Technische Hochschule Ingolstadt

ZD AUTOMOTIVE
Focus on Technology Shaping the Future

Konzeption und Implementierung eines auf maschinellem Lernen basierenden Algorithmus zur Verbesserung der Effizienz der Testautomatisierung für automotive Infotainmentsysteme

Abschlussarbeit zur Erlangung des akademischen Grades

Bachelor of Engineering

vorgelegt von
Chuxuan Li

Studiengang: Fahrzeugtechnik
Fakultät: Maschinenbau

Erstprüfer	Prof. Dr.-Ing. Daniel Großmann
Zweitprüfer	Prof. Dr.-Ing. Markus Bregulla
Betreuer	Dipl.-Ing. Qiang Zhou
Ausgabedatum	19.12.2017
Abgabedatum	28.02.2018

Sperrvermerk

Die vorliegende Arbeit „Konzeption und Implementierung eines auf maschinellem Lernen basierenden Algorithmus zur Verbesserung der Effizienz der Testautomatisierung für automotive Infotainmentsysteme“ beinhaltet interne vertrauliche Informationen der ZD Automotive GmbH. Die Weitergabe des Inhalts der Arbeit im Gesamten oder in Teilen sowie das Anfertigen von Kopien oder Abschriften – auch in digitaler Form – sind grundsätzlich untersagt. Ausnahmen bedürfen der schriftlichen Genehmigung der ZD Automotive GmbH.

Erklärung

Ich erkläre hiermit, dass ich die Arbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen oder Hilfsmittel benützt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Ingolstadt, 28. Februar 2018

(Unterschrift)
Vorname Name

Kurzfassung

Heutzutage spielt das Infotainmentsystem eine immer wichtigere Rolle beim Informationsaustausch zwischen Fahrzeug und der Außenwelt. Wegen der ansteigenden Komplexität und Interaktion von Infotainment-Funktionen im Fahrzeug ist der Absicherungsaufwand des Infotainmentsystems stetig größer geworden. Eine Automatisierung der Absicherungsprozesse kann dazu beitragen, die Testeffizienz zu verbessern.

Im Rahmen dieser Arbeit wird ein Algorithmus entwickelt und implementiert, um die automatisierte Absicherung der chinesischen HMI-Textanzeige (HMI: Human Machine Interface) zu realisieren. Eine Literaturrecherche zu den Bilderkennungstechniken im Kontext mit ‚künstlicher Intelligenz' und ‚Digitalisierung' wurde vor der Konzeption durchgeführt. Die Bildverarbeitung gestattet es, die Textbereiche in den zu absichernden Screenshots automatisch zu lokalisieren. Auf dieser Basis werden die Texte in Screenshots erkannt. Da das existierende Texterkennungsmodul für Chinesisch zu langsam ist, wird ein neues Erkennungsmodul mithilfe von maschinellem Lernen spezifisch für das Infotainmentsystem trainiert. Ein Vergleich zwischen den erkannten Texten und Soll-Informationen wird durchgeführt und das Vergleichsergebnis wird anschließend dargestellt. Als Ergebnis ist die Erkennungsgeschwindigkeit des Erkennungsmoduls deutlich erhöht. Gleichzeitig behaltet die Erkennungsgenauigkeit weiterhin auf hohem Niveau.

Inhaltsverzeichnis

Sperrvermerk i

Erklärung ii

Kurzfassung iii

1 Einführung 1
- 1.1 Motivation . . . 1
- 1.2 Konkretisierung der Aufgabenstellung . . . 1
- 1.3 Methodisches Vorgehen . . . 2

2 Stand der Technik 3
- 2.1 Hintergrund künstliche Intelligenz . . . 3
  - 2.1.1 Definition und Geschichte . . . 3
  - 2.1.2 Typische Bestandteile der künstlichen Intelligenz . . . 4
  - 2.1.3 Anwendungsgebiete der künstlichen Intelligenz . . . 5
- 2.2 Digitalisierung . . . 6
  - 2.2.1 Definition und Entwicklungsgeschichte . . . 6
  - 2.2.2 Automobile Digitalisierung . . . 6
  - 2.2.3 Infotainmentsystem . . . 7
- 2.3 Texterkennung (OCR) . . . 8
  - 2.3.1 Methoden und Prinzipien des OCR-Systems . . . 8
  - 2.3.2 Tesseract . . . 10

3 Maschinelles Lernen 11
- 3.1 Definition und Entwicklung . . . 11
- 3.2 Grundansätze des maschinellen Lernens . . . 12
  - 3.2.1 Überwachtes Lernen (supervised learning) . . . 12
  - 3.2.2 Unüberwachtes Lernen (unsupervised learning) . . . 12
  - 3.2.3 Verstärkendes Lernen (reinforcement learning) . . . 13
- 3.3 Klassifikation der Methoden des maschinellen Lernens und der zugehörigen Modellen . . . 13
  - 3.3.1 Regression . . . 13
  - 3.3.2 Klassifikation . . . 14
  - 3.3.3 Clustering . . . 15
- 3.4 Aktuelle konsolidierte Software-Frameworks für machinelles Lernen . . . 15
  - 3.4.1 Scikit-Learn . . . 15
  - 3.4.2 TensorFlow . . . 16
  - 3.4.3 OpenCV . . . 16

4 Entwicklung eines Algorithmus zur Bilderkennung und Fehlererkennung 17
- 4.1 Anforderungsanalyse . . . 17
- 4.2 Analyse der HMI-Screenshots . . . 18

4.2.1 Funktionsweise automatischer Screenshots . . . 19
4.2.2 Komponenten der Screenshots . . . 20
4.3 Vorstellung des Grundkonzepts . . . 20
4.4 Bilderkennung . . . 22
4.4.1 Konzeption Verfahren der Bildverarbeitung . . . 22
4.4.2 Konzeption Erstellung des Texterkennungsmoduls . . . 23
4.5 Fehlererkennung . . . 24

5 Implementierung und Validierung des Algorithmus zur HMI-Absicherung 26
5.1 Durchführung der Bildverarbeitung . . . 26
5.2 Training des OCR Erkennungsmoduls . . . 29
5.3 Programmierung . . . 31
5.4 Validierung des Algorithmus zur automatischen Fehlererkennung . . . 35
5.5 Auswertung der Ergebnisse . . . 37

6 Zusammenfassung und Ausblick 39

Literatur 41

Abbildungsverzeichnis

2.1 Entwicklung der Technologien Spracherkkennung und Verstehen natürlicher Sprache[8] . 6
2.2 vier industrielle Revolutionen - grobe sachliche und zeitliche Eingrenzung [12] 7
2.3 Beispiel Anzeigeelement [1] 8
2.4 Beispiel Bedienelement [2] 8
2.5 Tesseract-Struktur für Latein [36] 10

3.1 Vom Algorithmus zum Use Case - Eine Definition [5] 11

4.1 Anforderungsliste 18
4.2 Screenshot: Status A 19
4.3 Screenshot: Status B 19
4.4 Screenshot: Status C 19
4.5 Darstellung des Grundkonzepts 21
4.6 Bildverarbeitungsverfahren 23
4.7 Drei Situationen der Texterkennungsergebnisse 24
4.8 Abläufe der Fehlererkennung 25

5.1 Visualisierung der Binarisierung (Schwellenwert: 220) 27
5.2 Visualisierung der Binarisierung (Schwellenwert: 50) 27
5.3 Screenshot nach Erosion 28
5.4 Kontur der Erosionsbereichen finden 28
5.5 Farbige Visualisierung der Konturen 29
5.6 Visualisierung der TIF/BOX-Dateipaar in Software jTessBoxEditor 30
5.7 Format der Merkmalen eines Schriftzeichens[33] 31
5.8 Fehlermeldung beim Training des Erkennungsmoduls 31
5.9 Ablaufdiagramm für die Wiederholungsanalyse der Screenshots 32
5.10 Ablaufdiagramm für die Fehlererkennung (Ist-Texte als Referenz) 34
5.11 Ablaufdiagramm für die Fehlererkennung (Soll-Texte als Referenz) 35
5.12 Beispielanzeige für Fehlererkennungsergebnisse 36
5.13 Ergebnisinformationen bei der Praxistests 37

1 Einführung

1.1 Motivation

In den letzten Jahren haben sich die Kernkompetenzen der Automobilbranche sich von der Motoren- und Getriebetechnik hin zur Digitalisierung und Mobilität sowie dem autonomen Fahren gewandelt. In diesem Hintergrund wird das Infotainmentsystem im Auto als eine wichtige Schnittstelle zwischen Fahrzeug und digitaler Außenwelt immer mehr an Bedeutung gewinnen. Aufgrund der rasanten Entwicklungen von Infotainmentsystem ist das Infotainment fähig, die Interaktion zwischen Fahrzeug und Fahrer oder Passagier optimal zu unterstützen. Dabei werden zahlreiche neue Funktionen in die einzelnen Infotainmentsysteme integriert.

Jedoch stellt die ansteigende Komplexität und Interaktion von Infotainment-Funktionen im Fahrzeug für die Entwickler eine große Herausforderung dar. Zum einen muss eine hohe Qualität des Systems gewährleistet sein, zum anderen soll die Entwicklung der Software immer effizienter werden. Um den immer größer werdenden Absicherungsaufwand zu decken und die Testeffizienz zu erhöhen, müssen Softwareabsicherungen möglichst automatisiert werden. Hierfür wurde eine Testtool-Software von ZD Automotive GmbH entwickelt, mit deren Hilfe einzelne Testaufgaben wie zum Beispiel die HMI-Absicherung automatisch durchgeführt werden sollen. Diese Automatisierung soll mithilfe von maschinellem Lernen und Künstlichen Intelligenz realisiert werden. Um die Testeffizienz zu erhöhen, soll in dieser Arbeit eine automatisierte Fehlererkennungsfunktion für die HMI-Absicherung entwickelt werden.

1.2 Konkretisierung der Aufgabenstellung

Die HMI-Absicherung ist ein hauptsächlicher Bestandteil der Absicherung des Infotainmentsystemtests. Eine dessen Hauptaufgaben ist die Überprüfung der HMI-Textanzeigen. Das ursprüngliche Testverfahren erfolgte durch manuellen Vergleich zwischen den angegebenen Soll-Texten, die in XML oder Excel-Dateien gespeichert werden und den tatsächlich auf dem HMI-Bildschirm angezeigten Texten. Wenn ein Fehler erkannt ist, sollen die betroffenen Texte von Tester für weitere Fehlerkorrekturen manuell protokolliert werden. Danach wird eine Software entwickelt. Diese Software extrahiert die Soll-Texte aus der XML-Datei, fügt die erhaltenden Texte mit dem entsprechenden Screenshot des Bildschirms zusammen und lässt die beide Teile synchron anzeigen. Dadurch wird die Zeit für die Zusammenstellung von Soll- und Ist-Informationen gespart und ein klares Mapping garantiert. Aber bei der Überprüfung von Texten ist noch menschlicher Einsatz notwendig. Da nach jeder Softwareveränderung des Infotainmentsystems erneut die HMI abgesichert werden soll, ist es sehr zeit- und kostenaufwendig diese Arbeit zu wiederholen.

Um den Fehlererkennungsprozess zu automatisieren soll in dieser Arbeit ein Algorithmus entwickelt und implementiert werden. Dieser Algorithmus soll die Textbereiche in jedem Screenshot automatisch lokalisieren, die Texte erkennen und auf ihre Richtigkeit überprüfen. Zur Realisierung der Funktion soll die Technik ‚Texterkennung' verwendet werden. Da das existierende Texterkennungsmodul für Chinesisch trotz einer relativ hohen Genauigkeit sehr zeitaufwendig ist, ist es notwendig, die Erkennungseffizienz

zu erhöhen. Da die Soll-Texte in dem Infotainmentsystem bereits bekannt sind, wird hierbei ein Texterkennungsmodul mithilfe des maschinellen Lernens trainiert, das spezifisch zu dem Infotainmentsystem passt.

1.3 Methodisches Vorgehen

Um die Aufgaben sorgfältig und systematisch bearbeiten zu können, wird diese Arbeit in vier Arbeitspakete aufgeteilt:

- Vorstudie zum Stand der Technik in Themen ‚künstliche Intelligenz' und ‚Digitalisierung'
- Literaturrecherche zu den aktuellen Methoden von ‚maschinellem Lernen' und Klassifikation
- Entwicklung eines Algorithmus zur Fehlererkennung von Texten auf der HMI-Anzeige
- Implementierung der entwickelten Algorithmen und entsprechende Auswertung

Im Rahmen dieser Arbeit sollen zuerst durch eine umfassende Literaturrecherche die wichtigsten Entwicklungen und Trends im Bereich künstliche Intelligenz und Digitalisierung herausgearbeitet werden. Aufgrund des Bedarfs an Einsatz von Bildverarbeitungsverfahren werden Methoden zur Bildverarbeitungen ebenfalls recherchiert. Anschließend werden Recherchen zu maschinellem Lernen und dessen Klassifikation sowie eine Analyse von den Methoden durchgeführt. Die Literaturrecherche wird hauptsächlich im Internet unternommen. Mithilfe von wissenschaftlichen Suchmaschinen wie Google Scholar und Datenbanken wie IEEE Xplore Digital Library werden die veröffentlichen Thesen und Artikel gefunden. Als weitere zusätzliche Informationsquellen werden Nachschlagewerke und Lehrbücher von der Bibliothek verwendet. Die Literaturen werden je nach Schlagwort und Inhalt zu Gliederungspunkten geordnet, um die Arbeit und das Schreiben anzupassen.

Weiterhin wird in dieser Arbeit ein Algorithmus zur Bilderkennung entworfen. Die in den Bildern beinhaltenden Informationen (Textbereiche, Koordinaten, usw.) werden durch das Bildverarbeitungstool OpenCV extrahiert und gespeichert. Demnach wird das OCR(Optical Charactor Recognition)-Engine Tesseract zur Texterkennung jedes Textbereichs verwendet. Als Trainingsdaten werden die Soll-Texte benutzt. Mittels maschinellem Lernen wird ein systemspezifisches Erkennungsmodul in Tesseract erzeugt und trainiert. Weiterhin wird ein Algorithmus zur Fehlererkennung der Textanzeige entwickelt.

Abschließend wird das entwickelte Algorithmus implementiert. Die Funktionsfähigkeit der Implementierung wird durch Praxisversuche und deren Auswertung evaluiert.

2 Stand der Technik

Die Diskussion über den Stand der Technik erfordert die Untersuchung von **künstliche Intelligenzsystemen** und deren modernen Anwendungen im Kontext des **Digitalisierungsdebatte**.

2.1 Hintergrund künstliche Intelligenz

2.1.1 Definition und Geschichte

Künstliche Intelligenz (KI), auch Artificial Intelligence(AI) genannt, ist ein Teilgebiet der Informatik, die sich mit anderen Disziplinen wie der Statistik, der Linguistik, der Psychologie und der Neurowissenschaften integriert.[10] Das Ziel dieses Fachgebiets ist es, "menschliche Erkennungs- und Denkprozesse zu formalisieren und einem Rechner zu übertragen"[22].

Der Begriff ‚Künstliche Intelligenz' wurde ursprünglich von John McCarthy, Marvin Minsky und Claude Shannon in einem Konferenz während Sommer 1956 am Dartmouth College in Hanover, New Hampshire in die Welt gerufen. Nach der Vorstellung des Begriffs erwartete man zuerst, dass die KI allgemeine Aufgaben lösen kann. Bald stellte sich heraus, dass die Komplexität und Schwierigkeit der Realisierung der Funktion sehr hoch ist. Danach wurden in den siebziger Jahren die Ziele der KI konkretisiert. Sie beschränkte sich auf bestimmte Aufgaben, die basierend auf den aus gespeicherten Fachwissen gelernten Regeln lösbar sind. Diese Entwicklungsphase wurde durch ‚Knowledge-Based Approach'[22] bezeichnet. Wegen dieser Veränderung wurde bei Techniken der Wissensrepräsentation und in der Systemarchitecktur große Fortschritte erzielt. Daher wurde die künstliche Intelligenz einfacher zu applizieren. Wegen der Beschränkung der Rechenkomplexität, unerfüllbarer Rechnerperformance und ehlenden Daten, können viele Probleme wie beispielsweise maschinelles Sehen oder Bildverstehen, jedoch noch nicht gelöst werden. Solche Vorgänge sind zwar für die Menschen unkompliziert, aber in der Maschine sehr schwer nachzubilden. Heutzutage stellen bessere Rechenfähigkeiten, komplexere mathematischen Werkzeuge und fortgeschrittene Algorithmen wie Deep Learning für künstliche Intelligenz zur Verfügung. KI ist in der Lage, immer mehr Aufgaben zu lösen und die Effizienz der Aufgabenlösung stets zu erhöhen.

2.1.2 Typische Bestandteile der künstlichen Intelligenz

Typische Bestandteile der künstlichen Intelligenz, die heute diskutiert werden, sind:

- Big Data
- Deep Learning
- Cloud Computing

Big Data

Big Data sind Datensätze, die nicht mehr mit manuellen Methoden oder auf einem einzelnen Rechner verarbeitet werden[7]. Big Data ist ein Grundstein zur Realisierung der KI, weil die heutige Denkweise beim Umgang mit KI-Problemen die Umwandlung der intelligenten Fragen in den datenbasierten Aufgaben ist. Mit der Anhäufung von informativen Daten und Entwicklung von Rechenleistung wird die Performance des Systems stets verbessert.

Deep Learning

Deep Learning ist eine Teilmenge des maschinellen Lernens. Basierend von traditionellen Methoden des maschinellen Lernens wird Deep Learning entwickelt. Es ist eine Unterklasse von intelligenten neuronalen Netzen, in den die Datenverarbeitungsprozesse mit mehreren tiefen Zwischenlagen zwischen Eingabeschicht und Ausgabeschicht formuliert wird. Mit diesen hierarchischen Schichten werden die Prozesse des maschinellen Lernens durchgeführt. Die Begriffe im Bezug auf ‚Maschinelles Lernen' werden in Kapitel 3 weiter erklärt.

Wie im Abschnitt 2.1.1 erläutert wird, geht es in derzeitigen KI Studien hauptsächlich um Lösungen für Probleme, die Menschen intuitiv lösen können. Deep Learning zeigt bis jetzt gute Performance bei der Behandlung solcher Probleme. Es ermöglicht einen Rechner aus Erfahrungen zu lernen und Konzepte der Umgebung im Sinne einer Hierarchie zu bilden. Durch das Sammeln von Wissen aus Erfahrung ist die Implementierung des gesamten Wissens durch Menschen in dem Rechner nicht mehr notwendig. [11]

Cloud Computing

Laut der Definition von NIST National Institute of Standards and Technology ist Cloud Computing[23]:

"a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

Die Verarbeitung von Big Data erfordert enorme Rechenleistung. Das Deep Learning verlangt kein vollständiges Verständnis von den Verarbeitungsprozessen des Rechners. Bei dem Aufbau eines Deep Learning Modells wird ein mehrschichtiges neuronales Netzwerk gebildet. Hierbei müssen die Anzahl der Schichten und künstlicher Neuronen jeder Schicht festgelegt werden. Es müssen in sehr vielen Fällen Versuche stattfinden. Dieser Vorgang erfordert viel Zeit und Rechenleistung um die Parameter anzupassen. In diesen Hinblick spielt die Cloud-Computing-Performance eine wichtige Rolle bei KI. Je

stärker das Cloud Computing-Performance ist, desto mehr Versuche können in einem gleichen Zeitraum durchgeführt werden. Durch mehr Versuche werden mehr Erfahrungen gesammelt und mehr Wissen erlangt.

Die oben genannten drei Begriffe: Big Data, Deep Learning und Cloud Computing sind wichtige Bestandteile bei der Realisierung des Künstlichen Intelligenz. Aus der Bearbeitung der Big Data werden nützliche und relevante Daten extrahiert. Mit der Deep Learning Methode erfasst der Rechner zuerst diese bearbeiteten Daten und entwickelt daraus die unterliegenden Regeln. Schließlich wird die Aufgabe mit dieser Regel behandelt. Eine hohe Cloud Computing Performance beschleunigt diesen Prozess und steigert die rechnerische Effizienz.

2.1.3 Anwendungsgebiete der künstlichen Intelligenz

Das Forschungsgebiet der künstlichen Intelligenz kann aufgrund deren unterschiedlichen Lösungsansätze in mehreren Teilgebieten untergegliedert werden[22], z.B.:

- Bildverarbeitung und Bilderkennung
- Verarbeitung natürlicher Sprache
- Lernen
- Expertensysteme
- Robotik
- KI-Hardware und KI-Software

In dieser Arbeit wird eine HMI-Absicherungsfunktion entwickelt, deswegen werden hierbei hauptsächlich die Bildverarbeitung und Bilderkennung berücksichtigt.

Die **Bildverarbeitung und Bilderkennung** hat sich in vielen Bereichen zu einer Schlüsseltechnologie entwickelt. Bei den herkömmlichen Bildverarbeitung- und Bilderkennungsprozessen werden maschinelles Lernen bereits angewendet. Mit Einsatz von Deep Learning wird es möglich, ständiges Lernen und Echtzeitanalysen durchzuführen. Die Performance vieler Funktionen wird dadurch auch erhöht, z.B.: Car-Barrier-Erkennung, Gesichtsanalyse, Personalverfolgung, Texterkennung usw. Bei der Anwendung heutiger künstlicher Intelligenz sind beide Modi sehr wichtig.

Aber wie oben erwähnt, ist Deep Learing eine datenangetriebene Methode. Beispielsweise werden bei der Bildklassifizierung Millionen Trainingsbeispiele benötigt um ein zufriedenes Modell mit passenden Parametern zu erstellen. Wenn keine große Menge von markierten Trainingsdaten zur Verfügung stehen, können sich keine guten Ergebnisse bei Deep Learning ergeben. Nach der Aufgabeanalyse werden in dieser Arbeit traditionelle Methoden von Bildverarbeitung und Bilderkennung verwendet, um die Bildqualität optimal zu verbessern und die erwarteten Ergebnisse zu bekommen. Die Methoden zur Bildverarbeitung werden in Abschnitt 2.3 weiter vorgestellt.

Außerdem ist **Spracherkennung und Verstehen natürlicher Sprache** eine der reifsten Anwendungen von Künstlicher Intelligenz. Zu dem gehört die automatische Übersetzung von Sprache, Erkennung von gesprochenen Sprachen. Dieses Teilgebiet spielt eine wichtige Rolle. Vor allem bei den natürlichsprachigen Schnittstellen zu Informationssystemen. Google hat seine Sprachtechnologie in den Anwendungen wie Eingabemethode, Sprachsteuerung, Übersetzung und Websuche integriert. Andere Anwendungsbeispiele können aus Abbildung 2.1 ersehen werden.

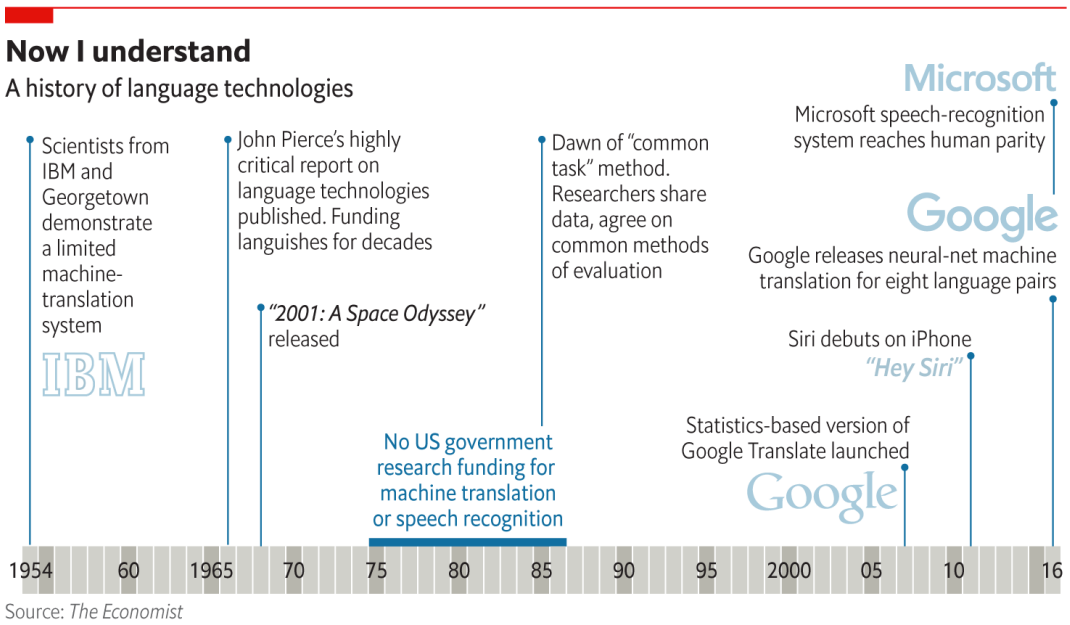

Abbildung 2.1: Entwicklung der Technologien Spracherkkennung und Verstehen natürlicher Sprache[8]

2.2 Digitalisierung

Nach Einschätzung wurden im Jahr 1986 nur ungefähr 1% der Daten in der Welt digital gespeichert[13]. Der digitale Anteil der gespeicherten Daten steigert rasant und erreicht im Jahr 2016 bereits 98%[13]. Viele Bereiche wie zum Beispiel Industrie, Medizin, Wirtschaft und Politik erlebten eine rasante Veränderung. Die Entwicklung der Digitalisierung von Daten erhöht die gesamte Datenmenge und steigert die Produktivität der Gesellschaft.

2.2.1 Definition und Entwicklungsgeschichte

Die Bedeutung von ‚Digitalisierung' wird im laufe der Zeit ständig erweitert. Im ursprünglichen Sinne bedeutete dieser Begriff die Digitalisierung von Daten, die Übermittlung von physischem Objekten oder analogen Medien in digitalen Daten, die mit nur wenigen festen Werten (z.B. Binärsystem) dargestellt werden.

Die weitere Entwicklung der Digitalisierung erfolgte während der 3. industriellen Revolution (siehe Abbildung 2.2). Die Digitalisierung beinhaltet einen Wandel zu digitalen Prozessen mittels Informations- und Kommunikationstechnik (information and communication technology, ICT) im Kontext der ständig steigenden digitalen Datenmenge[16]. Der Prozess der Digitalisierung vollzieht sich noch rasant. Digitale Informationen und Daten werden in elektronischen Datenverarbeitungssysteme gespeichert, verarbeitet und verteilt, die echte Welt wird durch digitalen Datenaustausch vernetzt. Laut der Prognose von Ericsson am Ende 2015 wird bis 2020 weltweit mehr als 90% der Menschen durch Mobilfunknetz abgedeckt.

2.2.2 Automobile Digitalisierung

Heutzutage ist die Digitalisierung im automobilen Bereich unumgänglich. Der Anteil der Kosten in Elektronik und Software eines Autos beträgt derzeit 35%[14]. Die elektronischen Systeme ermöglichen mehr

Ereignis	Beginn	Erläuterung der sachlichen Errungenschaften
Erste Industrielle Revolution	1760 ff., Schwerpunkt 19. Jahrhundert	Dampfmaschinen ersetzen in vielen Industrien bei schwersten Arbeiten die blanke Muskelkraft
Zweite Industrielle Revolution	1870 ff., Schwerpunkt ca. 1900 bis 1970	Eisenbahn, Fließbänder, Gas- und Wasserversorgung, Telefonie, Schreibmaschinen, Autos und vor allem die flächig verfügbar gemachte Elektrizität verbessern die Lebensbedingungen und Produktion dramatisch
Dritte Industrielle Revolution, „digitale Revolution“	1950, Schwerpunkt ab 1970	Vermehrt Elektronik und Digitaltechnik in Einzelgeräten, erste Computer im Masseneinsatz von Wirtschaft und Verwaltung, Verbesserung von Abläufen
Vierte Industrielle Revolution	ca. 2000 f.	Über das Internet medienbruchfreie Kommunikation zwischen Menschen, Behörden und Unternehmen möglich, cyber-physische Systeme vernetzen Maschinen in Produktionswirtschaft, Haushalt und im mobilen Einsatz

Abbildung 2.2: vier industrielle Revolutionen - grobe sachliche und zeitliche Eingrenzung [12]

als 90% der Innovationen und neue Funktionen im Auto, z. B.: autonomes Parken, Spurhalteassistent und sensorbasiertes Reporting[14]. Darüber Hinaus sind die Informatik- und Kommunikationstechnik von wesentlicher Bedeutung für eine effizientere Automatisierung innerhalb der Fertigungswerken - sowohl in den Produkten als auch in der Produktion. Das nennt man die Digitalisierung in der Automobilindustrie.[17]

2.2.3 Infotainmentsystem

Der Begriff Infotainment besteht aus ‚Information‘ und ‚Entertainment‘. Von dem Namen kann man die Aufgaben des Infotainmentsystems ersehen: Informationen übersichtlich darzustellen und Unterhaltungsfunktionen zu ermöglichen. Die Grundfunktionen der Infotainmentsysteme umfassen Autoradio, CD-Player, Navigationssystem, Anzeige der Verkehrssituation, Fahrerassistenzsystem usw. Darüber hinaus werden individualisierte Funktionen wie Konfiguration des Fahrzeugs ins Infotainment integriert.[3] Das Infotainmentsystem ist eine wichtige digitale Schnittstelle nicht nur zwischen Fahrzeug und Fahrer oder Passagiere, sondern als auch zwischen Fahrzeug und Außenwelt. Der digitale Informationsaustausch wird durch Fahrzeugvernetzung realisiert, z.B. aus sozialen Netzwerken und Cloud-Diensten ins Fahrzeug und sogar ins Handy[30].

Die Komponenten des Infotainmentsystems sind: Haupteinheit, Bedienungselemente(2.4) und Anzeigeelemente (Abbildung 2.3). Die Menschen erhalten die Informationen über Anzeigeelemente und steuern das System durch Bedienungselemente. Bei der HMI-Absicherung werden mithilfe von Testtools das Menschenverhalten simuliert und die Anzeigeinformationen des Bildschirms gespeichert.

Abbildung 2.3: Beispiel Anzeigeelement [1]

Abbildung 2.4: Beispiel Bedienelement [2]

2.3 Texterkennung (OCR)

Texterkennung, auch OCR(englische Abkürzung von *Optical Character Recognition*), ist ein Teilgebiet der Informatik, die sich mit den Prozessen der automatischen Analyse und Erkennung von Texten und Layout-Informationen in Bilddateien beschäftigt. Je nach Art des Textes wird die OCR-Technik in gedruckte Texterkennung und handschriftliche Texterkennung unterteilt. Die Merkmale beider Textarten sind sehr unterschiedlich, deswegen sind die Behandlungsmethoden für zwei Texterkennungsarten nicht identisch. Da diese Arbeit sich nur mit Screenshots aus HMI-System mit gedruckten Texten befasst, wird hauptsächlich die Vorgehensweise der gedruckten Texterkennung untersucht.

2.3.1 Methoden und Prinzipien des OCR-Systems

Ein typischer Texterkennungsprozess läuft wie folgt ab:

- Bildaufnahme
- Bildvorverarbeitung
- Mustererkennung
- Ergebnisausgabe

Bildaufnahme

Bei der Bildaufnahme werden die Pixelbilder als Datenquellen in das Erkennungssystem eingelesen. edem Pixel im Bild wird eine bestimmte Position und ein Farbwert zugewiesen. Die Bildgröße, nämlich die Breite und Höhe der Bilder, werden in Pixel gemessen.[6] Die Farbinformationen jedes Pixels werden mit RGB-Primärfarben oder Grauwert dargestellt.

Bildvorverarbeitung

Zur Optimierung des Ergebnisses der Texterkennung soll ein eingegebenes Pixelbild zuerst vorverarbeitet werden, anschließend werden Binarisierung, Rauschunterdrückung, Neigungskorrektur, Layout-Analyse usw. durchgeführt.

Die Pixel von den meisten Bildern sind mit den drei Primärfarben (Rot, Grün und Blau) kodiert. Nach Binarisierung der Bilder werden die Pixel nur mit zwei Farben Schwarz und Weiß (1 oder 0) kodiert. Solche Bilder werden als Binärbilder bezeichnet.[6] Mithilfe von Binarisierung kann die Bildinformationen reduziert und die zu speichernde und verarbeitende Datenmenge gesenkt werden. In Texterkennungsprogrammen kann Binarisierung als erster Schritt eingesetzt werden, um die relevanten Bereiche von dem Hintergrund zu trennen[15].

Aufgrund von Wanken und Erschütterung bei der Aufnahme sind die Bilder oft mit ungewollter Neigung belastet. Diese Bildneigung muss vor der Mustererkennung korrigiert werden. Ansonsten werden die Erkennungsergebnisse von der Realität abweichen, z. B.: ein um 90 Grad nach links geneigtes “E” kann als “W” betrachtet werden.

Ein anderer wichtiger Prozess bei der Vorverarbeitung ist die Layout-Analyse der Bilder. Die relevanten Bereiche wie der Textbereich werden hierbei extrahiert.[26]

Die Phase der Bildvorverarbeitung ist für die Texterkennung sehr wichtig, da die Erkennungsgenauigkeit davon beeinflusst wird.

Mustererkennung

Mustererkennung ist die Kerntechnik der Texterkennung. Die Hauptprozesse sind Text-Feature-Extraktion und Klassifikation.

Die häufig verwendete Methode der Feature-Extraktion basieren auf die Analyse von Merkmalspunkten, Verteilung der Pixel, topologische Merkmale oder polygonale Annäherung[35][44]. Bei der Texterkennung soll eine Datenbank dieser Sprache vorher vorhanden sein. Diese Datenbank umfasst die kleinsten sprachlichen Elemente wie Buchstaben in lateinischen Sprachen oder chinesischen Schriftzeichen und dessen zugehörigen Merkmalen, die der Klassifikation dienen.

Eine Klassifikation von Texten wird zuerst auf der Buchstaben- und Zeichenebene durchgeführt. Mit bestimmtem Algorithmus werden die Pixel jedes Zeichens mit Mustern in der Datenbank verglichen[25].

Mit Hilfe von modifizierten Wörterbüchern, welche die häufig verwendeten Wörter oder Zeichen umfassen, werden die Klassifikationsergebnisse auf der Wortebene weiter optimiert. In diesem Teilprozess wird die Wortsegmentierung durchgeführt. Die erkannten Buchstaben oder Zeichen aus dem ersten Teilprozess werden zu Wörter zusammengesetzt und mit den Zeichenkombinationen von Wörterbüchern verglichen. Damit kann die Erkennungsrate und Richtigkeit des Systems verbessert werden.

Ergebnisse der Texterkennung werden je nach Anforderung in Form von Textdateien (TXT oder XML-Datei), oder in einer Datenbank, oder direkt auf den Quellenbilder dokumentiert.

2.3.2 Tesseract

Tesseract ist eine Open-Source-OCR-Engine zur Texterkennung, die zwischen 1984 und 1994 von der Firma Hewlett Packard (HP) entwickelt wurde. Die Software wurde ursprünglich als PhD-Forschungsprojekt in HP Labs Bristol für eine mögliche Software/Hardware Add-On in einer Produktlinie entwickelt.[35] HP veröffentlichte dann in 2005 Tesseract für Open Source Entwicklung und diese wird seit 2006 von Google unterstützt. Abbildung 2.5 zeigt die Funktionsweise der Tesseract für Latein. Die Funktion Layout-Analyse von Tesseract ist jedoch schwach, deswegen ist die Durchführung von einer besonderen Layout-Analyse (Bildverarbeitung) vor dem Anruf von Tesseract benötigt.

Tesseract wird lediglich über die Befehlszeilenschnittstelle ausgeführt und verfügt über keine GUI (GUI: englische Abkürzung für Graphical User Interface). Jedoch gibt es viele Projekte wie OCRFeeder und VietOCR, die an einer GUI für Tesseract entwickeln, Außerdem gibt es ebenso freundliche API wie tesseocr. Heutzutage unterstützt Tesseact die Texterkennung von mehr als 100 Sprachen[42]. Für Chinesisch ist bereits ein Texterkennungsmodul ‚chi_sim' vorhanden. Jedoch ist der Rechenaufwand dieses Moduls wegen der großen Menge von Schriftzeichenmuster in Datenbank im Vergleich zu lateinischen Sprachen sehr hoch. In Folge davon ist die Erkennungsgeschwindigkeit dieses Moduls langsam. Ein Merkmal dieser Software ist, dass sie lernfähig ist. Die Texterkennung lässt sich erweitern, indem man die Datenbank dementsprechend ausbaut. Das bedeutet, dass Tesseract in der Lage ist, ein neues Texterkennungsmodul für eine bestimmten Menge Schriftzeichen oder für eine neue Sprache zu erstellen.

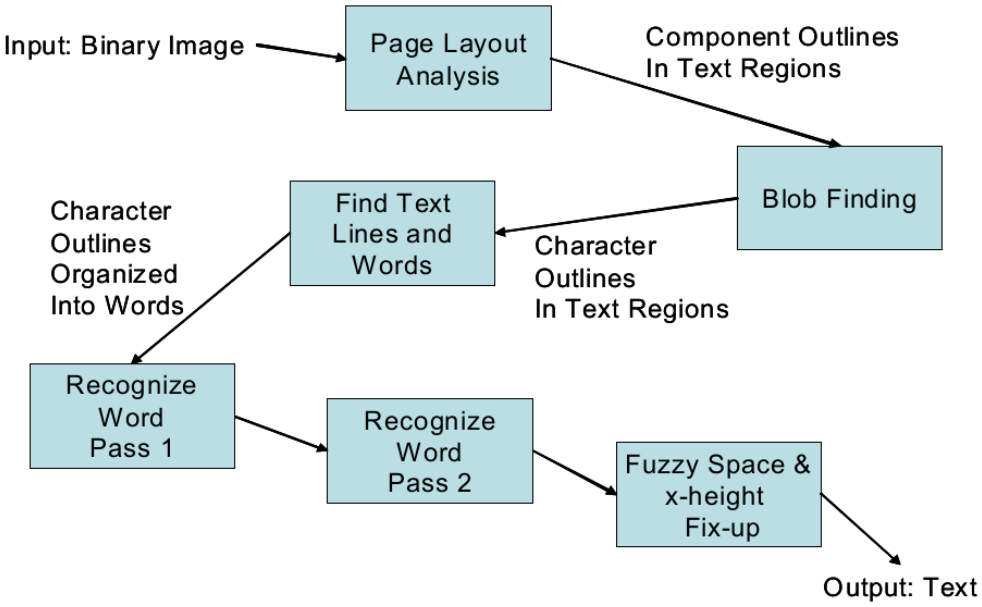

Abbildung 2.5: Tesseract-Struktur für Latein [36]

3 Maschinelles Lernen

In diesem Kapitel wird ein zusammenfassender Überblick über die Begriffe von maschinellem Lernen und dessen Klassifikation gegeben.

3.1 Definition und Entwicklung

Das Begriff maschinelles Lernen, auf Englisch Machine Learning genannt, stammt aus dem Bereich Informatik. Maschinelles Lernen ist eine rechnerische Methode, die es einem Rechner ermöglichen soll, menschliches Lernverhalten nachzubilden[24]. Damit kann der Rechner selbst Wissen und Fähigkeiten aus Erfahrungen lernen und sogar seine Leistung kontinuierlich verbessern. Maschinelles Lernen bietet KI die technischen Methoden zur Realisierung zahlreicher Funktionen. Die praktische Umsetzung von maschinellem Lernen ist im Form von Algorithmen realisiert. Nach Böttcher, Klemm und Velten erklärt Abbildung 3.1 die Beziehung zwischen dem Algorithmus von maschinellem Lernen und deren Anwendung im Kontext mit Künstlicher Intelligenz.

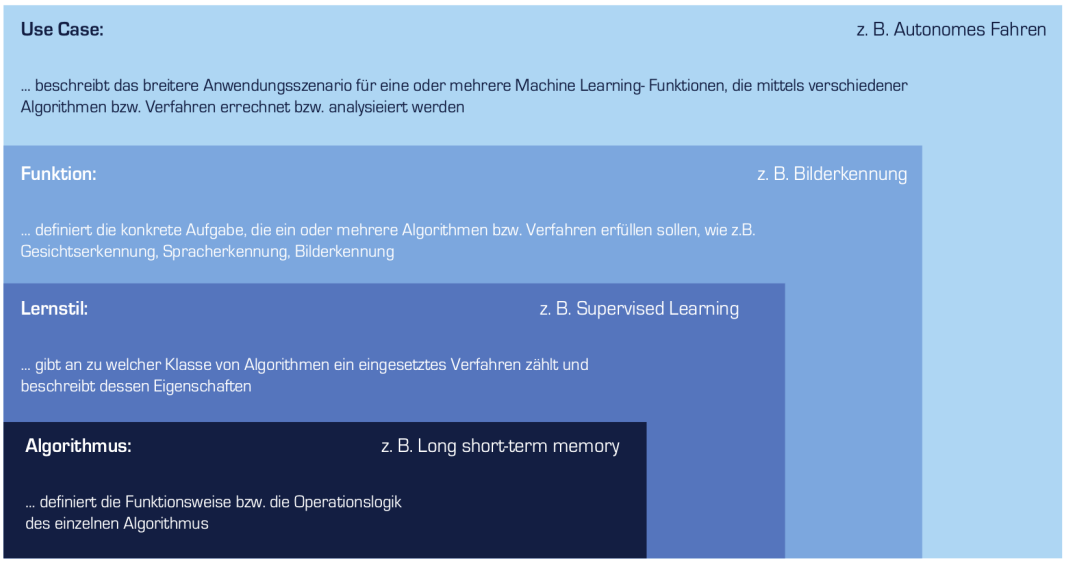

Abbildung 3.1: Vom Algorithmus zum Use Case - Eine Definition [5]

In 1959 stellte Arthur Samuel den Begriff "Machine Learning"bei IBM als Methode zur Realisierung künstlicher Intelligenz auf[18]. Wie das Gebiet der künstlichen Intelligenz seine Ziele in den siebziger Jahren konkretisierte, hat maschinelles Lernen danach sein Fokus auf Methoden und Modellen gerichtet, die auf der Statistik und Wahrscheinlichkeitstheorie basieren[21]. Die Entwicklung von maschinellen Lernen profitiert von der zunehmenden Verfügbarkeit der digitalen Informationen. Wegen der größeren Datenmenge und digitaler Datenform können die Rechner und Maschinen effizientere und genauere Modelle trainieren, damit maschinelles Lernen besser applizierbar ist. Heutzutage ist maschinelles Lernen auf dem Weg in den IT- und Digitalisierung-Mainstream. Schon 64 % Prozent der deutschen Unternehmen

sind mit dem Thema aktiv beschäftigt und 20 % Prozent haben die Machine Learning-Technologien schon produktiv eingesetzt[5].

Zum besseren Verstehen der Funktionsweise des maschinellen Lernens werden hier einige Fachbegriffe vor allem vorgestellt[24]:

Beispiel: Elemente oder Instanzen von Daten, die zum Lernen oder zur Auswertung eines Algorithmus verwendet werden.

Merkmale: Eine oder mehrere Eigenschaften, die das Beispiel repräsentieren.

Label: Werte oder Kategorien, die den Beispielen zugewiesen sind.

Trainingsbeispiel: Beispiele zum Training eines Lernalgorithmus.

Validierungsbeispiel: Beispiele zum Einstellen der Parameter des Lernalgorithmus.

Testbeispiel: Beispiele zur Bewertung der Performance des Lernalgorithmus.

3.2 Grundansätze des maschinellen Lernens

Beim maschinellen Lernen werden eine Menge von Trainingsdaten als Eingaben benutzt, von denen das System eine Regel lernen kann. Je nach Lernstil werden die Algorithmen des maschinellen Lernens in überwachtes Lernen, unüberwachtes Lernen und verstärkendes Lernen unterteilt.

3.2.1 Überwachtes Lernen (supervised learning)

Bei dem Überwachten Lernen werden die Eingabedaten mit Labels markiert. Ein Label entspricht die richtigen Ausgaben[19], die richtigen Maßnahmen, die das System zu der Eingabesituation ergreifen sollte[38], z.B. richtig/falsch zu bestimmen, Kategorien zu identifizieren usw. Der Algorithmus lernt eine Funktion aus gegebenen Paaren von Ein- und Ausgabenwerten. Diese Funktion erstellt die Zusammenhänge zwischen Ein- und Ausgaben, genauer gesagt findet sie die Assoziationen zwischen den Merkmalen der Eingaben und den Ausgaben. Basiert darauf kann man bei künftigen Eingabewerten die entsprechenden Labels vorhersagen.

3.2.2 Unüberwachtes Lernen (unsupervised learning)

Bei dem unüberwachten Lernen sind die Eingabedaten nicht durch Labels markiert. Aus den ungelabelten Daten kann der Algorithmus unbekannte Strukturen ermitteln und Regeln ableiten, damit er bei künftigen Eingabedaten möglichst gleichen Modell repräsentieren kann.

3.2.3 Verstärkendes Lernen (reinforcement learning)

Die Definition nach Sutton und Barto besagt: "Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal."[38] Der Lernprozess basiert auf die Theorie dynamischer Systeme. Der Lerner muss in der Lage sein, den Zustand seiner Umgebung ständig zu spüren und entsprechende Maßnahmen zu ergreifen, um die Umgebung zu beeinflussen[38]. Anders als beim überwachten Lernen, wird beim verstärkenden Lernen der Lerner keine Labels gegeben. Die Unterscheidung zum unüberwachten Lernen ist, dass sich verstärkendes Lernen nicht auf die Ermittlung von den versteckten Regeln oder der Struktur in ungelabelten Daten konzentriert, denn das alleine schafft ein Maximum von ‚reward' nicht[38]. Durch die Interaktion mit der Umgebung soll das Lernsystem im Laufe der Zeit viel versuchen und selbst eine optimale Strategie ermitteln um sein Ziel zu erreichen.

3.3 Klassifikation der Methoden des maschinellen Lernens und der zugehörigen Modellen

Maschinelles Lernen wird jeden Tag auf unterschiedliche Anwendungen angewendet, wie zum Beispiel Sprach- und Texterkennung, Gesichtserkennung, Fahrzeugkontrolle und medizinische Diagnose. Solche Anwendungen entsprechen verschiedenen Lernproblemen. Nach der Lernaufgaben können die Methode des maschinellen Lernens grundsätzlich wie unten klassifiziert werden[24]:

3.3.1 Regression

Das Ziel der Regressionsanalyse ist es, Beziehungen (Funktion f) zwischen einer abhängigen Variable (y) und einer oder mehrerer unabhängigen Variablen (x1, x2, ...) zu modellieren. Basierend auf dem geschätzten Modelle können die neuen Eingabedaten den Ausgabewerte zugeordnet werden. Eine Menge von kontinuierlichen Werte wird als Ausgabewerte dieses Prozesses erwartet. Wenn die Labels und Merkmalen in den Beispielen eine abhängige Beziehung haben, kann die Methode Regressionsanalyse verwendet werden. Je nach Anzahl der unabhängigen Variablen lässt sich eine Regression in einfache Regression und multiple Regression unterteilen. Außerdem können Regressionsanalyse in lineare Regression und nichtlineare Regression unterteilt werden:

- **Lineare Regression** Das Modell der linearen Regression geht davon aus, dass sich die Werte einer abhängigen Variablen durch eine lineare Gleichung aus den Werten der unabhängigen Variablen und zufälligen Fehlern zusammensetzen[4]. Wenn es bei der Regression nur ein unabhängiger Variable gibt, ist die lineare Gleichung eine gerade Linie in der Hyperebene. Alle Abstände von den Punkte (x, y) zu der Linie sollen möglichst klein sein. Auf Basis von linearen Regression werden zahlreiche Algorithmen entwickelt, z.B.: orthogonale Regression, generalisierte lineare Modelle.
- **Nichtlineare Regression** In vielen Fällen ist das lineare Modell nicht in der Lage, die Beziehungen zwischen abhängigen Variable und unabhängigen Variablen zu modellieren, z.B.: Größe und Alter der Menschen, Drehmoment und Zylinderdruck eines Autos usw. Deswegen werden nichtlineare Regression eingeführt. Nach Ruckstuhl wird nichtlineare Regression so definiert: "In der nichtlinearen Regression werden Funktionen untersucht, die sich nicht als lineare Funktionen in den Parametern schreiben lassen."[31] Einige nichtlineare Modelle können durch mathematische Methode in lineare Regression transformiert werden, die andere aber nicht.

3.3.2 Klassifikation

Das Ziel der Klassifikation ist, jedem Eingaben eine Kategorie zuzuweisen. Um die Ergebnisse begrenzten Kategorien zuzuordnen, muss die Ausgabewerte diskret sein. Ein Muster oder Regel wird von den Trainingsbeispielen gelernt, die ein Klassifikationsmodell ermöglicht. Dieses Muster wird dann zur Klassifikation neuer und unklassifizierter Beispiele verwendet. Typische Algorithmen dazu gehören:

- **kNN (k-Nearest-Neighbors)** Das Prinzip des Algorithmus k-Nearest-Neighbors ist es, eine vordefinierte Anzahl (k) von Trainingsbeispiele zu finden, die dem neuen Punkt am nächsten sind, und die Label daraus vorherzusagen. Die Beispiele können als Punkte innerhalb eines n-dimensionalen Raums betrachtet werden, wobei jede Dimension eine Merkmal des Beispiels entspricht[20]. Dadurch werden die Abstände zwischen allen Beispielen messbar sein und können die am nächsten Beispiele durch Berechnung herausgefunden werden. Das ist eine der einfachsten, nicht-parametrischen Klassifikationsmethoden.
- **Entscheidungsbäume (Decision Tree)** Das Ziel der Methode Entscheidungsbäume (engl. Decision Tree) ist die Entscheidungsfindung des Outputs mithilfe eines Baumes, der die verschiedene Inputs (Merkmalswerten von Beispielen) in Klassen aufteilt. Entscheidungsbäume werden auch als Klassifikationsbäume bekannt. Das Merkmal, das die Trainingsbeispiel am besten teilen kann, wird als Wurzelknoten gesehen. Jeder Knoten in einem Entscheidungsbaum entspricht einem Merkmal in einer zu klassifizierenden Beispiel, bei dem eine Entscheidung getroffen wird, um die Beispiele weiter zu nächsten Knoten zuzuweisen.[20] Der Prozess endet sich, wenn alle Beispiele in Ziel-Klasse aufgeteilt werden. Jede Klasse wird als ein Blattknoten bezeichnet.
- **Naive Bayesian classification** Naive Bayes-Klassifikation ist eine Klassifikationsmethode mit überwachtem Lernstil. Sie basierend auf dem Satz von Bayes und der Annahme - die starke Unabhängigkeit zwischen Merkmalen (jedes Merkmal ist unabhängig voneinander). Basierend darauf wird die Verteilungswahrscheinlichkeit zwischen den Ein/Ausgaben erlernt. Durch dieses gelernten Modells werden die neuen Eingaben den Kategorien mit höchsten Wahrscheinlichkeiten zugeordnet. Wegen der Unabhängigkeitsannahme wird dieser Algorithmus "naive"genannt. Sie reduziert die Anzahl der Modelldimensionen und vermeidet übermäßig komplizierte Modelle. Deswegen hat dieses Modell ein Vorteil, dass es lediglich eine kleine Menge von Trainingsdaten benötigt, um die notwendigen Parameter anzupassen[32].
- **Support Vector Machines** Ursprünglich wurden Support Vector Machines als binäre Klassifikatoren entwickelt. Das Ziel der SVM besteht in der Berechnung einer Hyperebene, die so im Merkmalsraum liegt, dass sie die Trainingsdaten der beiden Klassen optimal trennen kann. Die Bezeichnung öptimal"bedeutet, dass die Punkte (Trainingsdaten) im Merkmalsraum durch eine trennende Hyperebene geteilt werden und diese Hyperebene einen möglichst breiten Rand um die Klassengrenzen herum hat[9].
- **Logistische Regression** Das Grundkonzept dieses Algorithmus wird aus der linearen Regression abgeleitet. Jedoch sind die Eingaben hier kategorisch. Der Algorithmus wird nur gut funktionieren, wenn die Daten linear diskret sind (d.h. die Daten können vollständig durch eine Hyperebene getrennt sein). Wenn die Ausgaben nur zwei Werte annehmen ("0ünd "1"), die die Ergebnisse wie "Ja/Nein", "Gewinn/Verlust", "gesund/krank"darstellen, ist es binäre logistische Regression. Wenn die Ausgaben mehr als zwei Ergebniskategorien aufweist, werden die Beispiele durch multinomiale logistische Regression analysiert.[43]

3.3.3 Clustering

Bei Clustering werden die Beispieldatensätze anhand von Ähnlichkeiten in Gruppen zusammengefasst. Anders als bei der Klassifikation, beim Clustering werden keine Kategorien (Labels) als Klassifikationsergebnisse vordefiniert. Der Lernstil ist unüberwachtes Lernen.

- **K-Means** K-means ist ein Prototyp-basiertes Clusterverfahren, das heißt die Cluster werden durch Prototypen (z.B. Zentrum) repräsentiert. Die Anzahl der Clustern K wird vordefiniert. Die Aufgabe dieses Modells ist es die Eingabedatensatz in K Cluster zu unterteilen. Während des Clusterverfahrens werden zuerst K verschiedene Punkte als Anfangszentren gewählt. Dann wird jeder Datenpunkt den ihm nähesten Zentrum zugeordnet. Basierend auf den jetzigen Clustern wird das Zentrum jedes Clusters neu berechnet. Danach wird jeder Datenpunkt wieder den ihm nähesten Cluster zugeordnet. Dieser Vorgang wird wiederholt, bis die Zentren nicht mehr oder nur wenig verschieben.[40]
- **Hierarchisches Clustern** Hierarchisches Cluster bezeichnet das distanzbasierte Clusterverfahren zur Strukturentdeckung in Datenbeständen. Dieses Verfahren teilen die Datenmenge in eine abgestufte Folge von Clustern ein. Jedes Beispiel wird am Anfang als ein Cluster betrachtet. Eine Berechnung des Abstands zwischen allen Clustern wird durchgeführt. Anschließend werden die beide Clustern, die am nähesten zueinander sind, in einem Obercluster zusammengefasst. Dieser Vorgang wird wiederholt, bis alle Beispiele zu einem gleichen Cluster gehören.[40]
- **Dichteverbundenes Clustern (DBSCAN)** Dichteverbundenes Cluster, engl. Density-Based Spatial Clustering of Applications with Noise (DBSCAN), ist ein dichtebasiertes Clusterverfahren. Zwei Parameter werden vor Clustering festgelegt: Nachbarschaftslänge und Dicht. Wenn der Abstand zwischen einer Punkt und einer Kernpunkt weniger als ‚Nachbarschaftslänge' ist, kann diese Punkt mit der Kernpunkt gebunden. ‚Dicht' definiert dagegen, wann eine Punkt als Kernpunkt gesehen werden kann. Während des Verfahrens werden vor allem alle Datenpunkte laut der Nachbarschaftslänge und Dicht als sogenannte Kernpunkte, Randpunkt und Rauschpunkte benannt. Rauschpunkte werden bei der Berechnung entfernt. Die durch dieselben Kernobjekten mit einander verbundenen Objekte werden ein Cluster bilden.[40]

Zusammenfassend sind die oben erwähnten Begriffe: Regression, Klassifikation und Clustering drei Grundmethode für maschinelles Lernen. Basierend darauf werden zahlreiche Algorithmen zur Lösung spezifischer Aufgaben entwickelt. Beispielsweise werden einige Modelle im Anwendungsgebiet ‚machine-learned ranking (MLR)' entwickelt, um eine Rangfolge der Instanzen in neuen Listen zu erstellen. Ihr Grundprinzip lässt sich je nach den Anforderungen an die Ausgabedaten auf Klassifikation oder Regression zurückführen.

3.4 Aktuelle konsolidierte Software-Frameworks für machinelles Lernen

3.4.1 Scikit-Learn

Scikit-Learn ist eine freie Software-Bibliothek für maschinelles Lernen für die Programmiersprache Python[29]. Sie ist so konzipiert, dass es mit den numerischen Bibliotheken NumPy (Eine Basisdatenstruktur für Daten- und Modellparameter[29]) und wissenschaftlichen Bibliotheken SciPy (beinhaltet effiziente Algorithmen für lineare Algebra, spärliche Matrix-Darstellung, spezielle Funktionen und grundlegende statistische Funktionen[29]) von Python zusammenarbeitet. Scikit-Learn stellt hochmoderne

Implementierungen vieler Algorithmen für maschinelles Lernen bereit, während sie eine leicht nutzbare und eng in die Python-Sprache integrierte Oberfläche verfügt. Sie wird bis jetzt aktiv weiterentwickelt[29].

3.4.2 TensorFlow

TensorFlow ist eine von Google gegründeter Open-Source Programmbibliothek für künstliche Intelligenz bzw. maschinelles Lernen im Umfeld von Sprache und Bildverarbeitungsaufgaben. Mit den eingebauten Modulen wie TF.Learn und TF.Slim kann TensorFlow ein neues Netzwerk schnell entwickeln und eine Schnittstelle zu Scikit-Learn erstellen. Wegen einer flexiblen Architektur können die Berechnungen auf einer oder mehreren CPUs oder GPUs auf einem Desktop, Server oder mobilen Gerät mit einer einzigen API bereitgestellt werden[41]. TensorFlow unterstützt Linux, Mac und Windows. Die durch Tesseract trainierten Modelle können reibungslos auf eine Vielzahl von Hardware- und Betriebssystemplattform verwendet werden.

3.4.3 OpenCV

OpenCV (Open Source Computer Vision Library) ist eine freie Programmbibliothek mit Algorithmen für die Bildverarbeitung und maschinelles Sehen. Sie beinhaltet ferner Bibliothek für maschinelles Lernen mit Funktionen wie kNN, Entscheidungsbäume, Bayes-Klassifikator, Künstliche neuronale Netze usw[27]. Außerdem werden die maschinelles Lernen Modells in die Parameter mehrerer Funktionen eingebaut werden.

Außerdem stellen noch viele Frameworks eine breite Menge von modernen Algorithmen des maschinellen Lernens zur Verfügung, z.B. Brain für die Programmiersprache JavaScript, Go Learn für die Programmiersprache Go, Torch7 für Lua usw. Caffe(Convolutional Architecture for Fast Feature Embedding), Keras, Theano und CNTK(Computational Network Toolkit) konzentrieren sich wie TensorFlow auf das Deep learning. OpenAI Roboschool, Nervana Coach und Unity ML Agents sind spezifisch für verstärkendes Lernen entwickelt. Durch Gestaltung der vorhandenen Modulen in den Bibliotheken reduzieren solche Frameworks den Aufwand beim Coding und Verstehen von Coding.

4 Entwicklung eines Algorithmus zur Bilderkennung und Fehlererkennung

In diesem Kapitel wird der entwickelte Algorithmus zur Absicherung der HMI-Textanzeige vorgestellt. Zu diesem Algorithmus gehören die Entwicklung automatischer Bildverarbeitung und -erkennung und die Entwicklung eines Algorithmus zur automatischen Textfehlererkennung. Nach der Analyse von Eingabedaten wird ein Grundkonzept zur Lösung der Aufgaben entwickelt.

4.1 Anforderungsanalyse

Für die Software-Produktentwicklung ist eine Anforderungsanalyse notwendig. Im Folgenden werden die fundamentalen Anforderungen des Programms noch einmal konkret aufgezeigt. Alle qualitativen und quantitativen Anforderungen können in einer Anforderungsliste geordnet und in eine Zusammenstellung geschrieben werden.

Eine Anforderungsliste besteht normalerweise aus den Hauptmerkmalen, Anforderungen, detaillierten Beschreibungen, Forderungen/Wünschen und Zuständigen. Jede Anforderung besitzt eine Identifizierungsnummer. Hauptmerkmale helfen dabei, eine Anforderungsliste zu erstellen. Anforderungen werden in Anforderungslisten meist stichpunktartig beschrieben. Zur Vermeidung von Fehlinterpretationen empfiehlt sich die Angabe einer detaillierten Beschreibung. Forderungen müssen unter allen Umständen erfüllt werden. Wünsche hingegen sollen nach Möglichkeit berücksichtigt werden, müssen jedoch nicht zwangsweise erfüllt werden.[28]

In dieser Arbeit wird ein Algorithmus erstellt und implementiert, der eine automatisierte Fehlererkennung der Textanzeige ermöglicht. Die Anforderungsliste für diesen Algorithmus ist in Abbildung 4.5 dargestellt. Zur Erarbeitung der Anforderungen werden Hauptmerkmale aus den Normen ISO 9126 und DIN 66272 herangezogen. Sie lauten: Funktionalität, Benutzbarkeit, Effizienz, Zuverlässigkeit, Änderbarkeit und Übertragbarkeit. Diese Liste wird auch durch selbstdefiniertes Merkmale, wie zum Beispiel, die zur Verfügung stehende Software und Dokumentation ergänzt.

Zu der Funktionalität des Softwareprodukts gehören die einzelnen Funktionen für die automatisierte Absicherung der Textanzeigen. Es wird gefordert, dass die Fehlererkennung ohne Tester durchgeführt werden kann. Dazu müssen vor allem die vorverarbeiteten Screenshots in einer saubereren Form zur Verfügung stehen. Die Ist-Texte sollen dann automatisch nach der Bilderkennung in einer definierten Dateiform gespeichert und weiter bei der Fehlererkennung verwendet werden. Eine einfache statistische Auswertung der Fehlerkennungsergebnisse soll auch realisiert werden.

Im Bereich der Benutzbarkeit wird die Anzeige aller relevanten Daten bei der Fehlererkennung gefordert. Dazu zählen u.a. die aktuellen Soll-Texte für alle Screenshots unter einem Bildschirmnamen sowie die Ist-Texte und die Vergleichsergebnisse. Diese Informationen werden dem Nutzer dargeboten.

Zu der Dokumentation gehören die Visualisierung der Funktionsweise des Fehlererkennungsprogramms und eine Beschreibung der Trainingsverfahren eines Texterkennungsmodells. Diese Informationen werden

bei der Erweiterung von Funktionen wie zum Beispiel Training eines Erkennungsmoduls neuer Sprache funktionieren.

Hauptmerkmale	Anforderung	Nr.	Detaillierte Beschreibung	F/W
Funktionalität	Bilderkennung	1	Automatisierte Bildverarbeitung der Input-Screenshots	F
		2	Training eines systemorientierten Texterkennungsmodells	F
		3	Ist-Texte durch Texterkennung aus den Screenshots extrahiert	F
	Fehlererkennung	4	Statistische Auswertung der Anzeigetexte	F
Benutzbarkeit	Anzeige aller notwendigen Daten zur Ergebnissedarstellung	5	Soll-Texte der HMI-Absicherung	F
		6	Richtige Ist-Texte bei der Absicherung	F
		7	Falsche Ist-Texte bei der Absicherung	F
	Sprache	8	Programm- und Kommentarsprache ist Englisch	F
		9	Texterkennung von Chinesisch	F
		10	Texterkennung anderer Sprache	W
Software	Tesseract OCR-Engine	11	Das Texterkennungsverfahren wird durch Texterkennungsengine Tesseract durchgeführt	F
	Eingabedaten	12	Eine Gruppe von Screenshots des Infotainmentsystems unter gleichen Bildschirmnamen	F
		13	Soll-Texte in XML-Dateiform	F
	Ausgabedaten	14	Ist-Texte und dessen Koordinaten im CSV-Dateiform	F
		15	Ergebnisse der Fehlererkennung	F
Änderbarkeit	Parameter der Bilder- und Fehlererkennung	16	Variable Parameter für Bildquelle	F
		17	Variable Parameter für Texterkennungsmodel	F
Effizienz	Zykluszeit	18	Verarbeitungszeit jedes Bildes unter 2 Sekunde	W
Dokumentation	Ablaufsdiagramm	19	Die Funktionsweise des Fehlererkennung Programms wird mithilfe von Ablaufdiagrammen veranschaulicht	F
	Trainingsverfahren	20	Trainingsverfahren des Texterkennungsmodells wird dokumentiert	F
	Programm	21	Funktionsblocke werden erklärt	F

Abbildung 4.1: Anforderungsliste

4.2 Analyse der HMI-Screenshots

Die Aufgabe dieser Arbeit ist es, eine Funktion zu entwickeln, durch die die HMI-Textanzeige automatisch überprüft werden kann. ZD Automotive hat ein Screenshot-Tool entwickelt, welches in der Lage ist, automatisch HMI-Menüstrukturen zu generieren und Screenshots jedes Bildschirms zu erstellen. Dadurch stellen die digital gespeicherten HMI-Screenshots die Voraussetzung für die automatische Fehlererkennung dar. Zur Entwicklung eines auf das HMI-Absicherungssystem anpassenden Algorithmus, sollen vor allem die Screenshots analysiert werden.

4.2.1 Funktionsweise automatischer Screenshots

Das Screenshot-Tool funktioniert je nach den Namen der Screenshots (Screen-ID oder Bildschirmname). Ein Signal ‚Klick' simuliert das Klickverfahren auf dem Bildschirm von Menschen. Das Anzeigeelement des Infotainmentsystems - der Bildschirm - hat eine Auflösung von 1280 * 660 Pixeln. Manchmal können nicht alle der Inhalte unter einem Bildschirmnamen (Screen Name) auf einmal in dem Bildschirm angezeigt werden. Deswegen simuliert das Tool noch ein Zugverfahren, um die nicht angezeigten Bereiche herauszuziehen. Wenn der Status der HMI-Anzeige geändert wird, wird ein neuer Screenshot erstellt - zum Beispiel: nach einem Klicksignal wird Status A (siehe Abbildung 4.2) in Status B (siehe Abbildung 4.3) umgewandelt, nach einem Zugverfahren nach oben wird Status B in Status C (siehe Abbildung 4.4) umgewandelt. Unter jedem Status wird ein Screenshot erstellt.

Abbildung 4.2: Screenshot: Status A

Abbildung 4.3: Screenshot: Status B

Abbildung 4.4: Screenshot: Status C

4.2.2 Komponenten der Screenshots

In jedem Screenshot sind viele Informationen in unterschiedlichen Formen enthalten. Ein Screenshot besteht aus einem schwarzen Hintergrund, einem Kopfbereich mit Uhrzeit- und Bluetooth-Info usw., einem Link-Bereich der zu einem bestimmten Menü führt, und einem Hauptbereich. Bei der Textabsicherung wird nur der Hauptbereich überprüft.

Die Komponenten im Hauptbereich lassen sich im System in zwei Typen klassifizieren: ‚statische' Komponenten und ‚dynamische' Komponenten. Wenn man auf eine ‚statische' Komponente klickt, wird nichts passieren. Solche Komponenten dienen als Darsteller der Informationen und haben keine Verbindungen mit anderen Screens. Die ‚dynamischen' Komponenten dienen als Statusveränderungstaste. Wenn man auf solche Komponenten klickt, wird entweder ein Sprung zu einem anderen Screen in der Menüstruktur oder ein Informationsaustausch zwischen dem Infotainmentsystemen und den Steuergeräten des Fahrzeuges ausgelöst.

Da Screenshots statische Bilder sind und keine dynamischen Informationen, wie zum Beispiel Bildschirmsprung oder Informationsaustausch tragen können, werden die Komponenten auch nur statisch analysiert. Es werden alleine die visuellen Informationen verwendet. Die dynamischen Informationen können hierbei nicht helfen. Aus visueller Sicht lassen sich die Komponenten im Hauptbereich wie folgend aufteilen:

- **Bilder** Zu den Bildern gehören z.B. Icons, Zurück-Taste und Trennlinien. Manchmal werden die Bilder neben die Texte gelegt, um die Bedeutung der Texte in einer intuitiven Weise zu wiederholen. Manchmal übernehmen die Bilder alleine den Auftrag, Informationen zu tragen.
- **Texte** Die Textbereiche sind die Zielkomponenten dieser Arbeit. Da das Infotainmentsystem verschiedene Versionen in unterschiedlichen Sprachen zur Verfügung stellt, gibt es bei einer Komponente je nach Sprache eine Version. Diese Arbeit richtet sich nach der chinesischen Version. Manchmal sind die Texte, Wörter - eine Kombination von zwei oder mehreren chinesischen Schriftzeichen. Manchmal bestehen die Texte aus einem oder mehreren Sätzen.

4.3 Vorstellung des Grundkonzepts

Die Eingabe- und Ausgabedaten der Absicherungsfunktion werden überlegt. Als Eingabedaten stehen die aktuell aus dem Infotainmentsystem entnehmbaren Informationen zur Verfügung: Screenshots mit einem Dateinamen in Form von Screen-ID, Schriftart-Datei der Anzeigetexte im Infotainmentsystem. Außerdem sind die Inhalte der Soll-Anzeigetexte in Form von XML-Dateien verfügbar. Als Ausgabe wird eine Darstellung der Ergebnisse der Fehlererkennung erwartet. Die Ergebnisse sollen zeigen, welche Texte richtig oder falsch sind.

In das Programm werden die Quelle der Screenshots und die XML-Datei (mit Soll-Texten) eingegeben. Aufgrund der Funktionsweise automatischer Screenshots werden ein Paar Screenshots unter dem gleichen Bildschirmnamen voneinander wiederholt. Die Struktur des Bildes und die Textinhalte solcher Screenshots sind gleich. Sie werden nur wegen der Statusveränderung durch ein Klickverfahren alle als Screenshots gespeichert. Wenn alle diese Screenshots mit gleichen Texten verarbeitet werden, wiederholt das Programm zu viele unsinnige Verfahren, was dazu führt, dass das Programm sehr zeitaufwendig wird. Zur Vermeidung von wiederholenden Verarbeitungen wird vor allem eine Wiederholungsanalyse für Screenshots ins Programm eingeführt. Die gleichen Screenshots unter einem Bildschirmnamen werden gefiltert. Nur ein Screenshot davon wird weiter an nächsten Schritt gegeben.

Zur Realisierung der automatischen Fehlererkennung, sollen zuerst die Texte in den Screenshots erkannt werden. Hierbei soll die Texterkennungstechnik verwendet werden. Demnach können erst die erkannten Texte mit den Soll-Texten verglichen werden. Im Bereich Texterkennung gibt es verschiedene kommerzielle Software (z.B. ABBYY Finereader, Havon OCR), die sehr teuer ist und keine offenen Strukturen aufweist. Trotzdem liefert sie eine allgemeine Texterkennungsfunktion, kann aber nicht den spezifischen Situationen angepasst und verbessert werden. Außerdem ist das Preis-Leistungsverhältnis relativ hoch. Im diesem Hinblick wird eine Open-Source Software verwendet. Es handelt sich dabei um die Textererkennungsengine Tesseract-OCR 3.04. Es ist eine freie Software, die unter der Apache-Lizenz, Version 2.0 zu haben ist. Diese Software wurde nicht nur wegen der Kostenfreiheit gewählt, sondern auch wegen ihrer Plattformunabhängigkeit und guten Erweiterbarkeit. Für chinesische Schriftzeichen ist bereits ein Erkennungsmodul in der Datenbank der Tesseract vorhanden. Wegen des riesigen Rechenaufwands bei der Mustererkennung, ist die Erkennungsgeschwindigkeit dieses Moduls jedoch sehr langsam. Zur Erhöhung der Erkennungseffizienz muss Tesseract auf Basis von Soll-Anzeigetexten und Schriftart-Dateien ein neues Erkennungsmodul trainieren. Hier wird Tesseract sowohl bei Texterkennung angerufen als auch bei Training neues Erkennungsmoduls verwendet.

Da sich die Funktionen der Tesseract auf die Texterkennung konzentrieren und es nur begrenzte und schwache Verfahren zur Unterstützung der Bildverarbeitung (mithilfe der Bibliothek Leptonica) gibt, stellt Tesseract hohe Anforderungen an die Qualität der Eingabebilder, z.B. der Vordergrund soll vom Hintergrund durch große Helligkeitsunterschiede eindeutig getrennt werden, da ansonsten nur Texte erkannt werden. Deshalb ist eine Vorverarbeitung der Bilder der Texterkennung von Tesseract notwendig, um eine höhere Qualität der Ausgaben von Tesseract zu ermöglichen. Bei der Bildverarbeitung für die Screenshots kann das Ziel der Verarbeitung konkreter definiert werden: die Texte aus komplexem Hintergrund eindeutig zu trennen und gleichzeitig die Texte selber möglichst vollständig zu halten, die anderen Komponenten des Screenshots (z.B. Bilder) möglichst viel zu entfernen. Zur Umsetzung der Bildverarbeitung wird die freie Programmbibliothek Open Computer Vision Library (OpenCV) verwendet. Wegen ihrer schnellen Geschwindigkeit bei der Bildverarbeitung und ihrer vollständigen und fortschrittlichen Algorithmen wird OpenCV gewählt. Ihre Stärke liegt auch in der Plattformunabhängigkeit. Die Version ist OpenCV 2.4.13.4.

Das Bilderkennungsverfahren setzt sich zusammen aus Bildverarbeitung und Texterkennung. Die Ausgaben davon - die erkannten Texte (Ist-Texte)- werden als Eingaben für die Fehlererkennung dienen.

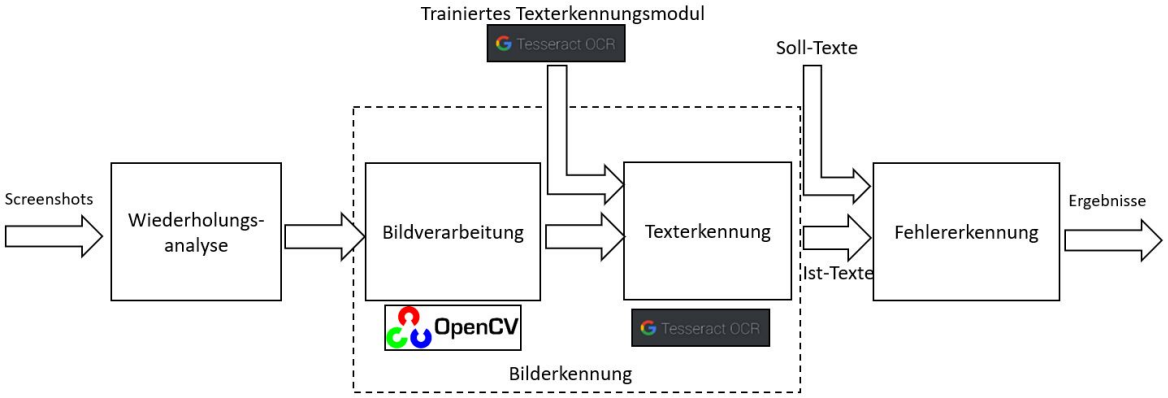

Abbildung 4.5: Darstellung des Grundkonzepts

Anschließend soll das Fehlererkennungsverfahren durchgeführt werden. Die Inhalte der Soll-Anzeigetexte sollen aus der XML-Datei extrahiert werden. Ein Vergleichsverfahren zwischen Ist-Texte und Soll-Texte wird erstellt. Die Vergleichsergebnisse sollen für eine weitere Überprüfung und Korrektur gespeichert und angezeigt werden. Das ist ein vollständiges Fehlererkennungsverfahren für die Textanzeige-Absicherung der Screenshots unter einem Bildschirmnamen: Wiederholungsanalyse, Bildverarbeitung, Texterkennung und Fehlererkennung. Nachdem alle Daten gespeichert wurden, fängt das Verfahren für Screenshots unter einem anderen Bildschirmnamen wieder von vorne an. Abbildung 4.5 zeigt das Grundkonzept der Fehlererkennungsvorgänge.

4.4 Bilderkennung

Hier wird das entwickelte Verfahren zur Mission Bilderkennung vorgestellt. Die Screenshots, in Form von Pixelbilder, werden in dem Erkennungssystem eingelesen. Jeder Pixel lässt sich in 2D-Bildkoordinaten beschreiben. Die Bildkoordinaten werden mit x, y bezeichnet. Im ersten Schritt werden die Eingabebilder, nämlich Screenshots, vorverarbeitet. Wenn die Anforderungen an die Qualität der Eingabebilder erfüllt sind, wird die Texterkennung durch OCR-Engine Tesseract durchgeführt.

4.4.1 Konzeption Verfahren der Bildverarbeitung

Laut des Grundkonzepts sollen die Texte aus komplexem Hintergrund eindeutig getrennt werden. Die Pixeln der Textbereiche sollen möglichst vollständig gehalten, und die anderen Komponenten des Screenshots (z.B. Bilder, Hintergrund der Textbereiche) sollen möglichst viel entfernt werden. Wie im Kapitel Grundlagen erwähnt, werden normalerweise bei der Bildverarbeitung im wesentlichen Binarisierung, Rauschunterdrückung, Neigungskorrektur und Layout-Analyse durchgeführt. Basierend auf der Analyse des HMI- Screenshots wird der Ablauf der Bildverarbeitungsverfahren hier nochmal spezifisch dargelegt (Abbildung 4.6).

Durch Binarisierung wird ein Bild in zwei Farben konvertiert, schwarz und weiß. Dieser Prozess ist notwendig, weil die Eingabe von Tesseract ein Binärbild sein muss. Das Ergebnis einer traditionellen Binarisierung kann suboptimal sein, wenn der Bildhintergrund ungleichmäßig dunkel ist. Das kann dazu führen, dass der Bildbereich mit dunklerem Hintergrund total schwarz wird und alle Bildinformationen in diesem Bereich verdeckt sind. Bei vielen Screenshots ist es genauso: der Hintergrund vieler Textbereiche ist ungleichmäßig dunkel. Bei der Verarbeitung müssen diese potenziellen Probleme beachtet werden.

Rauschunterdrückungsverfahren können das Rauschen von Bildern minimieren. Laut der ursprünglichen Bedeutung von "Rausch" entstehen die Räusche wegen der begrenzten Auflösung der Bildsensoren bei der Bildaufnahme. Theoretisch werden die Screenshots aus dem Infotainmentsystem kein Rauschen haben, denn die originalen Bilder sind digital und dessen Pixeln werden eins zu eins aufgenommen. Eine Verschlechterung des eigentlichen Bildinhalts wird während der Aufnahme der Screenshots nicht ausgelöst. Aber dieses Verfahren wird nicht vom Anfang an aufgegeben. Es ist möglich, dass der ungleichmäßige Hintergrund als Rauschen betrachtet und durch Rauschunterdrückung behandelt werden kann.

Da bei der Aufnahme von Screenshots keine Neigung entsteht, ist eine Neigungskorrektur unnötig.

Layout-Analyse ist auch ein wesentliches Verfahren. Die relevanten Textbereiche sollen hierbei analysiert und extrahiert werden. Idealerweise werden der Hintergrund und andere unnötige Informationen in den

Bildern nach den oben genannten Verfahren schon entfernt. Durch Basisoperation in der Morphologie, (auch mathematische Morphologie) wie Erosion und Dilatation mit den vordefinierten Parametern, können dann die nebeneinanderstehenden Schriftzeichen verbunden werden. Mit Hilfe des k-Means Clustering Algorithmus werden die Konturen dieser verbundenen Textbereiche eindeutig mathematisch beschrieben: Koordinaten des Anfangspunktes (x, y), die Breite der Kontur w, die Höhe der Kontur h. Solche verbundenen Textbereiche, die den Komponenten des HMI-Bildschirms entsprechen, lassen sich auf diese Weise in den Bildern lokalisieren.

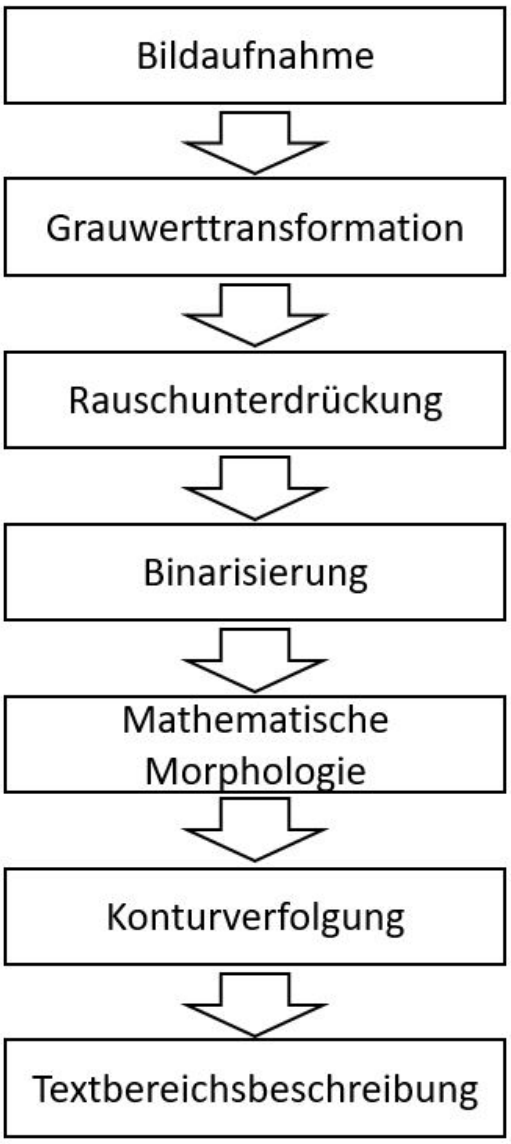

Abbildung 4.6: Bildverarbeitungsverfahren

4.4.2 Konzeption Erstellung des Texterkennungsmoduls

Nachdem die Textbereiche in den Bildern bekannt sind, kann der Texterkennungsprozess durchgeführt werden. Zudem muss ein Texterkennungsmodul vorhanden sein. Zur Erstellung des Texterkennungsmoduls werden die Methoden von maschinellem Lernen verwendet.

Mit Verwendung der vorhandenen Schriftart-Datei, der Anzeigetexte und der Datei mit den Inhalten der Soll-Texte, lässt sich im ersten Schritt eine Datei mit den Trainingsbeispielen generieren. Ein Trainingsbeispiel entspricht einem Schriftzeichen in definierter Schriftart. Demnach soll eine Analyse der Umrisse jedes Trainingsbeispiels durchgeführt werden, in der die relevanten Informationen (Merkmale) vom Trainingsbeispiel gespeichert sind, z.B. Schriftart, Zeicheneigenschaften. Das UTF-8-codierte Schriftzeichen ist das Label, das zu den Merkmalen passt. Nachdem alle Merkmale des Schriftzeichens extrahiert und gespeichert wurden, müssen sie gruppiert werden, um die Prototypen zu erstellen. Alle Prototypen werden zusammengestellt und als ein Modul gepackt. Auf diese Weise wird ein Erkennungsmodul erstellt. Es dient als Erkennungsmuster/Klassifikator bei der Klassifikation des Schriftzeichens.

Zur Erhöhung der Erkennungsrate können Wörterbücher verwendet werden. Die Soll-Texte bei der Texterkennung sind bereits bekannt. Das heißt, die Kombinationen zwischen unterschiedlichen Schriftzeichens sind bereits festgelegt. Diese Informationen können ausgenutzt werden, um die ‚Wörter', nämlich die Zeichenkombinationen im Wörterbuch zu erweitern. Die Wörterbücher sollen nach ihrer Entstehung ebenfalls mit den Prototypen zusammengepackt werden.

Der letzte Schritt der Bilderkennung ist die Texterkennung. Für jeden durch Bildverarbeitung gefundenen Textbereich wird Tesseract OCR-Engine benutzt. Die Texte in den Textbereichen werden auf Zeichen- und Wörterebene mit Mustern (Prototypen und Wörterbücher) verglichen. Laut der kNN-Methode wird das Schriftzeichen mit der höchsten Konfidenz als Ausgabetext (Ergebnis der Texterkennung) festgelegt. Alle erkannten Texte aus einem Bild werden in einer Datei für weitere Fehlererkennungsverfahren gespeichert.

4.5 Fehlererkennung

Bei der Fehlererkennung wird erwartet, dass die erkannten Texte aus der Texterkennung mit den Soll-Texten verglichen werden und anschließend die Vergleichsergebnisse dargestellt werden. Die Erkennungskriterien werden hierbei entwickelt.

Die möglichen Ergebnisse aus der Texterkennung enthalten:

a) Texte in Soll-Liste, Texte in Ist-Liste

b) Texte in Soll-Liste, Texte nicht in Ist-Liste

c) Texte nicht in Soll-Liste, Texte in Ist-Liste

Soll-Texte	Ist-Texte
✓	✓
✓	✗
✗	✓

Abbildung 4.7: Drei Situationen der Texterkennungsergebnisse

Fall a) bedeutet, dass die Texte richtig auf dem Bild erkannt werden. Es beweist auch, dass die erwarteten Texte auf dem Bildschirm richtig gezeigt werden. Dieser Testfall kann als bestanden (richtig) betrachtet werden, weil die Soll-Texte richtig im HMI-Anzeigeelement gezeigt werden.

Fall b) bedeutet, dass die erwarteten Texte nicht auf den Screenshots richtig gezeigt werden. Mögliche Gründe dafür sind: erstens, die Texte stehen doch im Screenshots aber werden im Texterkennungsverfahren nicht richtig erkannt. Zweitens, die Texte sind auf dem Bildschirm des Infotainmentsystems, aber in den Screenshots nicht enthalten. Das impliziert, dass bei der Generierung von Screenshots nicht alle Inhalte unter einer Screen-ID gespeichert werden. Drittens, die erwarteten Texte werden weder in den Screenshots noch auf dem Bildschirm gezeigt werden. Das ist dann ein nicht bestandener Testfall. Der zweite und der dritte Grund lassen sich nicht durch diesen Algorithmus unterscheiden, weil die Eingabedaten, Screenshots sind und der Status auf dem Bildschirm nicht erlangt werden kann.

Fall c) bedeutet, dass die bei der Texterkennung erkannten Ist-Texte von der Soll-Datei abweichen. Entweder ist es ein nicht bestandener Testfall oder die Ist-Texte wurden falsch erkannt.

Auf Basis von der obigen Analyse soll die Fehlererkennung in zwei Richtungen durchgeführt werden:

1. Ist-Texte als Referenz, basierend darauf werden Soll-Texte überprüft. Die Fehler werden als Absicherungsergebnisse betrachtet.
2. Soll-Texte als Referenz, basierend darauf werden Ist-Texte überprüft. Die betroffenen Fehler wirken als Hinweise zur Verbesserung der Generierung von Testfällen.

Als Ergebnisse der Fehlererkennung sollen die beiden richtigen und falschen Texte angezeigt werden. Abbildung 4.8 zeigt die Abläufe der Fehlererkennung.

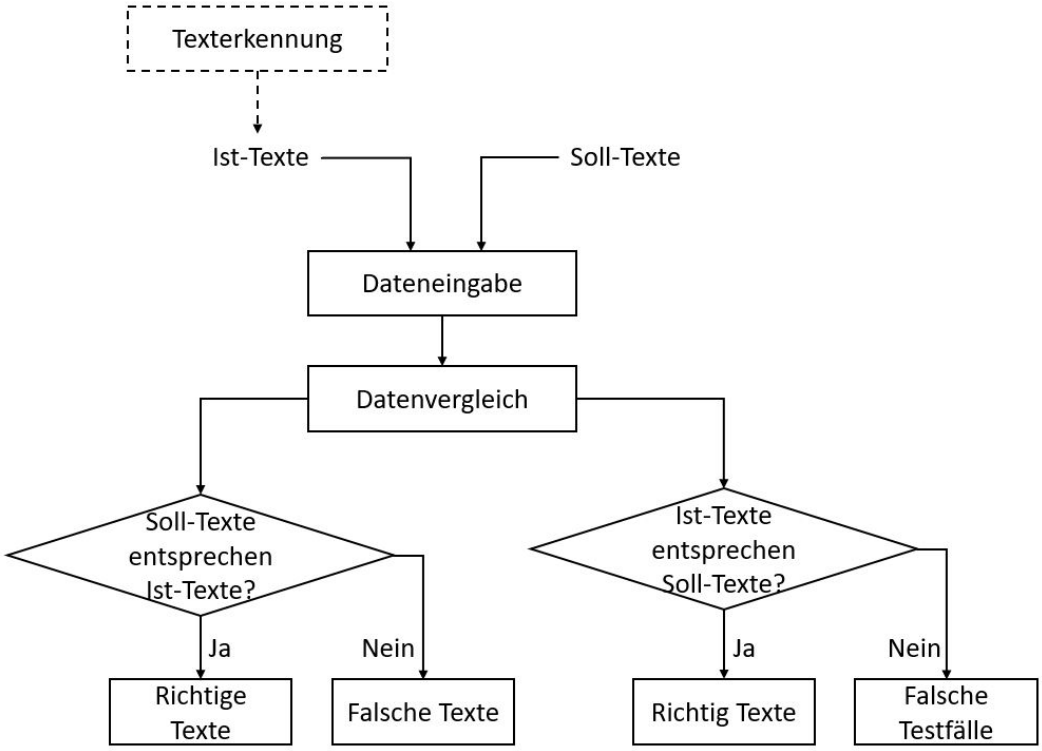

Abbildung 4.8: Abläufe der Fehlererkennung

5 Implementierung und Validierung des Algorithmus zur HMI-Absicherung

Im Rahmen dieses Kapitels werden die Prozesse zur Umsetzung des im Kapiteapitel 4 entwickelten Algorithmus dokumentiert. Basierend auf den Ergebnissen der Implementierung wird eine Auswertung des Algorithmus erstellt.

5.1 Durchführung der Bildverarbeitung

Das Ergebnis der Bildverarbeitung soll eine Menge von Merkmalen des Eingabebildes sein. Zur erfolgreichen Ermittlung der Ergebnisse muss die Bildqualität laut Nachfrage verbessert werden.

Mithilfe von OpenCV werden die Bilder verarbeitet. Die Screenshots werden in JPG-Form unter Anwendung der OpenCV-Funktion ‚imread' aus der angegebenen Quelle aufgenommen. Anstatt als Graustufenbilder werden die Screenshots zur besseren Ergebnisdarstellung als Farbbilder aufgenommen. Die Farbbilder werden dekodiert und die Farbkanäle werden in der Reihenfolge B, G, R gespeichert[39]. Ein originaler Screenshot ist eine Kombination von Screenshots mit den gleichen Inhalten, aber in unterschiedlichen Sprachen. Da das Erkennungsmodul lediglich für Chinesisch entwickelt wurde, werden nur die chinesischen Teile der Screenshots für weitere Verarbeitungen ausgeschnitten.

Anschließend wird eine Grauwerttransformation mit der Funktion ‚cvtColor' durchgeführt. Alle Pixel in den Bildern werden in Grauwerten [0, 255] beschrieben. Dieser Prozess stellt die Bilder zur Binarisierung bereit. Da die Screenshots aus schwarzem Hintergrund bestehen, wird noch ein Schritt vor der Binarisierung durchgeführt – das Farbenumkehren. Dadurch wird der Hintergrund von schwarz in weiß und die Pixel mit Informationen werden in schwarz oder grau umgewandelt.

In der Konzeptionsphase wurde vermutet, dass der ungleichmäßige Hintergrund des Screenshots vielleicht als Rauschen betrachtet und durch Rauschunterdrückung behandelt werden könnte. Jedoch wurde nach vielen Versuchen klar, dass Rauschunterdrückung lediglich kleine Veränderung beim Hintergrund bewirkt. Diese Veränderungen können auch ohne Rauschunterdrückung durch Binarisierung realisiert werden. Außerdem ist dieser Prozess sehr zeitaufwendig – für jedes Bild dauert dieses Verfahren ungefähr 1,2 Sekunden. Aus diesen Gründen wird auf Rauschunterdrückung bei der Bildverarbeitung verzichtet.

Die Binarisierung spielt eine wichtige Rolle bei der Verarbeitungsqualität. Mit der Funktion ‚threshold' kann dieses Verfahren entscheiden, welche Bildinformationen verworfen werden sollen. Durch ein kleines Visualisierungsprogramm mit Schwellenwert-Laufleiste kann die Bildveränderung mit unterschiedlichen Schwellenwerten deutlich sichtbar gemacht werden. Aus den Abbildungen 5.1, 5.2 wird ersichtlich, dass mit einem höheren Schwellenwert mehr Bildinformationen bestehen bleiben. Mit dem Wert 220 bleiben beispielsweise alle Texte im Screenshot erhalten. Einige Komponenten, wie Trennlinien, werden teilweise oder vollständig entfernt, aber die meisten Komponenten sind noch sichtbar. Die verbliebenen Hintergrundinformationen können die Texterkennung irreleiten. Im Vergleich zur Abbildung 5.1 werden in Abbildung 5.2 mit dem Schwellenwert 50 fast alle Hintergrundinformationen entfernt. Jedoch fehlt auch

ein Textbereich – die verlorenen Texte werden in Rot angezeigt und die zugehörigen Pixel hatten nach der Grauwerttransformation im Vergleich zu den schwarzen Texten einen höheren Grauwert. Deswegen werden sie mit einem höheren Schwellenwert entfernt. Es ist nicht erstrebenswert, dass die zu überprüfenden Texte bereit bei der Bildverarbeitung verloren gehen. Aus diesem Grund darf der Schwellenwert nicht zu niedrig eingestellt werden. Im Gegensatz dazu sind zu viele Hintergrundinformationen ebenfalls nicht gewollt, denn sonst werden die Ergebnisse der Texterkennung von ungewollten Bildkomponenten gestört. Vor diesem Hintergrund wird die Binarisierung in zwei verschiedenen Bereichen eines Screenshots unterschiedlich durchgeführt: der gesamte Hauptbereich des Screenshots wird mit einem niedrigen Grauwert transformiert und der Screenshot dadurch möglichst sauber gefiltert. Der ‚Namens-Bereich', der den Bildschirmnamen und farbige Texte beinhaltet, wird durch eine zusätzliche Binarisierung verarbeitet, damit die Farbtexte nicht verloren werden. Durch mehrere Versuche wurden die Schwellenwerte für beide Binarisierungen folgendermaßen festgelegt: 0 und 224.

Abbildung 5.1: Visualisierung der Binarisierung (Schwellenwert: 220)

Abbildung 5.2: Visualisierung der Binarisierung (Schwellenwert: 50)

Nach der Binarisierung muss der Prozess ‚mathematische Morphologie' durchgeführt werden. Die Grundidee dieses Prozesses ist es, die in einem Textbereich nebeneinanderstehenden Schriftzeichen

zu verbinden, damit das System eine Kontur dieses Bereichs finden kann. Mit den Konturen können die Koordinateninformationen der Textbereiche erhalten werden. Dadurch lassen sich die einzelnen Textbereiche lokalisieren und aus dem Screenshot extrahieren. Danach kann das Texterkennungsverfahren mit jeder Kontur durchgeführt werden. Anstatt einer einmaligen Erkennung des ganzen Screenshots ermöglicht die auf Konturverfolgung basierende Texterkennung eine Darstellung der Ergebnisse mit Layout-Informationen. Das vermeidet gleichzeitig eine falsche Erkennung von unnötigen Informationen. Nach dem Farbumkehren und der Binarisierung bestehen die Schriftzeichen aus schwarzen Pixeln. Um die Schriftzeichen zu verbinden, müssen die schwarzen Bereiche erweitert werden, wofür die Basisoperation der mathematischen Morphologie – ‚Erosion' – verwendet wird. Einerseits sollen die Schriftzeichen verbunden werden, anderseits sollen die unterschiedlichen Textbereiche separiert bleiben. Da sich alle Schriftzeichen auf ein Rechteck mit der Größe 40*40 Pixel begrenzen, wird die Kerngröße dieses Verfahrens in (20, 20) definiert. Die Größeangabe ist ebenfalls Pixel. Dadurch wird sichergestellt, dass sich auch das kleinste Schriftzeichen mit seinem Nachbar verbinden kann. Ein visueller Effekt der Erosion lässt sich in Abbildung 5.3 darstellen.

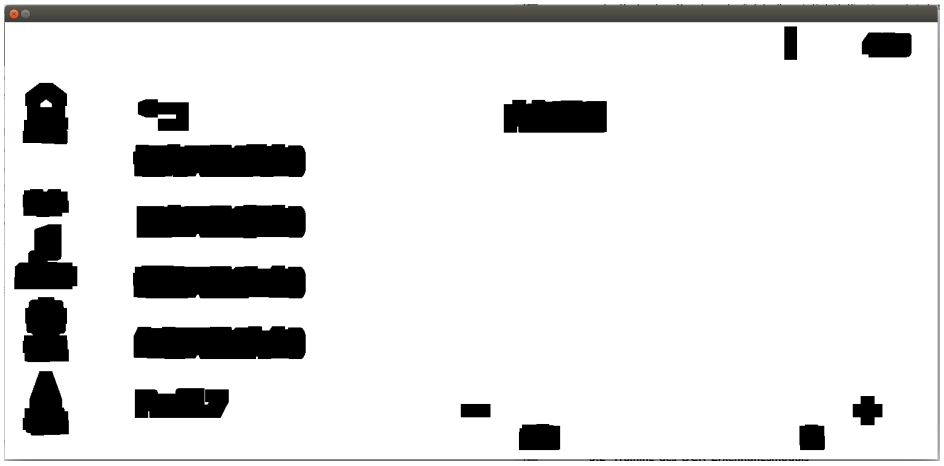

Abbildung 5.3: Screenshot nach Erosion

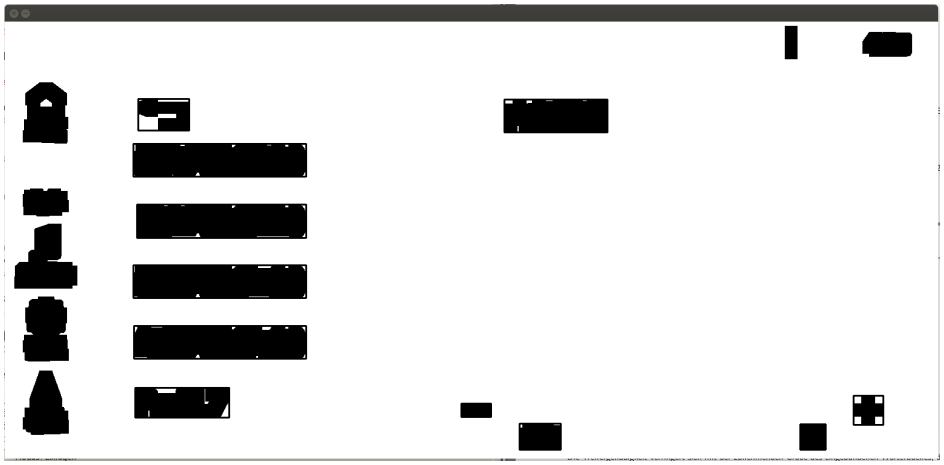

Abbildung 5.4: Kontur der Erosionsbereichen finden

Im Anschluss wird die Funktion ‚findContours' verwendet, um die Konturen aller Bereiche zu finden. Hierbei ist nur die Außenkontur der Textbereiche wichtig, weshalb der Parameter ‚RETR_EXTERNAL' gewählt wird. Die visuelle Darstellung der Konturen auf dem Binärbild zeigt Abbildung 5.4, eine farbige Darstellung zeigt 5.5. Diese Funktion sendet eine Variable zurück, die die Konturen als Vektoren von

Punkten speichert.

Abbildung 5.5: Farbige Visualisierung der Konturen

5.2 Training des OCR Erkennungsmoduls

Für jeden Textbereich erkennt die Texterkennung mithilfe der Konturen entsprechende Texte. Dieses Erkennungsverfahren wird durch die Open-Source-Engine Tesseract ermöglicht. Zur Erkennung unterschiedlicher Sprachen soll ein korrespondierendes Erkennungsmodul verwendet werden. Dazu wird ein zu dem Infotainmentsystem passendes Erkennungsmodul trainiert. Die Trainingsbeispiele sollen dabei manuell definiert und generiert werden. Die Texte in den Screenshots des Infotainmentsystems stellen die nötigen Testbeispiele dar, um die Performance des Erkennungsmoduls bewerten zu können. Die Trainingsprozesse sind wie folgt aufgebaut[33]:

Generierung von Trainingsbeispielen

Aus den Trainingsbeispielen erlernt die Tesseract-Engine ein Muster, das bei der Mustererkennung als Klassifikator der Schriftzeichen dient. Deswegen stellen der Umfang und die Qualität der Trainingsbeispiele einen wichtigen Einflussfaktor auf die Performance der Texterkennung dar. Je größer die Muster-Datenbank ist, desto langsamer ist die Geschwindigkeit der Texterkennung, denn desto mehr Muster müssen bei der Mustererkennung miteinander verglichen werden. Um die Texte so schnell wie möglich zu erkennen und zu vermeiden, dass die Texte als andere, im Infotainmentsystem nicht erscheinende Schriftzeichen erkannt werden, wird der Umfang der Trainingsbeispiele in Soll-Texten begrenzt. Alle chinesischen Schriftzeichen werden vor allem aus der XML-Datei extrahiert. Diese Schriftzeichen werden in einer TXT-Datei zusammengefasst und als Trainingstexte gespeichert. In den Trainingstexten sollten mehr Beispiele der häufigeren Schriftzeichen erscheinen[33]. Da die extrahierten Daten vollständige Texte im Infotainmentsystem sind, verändert die Häufigkeit jedes Schriftzeichens nichts. Außer der XML-Datei ist die Schriftart-Datei im Infotainmentsystem verfügbar. Mit dieser Datei ist Tesseract in der Lage, Trainingsdateien mit der gleichen Schriftart der HMI-Anzeige zu erstellen. Das durch diese Trainingsdatei trainierte Erkennungsmodul ist besonders effektiv bei der Erkennung von HMI-Textanzeigen.

Wie in Abschnitt 2.3.2 erwähnt, wird Tesseract über Befehlszeilenschnittstelle ausgeführt. Durch den Befehl ‚text2image' erstellt Tesseract 3.04 automatisch auf Basis der Trainingstextdatei und der Schrift-

artdatei zwei neue Dateien: ein TIF/BOX-Dateipaar. Die TIF-Datei ist eine Bilddatei, in der alle Trainingstexte mit der definierten Schriftart formatiert werden. Die BOX-Datei ist eine Textdatei, in der die Schriftzeichen im Trainingsbild mit den Koordinaten der Bounding Box jedes Schriftzeichens aufgelistet werden[37]. Bisher werden alle Trainingsbeispiele erstellt. Die Beispiele und die Koordinateninformationen sind alle in einem Bild oder mehreren Bilder (wenn die Beispiele sehr zahlreich sind) beinhaltet.

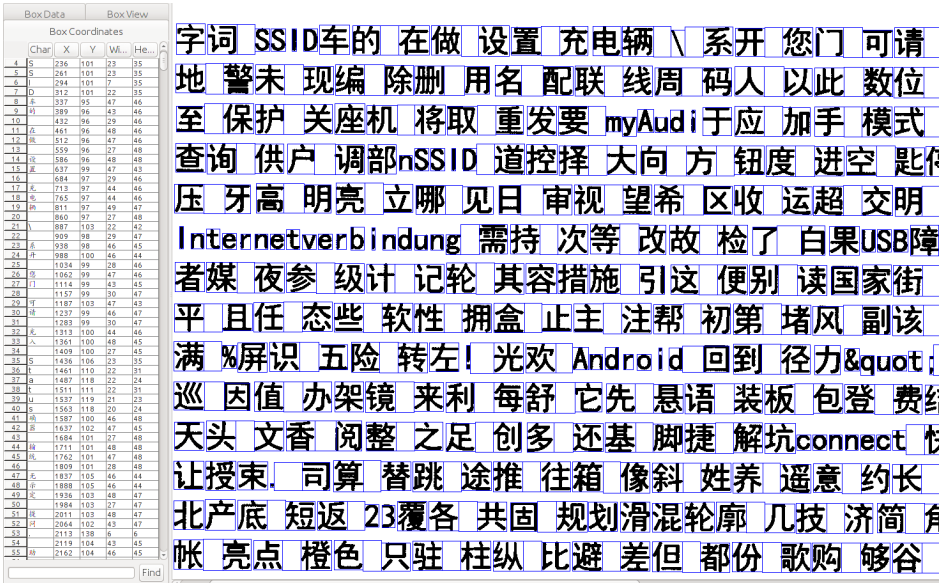

Abbildung 5.6: Visualisierung der TIF/BOX-Dateipaar in Software jTessBoxEditor

Training des Erkennungmoduls

Beim Training wird die Engine im Trainingsmodus ausgeführt. Nach der Ausführung des Befehls ‚tesseract [lang].[fontname].exp[num].tif [lang].[fontname].exp[num] box.train' wird eine neue ‚.tr'-Textdatei (Abbildung 5.7) erstellt, in der die Merkmale jedes Schriftzeichens beschrieben werden. Diese Datei wird dann im folgenden Schritt ‚Clustering' verwendet. Manchmal erscheinen Fehler bei den Ausgaben dieses Verfahren, was impliziert, dass die Merkmale der Schriftzeichen von Fehlermeldungen die Schriftzeichen nicht identifizieren können. Ein Grund dafür könnte es sein, dass die Bounding Box an der falschen Position ist. Solche BOX-Daten werden korrigiert. Eine andere Möglichkeit ist, dass die Merkmale von zwei Schriftzeichen sehr ähnlich sind und Tesseract die Merkmale nicht klassifizieren kann. Während des Trainings treten einige Fehlermeldungen immer wieder auf und können nie entfernt werden (siehe Abbildung 5.8 als Beispiel). Es wird davon ausgegangen, dass solche Schriftzeichen von Tesseract nicht klassifiziert werden können. Sie haben eine Gemeinsamkeit – die Außenstruktur, was dazu führt, dass die Größe und die Umrisse sich nicht voneinander unterscheiden lassen.

Da die unlösbaren Fehlermeldungen zunächst nicht aufgelöst werden können, geht das Training weiter. Mit den Befehlen ‚unicharset_extractor' und ‚set_unicharset_properties' erstellt Tesseract eine Datei zur Sammlung der Informationen jedes Zeichens, das von der Tesseract-Engine erlernt werden muss. Diese Datei dient zusammen mit der oben genannten ‚.tr'-Datei als Basis des Clusteringsverfahrens, denn in ihr wird die Zeichenform der Schriftzeichen beschrieben. Diese Merkmale werden durch die Befehle ‚mftraining' und ‚cntraining' in Gruppen zusammengefasst. Dadurch können die Prototypen der Schriftzeichen erstellt werden. Als Ausgabedaten werden zwei Binärdateien erstellt: ‚inttmp' dokumentiert den Formprototyp und ‚pffmtable' die Anzahl der Soll-Merkmale jedes Schriftzeichens.

```
<fontname> <character> <left> <top> <right> <bottom> <pagenum>
 4
mf <number of features>
<x> <y> <length> <dir> 0 0
...
cn 1
<ypos> <length> <x2ndmoment> <y2ndmoment>
if <number of features>
<x> <y> <dir>
...
tb 1
<bottom> <top> <width>
```

Abbildung 5.7: Format der Merkmalen eines Schriftzeichens[33]

```
FAIL!
APPLY_BOXES: boxfile line 379/国 ((1138,4283),(1180,4327)): FAILURE! Couldn't find a matching blob
FAIL!
APPLY_BOXES: boxfile line 402/围 ((2364,4277),(2404,4321)): FAILURE! Couldn't find a matching blob
FAIL!
APPLY_BOXES: boxfile line 461/国 ((2462,4215),(2505,4259)): FAILURE! Couldn't find a matching blob
FAIL!
APPLY_BOXES: boxfile line 509/图 ((1463,4157),(1504,4202)): FAILURE! Couldn't find a matching blob
FAIL!
APPLY_BOXES: boxfile line 544/因 ((188,4102),(228,4145)): FAILURE! Couldn't find a matching blob
FAIL!
APPLY_BOXES: boxfile line 707/图 ((2462,3967),(2503,4011)): FAILURE! Couldn't find a matching blob
FAIL!
APPLY_BOXES: boxfile line 738/固 ((636,3913),(678,3958)): FAILURE! Couldn't find a matching blob
FAIL!
APPLY_BOXES: boxfile line 993/图 ((1561,3661),(1602,3705)): FAILURE! Couldn't find a matching blob
APPLY_BOXES:
   Boxes read from boxfile:    1093
   Boxes failed resegmentation:       8
```

Abbildung 5.8: Fehlermeldung beim Training des Erkennungsmoduls

Wenn alle während des Trainings erstellten Dateien verbunden sind, wird die Datei des Erkennungsmoduls erstellt.[33]

5.3 Programmierung

Die programmtechnische Umsetzung der im vorangegangenen Kapitel beschriebenen Bilderkennung sowie Fehlererkennung wird mit der Software PyCharm realisiert. PyCharm ist eine integrierte Entwicklungsumgebung für die Programmiersprache Python. Mithilfe der Darstellung des Programmablaufs wird das Programm erstellt.

Wiederholungsanalyse

Vor der Bildverarbeitung sollte zuerst eine Wiederholungsanalyse (Abbildung 5.9) durchgeführt werden, um die wiederholenden Screenshots zu filtern. Die Screenshots mit einem Bildschirmnamen werden in einen Ordner gelegt und die Dateinamen werden mit Bildschirmnamen sowie Reihenfolgenummer bezeichnet. Ein Screenshot $images[i]$ und der darauffolgende Screenshot $images[i+1]$ werden gleichzeitig ins Programm eingelesen. Die chinesischen Teile beider Screenshots werden danach extrahiert und anschließend in Graustufenbilder umgewandelt. Zum schnellen und groben Vergleich werden die beiden Bilder danach auf eine Größe von 8*8 Pixeln komprimiert. Die Ähnlichkeit der beiden komprimierten Bilder wird durch die Methode struktureller Ähnlichkeit (SSIM, engl. structural similarity) geschätzt. Diese Funktion kehrt einen

Index zurück: die Ähnlichkeitswerte k im Bereich von 0 bis 1. Je größer der Index k ist, ist, desto ähnlicher sind die Bilder. Solange die Ähnlichkeitswerte kleiner als 0,99 sind, wird das Bildverarbeitungsverfahren für beide Screenshots durchgeführt. Ansonsten wird die Verarbeitung des Screenshots $images[i+1]$ übersprungen.

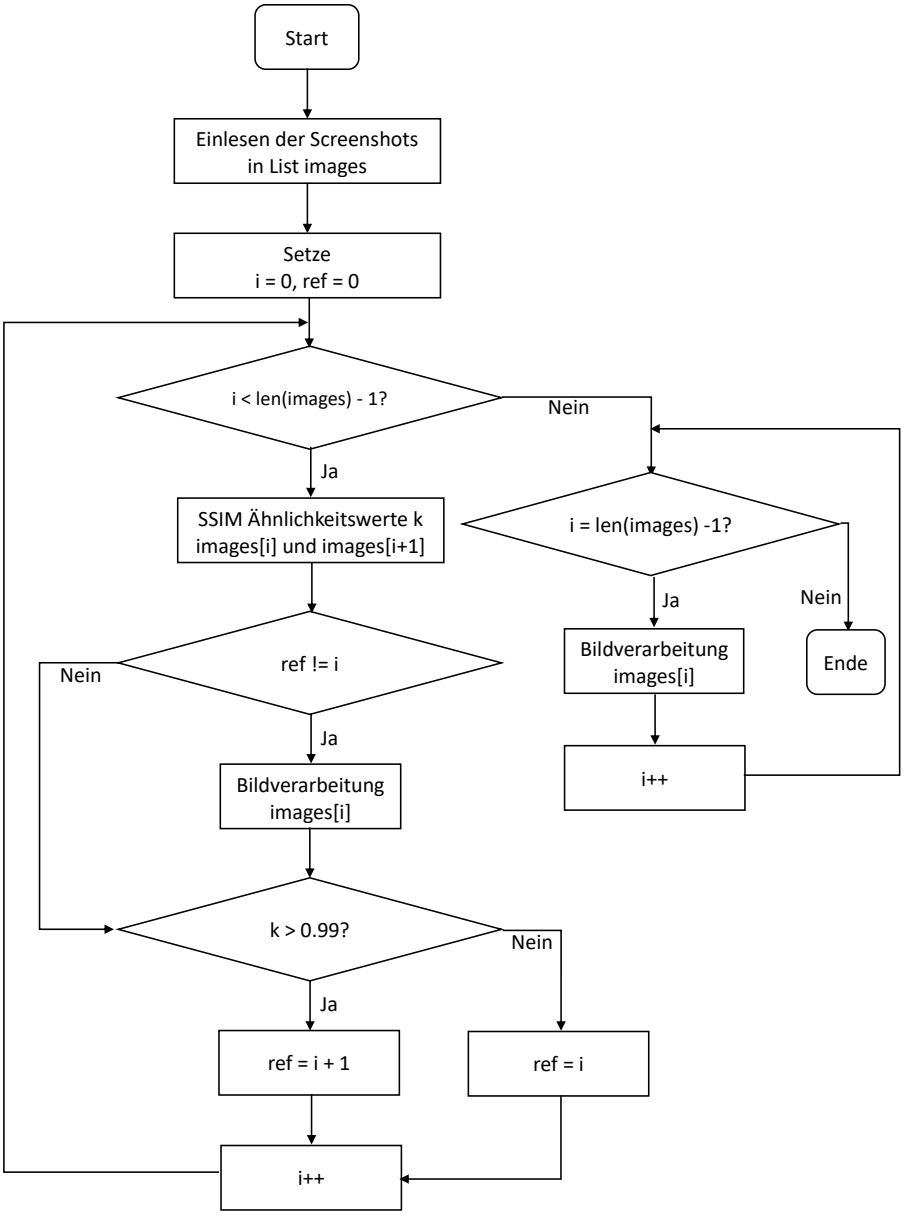

Abbildung 5.9: Ablaufdiagramm für die Wiederholungsanalyse der Screenshots

Nach der Bildverarbeitung werden die Kontureninformationen ermittelt. Dafür wird ein vorläufiges Bild für jeden Textbereich erzeugt. Die Texterkennung wird mithilfe des trainierten Erkennungsmoduls in diesem temporären Bild durchgeführt. Die erkannten Texte werden zusammen mit deren Koordinaten- und Größeninformationen (x, y, w, h) in einer CSV-Datei gespeichert.

Fehlererkennung

Anhand der Konzeption von Fehlererkennung wird eine Beispielimplementierung im Programm realisiert. Die Fehlererkennung wird in zwei Richtungen durchgeführt: entweder mit Ist-Texten als Referenz oder mit Soll-Texten als Referenz. Der entsprechende Prozess ist im Ablaufdiagramm von Abbildungen 5.10, 5.11 visualisiert.

Vor allem sind die Soll-Texte und Ist-Texte bereitzustellen. Zwei Listen werden für die Soll-Texte a und Ist-Texte b erstellt. Aus der XML-Datei werden die Soll-Texte entnommen. Jeder Soll-Text wird als ein Element der Soll-Liste gespeichert. Um doppelt abzusichern, dass die Texte der sich wiederholenden Screenshots nicht zweimal erkannt werden, filtert das Programm vor der Extrahierung von Ist-Texten noch einmal die Ergebnisse der Texterkennung. Die Texte, die die gleichen Koordinaten und Größe haben, werden nur einmal mit den Koordinaten- und Größeninformationen in Datei Ist-Element gespeichert. Danach werden die Ist-Texte ebenfalls aus der Ist-Element-Datei extrahiert. Die von jedem Textbereich erkannten Texte werden als ein Element der Ist-Liste eingelesen. Bei der Fehlererkennung werden alle Elemente beider Listen eins zu eins verglichen.

Wenn die Ist-Texte die Referenz bei der Fehlererkennung sind, vergleicht das Programm jedes Element in Ist-Liste $b[i]$ mit allen Elementen in Soll-Liste $a[j]$. Wenn die beiden Elemente $b[i]$ und $a[j]$ identisch sind, schätzt das Programm, wie die Soll-Texte richtig auf dem Bildschirm angezeigt werden. Das Programm fügt die richtigen Ist-Texte in die Liste $correct$ hinzu. Die Fehlerschätzung für dieses Ist-Element endet hier und das Programm fängt die Schätzung für das nächste Element $b[i+1]$ an. Falls die beiden Elemente nicht identisch sind, wird ein weiterer Vergleich ausgelöst. Die Texte werden dann in einzelne Schriftzeichen aufgeteilt und es wird ein Vergleich zwischen den Schriftzeichen in $b[i]$ und $a[j]$ durchgeführt. Nach dem Vergleich berechnet das Programm die Ähnlichkeit $siml$ anhand des Elements $b[i]$.

$$siml_{\mathrm{b[i]a[j]}} = \frac{len(richtige\ Schriftzeichen)_{\mathrm{b[i]a[j]}}}{len(alle\ Schriftzeichen)_{\mathrm{b[i]}}}. \tag{5.1}$$

Das Element $b[i]$ vergleicht sich dann mit dem nächsten Element in der Soll-Liste $a[j+1]$:

$$siml_{\mathrm{b[i]a[j+1]}} = \frac{len(richtige\ Schriftzeichen)_{\mathrm{b[i]a[j+1]}}}{len(alle\ Schriftzeichen)_{\mathrm{b[i]}}}. \tag{5.2}$$

Falls die Ähnlichkeit $siml_{\mathrm{b[i]a[j+1]}}$ größer als $siml_{\mathrm{b[i]a[j]}}$ ist, erneuert das Programm die Variable $similarity$ und ref_text. Wenn es keine gleichen Elemente in der Ist- und Soll-Liste gibt, wird für jedes Element $b[i]$ ein Element in der Soll-Liste gefunden, das die höchste Ähnlichkeit $similarity$ mit $b[i]$ aufweist. Die entsprechenden Soll-Texte ref_text werden ebenso gespeichert. as Programm schreibt dann jedes Element und seine $similarity$ und ref_text in die Liste $wrong$. Nachdem alle Elemente in der Ist-Liste mit den Elementen in der Soll-Liste verglichen wurden, gibt das Programm die Ergebnislisten $correct$ und $wrong$ aus. Diese Ergebnisse stellen dar:

- welche Elemente in der Ist-Liste identisch zu denen in der Soll-Liste sind;
- zu welchen Elementen in der Ist-Liste keine entsprechenden Elemente in der Soll-Liste gefunden werden können. Gleichzeitig zeigt das Programm das ähnlichste Element in der Soll-Liste und die Ähnlichkeitswerte an.

Umgekehrt wird die Fehlererkennung in die andere Richtung durchgeführt, sodass die Soll-Texte die Referenz sind. In diesem Fall findet das Programm heraus, welche Texte nur in der Soll-Liste aber nicht in der Ist-Liste auftreten. Die Logik und die programmtechnische Umsetzung dieses Verfahrens ähnelt den bereits oben beschriebenen. Der Unterschied liegt lediglich in der Referenzliste. Die Ähnlichkeit *similarity* wird mithilfe der Soll-Liste *a* berechnet. Die *ref_text* besteht hier aus Ist-Texten. Die Ausgabeergebnisse sind ebenfalls die Elemente der Soll-Liste. Diese Ergebnisse stellen dar:

- welche Elemente in der Soll-Liste in Screenshots nicht gefunden werden. Gleichzeitig zeigt das Programm das ähnlichste Element aus der Ist-Liste und die entsprechenden Ähnlichkeitswerte an.

$$simil_{\text{a[i]b[j]}} = \frac{len(richtige\ Schriftzeichen)_{\text{a[i]b[j]}}}{len(alle\ Schriftzeichen)_{\text{a[i]}}}. \tag{5.3}$$

Abschließend wird eine Auszählung wiederholter Ergebnisse durchgeführt, die ergibt, wie viele Male sich die Texte in Screenshots wiederholen.

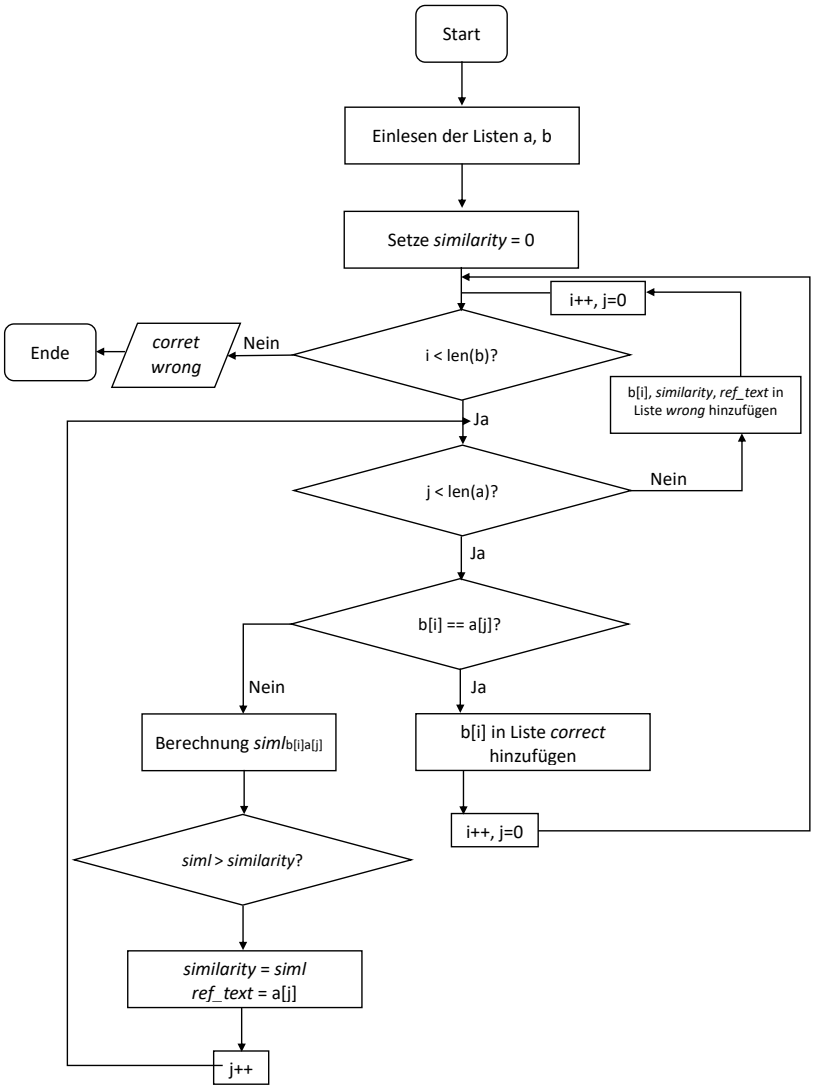

Abbildung 5.10: Ablaufdiagramm für die Fehlererkennung (Ist-Texte als Referenz)

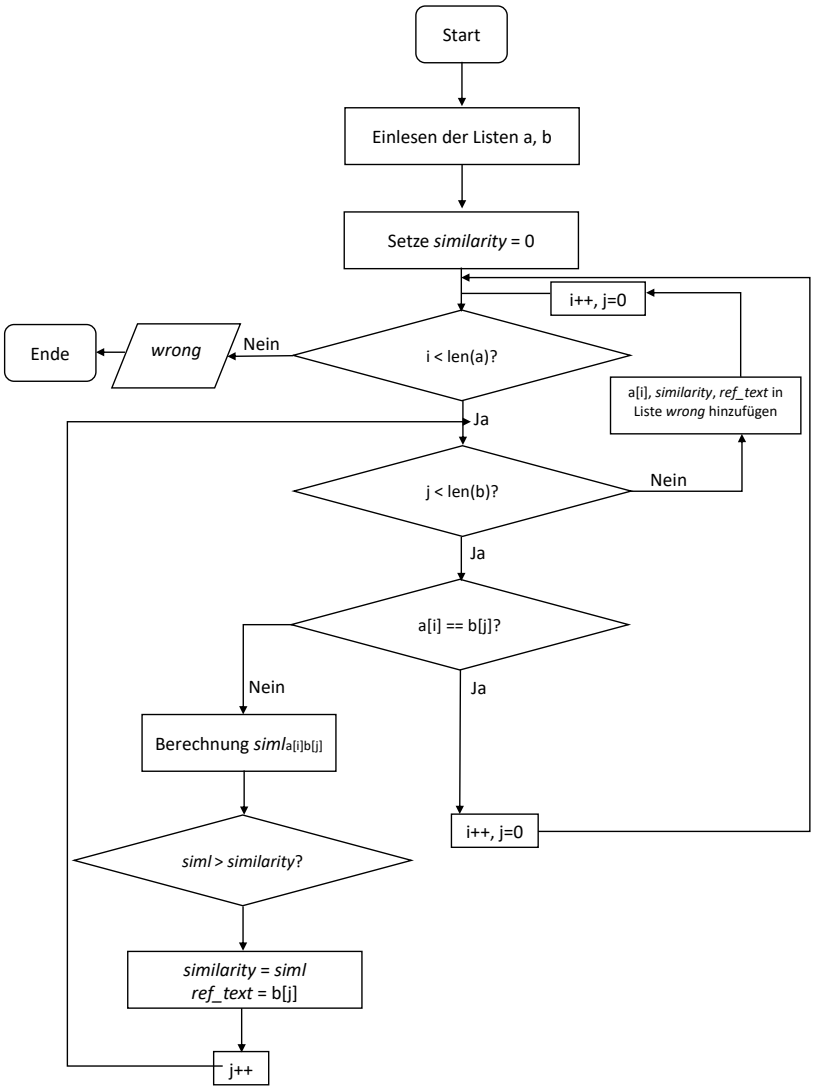

Abbildung 5.11: Ablaufdiagramm für die Fehlererkennung (Soll-Texte als Referenz)

5.4 Validierung des Algorithmus zur automatischen Fehlererkennung

Zur Überprüfung der automatischen Fehlererkennung wird zunächst das Bildverarbeitungsverfahren getestet. Durch die Visualisierungsfunktion ‚imshow' im OpenCV werden die Ergebnisbilder jedes Schrittes angezeigt. Nach einer reibungslosen Verarbeitung werden die Textbereiche automatisch lokalisiert und die Texte werden automatisch erkannt und gespeichert.

Nach der Implementierung und Einbindung des Programms wird das vollständige Programm getestet. Nachdem die Quelle der Screenshots, die Soll-Datei und das Erkennungsmodul manuell zur Verfügung gestellt wurden, können die Screenshots durch das Programm verarbeitet und die Texte erkannt und analysiert werden. Die erwarteten Ausgabedateien werden erfolgreich generiert. Während des Tests wird auch die Verarbeitungszeit zur Evaluierung der Effizienz ermittelt.

Abbildung 5.12 zeigt das Ergebnis eines Tests anhand einer Gruppe von Screenshots unter einem Bildschirmnamen. Die Ergebnisse werden in drei Gruppen aufgeteilt. Die erste Gruppe zeigt die Texte, die als richtig eingeschätzt wurden. Solche Texte wurden sowohl in der Soll-Datei als auch in der Ist-Datei gefunden. Die zweite Gruppe zeigt die Ist-Texte, die nicht in der Soll-Datei erwartet wurden. Die drei Ist-Elemente mit der höchsten Ähnlichkeit weisen maximal 20 % Ähnlichkeit mit einem Soll-Element auf. Möglicherweise werden hier nicht entfernte Bilder oder ‚Teiltexte' erkannt (‚Teiltexte' bezeichnet die Texte, von denen nur ein Oberteil oder Unterteil im Screenshot angezeigt wurde). Die dritte Gruppe beinhaltet die Soll-Texte, die nicht in Screenshots gefunden wurden. Das bedeutet, dass die Soll-Texte stehen nicht in Ist-Liste. Beispielsweise ist das Soll-Element ‚ausgewogen' ein deutsches Wort und es wird nicht in Screenshots erkannt und deshalb nicht in die Ist-Liste hinzugefügt. In der chinesischen Version ist das Wort nicht erwartet. Es wird demnach als falsche Soll-Information betrachtet.

```
==========================================================
Correct:

经济  found 1 times
舒适  found 4 times
个性化设置  found 3 times
悬挂  found 2 times
转向  found 3 times
安静  found 1 times
当前  found 1 times
自动  found 1 times
运动  found 5 times
车辆  found 1 times
平衡  found 5 times
发动机声音  found 1 times
驾驶  found 1 times

==========================================================
Recognized texts from screenshots are not found in the xml file:

口工2柱
(similarity: 0)  found 1 times
带开力加宙音
(similarity: 0.166666666667)  found 1 times
2单Av纸
(similarity: 0.2)  found 1 times

==========================================================
Test cases in the xml file are not found in screenshots:

Audi 驾驶模式选择
(similarity: 0.0909090909091)
ausgewogen
(similarity: 0)
komfortabel
(similarity: 0)
ausgewogen
(similarity: 0)

processing time: 7.520642s
```

Abbildung 5.12: Beispielanzeige für Fehlererkennungsergebnisse

5.5 Auswertung der Ergebnisse

Beim Test sind insgesamt 297 Screenshots unter verschiedenen Bildschirmnamen von Menü ‚CAR' vorhanden. Sie werden alle ins Programm eingegeben und verarbeitet. Die entsprechenden Informationen über die Fehlererkennung werden gesammelt. Abbildung 5.13 zeigt die Ergebnisinformationen der Praxistests. Die linke dunkelblaue Säule zeigt die Ergebnisinformationen des originalen Erkennungsmoduls der Engine Tesseract ‚chi_sim'; die rechte hellblaue Säule gibt die Ergebnisinformationen von dem in dieser Arbeit trainierten Erkennungsmodul ‚chi_test' an.

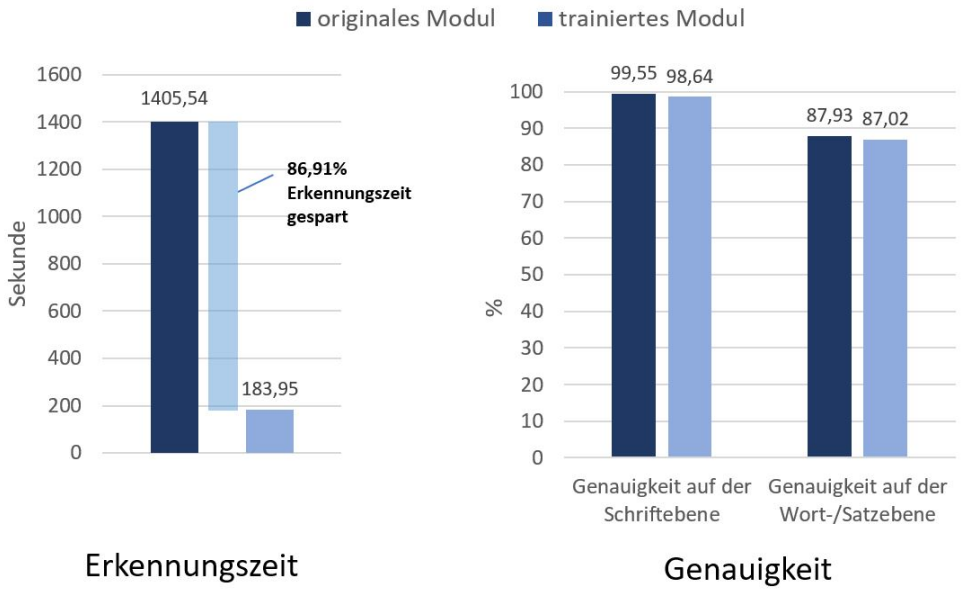

Abbildung 5.13: Ergebnisinformationen bei der Praxistests

Aus der linken Grafik wird deutlich, dass die Geschwindigkeit des Moduls ‚chi_test' deutlich schneller als die von ‚chi_sim' ist. Die Gesamtzeit für alle Screenshots beträgt 183,95 Sekunden und die durchschnittliche Verarbeitungszeit für einen Screenshot beträgt 0,62 Sekunden, während das Erkennungsmodul ‚chi_sim' eine Gesamtzeit von 1405,54 Sekunden (durchschnittliche Verarbeitungszeit von 4,73 Sekunden) hat. Die Erkennungszeit wird um 87 % reduziert. Da nur eine begrenzte Menge chinesischer Schriftzeichen im Infotainmentsystem erforderlich ist, sind solche unterschiedlichen Schriftzeichen nur einen kleinen Teil vollständiger chinesischer Schriftzeichen. Die Geschwindigkeitssteigerung ist dementsprechend der Verkleinerung der Musterdatenbank zu verdanken.

Die Genauigkeit des Erkennungsmoduls lässt sich in Genauigkeit der Schriften und Genauigkeit der Wörter/Sätze beurteilen. Laut der offiziellen Unterlagen von Tesseract hat das Modul ‚chi_sim' eine Fehlerquote von 8,25 % auf der Schriftebene und 10,82 % auf der Wort-/Satzebene[34]. Umgerechnet ist die Genauigkeit des Moduls ‚chi_sim' 91,75 % auf der Schriftebne und 89,18 % auf der Wort/Satzebene. Da Tesseract die Konfidenz jedes Schriftzeichens in den Texterkennungsergebnissen nicht vorlegen kann, wird die Bewertung der Genauigkeit in dieser Arbeit auf Basis der Ähnlichkeit *similarity* im Abschnitt 5.3 im Programm realisiert. Es gibt keinen großen Unterschied zwischen der Genauigkeit der Module ‚chi_sim' und ‚chi_test'. Beide Module beweist gute Performance auf der Schriftebene mit der Genauigkeit 99,55 % und 98,64%. Auf der Wort-/Satzebene werden die Erkennungsergebnisse von meheren Faktoren beeinflusst, zum Beispiel die Qualität der Layout-Analyse oder die Begrenzungen der Engine.

Es ist auffällig, dass einige Ergebnisse der Texterkennung von den Soll-Texten abweichen. Dieses Problem tritt nicht nur mit dem Modul ‚chi_test', sondern auch mit ‚chi_sim' auf. Wie im Abschnitt Training des Erkennungsmoduls erwähnt, haben manche Schriftzeichen ähnliche Umrisse und Größe, weswegen die Tesseract-Engine ihre Merkmale nicht unterscheiden kann. Bei der Erkennung solcher Schriftzeichen kann Tesseract die Muster nicht erkennen und die Schriftzeichen in Folge nicht klassifizieren. Außerdem hat Tesseract keine stabile Performance bei der Erkennung von Text aus einzelnem Schriftzeichen oder bei der Erkennung von langen Sätze. In einige Fällen kann Tesseract solche einzelne Schriftzeichen oder lange Sätze 100 Prozent richtig erkennen. Aber in andere Fällen werden solche einzelne Schriftzeichen als anders betrachtet oder die Sätze werden teilweise von Tesseract falsch erkannt. Die Fehler können leider nicht vorhergesagt werden.

Außerdem wäre eine intelligentere Methode zur Layout-Analyse notwendig, denn die Bildverarbeitung der Screenshots konnte nicht in allen Fällen die Textbereiche zu 100 Prozent richtig finden. Aus dem Infotainmentsystem können tausende Screenshots erstellt werden. Die daraus resultierende Kombination von Komponenten in Screenshots wie Texten und Bildern ist dabei sehr umfangreich. Die allgemeinen Bildverarbeitungsverfahren sind nicht für alle Layout-Strukturen geeignet, z. B. sind die Parameter von Erosion auf bestimmte Werte festgelegt. In einigen Fällen werden die kleinen Bilder vor dem ersten Schriftzeichen eines Textbereichs während der Erosion mit dem Textbereich verbunden, weswegen dieses Bild auch als Textbereich eingestuft wird. Wegen der zeitlichen Beschränkung kann eine effizientere Layout-Analyse leider nicht entwickelt werden. Aber es kann eine Sonderfunktion ins Programm eingefügt werden, durch die bei dem oben genannten Fall die Texte auch als richtig betrachtet werden können. Es wird trotzdem empfohlen, in Zukunft die Layout-Analyse zu optimieren. Es wäre auch möglich, die Textbereiche auf andere Art und Weise vor Durchführung dieses Programms zu lokalisieren.

6 Zusammenfassung und Ausblick

Die vorliegende Arbeit beschäftigt sich mit der Entwicklung und Implementierung eines automatisierten Fehlererkennungsprozesses für die Absicherung der HMI-Textanzeige. Eine Literaturrecherche zu den Bilderkennungstechniken im Kontext mit ‚künstlicher Intelligenz' und ‚Digitalisierung' wird vor allem durchgeführt. Außerdem wird eine Klassifikation der Methoden und Modellen von maschinellem Lernen dargestellt. Aufgrund davon lassen sich zwei Hauptaufgaben dieser Arbeit identifizieren. Zum einen ist ein Fehlererkennungssystem zu entwickeln, das die Screenshots aus dem Infotainmentsystem verarbeitet und die Texterkennungsergebnisse prüft. Zum anderen muss ein spezifisch zu dem Infotainmentsystem passendes Erkennungsmodul auf Basis von Theorie des maschinellen Lernens entwickelt werden.

Die Klassifikation von Methoden und den zugehörigen Modellen des maschinellen Lernens ist anhand von zwei Aspekten durchgeführt, nämlich Lernstil und Lernaufgaben. Die zwei Merkmale sind in Bezug auf die Anforderungen und bieten theoretische Unterstützung zur Analyse der Aufgaben und Entscheidung zur Lernstrategien. Da neue Modelle von maschinellem Lernen stetig entwickelt und auf Basis von alten Algorithmen weiterentwickelt werden, wurde nicht jedes Modell in der Klassifikation analysiert. Trotzdem werden die typischen Modelle jeder Methode in dieser Arbeit so ausgewählt, dass diese Modelle als grundlegende Lösung zur Aufgaben dienen können.

Durch die Fehlererkennungsalgorithmus ist das System möglich, die Bildeigenschaften der Screenshots zu ändern und die erforderlichen Textbereiche zu lokalisieren. Dabei werden variable Einstellungen von Parametern wie Schwellenwert berücksichtigt. Nach der sogenannten Bildverarbeitung für die einzelnen Screenshots werden die erkannten Texte mit den Soll-Texten verglichen und die Ergebnisse dazu dargestellt. Darüber hinaus wurde ein neues Texterkennungsmodul entwickelt. Dieses Erkennungsmodul ist verwendet bei Erkennung der Texten und hat die Erkennungsgeschwindigkeit der OCR-Engine Tesseract für das Infortainmentsystem deutlich erhöht. Nach der Implementierung des Algorithmus in einem Programm werden Tests in Verbindung mit den Validierungsscreenshots durchgeführt, dadurch wurden Hinweise zur Optimierung der Parameter im Programm zusammenfasst und implementiert. Weiterhin wurden die restlichen Screenshots zum Testen ins Programm eingegeben. Eine Bewertung für die Testergebnisse wurden abschließend erstellt.

Die durchgeführten Tests haben die Funktionsweise des Algorithmus für die Bilderkennung sowie das Zusammenspiel von Bilderkennung und Fehlererkennung erfolgreich wiedergegeben. Mit dem implementierten Programm ist es in der Lage, die in Screenshots beinhaltenden Texten mit den bei der HMI-Absicherung erwarteten Texten automatisch zu vergleichen. Die Genauigkeit der Erkennungsergebnis wird jedoch durch die Begrenzung der ausgewählten Open-Source-Engine Tesseract in gewissem Maße begrenzt, da Tesseract wegen ihrer Funktionsweise bei der Merkmalextrahierung einige bestimmten chinesischen Schriftzeichen nicht klassifizieren kann.

Nachfolgende Arbeiten lassen sich für die automatische Fehlererkennung in anderen Sprachen erweitern. Mit der Dokumentation von Trainingsverfahren eines Texterkennungsmoduls ist es realistisch, die Trainingsbeipsiele und Wörterbücher für eine andere Sprache und dementsprechendes Erkennungsmodul zu erstellen. Für lateinische Sprachen stellen Tesseract schon Erkennungsmodule mit hohen Effizienz zur Verfügung. Das Training von Erkennungsmodulen für nicht lateinischen Sprachen wie Koreanisch und traditionelles Chinesisch ist gefordert und von größeren Bedeutung.

Ferner können die Einzelnen Funktionen des Algorithmus in künftigen Arbeiten als Schnittstelle dienen und in andere Programmen eingesteckt werden. Die Wiederholungsanalyse von Screenshots können in allen Situationen verwendet, in denen die Screenshots generiert werden. Eine reine Bilderkennung ohne Fehlererkennung ist auch von Bedeutung. Anderenfalls kann das Erkennungsmodul zur Texterkennung verwendet, wenn die Textbereiche schon in Screenshots festgelegt werden. Das vermeidet die Beschränkung von Qualitäten der Layout-Analyse und ermöglicht eine bessere Erkennungsqualität.

Literatur

[1] Audi AG. *Anzeigen*. 2017.
URL: https://www.audi-mediacenter.com/de/technik-lexikon-7180/anzeigen-7181.

[2] Audi AG. *Bedienung*. 2017.
URL: https://www.audi-mediacenter.com/de/technik-lexikon-7180/bedienung-7182.

[3] Continental AG. *Infotainment Systeme*. 2018. URL: https://www.continental-automotive.com/de-DE/Passenger-Cars/Interior/Infotainment-Systems.

[4] Hans-Jürgen Andreß, Jacques A Hagenaars und Steffen Kühnel.
Analyse von Tabellen und kategorialen Daten: log-lineare Modelle, latente Klassenanalyse, logistische Regression und GSK-Ansatz. Springer-Verlag, 2013.

[5] Björn Böttcher, Daniel Klemm und Dr. Carlo Velten. *Machine Learning im Unternehmenseinsatz*. Crisp Research AG, Jan. 2017.

[6] Wilhelm Burger und Mark James Burge.
Digitale Bildverarbeitung: Eine algorithmische Einführung mit Java. Springer-Verlag, 2009, S. 10–12.

[7] Wolfie Christl. "Kommerzielle digitale Überwachung im Alltag".
In: *Studie im Auftrag der österreichischen Bundesarbeitskammer* (2014).

[8] The Economist. *Language: Finding a voice*. 2017.
URL: http://www.economist.com/technology-quarterly/2017-05-01/language.

[9] Jasmin Fischer. "Support Vector Machines (SVM)". In: 2007.

[10] Günther Goerz und Josef Schneeberger. *Handbuch der Künstlichen Intelligenz*.
Walter de Gruyter, 2003.

[11] Ian Goodfellow, Yoshua Bengio und Aaron Courville. *Deep Learning*.
http://www.deeplearningbook.org. MIT Press, 2016.

[12] Roland Heuermann, Andreas Engel und Jörn von Lucke.
"Digitalisierung: Begriff, Ziele und Steuerung".
In: *Digitalisierung in Bund, Ländern und Gemeinden*. Springer, 2018, S. 9–50.

[13] Martin Hilbert und Priscila López.
"The world's technological capacity to store, communicate, and compute information".
In: *science* 332.6025 (2011), S. 60–65.

[14] Evan Hirsh, Akshay Singh und Reid Wilk. *2015 Auto Industry Trends*. 2015.
URL: https://www.strategyand.pwc.com/trends/2015-auto-trends.

[15] Hendrik Horn und Jörgen Kosche. "Segmentierung von Bilddaten". In: *Seminarvortrag, Institut für Informationsverarbeitung und Kommunikation, Universität Potsdam* (2004).

[16] Henning Kagermann. "Change through digitization—Value creation in the age of Industry 4.0".
In: *Management of permanent change*. Springer, 2015, S. 23–45.

[17] Henning Kagermann u. a.
Recommendations for implementing the strategic initiative INDUSTRIE 4.0: Securing the future of German manufacturing industry; final report of the Industrie 4.0 Working Group.
Forschungsunion, 2013.

[18] Ron Kohavi. "Glossary of terms". In: *Machine Learning* 30 (1998), S. 271–274.

[19] Sotiris B Kotsiantis, I Zaharakis und P Pintelas.
Superised machine learning: A review of classification techniques. 2007.

[20] Sotiris B Kotsiantis, I Zaharakis und P Pintelas.
Supervised machine learning: A review of classification techniques. 2007.

[21] Pat Langley. "The changing science of machine learning".
In: *Machine Learning* 82.3 (2011), S. 275–279.

[22] Jan Lunze. *Künstliche Intelligenz für Ingenieure: Methoden zur Lösung ingenieurtechnischer Probleme mit Hilfe von Regeln, logischen Formeln und Bayesnetzen*.
Walter de Gruyter GmbH & Co KG, 2016.

[23] Peter Mell, Tim Grance u. a. "The NIST definition of cloud computing". In: (2011).

[24] Mehryar Mohri, Afshin Rostamizadeh und Ameet Talwalkar. *Foundations of machine learning*.
MIT press, 2012.

[25] Shunji Mori, Ching Y Suen und Kazuhiko Yamamoto.
"Historical review of OCR research and development".
In: *Proceedings of the IEEE* 80.7 (1992), S. 1029–1058.

[26] Lawrence O'Gorman. "The document spectrum for page layout analysis".
In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15.11 (1993), S. 1162–1173.

[27] OpenCV. *Machine Learning Overview*. 2015.
URL: `https://docs.opencv.org/3.1.0/dc/dd6/ml_intro.html`.

[28] Gerhard Pahl und Wolfgang Beitz. *Konstruktionslehre: Methoden und Anwendung*.
Springer-Verlag, 2013.

[29] Fabian Pedregosa u. a. "Scikit-learn: Machine learning in Python".
In: *Journal of machine learning research* 12.Oct (2011), S. 2825–2830.

[30] Andreas Riener u. a. "Workshop Automotive HMI".
In: *Mensch und Computer 2016–Workshopband* (2016).

[31] Andreas Ruckstuhl. *Einführung in die nichtlineare Regression*. 2008.

[32] Scikit-learn. *Scikit-learn 0.19.1 (stable) documentation*. 2017.
URL: `http://scikit-learn.org/stable/modules/naive_bayes.html`.

[33] Nisarg Shah. *Training Tesseract*. Sep. 2017.
URL: `https://github.com/tesseract-ocr/tesseract/wiki/Training-Tesseract`.

[34] Ray Smith. *Tesseract blends old and new OCR technology*. 2016.

[35] Ray Smith und Google Inc. "An Overview of the Tesseract OCR Engine". In: (Sep. 2007).

[36] Ray Smith u. a. *Adapting the Tesseract Open Source OCR Engine for Multilingual OCR*.
Barcelona, Spain, Juli 2009.

[37] Solomennikm. *Training Tesseract – Make Box Files*. Feb. 2017.
URL: `https://github.com/tesseract-ocr/tesseract/wiki/Training-Tesseract-%E2%80%93-Make-Box-Files`.

[38] Richard S Sutton und Andrew G Barto. *Reinforcement learning: An introduction*. Bd. 1. 1. MIT press Cambridge, 1998.

[39] opencv dev team. *OpenCV 2.4.13.5 documentation*. 2018. URL: https://docs.opencv.org/2.4/doc/tutorials/introduction/display_image/display_image.html.

[40] Team Clusteranalyse an der Technischen Universität München. *Clustermethoden*. 2009. URL: http://www-m9.ma.tum.de/material/felix-klein/clustering/Methoden/Methoden.php.

[41] TensorFlow. *About TensorFlow*. 2018. URL: https://www.tensorflow.org/.

[42] *TESSERACT(1) Manual Page*. URL: https://github.com/tesseract-ocr/tesseract/blob/master/doc/tesseract.1.asc#languages.

[43] Strother H. Walker und David B. Duncan. "Estimation of the Probability of an Event as a Function of Several Independent Variables". In: *Biometrika* 54.1/2 (1967), S. 167–179. ISSN: 00063444. URL: http://www.jstor.org/stable/2333860.

[44] Yiming Yang und Jan O. Pedersen. *A Comparative Study on Feature Selection in Text Categorization*. 1997.