimales Ergebnisbild. Jegliche farblichen Flächen wurden durch die Bildverarbeitung entfernt. Übrig bleibt klar lesbarer Text mit einem hohen Kontrast zum Hintergrund. . . . .	5
2.2	Auszug aus den ungefilterten Ergebnisdaten bei Durchführung der Texterkennung in dem gezeigten Screenshot. . . . .	7
3.1	Ein Vergleich unterschiedlicher Resampling-Filter. Durch die Aufteilung der Fehler auf mehrere Pixel bleiben Details und Konturen bei Anwendung des Lanczos-Filters vergleichsweise gut erhalten und der Text ist gut lesbar. . . . .	11
3.2	Ein im Verhältnis zur Bildgröße zu großer einfärbiger Rahmen . . . . .	12
3.3	Anwendung des festen Schwellenwertverfahrens auf einen Beispielscreenshot. Bei einem passenden Schwellenwert und nur geringfügig verschiedenen Farbflächen ist der Textinhalt gut vom Hintergrund abgrenzbar. Der Schwellenwert im gezeigten Bild beträgt 60 %. . . . .	13
3.4	Anwendung des festen Schwellenwertverfahrens auf einen Beispielscreenshot. Bei einem falsch gewählten Schwellenwert oder komplexen UI-Elementstrukturen ist der Text nicht erkennbar. Der Schwellenwert im gezeigten Bild beträgt 80 %. . . . .	13
3.5	Anwendung der adaptiven Schwellenwertmethode auf einen Beispielscreenshot. Die Blockgröße ist gut an den Bildinhalt angepasst und alle Details bleiben erhalten. Dieses Verfahren punktet hier besonders bei den farbigen „Energy Labels“, deren Textinhalte sonst mittels keinem anderen Verfahren komplett erkannt wurden. . . . .	14
3.6	Anwendung der adaptiven Schwellenwertmethode auf einen Beispielscreenshot. Die bei unangepasster Blockgröße entstehenden Artefakte schränken die Funktionsweise des Texterkennungssystems deutlich ein. . . . .	15
3.7	Beispiel eines unimodalen Histogramms mit einem deutlichen Spitzenwert [45] . . . . .	15
3.8	Anwendung der Dreiecks-Schwellenwertmethode auf einen Beispielscreenshot. Gleichen sich die Inhalte farblich, werden einige Details extrahiert. Durch kleinste Farbvariationen im Bild weicht der Schwellenwert jedoch vom Optimum ab und die Texte sind nur schwer zu erkennen. . . . .	16

3.9	Anwendung der Dreiecks-Schwellenwertmethode auf einen Beispielscreenshot. Bereits bei mäßiger Variation der Helligkeit verschwinden die ersten Details. . . . .	16
3.10	Beispiel eines bimodalen Histogramms [44] . . . . .	17
3.11	Anwendung der Schwellenwertmethode nach Otsu [17] auf einen Beispielscreenshot. Wird ein passender Schwellenwert ermittelt, lässt sich der Text gut vom Hintergrund trennen. . . . .	17
3.12	Anwendung der Schwellenwertmethode nach Otsu [17] auf einen Beispielscreenshot. Bei komplexen Strukturen im User-Interface gehen aufgrund des globalen Schwellenwerts Details verloren. . . . .	18
3.13	Anwendung der Schwellenwertmethode nach Kapur auf einen Beispielscreenshot. Trotz der vielen verwendeten Farben wird der Inhalt gut dargestellt. . . . .	18
3.14	Anwendung der Schwellenwertmethode nach Kapur auf einen Beispielscreenshot. Trotz der eigentlich einfach erscheinenden Oberfläche erzeugt das Verfahren Artefakte, die die Texterkennung deutlich erschweren. . .	19
3.15	Auszug aus den Ergebnisdaten der Texterkennung aus 2.2 nach der Confidence-Filterung. Alle Wörter unter dem Schwellenwert werden entfernt. . . . .	19
3.16	Auszug aus den Ergebnisdaten der Texterkennung aus 2.2 nach der Normalisierung. Alle Wörter beinhalten nun ausschließlich Kleinbuchstaben. . . . .	20
3.17	Auszug aus den Ergebnisdaten der Texterkennung aus 2.2 nach der Duplikatentfernung. . . . .	20
3.18	Auszug aus den Ergebnisdaten der Texterkennung aus 2.2 nach der Wortlängenfilterung. Alle Wörter, die kürzer sind als der Schwellenwert, werden aus den Ergebnisdaten entfernt. . . . .	20
3.19	Auszug aus den Ergebnisdaten der Texterkennung aus 2.2 nach der Filterung mit Regular Expressions. Findet der sprachabhängige Filter keine Treffer, wird das Wort aus den Ergebnisdaten entfernt. . . . .	21
4.1	Beispielhafter Ablauf eines Texterkennungsprogramms . . . . .	22
4.2	Beispielhafter Ablauf der Vorverarbeitungskette . . . . .	25
4.3	Beispielhafter Ablauf der Nachbearbeitungskette . . . . .	25
4.4	Ein gut für die Texterkennung geeigneter Screenshot. Die wesentlichen Inhalte weisen einen guten Kontrast zum Hintergrund auf und befinden sich in Bereichen mit gleichmäßiger Helligkeit. . . . .	28
4.5	Ein schlecht lesbarer Screenshot. Aufgrund der vielen Symbole und der bunten Flächen stellt dieses Bild eine Herausforderung für das Texterkennungssystem dar. . . . .	29
4.6	Ein Screenshot und die daraus manuell extrahierten Schlagworte. . . . .	29

# Tabellenverzeichnis

5.1	Auszug aus der „Processor Stats“ Tabelle im generierten Bericht. Die Eigenschaften der Originalbilder im Vergleich zu den verarbeiteten Bildern geben Aufschluss über die Arbeitsweise und Effektivität des Prozessors. Im obersten Bild sind die Inhalte gut zu erkennen, wogegen in der untersten Reihe nach der Filterung nur wenige Artefakte zurückbleiben. .	34
5.2	Auszug aus der „Scan Results“ Tabelle im generierten Bericht. Für jede Ausgabedatei werden sämtliche Statistiken aufgelistet. . . . .	35
5.3	Auszug aus der „Scan Results“ Tabelle im generierten Bericht. Neben der numerischen Statistik erfolgt zudem ein Vergleich der tatsächlichen Inhalte mit den erkannten Wörtern. „-“ steht hierbei für ein nicht erkanntes Wort. . . . .	36
5.4	Auszug aus der „Processing Summary“ Tabelle im generierten Bericht: Auflistung der Verfahren mit den durchschnittlich besten und schlechtesten Ergebnissen auf Basis der Word Error Rate. Die jeweilige Verarbeitungsmethode ist in der Spalte „Processor“ zu finden, die Wortfehlerrate und die Standardabweichung in „Error“ und „Deviation“. . . . .	36
5.5	Auszug aus der „Processing Summary“ Tabelle im generierten Bericht: Auflistung der Verfahren mit den durchschnittlich besten und schlechtesten Ergebnissen auf Basis der Character Error Rate. Die jeweilige Verarbeitungsmethode ist in der Spalte „Processor“ zu finden, die Zeichenfehlerrate und die Standardabweichung in „Changes“ und „Deviation“. . . .	37
5.6	Auszug aus der „Processing Summary“ Tabelle im generierten Bericht: Auflistung der Verfahren mit den durchschnittlich besten und schlechtesten Ergebnissen auf Basis der Laufzeit. Die jeweilige Verarbeitungsmethode ist in der Spalte „Processor“ zu finden, die Laufzeit und die Standardabweichung in „Time“ und „Deviation“. . . . .	37

# Quellenverzeichnis

## Literatur

- [1] AMAM Asif u. a. „An overview and applications of optical character recognition“. *International Journal of Advance Research In Science And Engineering* 3.7 (2014). (Besucht am 18.02.2024) (siehe S. 4).
- [2] Thibaud Briand und Pascal Monasse. „Theory and practice of image B-spline interpolation“. *Image Processing On Line* 8 (2018), S. 99–141 (siehe S. 11).
- [3] M Cayrol, H Farreny und H Prade. „Fuzzy pattern matching“. *Kybernetes* 11.2 (1982), S. 103–116 (siehe S. 38).
- [4] K.R. Chowdhary. „Natural language processing“. *Fundamentals of artificial intelligence* (2020) (siehe S. 6).
- [5] Kenneth W. Church und Lisa F. Rau. „Commercial Applications of Natural Language Processing“. *Commun. ACM* 38.11 (1995). DOI: church1995. (Besucht am 18.02.2024) (siehe S. 6).
- [6] Line Eikvil. „Optical character recognition“. *citeseer.ist.psu.edu/142042.html* 26 (1993) (siehe S. 4).
- [7] Shreyas Fadnavis. „Image interpolation techniques in digital image processing: an overview“. *International Journal of Engineering Research and Applications* 4.10 (2014), S. 70–73 (siehe S. 11).
- [8] Urvashi Gupta und Rohit Sharma. „Comparison of Different Cloud Computing Platforms for Data Analytics“. In: Sep. 2023. DOI: 10.1007/978-981-99-3716-5\_7 (siehe S. 4).
- [9] Noman Islam, Zeeshan Islam und Nazia Noor. „A survey on optical character recognition systems“. *arXiv preprint* (2017). URL: <https://doi.org/10.48550/arXiv.1710.05703> (besucht am 18.02.2024) (siehe S. 4).
- [10] Krishna Prakash Kalyanathaya, D Akila und P Rajesh. „Advances in natural language processing: a survey of current research trends, development tools and industry applications“. *International Journal of Recent Technology and Engineering* 7.5C (2019) (siehe S. 6).
- [11] Jagat Narain Kapur, Prasanna K Sahoo und Andrew KC Wong. „A new method for gray-level picture thresholding using the entropy of the histogram“. *Computer vision, graphics, and image processing* 29.3 (1985) (siehe S. 16).



- [12] Romain Karpinski, Devashish Lohani und Abdel Belaid. „Metrics for complete evaluation of ocr performance“. In: *IPCV'18 - The 22nd Int'l Conf on Image Processing, Computer Vision, & Pattern Recognition*. 2018. URL: <https://inria.hal.science/hal-01981731> (siehe S. 2, 7, 8).
- [13] Nalin Kumar und M Nachamai. „Noise removal and filtering techniques used in medical images“. *Oriental Journal of Computer Science and Technology* 10.1 (2017), S. 103–113 (siehe S. 5).
- [14] Vladimir I Levenshtein u. a. „Binary codes capable of correcting deletions, insertions, and reversals“. In: *Soviet physics doklady*. Bd. 10. 8. Soviet Union. 1966, S. 707–710 (siehe S. 7, 8).
- [15] Xiaoqing Liu und Jagath Samarabandu. „Multiscale edge-based text extraction from complex images“. In: *2006 IEEE International Conference on Multimedia and Expo*. IEEE. 2006, S. 1721–1724 (siehe S. 38).
- [16] Lily Rojabiyati Mursari und Antoni Wibowo. „The effectiveness of image pre-processing on digital handwritten scripts recognition with the implementation of OCR Tesseract“. *Computer Engineering and Applications Journal* 10.3 (2021) (siehe S. 5).
- [17] Nobuyuki Otsu. „A threshold selection method from gray-level histograms“. *IEEE transactions on systems, man, and cybernetics* 9.1 (1979). DOI: 10.1109/TSMC.1979.4310076. (Besucht am 18.02.2024) (siehe S. 15, 17, 18).
- [18] Youngja Park u. a. „An empirical analysis of word error rate and keyword error rate.“ In: Sep. 2008. DOI: 10.21437/Interspeech.2008-537 (siehe S. 7, 8).
- [19] Sandip Rakshit, Subhadip Basu und Hisashi Ikeda. „Recognition of handwritten textual annotations using tesseract open source ocr engine for information just in time (ijit)“. *arXiv preprint arXiv:1003.5893* (2010) (siehe S. 4).
- [20] Prasanna K Sahoo, SAKC Soltani und Andrew KC Wong. „A survey of thresholding techniques“. *Computer vision, graphics, and image processing* 41.2 (1988) (siehe S. 13, 14).
- [21] Ryo Seta, Kan Okubo und Norio Tagawa. „Digital image interpolation method using higher-order Hermite interpolating polynomials with compact finite-difference“. In: *Proceedings: APSIPA ASC 2009: Asia-Pacific Signal and Information Processing Association, 2009 Annual Summit and Conference*. Asia-Pacific Signal und Information Processing Association. 2009, S. 406–409 (siehe S. 11).
- [22] Ray Smith. „An Overview of the Tesseract OCR Engine“. In: *Ninth international conference on document analysis and recognition (ICDAR 2007)*. Bd. 2. IEEE. 2007. URL: <https://ieeexplore.ieee.org/document/4376991> (besucht am 18.02.2024) (siehe S. 4).
- [23] Dan Sporici, Elena Cuşnir und Costin-Anton Boiangiu. „Improving the accuracy of Tesseract 4.0 OCR engine using convolution-based preprocessing“. *Symmetry* 12.5 (2020) (siehe S. 5).

- [24] Xiang Tong und David A. Evans. „A Statistical Approach to Automatic OCR Error Correction in Context“. In: *Fourth Workshop on Very Large Corpora*. Hrsg. von Donia Scott. Herstmonceux Castle, Sussex, UK: Association for Computational Linguistics, Juni 1996. URL: <https://aclanthology.org/W96-0108> (siehe S. 7).
- [25] William Ughetta und Brian W. Kernighan. „The Old Bailey and OCR: Benchmarking AWS, Azure, and GCP with 180,000 Page Images“. English (US). In: Association for Computing Machinery, Inc, Sep. 2020. DOI: 10.1145/3395027.3419595 (siehe S. 4).
- [26] Michael Unser. „Splines: A perfect fit for signal and image processing“. *IEEE Signal processing magazine* 16.6 (1999), S. 22–38 (siehe S. 11).
- [27] Ye-Yi Wang, Alex Acero und Ciprian Chelba. „Is word error rate a good indicator for spoken language understanding accuracy“. In: *2003 IEEE workshop on automatic speech recognition and understanding (IEEE Cat. No. 03EX721)*. IEEE. 2003, S. 577–582 (siehe S. 7).
- [28] W John Wilbur und Karl Sirotkin. „The automatic identification of stop words“. *Journal of information science* 18.1 (1992) (siehe S. 6).
- [29] Gregory W Zack, William E Rogers und Samuel A Latt. „Automatic measurement of sister chromatid exchange frequency.“ *Journal of Histochemistry & Cytochemistry* 25.7 (1977) (siehe S. 14).
- [30] Inc. Amazon Web Services. *Amazon Textract - Homepage*. eng. 23. Mai 2023. URL: <https://aws.amazon.com/textract> (besucht am 18.02.2024) (siehe S. 4).
- [31] Inc. Amazon Web Services. *Amazon Textract - Pricing*. eng. 23. Mai 2023. URL: <https://aws.amazon.com/textract/pricing/> (besucht am 18.02.2024) (siehe S. 10).
- [32] Microsoft Corporation. *Azure AI Vision - Homepage*. eng. 23. Mai 2023. URL: <https://azure.microsoft.com/en-us/products/ai-services/ai-vision> (besucht am 18.02.2024) (siehe S. 4).
- [33] Microsoft Corporation. *Azure AI Vision - Pricing*. eng. 23. Mai 2023. URL: <https://azure.microsoft.com/en-gb/pricing/details/cognitive-services/computer-vision/> (besucht am 18.02.2024) (siehe S. 10).
- [34] Anyline GmbH. *Anyline - Homepage*. eng. 23. Mai 2023. URL: <https://anyline.com> (besucht am 18.02.2024) (siehe S. 4).
- [35] Ing. Punzenberger COPA-DATA GmbH. *COPA-DATA zenon - Homepage*. eng. 23. Mai 2023. URL: <https://www.copadata.com/en/product/zenon-software-platform-for-industrial-automation-energy-automation/> (besucht am 18.02.2024) (siehe S. 1).
- [36] Google LLC. *Google Cloud Vision - Homepage*. eng. 23. Mai 2023. URL: <https://cloud.google.com/vision> (besucht am 18.02.2024) (siehe S. 4).
- [37] Google LLC. *Google Cloud Vision - Pricing*. eng. 23. Mai 2023. URL: <https://cloud.google.com/vision/pricing> (besucht am 18.02.2024) (siehe S. 10).
- [38] ImageMagick Studio LLC. *ImageMagick Homepage*. eng. 23. Mai 2023. URL: <http://www.imagemagick.org/> (besucht am 18.02.2024) (siehe S. 10, 25).

- [39] Iron Software LLC. *IronOCR for .NET - Homepage*. eng. 23. Mai 2023. URL: <https://ironsoftware.com/csharp/ocr/> (besucht am 18.02.2024) (siehe S. 4).
- [40] *Magick.NET - NuGet*. eng. 23. Mai 2023. URL: <https://www.nuget.org/packages/Magick.NET.Core> (besucht am 18.02.2024) (siehe S. 23).
- [41] *Tesseract - NuGet*. eng. 23. Mai 2023. URL: <https://www.nuget.org/packages/Tesseract> (besucht am 18.02.2024) (siehe S. 23).
- [42] *Tesseract Documentation*. eng. 23. Mai 2023. URL: <https://tesseract-ocr.github.io/> (besucht am 18.02.2024) (siehe S. 4, 5, 10, 11).
- [43] tesseract-ocr. *Tesseract Repository*. eng. 23. Mai 2023. URL: <https://github.com/tesseract-ocr/tesseract> (besucht am 04.01.2024) (siehe S. 4).

## Medien

- [44] Wikimedia Commons. *Example of a histogram exhibiting bimodality*. 2014. URL: <https://commons.wikimedia.org/wiki/File:Bimodal-histogram.png> (besucht am 18.02.2024) (siehe S. 17).
- [45] Wikimedia Commons. *Histogram of tips given in a restaurant*. 2014. URL: <https://commons.wikimedia.org/wiki/File:Tips-histogram1.png> (besucht am 18.02.2024) (siehe S. 15).