Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Automatische Erkennung von Textelementen in Bildschirmaufnahmen für die Eye Tracking Datenanalyse

Automatically Detecting Text Elements in Screen Recordings for Analysis of Eye Tracking Data

Bachelorarbeit

im Studiengang Informatik

von

Sebastian Eggers

Prüfer: Prof. Dr. Kurt Schneider
Zweitprüfer: Dr. Jil Klünder
Betreuer: Maike Ahrens

Hannover, 12.08.2021

Erklärung der Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und ohne fremde Hilfe verfasst und keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet habe. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 12.08.2021

_____

Sebastian Eggers

Zusammenfassung

Requirements Traceability ist ein Werkzeug des Requirements Managements, welches häufig viel zu wenig Beachtung erfährt. Dies ist auf den hohen Aufwand zurückzuführen, der mit dem Aufrechterhalten der Trace Links einhergeht. Deshalb ist es wünschenswert Aufgaben wie das Aufrechterhalten und Wiederherstellen als auch das Erstellen von Trace Links zu automatisieren. Dabei reichen Ansätze, die mit Information Retrieval Methoden arbeiten, auf Grund ihrer häufig niedrigen Relevanz und Sensitivität nicht aus. Aus diesem Grund sollen diese Tätgkeiten mit Hilfe von Eye Tracking automatisiert werden. Die manuelle Auswertung von Eye Tracking Analysen ist allerdings ebenso aufwändig, denn es müssen die Areas of Interest (AOI) manuell eingetragen werden. Mit der in dieser Arbeit vorgestellten Software soll ein Schritt in Richtung der automatisierten Auswertung von Eye Tracking Analysen im Kontext von Requierements Traceability gemacht werden. Dabei wird Text in Bildschirmaufnahmen von Eye Tracking Analysen erkannt und in ein maschinenlesbares Format gebracht. Aus diesem Text entstehen AOIs. Diese AOIs werden über die gesamte Länge des Videos verfolgt und Veränderungen von diesen werden vermerkt. In einem letzten Schritt werden diese AOIs mit den Daten der Eye Tracking Analyse abgeglichen und es wird vermerkt, ob Fixationen innerhalb der Bereiche der AOIs stattgefunden haben.

Abstract

Automatically Detecting Text Elements in Screen Recordings for Analysis of Eye Tracking Data

Requirements traceability is a requirements management tool that often receives far too little attention. This is due to the high effort involved in maintaining trace links. Therefore, it is desirable to automate tasks such as maintaining and restoring but also creating trace links. In this context, approaches using information retrieval methods are not sufficient due to their often low precision and recall values. For this reason, these activities are to be automated with the help of eye tracking. However, the manual evaluation of eye tracking analyses is just as time-consuming, because the areas of interest (AOI) have to be entered manually. The software presented in this thesis is a step towards the automated evaluation of eye tracking analyses in the context of Requierements Traceability. Text in screen recordings of eye tracking analyses is recognized and converted into a machine-readable format. AOIs are created from this text. These AOIs are tracked over the entire length of the video and changes to them are noted. In a final step, these AOIs are compared with the data from the eye tracking analysis and it is noted whether fixations have taken place within the range of the AOIs.

Inhaltsverzeichnis

Kapitel 1

Einleitung

In der Softwareentwicklung gibt es viele Fallstricke, die dazu führen können, dass ein Projekt scheitert. Der Grund für das Scheitern kann in der Organisationsstruktur oben im Management angesiedelt sein oder aber viele Ebenen tiefer liegen. Ein Beispiel dafür ist schlechtes Requirements Engineering, was die Bedeutung des Gebiets für die Softwareentwicklung erklärt [20, 24]. Der CHAOS Report 1995[1] führt beispielsweise fünf der zehn häufigsten Gründe für das Scheitern von Projekten auf schlechtes Requirements Engineering zurück. Zum Beispiel kann es sein, dass ein Projekt das vorgegebene Zeitlimit nicht einhalten kann, weil Anforderungen von Entwicklern fehlinterpretiert wurden. Dies führt dazu, dass nachgebessert werden muss, weil der Kunde mit der Software nicht zufrieden ist [35]. Um solche Fehler zu vermeiden, sollte eine Organisation um gutes Requirements Engineering bemüht sein. Bei dem Requirements Engineering auf Projekt-Ebene geht es um alles was mit dem Erheben, Dokumentieren und Managen von Anforderungen zu tun hat [25, 35]. Besonders die *Requirement Traceability* (Rückverfolgbarkeit von Anforderungen) ist dabei ein wichtiger Bestandteil des Requirements Engineerings, bei dem es um die Fähigkeit geht den Lebenszyklus einer Anforderung in Vorwärts- und Rückwärtsrichtung beschreiben und verfolgen zu können, so Gotel et al. [11]. Es werden dabei unterschiedliche Objekte mit inhaltlichem Zusammenhang durch *Trace Links* (Verknüpfungen) verbunden. Beispielsweise werden Spezifikationen unterschiedlicher Abstraktionslevel oder Anforderungen mit dem zugehörigen Source-Code verknüpft. Dabei hilft *Requirement Traceability* diese Verbindung von Spezifikationen auf höheren Abstraktionslevel bis hin zum Testfall nachzuvollziehen [24]. Dies findet zum Beispiel in der Sicherheitsanalyse oder Testfall-Auswahl Anwendung. Allerdings wird das Tracing trotz bekannter Vorteile nicht häufig verwendet. Da ein Softwaresystem häufig überarbeitet oder erweitert wird, ist es schwer die *Trace Links* aktuell zu halten [22]. So sind *Information Retrieval* Methoden eine Möglichkeit *Trace Links* zwischen Anforderungen oder auch

[1] Quelle: The Standish Group International, Inc., www.standishgroup.com

zwischen Dokumentation und Source-Code herzustellen und zu aktualisieren. Allerdings sind diese Methoden nicht optimal, da die *Precision* (Relevanz) und *Recall* (Sensitivität) meist gering sind [2, 3]. Eine weitere Methode ist das *Eye Tracking*, also das Aufzeichnen von Sakkaden (Augenbewegungen) und Fixationen (Fixierungspunkte) des menschlichen Auges. Mit *Eye Tracking* kann die Arbeit des Entwicklers im Alltag aufgezeichnet werden. So können beispielsweise neue *Trace Links* durch ein häufiges Betrachten einer Anforderung und der nachfolgenden Bearbeitung einer bestimmten Stelle im Source-Code entstehen, da diese in Folge dieser Ereigniskette in einem Zusammenhang stehen. Außerdem kann *Eye Tracking* dabei helfen *Trace Links* zu verifizieren und aktuell zu halten, da nachvollziehbar ist wie sich diese *Trace Links* während der Arbeit eines Entwicklers verändern [3].

1.1 Problemstellung

Bisher erfordert das Auswerten von Eye Tracking Daten sehr viel manuellen Aufwand. So müssen AOIs (Areas of Interest) manuell in einem Programm für die Analyse von Eye Tracking Daten in den Bildschirmaufnahmen eingetragen werden. Diese AOIs sind zum Beispiel Bereiche in der Aufnahme, an dem eine Anforderung oder Source-Code zu sehen ist. Zwischen diesen Objekten können dann anhand von Blickmustern oder einer häufigen, gemeinsamen Betrachtung Trace Links hergestellt werden. Dies lässt sich in dynamischen Umgebungen wie der Softwareentwicklung allerdings nur unter sehr viel Aufwand betreiben, da im Alltag eines Entwicklers sehr viel zwischen Anforderungen, Dokumentationen, Testfällen und Code-Dateien gewechselt wird. Innerhalb dieser Dokumente kann gescrollt, Text hinzugefügt oder gelöscht werden. All diese Aktionen erzeugen neue oder veränderte AOIs, die manuell eingetragen werden müssen. Mit dieser Arbeit soll ein Schritt in Richtung der automatischen Erstellung und Verfolgung von Trace Links getan werden.

1.2 Lösungsansatz

Um den hohen manuellen Aufwand zu verringern, der bei der Erstellung der AOIs entsteht, sollen diese automatisiert gefunden und verfolgt werden. Dabei werden die Frames der Bildschirmaufnahmen auf Textelemente untersucht. In einem zweiten Schritt werden die gefundenen Textelemente analysiert, sodass der Text in einem maschinenlesbaren Format und die zugehörigen Koordinaten gespeichert und verfolgt werden kann. Aus diesen Textelementen entstehen dann die AOIs. Dabei wird auch abgeglichen, ob die AOI bereits erfasst wurde, eine bereits erfasste AOI nicht mehr sichtbar ist oder bearbeitet wurde. Es wird auch die Differenz zwischen zwei aufeinanderfolgenden Frames berechnet, da eine Detektion und spätere

Erkennung von Textelementen auf jeden Frame sehr zeitaufwendig wäre. In einem letzten Schritt werden die gefundenen AOIs mit den Eye Tracking Daten abgeglichen und es wird vermerkt, ob und zu welchem Zeitpunkt eine AOI angesehen wurde. Die gefundenen AOIs werden in einem Format gesichert, welches dem der manuellen Erstellung mit der Software Tobii Pro Lab gleicht, sodass diese mit weiterer Software, wie zum Beispiel der von Arafat [4], automatisiert verarbeitet werden können.

1.3 Struktur der Arbeit

In Kapitel 2 werden die Grundlagen, welche zum Verständnis der Arbeit beitragen, erläutert. Im folgenden Kapitel 3 wird diese Arbeit von verwandten Arbeiten abgegrenzt. Kapitel 4 handelt von dem zugrunde liegenden Konzept der Software. Es werden dort die einzelnen Schritte von der Eingabe bis zur Ausgabe erläutert. In Kapitel 5 werden die verwendeten Tools und die Architektur der Software beschrieben. Danach wird das Ergebnis in Kapitel 6 einer Evaluation unterzogen. Hier werden besonders Laufzeit und Genauigkeit betrachtet, also wie viele der durch einen Menschen erstellten AOIs durch die Software erkannt werden. Im letzten Kapitel 7 werden die Ergebnisse dieser Arbeit zusammengefasst und es wird ein Ausblick auf mögliche Erweiterungen der Software gegeben.

Kapitel 2

Grundlagen

Dieses Kapitel beinhaltet die für das Verständnis dieser Arbeit notwendigen Grundlagen. Dabei wird zunächst auf den größeren Kontext der Arbeit, der Requirements Traceability, eingegangen. Danach folgen Erklärungen zum Eye Tracking. Um die in dieser Arbeit verwendeten Verfahren für die Texterkennung zu verstehen, werden weiter Konzepte der Künstlichen Intelligenz erklärt. Als letzter Teil dieser Grundlagen wird auf die Berechnung des Unterschiedes zwischen zwei Bildern und das Representational State Transfer (REST) Architekturmodell eingegangen. Das Verständnis für die absolute Differenz zwischen zwei Bildern hilft beim Verständnis des Verfahrens zum Erhöhen der Effizienz in Bezug auf die Laufzeit. Das REST-Architekturmodell wird von dem User Interface im Backend verwendet.

2.1 Requirements Traceability

Requirements Traceability ist ein Werkzeug des Anforderungsmanagements innerhalb des Requirements Engineerings. Gotel et al. definieren Requirements Traceability als Fähigkeit eine Anforderung während ihres gesamten Lebenszyklus in Vorwärts- und Rückwärtsrichtung zu beschreiben und zu verfolgen [11]. Dies ist gerade für die Softwareentwicklung sehr interessant, da sie den Projektbeteiligten einen guten Überblick über Entscheidungen und ihre Hintergründe während der Entwicklung eines Systems bieten [18]. Außerdem ermöglicht die Requirements Traceabality die Verfolgung einer Anforderung von der Spezifikation bis zum Testfall [24]. So werden mit Trace Links zwei Objekte mit inhaltlichem Zusammenhang miteinander verbunden. Über diese Trace Links können beispielsweise Spezifikationen unterschiedlicher Abstraktionslevel miteinander verknüpft werden. Es können auch Anforderungen und deren Überarbeitungen verbunden werden, was beim Verständnis hilft, wie sich eine Anforderung über die Zeit eines Projektes entwickelt. Eine weitere Möglichkeit ist das Verknüpfen von Source-Code mit Anforderungen. Aus diesen Verknüpfungen lassen

sich zum Beispiel Entscheidungen ableiten, die während der Entwicklung in Bezug auf die Anforderungen getroffen wurden [18, 24]. Anwendung findet die Requirements Traceability beispielsweise in der Sicherheitsanalyse von Software [22]. Durch die Requirements Traceability kann sichergestellt werden, dass alle sicherheitskritischen Anforderungen erfüllt wurden, da diese bis zum Testfall verfolgt und überprüft werden können.

2.2 Eye Tracking

Beim Eye Tracking (Blickaufzeichnung) werden Fixationen und Sakkaden des menschlichen Auges aufgezeichnet, um zu analysieren wohin eine Person ihren Blick richtet. Als Fixationen bezeichnet man dabei einen relativen Stillstand des Auges in Bezug auf ein angesehenes Objekt. Das Objekt wird fixiert und es werden Informationen vom Auge aufgenommen. Dabei ist es möglich, dass sich ein fixiertes Objekt bewegt. Dies führt dazu, dass "gleitende Folgebewegungen" ausgeführt werden, sodass hier nur von einem relativen Stillstand des Auges gesprochen werden kann [6, 14]. Sakkaden sind die Blicksprünge zwischen zwei Fixationen. Sie werden durch einen aktiven Prozess (z.B. die Absicht auf den linken Bildschirm zu sehen) oder passiv durch Veränderung der Umwelt ausgelöst. Um diese Daten zu bestimmen, werden heutzutage meist Kamerasysteme verwendet [14]. Beispielsweise werden diese Daten genutzt, um zu verstehen wie gut UML-Modelle oder Source-Code von Entwicklern verstanden werden und um mögliche Debugging-Methoden zu beschreiben [27].

2.3 Künstliche Intelligenz

Um das verwendete Verfahren für die Erkennung von Textregionen zu verstehen, wird in diesem Abschnitt auf Artificial Neural Networks (ANN) und Convolutional Neural Networks (CNN) eingangen. Die Textregionen werden in einem zweiten Schritt in ein maschinenlesbares Format gebracht. Für diesen Schritt wird ein Convolutional Recurrent Neural Network (CRNN) verwendet. Die Funktionsweise eines CRNNs wird in 2.3.3 erklärt.

2.3.1 Artificial Neural Network

Artificial Neural Networks (ANNs) sind der Biologie von menschlichen neuronalen Netzwerken angelehnt [12, 15]. Es wird versucht die Struktur und Funktion der menschlichen Neuronen im Gehirn künstlich nachzubilden. Dabei besteht ein ANN aus einem Netzwerk von vielen künstlichen Neuronen. Diese Neuronen können in klassischen ANNs drei Aktionen durchführen. In einem ersten Schritt werden die Eingabewerte mit den Gewichten multipliziert. Das Ergebnis dieser Operation wird addiert. Im dritten Schritt

wird eine Aktivierungsfunktion angewandt, welche die Ausgabe des Neurons bestimmt [15]. Ein Beispiel für eine solche Funktion ist die Sigmoid-Funktion:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

Diese Neuronen werden in ANNs über mehrere Schichten (Layer) miteinander verbunden. Wie viele Schichten verwendet werden und ob jedes Neuron mit allen Neuronen des Folgelayers verbunden ist (ein "Fully-Connected-Network") oder ob Verbindungen ausgelassen werden, hängt von der Konfiguration und dem Anwendungsfall des Netzwerkes ab. Die Eingabeschicht wird Input-Layer genannt. Die Schichten zwischen dem Input-Layer und der Ausgabeschicht, dem Output-Layer, werden Hidden-Layer genannt. ANNs können zum Beispiel bei der Klassifikation von Daten und in der Bilderkennung angewandt werden.

2.3.2 Convolutional Neural Network

Convolutional Neural Networks (CNNs) werden häufig in der Erkennung von Mustern genutzt. Dabei haben CNNs besonders in der Bild- oder Spracherkennung den großen Vorteil, dass sie die Zahl der Gewichte im Vergleich zu herkömmlichen neuronalen Netzen drastisch reduzieren, so Albawi et al. [1]. So hat bereits ein Bild im 32x32 Format 1024 Pixel. Diese Pixel sind in einer 2D-Matrix angeordnet, falls das Bild schwarz-weiß ist, oder in einer 3D-Matrix angeordnet, falls das Bild in Farbe ist, da jeder Eintrag in der Matrix zusätzlich einen Rot-, Grün- und Blauwert enthält. In einem *Fully-Connected-Network*, also einem ANN, in dem jedes Neuron einer Schicht mit jedem Neuron anderer Schichten verbunden ist, hätte somit jedes Neuron des ersten *Hidden Layers* 32 x 32 x 3 = 3072 Gewichte. Diese Problematik kann durch CNNs umgangen werden. Ein CNN besteht typischerweise aus 3 Bestandteilen [19]:

1. **Convolution**: Das Bild wird Region für Region abgetastet und mit Kerneln in Form von Matrizen elementweise multipliziert und addiert. Diese Kernel können auch als Filter beschrieben werden und werden während des Trainings eines CNNs gelernt. Durch diese Aktion entsteht eine Reduktion der Eingabegröße. In Formel 2.2 ist dabei eine 5x5 Beispieleingabe zu finden, die durch Padding aufgefüllt wurde. Diese wird mit einem Kernel in Formel 2.3 elementweise multipliziert und summiert. Das Ergebnis dieser Aktion mit der grauen Region und dem Kernel ist in Formel 2.4 grau hinterlegt.

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 4 & 1 & 0 \\ 0 & 5 & 1 & 1 & 3 & 2 & 0 \\ 0 & 2 & 3 & 3 & 1 & 1 & 0 \\ 0 & 1 & 2 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (2.2) \qquad \begin{pmatrix} 1 & 0 & -1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \quad (2.3)$$

$$\begin{pmatrix} 3 & 7 & 0 \\ 4 & 0 & 3 \\ -1 & 5 & 1 \end{pmatrix} \quad (2.4)$$

2. **Pooling** (oder Subsampling): Beim Pooling werden weitere nicht benötigte Informationen durch Downsampling entfernt. Eine häufig verwendete Technik ist das Max-Pooling. Beim Max-Pooling wird die Eingabe-Matrix in einer zuvor definierten Fenstergrößte iteriert und nur das Maximum des Fensters wird in die Ausgabe übernommen. Formel 2.5 zeigt dabei eine beispielhafte Eingabe mit einer Fenstergröße von 2x2 und Formel 2.6 der zugehörigen Ausgabe des Max-Poolings.

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 1 & 1 & 3 \\ 2 & 3 & 3 & 1 \\ 1 & 2 & 1 & 1 \end{pmatrix} \quad (2.5) \qquad \begin{pmatrix} 5 & 4 \\ 3 & 3 \end{pmatrix} \quad (2.6)$$

3. **Fully-Connected**: In der Fully-Connected-Schicht wird die Ausgabe der vorherigen Schichten durch ein klassisches ANN klassifiziert.

Abbildung 2.1 zeigt dabei die Architektur eines Convolutional Neural Networks für die Erkennung von Zahlen. Die Eingabebilder der Größe 32x32 Pixel werden zunächst in einer Convolution Schicht abgetastet. Dadurch entstehen 28x28 Pixel Feature Maps. Danach werden diese Feature Maps durch Subsampling auf 14x14 Pixel verkleinert. Diese werden durch eine weitere Convolution Schicht mit folgendem Subsampling auf 5x5 Pixel Feature Maps reduziert. Als letzte Aktion werden diese durch Convolution zu 120 1x1 Pixel Feature Maps verkleinert und in ein *Fully-Connected-Network* mit 120 Input-Neuronen, einem Hidden-Layer mit 84 Neuronen und einem Output-Layer mit 10 Neuronen eingegeben und klassifiziert [16].

2.3.3 Convolutional Recurrent Neural Network

Ein Convolutional Recurrent Neural Network ist eine Kombination aus einem Convolutional Neural Network und einem Recurrent Neural Network (RNN), bei dem die Convolution und Pooling Schichten eines CNNs mit einem RNN statt einem klassischen künstlichen neuronalen Netzwerk verbunden werden [32]. Der Unterschied von einem RNN zu einem klassischen künstlichen

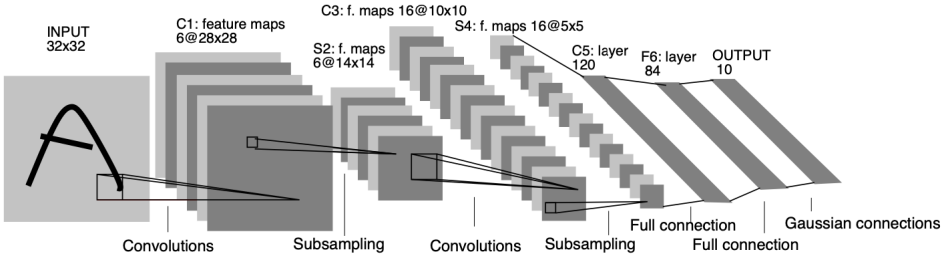

Abbildung 2.1: Architekturbild des LeNet-5 von LeCun et. al. [16]

neuronalen Netzwerk ist das Verbinden der Schichten zusätzlich mit den Ausgaben von früheren Zeitpunkten. Dies führt im Vergleich zu künstlichen neuronalen Netzen zu besseren Ergebnissen bei sequenziellen Daten [30].
Abbildung 2.2 zeigt dabei auf der oberen Hälfte ein ANN mit einer Eingabe, zwei Hidden Layern mit jeweils zwei Neuronen und einer Ausgabe. Die Schichten sind dabei "fully connected". Auf der unteren Häflte ist ein Recurrent Neural Network mit demselben Aufbau zu sehen. Die Hidden Layer zum Zeitpunkt t2 sind zusätzlich mit den Ausgaben des vorherigen Zeitpunktes t1 verbunden.

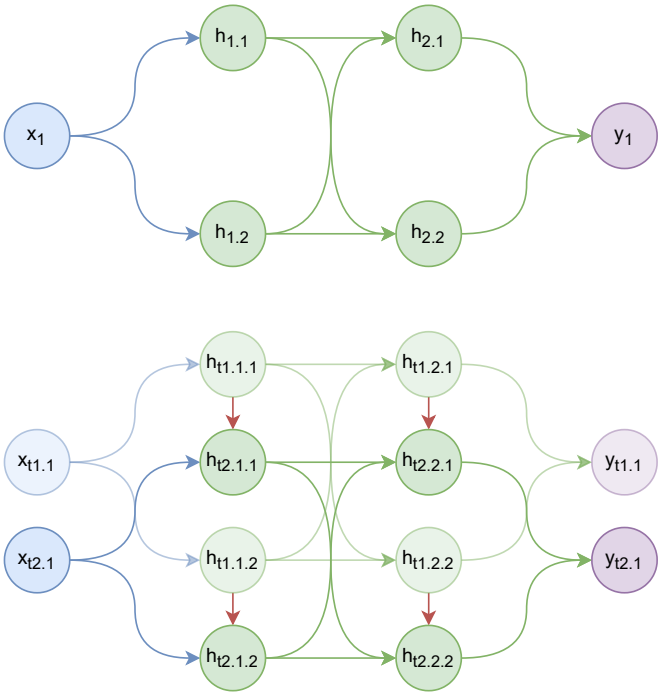

Abbildung 2.2: Schema eines klassischen künstlichen neuronalen Netzwerkes (oben) und eines Recurrent Neural Networks (unten)

2.4 Unterschied zwischen zwei Bildern

Da es bei der Analyse von Bildschirmaufnahmen nicht sinnvoll ist jeden Frame des Videos neu zu analyiseren, soll nur unter bestimmten Bedingungen der Frame neu analysiert werden. Eine dieser Bedingungen ist die Differenz zwischen zwei Frames. Ein Bild (oder Frame) kann wie in 2.3.2 beschrieben als 2D- oder 3D-Matrix dargestellt werden. Um den Anteil der unterschiedlichen Pixel zwischen zwei Bildern zu berechnen, muss zunächst die absolute Differenz der Matrizen berechnet werden. Dafür wird elementweise die absolute Differenz gebildet.

$$absdiff(A, B) = |A - B| \tag{2.7}$$

Sind die Bilder gleich, entsteht hier die Nullmatrix. Um an den Anteil der unterschiedlichen Pixel zwischen den Bildern zu gelangen, werden die Anzahl der Einträge größer 0 der resultierenden Matrix durch die Größe der Matrix (Anzahl aller Einträge) geteilt. Ein Algorithmus könnte folgendermaßen aussehen:

function DIFFERENCE(A, B)
  if len(A) == 0 or len(A) != len(B) **then**
    return
  end if
  $absdiff \leftarrow absdiff(A, B)$
  $not\_null \leftarrow 0$
  for all absdiff entries > 0 **do**
    $not\_null \leftarrow not\_null + 1$
  end for
  return $not\_null/size(absdiff)$
end function

2.5 Representational State Transfer

Representational State Transfer (REST) ist ein beliebtes Architekturmuster für Webanwendungen, dessen Client-Server Kommunikation zustandslos ist. Dies bedeutet, dass jede Anfrage vom Client an den Server alle Informationen enthalten muss, die dieser benötigt um die Anfrage zu verarbeiten. Die REST Architektur besteht dabei aus drei Elementen, die Fielding[8] wie folgt definiert:

1. **Daten (Data Elements)**: Ressourcen, Resourcenbezeichner, Ressourcenmetadaten, Repräsentationen, Repräsentationsmetadaten, Steuerdaten

2. **Konnektoren (Connectors)**: Client, Server, Cache, Resolver, Tunnel

3. **Komponenten (Components)**: Origin Server, Gateway, Proxy, User Agent

In dieser Architekturform führen Komponenten Aktionen auf den Daten aus. Daten sind dabei beispielsweise Ressourcen (beispielsweise die aktuellen Wetterdaten für Hannover), die durch einen Ressourcenbezeichner (beispielsweise eine URL) identifiziert werden. Die Repräsentationen werden als aktueller Zustand einer Ressource verwendet, damit Aktionen auf diesen ausgeführt werden können. Komponenten verwenden Konnektoren um Anfragen und Antworten auf diese Anfragen zu versenden und zu empfangen. Eine Komponente kann dabei zum Beispiel der User Agent eines Webbrowsers sein, der über JavaScript (im weiteren Sinne ein Konnektor) einen Request an einen Origin Server sendet [23]. REST beschränkt dabei nicht die Kommunikation auf ein bestimmtes Protokoll. Die unterstützten Protokolle sind dabei abhängig von den Konnektoren. Im Web wird jedoch meist das HTTP-Protokoll verwendet. Wichtige HTTP-Methoden sind dabei [8, 23]:

1. **GET**: Ruft die Repräsentation einer Ressource ab.
2. **POST**: Erstellt eine Ressource. In der Praxis wird diese Methode aber auch zum Aktualisieren und Löschen von Ressourcen verwendet, da es üblich ist nur GET und POST in einer Anwendung zu unterstützen, so Richard [23].
3. **PUT**: Aktualisiert eine Ressource.
4. **DELETE**: Löscht eine Ressource.

Kapitel 3

Verwandte Arbeiten

Texterkennung ist ein Thema, welches bereits seit dem 19. Jahrhundert erforscht wird. Fortschritte wurden allerdings erst mit der industriellen Revolution gemacht [7]. Heutzutage wird Texterkennung nicht mehr nur durch Algorithmen wie das Suchen von scharfen Kanten in Bildern durchgeführt, sondern durch künstliche neuronale Netzwerke übernommen. Außerdem ist die Texterkennung nicht mehr nur auf Bilder beschränkt, auch in Videos wird sie angewandt. Arbeiten zu diesem Thema werden in Abschnitt 3.1 vorgestellt und von dieser Arbeit abgegrenzt. Eye Tracking ist ebenfalls kein neues Thema. So wurden bereits 1950 von Fitts et al. die Blickbewegungen von Piloten beim Landevorgang aufgezeichnet und ausgewertet [9, 17]. In der Softwareentwicklung hingegen findet Eye Tracking auch zunehmend Verwendung. Beispiele dafür werden in Abschnitt 3.2 vorgestellt. Auch wird Eye Tracking bereits im Kontext von Requirements Traceability eingesetzt. Arbeiten in diesem Kontext werden in Abschnitt 3.3 erläutert und von dieser Arbeit abgegrenzt.

3.1 Texterkennung in Bildern und Videos

Zhou et al.[37] haben mit *EAST (Efficient and Accurate Scene Text Detector)* eine Pipeline für die Erkennung von Text in natürlichen Szenen vorgestellt, die mit einem Convolutional Neural Network und "Non-maximum suppression" arbeitet. Bedeutend ist diese Arbeit gewesen, da sie durch deutlich weniger Verarbeitungsschritte schneller als vergleichbare Arbeiten bei vergleichbarem F1-Wert war. Dabei erkannte das schnellste Model auf einer NVIDIA Titax X Grafikkarte mit Maxwell Architektur und Intel E5-2670 v3 CPU 16.8 FPS (Frames per Second) mit einem F1-Wert von 0.757. Für das Benchmark wurde der COCO-Text[1] Datensatz genutzt[37]. Dieser enthält Text in natürlichen Szenen. Der Vergleich mit Bildschirmaufnahmen ist nicht ganz einfach, da im Bereich der Softwareentwicklung sehr viel Text

[1] https://bgshih.github.io/cocotext/

vorliegt, wohingegen die Bilder des Datensatz meist wenig bis gar keinen Text enthalten [33]. Diese Ergebnisse wurden mit einer 720p (1280×720 Pixel) Auflösung erreicht und der Text liegt nicht in einem maschinenlesbaren Format vor, es sind lediglich die Textregionen bekannt [37]. Für diese Arbeit wird der Text in maschinenlesbarem Format benötigt. Ebenso kann die Auflösung bei Bildschirmaufnahmen mit 1080p (1920x1080 Pixel) höher sein wodurch die selbe Geschwindigkeit nicht reproduziert werden kann.
Baek et al.[5] haben mit *CRAFT (Character Region Awareness for Text Detection)* eine Architektur für ein Convolutional Neural Network vorgestellt, welches sowohl einen Region Score als auch einen Affinity Score für Textregionen produziert. Dabei werden die Region Scores verwendet um einzelne Buchstaben zu lokalisieren und anhand der Affinity Scores werden diese Buchstaben zu Gruppen zusammengefasst. In einem Experiment wurden unterschiedliche Methoden zur Texterkennung verglichen. Für einen Vergleich wurde mitunter das ICDAR 2015 Dataset[2] verwendet, welches ebenso wie COCO-Text Bilder mit Text in natürlichen Szenen enthält. Dabei hat CRAFT auf dem Datensatz einen F1-Wert von 0.869 bei einer FPS Rate von 8.6 erreichen können[5]. Auch hier ist der Vergleich mit Bildschirmaufnahmen schwer, da zwar mehr als nur eine Zeile Text in dem Datensatz vorhanden sein kann, allerdings ist in der Gesamtheit weniger Text als beispielsweise bei einem Dokument vorhanden. Außerdem liegt auch hier der Text nicht in einem maschinenlesbaren Format vor, die Textregionen sind lediglich bekannt. Im Vergleich schnitt CRAFT jedoch besser als EAST ab [5].
Shi et al.[31] haben eine Architektur für ein CRNN vorgestellt, welches aus drei Schichten besteht. Der Convolutional Layer erstellt aus einem Eingabebild durch ein normales CNN (ohne die "fully-connected" Schicht) unterschiedliche Feature Maps. Diese Feature Maps werden in den Recurrent Layer weitergegeben. Im Recurrent Layer wird diesen Features ein Label in Form eines alphanumerischen Zeichens durch ein LSTM Netzwerk zugewiesen. In dem Transcription Layer werden die Label in einem letzten Schritt zu einer Sequenz (z.B. einem Wort) zusammengefügt und ausgegeben. Die Genauigkeit auf dem ICDAR 2013 Dataset[3] liegt bei 86.7%. Der Text liegt am Ende dieses Netzwerkes in einem maschinenlesbaren Format vor. Allerdings wurden auch hier nur Bilder aus natürlichen Szenen verwendet. Besonders bei sehr viel Text nebeneinander (z.B. wenn zwei Fenster mit Dokumentationen nebeneinander auf der Aufnahme sind) hat diese Methode Probleme, da die Vorhersage anhand von Wörterbüchern mit mehr als einem Wort schwer ist [31].
Yusufu et al.[36] haben ein System zur Text Detektion und Tracking in Videos vorgestellt. Dabei wird in einzelnen Videoframes zunächst Text

[2]https://rrc.cvc.uab.es/?ch=4
[3]https://rrc.cvc.uab.es/?ch=2

erkannt und lokalisiert. Dies geschieht durch das Erkennen von zusammenhängenden Komponenten, das Erkennen von Kanten und das Erkennen von ähnlichen Texturen. Es wird davon ausgegangen, dass Texte in einem Frame meist eine ähnliche Farbe und Größe haben und dass diese in räumlichem Zusammenhang stehen. Dabei wird über fünf verschiedene Test Sets ein F1-Wert von 0.882 mit einer Genauigkeit von 81.5% erreicht [36]. Diese Methode ist problematisch, da die Annahme, dass Texte meist eine ähnliche Farbe im Kontext von Bildschirmaufnahmen im Bereich der Softwareentwicklung haben, falsch ist. So haben unterschiedliche Keywords in der Programmierung innerhalb einer IDE häufig andere Farben als Text, der auf der selben Zeile steht. Ebenso werden in der Arbeit meist Texte aus Fernsehsendungen mit Overlay verwendet. Diese befinden sich meist auf wenigen Zeilen mit ähnlicher Schriftgröße. Die Schriftgröße kann bei Bildschirmaufnahmen aber deutlich variieren. Es kann im Browser z.B. die Schrift vergrößert werden während auf dem Desktop noch Icons zu sehen sind, deren Schrift sehr klein ist. Außerdem muss der Text in einem dritten Schritt in ein maschinenlesbares Format gebracht werden, was nicht Teil der vorgestellten Methode ist. Es sind nur die Textregionen bekannt. Für diese Arbeit wird allerdings der Text in einem maschinenlesbaren Format benötigt.

3.2 Eye Tracking in der Softwareentwicklung

Eye Tracking wird in der Softwareentwicklung häufig im Bereich des Modell- und Codeverständnisses verwendet. So nutzen Sharif und Maletic beispielsweise Eye Tracking um zu erklären, ob der camelCase-Codestil und under_scores-Stil das Verständnis von Code beeinflussen [29]. Porras und Guéhéneuc nutzen Eye Tracking um das Verständnis von neuen visuellen Repräsentationen von Architekturmustern mit einer äquivalenten UML-Darstellung zu vergleichen [21]. Auch das Verständnis von Anforderungen unterschiedlicher Formen kann mit Eye Tracking untersucht werden. So nutzen Sharifi et al. Eye Tracking um das Verständnis von Anforderungen in graphischer und textueller Form miteinander zu vergleichen [26]. Eye Tracking kann aber auch beim Verständnis des Debugging-Prozesses helfen. So untersuchen Hejmady und Narayanan welche Rolle graphische und textuelle Elemente wie beispielsweise die dynamische Variablenübersicht und Code beim Debugging spielen [13].

3.3 Eye Tracking und Requirements Traceability

Auch innerhalb der Requirements Traceability wird Eye Tracking bereits genutzt. So stellen Ali et al.[3] eine Methode vor, wie die Genauigkeit von Information Retrieval Methoden in der automatischen Wiederherstellung von Trace Links durch Eye Tracking verbessert werden kann. Dabei stellen sie mit

SE/IDF und *DOI/IDF* zwei neue Gewichtungsschemata vor. Diese basieren auf den Ergebnissen, welche die Autoren in einem Eye Tracking Experiment zuvor gewinnen konnten [3]. Diese Arbeit soll allerdings einen Schritt in Richtung der automatischen Erstellung und Verfolgung der Trace Links mit Hilfe von Eye Tracking Analysen und deren Bildschirmaufnahmen machen. Dies soll durch automatisierte Analyse der Bildschirmaufnahmen geschehen. Dabei soll zunächst nicht auf den Inhalt des Textes geachtet werden.
Walters et al.[34] stellen mit iTrace, einem Plugin für die Eclipse IDE, eine Methode vor, die mit Blickaufzeichnungen aus Eye Tracking Analysen Trace Links zwischen unterschiedlichen Artefakten herstellt. Dabei sollen Aufgaben wie Link Wiederherstellung, Link Verfolgung und Link Visualisierung automatisiert werden. Außerdem werden neue Trace Links durch Interaktionsmuster zwischen verschiedenen Artefakten hergestellt [34]. Auch hier werden Information Retrieval Methoden verwendet. Diese Arbeit soll ohne solche Methoden auskommen. Es soll durch Analyse von Bildschirmaufnahmen und dem späteren Abgleich der Eye Tracking Daten die automatische Auswertung von Eye Tracking Analysen unterstützt werden. Aus den Daten dieser automatischen Auswertung sollen später Trace Links erstellt und aktuell gehalten werden. Dabei soll diese Arbeit unabhängig von spezifischen Anwendungen wie beispielsweise einer IDE sein.
Sharif und Kagdi[28] stellen Möglichkeiten vor wie Eye Tracking Methoden Requirements Traceability Aufgaben unterstützen können. Dabei gehen sie unter anderem auf die Wiederherstellung und Aufrechterhaltung von Trace Links ein. Für die Wiederherstellung und Aufrechterhaltung werden die Daten aus den Fixationen und Sakkaden, also die Auswertung von Blickmustern, vorgeschlagen. Dabei gehen die Autoren allerdings nicht auf die Auswertung von Eye Tracking Daten ein [28].

Kapitel 4

Anforderungen und Konzept

In diesem Kapitel geht es um die ermittelten Anforderungen und die grundlegenden Überlegungen für die Umsetzung dieser. Dabei werden zunächst in Abschnitt 4.1 die Anforderungen erläutert. Abschnitt 4.2 erklärt das Konzept, mit dem die zuvor erläuterten Anforderungen erfüllt werden.

4.1 Anforderungen

Um das Ziel der Texterkennung in Bildschirmaufnahmen von Eye Tracking Analysen im Kontext von Requirement Traceability zu erfüllen, wurden folgende Anforderungen ermittelt. Diese müssen in funktionale und nichtfunktionale Anforderungen aufgeteilt werden.

4.1.1 Funktionale Anforderungen

R1 Texterkennung in Bildschirmaufnahmen: Innerhalb von Bildschirmaufnahmen zugehörig zu Eye Tracking Analysen soll die Software Text erkennen und in ein maschinenlesbares Format wie Unicode bringen.

R2 Erstellen einer .aois-Datei: Der Text soll von der Software zu AOIs zusammengefasst und in das .aois-Format gebracht werden. Eine solche beispielhafte .aois-Datei ist im Anhang unter Abschnitt B zu finden. Das .aois-Format würde andernfalls auch bei der manuellen Analyse durch Tobii Pro Lab entstehen.

R3 Abgleich mit Eye Tracking Daten: Die gefundenen AOIs sollen von der Software in die Daten der Eye Tracking Analyse eingetragen werden. Dabei soll die Software auch abgleichen, ob Fixationen in den Bereichen der AOIs stattgefunden haben.

R4 Veränderungen gefundener AOIs müssen erkannt werden: Innerhalb von Dokumenten kann gescrollt, Textdokumente können

bearbeitet und Fenster können verschoben werden. Die Software soll diese Veränderungen von bereits erkannten AOIs detektieren. Dabei soll die Software Aktionen wie Bearbeiten, Scrollen und Verändern der Koordinaten einer AOI unterstützen und in der AOI mit dem Zeitpunkt der Änderung vermerken.

4.1.2 Nichtfunktionale Anforderungen

R5 **Qualität der Ergebnisse ist wichtig**: Da die Ergebnisse in späteren Schritten außerhalb des Rahmens dieser Arbeit im Kontext von Requirements Traceability weiterverwendet werden sollen, muss die Qualität der Ergebnisse der Software hoch sein.

R6 **Geschwindigkeit der Analyse ist weniger wichtig**: Die Geschwindigkeit der Software ist weniger wichtig als die Qualität der Ergebnisse.

R7 **Genauigkeit muss bei Textdokumenten hoch sein**: In Textdokumenten sind die Anforderungen enthalten. Die Ergebnisse der Software müssen für diese Dokumentenart gut sein.

R8 **Modularer Aufbau der Software**: Die Software muss modular aufgebaut sein, damit sie später in Kombination mit anderen Komponenten zusammenarbeiten kann und Teile der Software einfach wiederverwendet werden können.

R9 **Wartbarkeit der Software**: Die Software muss einfach wart- und anpassbar sein, damit zu einem späteren Zeitpunkt weitere Funktionalitäten hinzugefügt werden können.

4.2 Konzept

Das Konzept der Software muss grundsätzlich in mehrere Abschnitte unterteilt werden. Abschnitt 4.2.1 Videoverarbeitung erklärt dabei wie mit dem eingegeben Video verfahren wird. Der Abschnitt 4.2.2 Optical Character Recognition erklärt dabei wie innerhalb der einzelnen Videoframes der Text in maschinenlesbares Format gebracht wird. Diese Ergebnisse werden danach durch das Merging zusammengefügt und in einem weiteren Schritt, dem Tracking, über die gesamte Länge des Videos verfolgt. In einem letzten Schritt werden die Daten gespeichert und mit den Eye Tracking Daten abgeglichen. Das Konzept hinter diesem Vorgang erläutert Abschnitt 4.2.5 Zuweisung der Daten.

4.2.1 Videoverarbeitung

Die Videoverarbeitung ist der erste Abschnitt der Software. Bei der Videoverarbeitung geht es grundsätzlich um das Auslesen der Metadaten und

einzelenen Frames. Zunächst müssen für das .aois-Format die Metadaten wie die Anzahl der Frames, Frames per Second (FPS), Pixelhöhe und Pixelbreite ausgelesen werden. Aus der Anzahl der Frames und den FPS errechnet sich die Gesamtlänge in Mikrosekunden:

$$Videolänge_{\mu s} = \frac{|Frames|}{FPS} * 1000000 \tag{4.1}$$

Für die Metadaten des .aois-Formats werden die Gesamtlänge in Mikrosekunden, die Anzahl der Frames, FPS und auch die Anzahl der Videos und der Typ der Videos benötigt. Bei der Anzahl und Typ der Videos werden Zahlenwerte erwartet. Folglich können die Werte auf 1 gesetzt werden, da wir nicht mehrere Videos hintereinander erlauben, weil der Benutzer nur Paare von Bildschirmaufnahmen und Eye Tracking Daten eingeben kann. Wenn die Metadaten ausgelesen wurden und das Video ein valides Video ist, werden die Frames einzeln aus dem Video ausgelesen. Bevor diese Frames zu der Optical Character Recognition weitergegeben werden, wird die Differenz zwischen dem vorherigen und dem aktuellen Frame berechnet. Dabei wird der Frame in ein Raster von Zellen aufgeteilt und die einzelnen Zellen werden miteinander verglichen. Der Vergleich geschieht durch das Berechnen der absoluten Differenz zweier Zellen. Ist die Anzahl der Zellen, die auf Grund der absoluten Differenz als geändert gelten, kleiner als ein Grenzwert, welcher vom Benutzer angegeben werden kann, wird der aktuelle Frame ignoriert und nicht weitergegeben. Die Software fährt mit dem Auslesen des nächsten Frames fort. Diese Operation wird ausgeführt, damit zwei sich zu sehr ähnelnde Frames nicht neu analysiert werden. Wenn der Unterschied beispielsweise kleiner als 1% ist, kann sich die Maus bewegt haben, der restliche Bildschirminhalt allerdings nicht. Eine neue Analyse des Frames wäre an dieser Stelle überflüssig. Abbildung 4.1 zeigt dabei das Flussdiagramm für die Videoverarbeitung. Um die Videovearbeitung im größeren Zusammenhang zu sehen, ist in Abschnitt A.1 ein Flussdiagramm zu finden, welches das gesamte Konzept umspannt.

4.2.2 Optical Character Recognition

Bei der Optical Character Recognition (OCR) werden innerhalb eines Frames zunächst die Textregionen erkannt. Dies geschieht mithilfe eines CNNs auf Basis des CRAFT Papers[5]. Die Textregionen bestehen aus den vier Koordinaten der Randpunkte der Region und einem Confidence-Level, welches anzeigt, wie sicher das CNN mit der gefundenen Region ist. Die Werte dieses Confidence-Level sind zwischen 0 und 1. Der Benutzer kann ebenfalls angeben, ab welchem Confidence-Level die Textregionen berücksichtigt werden sollen. Da wir allerdings Bildschirmaufnahmen und keine natürliche Szenen analysieren, kann das Level sehr niedrig sein. Bei Tests, die im Rahmen der Arbeit durchgeführt wurden, hat sich ergeben,

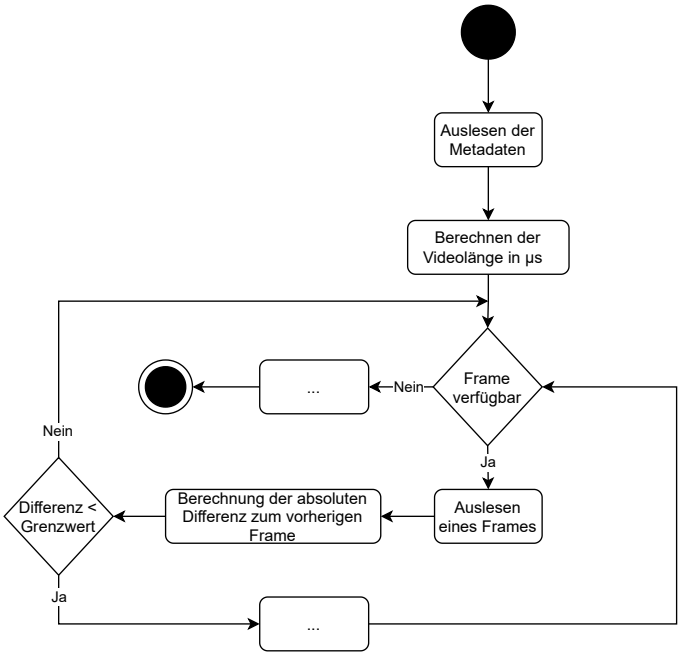

Abbildung 4.1: Flussdiagramm für die Videoverarbeitung - die "..."-Elemente markieren dabei Aktionen, die nicht zu der Videoverarbeitung gehören

dass die besten Ergebnisse im Kontext einer IDE zwischen einem Wert von 0.1 und 0.3 produziert werden. Bei einem zu niedrigen Level wird auch Text in Icons erkannt, die keinen Text enthalten. Innerhalb dieser Regionen wird dann in einem zweiten Schritt der Text erkannt. Das Erkennen des Textes wird von einem CRNN, welches auf dem Paper von Shi et al.[31] basiert, erledigt. Der erkannte Text, die zugehörigen Koordinaten der Randpunkte und das Confidence-Level werden dann weiter an das Merging gegeben.

4.2.3 Merging

Beim Merging werden diese Textregionen verbunden. Dies wird benötigt, da viele kleine AOIs keinen Nutzen zur automatischen Eye Tracking Analyse beitragen würden. Auch zu große AOIs würden keinen Nutzen beisteuern, da sowohl zu kleine, als auch zu große AOIs durch einen Menschen manuell zusammengefügt oder getrennt werden müssen. Zunächst werden allerdings alle erkannten Boxen mit einem zu niedrigen Confidence-Level aus der weiteren Betrachtung ausgeschlossen. Danach werden die übrig gebliebenen Regionen verbunden. Dies geschieht durch die einfache Heuristik, dass nah beieinander stehende Objekte in einem Zusammenhang stehen. Ein Verbinden jeweils einer Region mit der ersten ihr ähnlichen Region (z.B. geringer Abstand der X und Y Koordinaten der Mittelpunkte) ist dafür nicht geeignet. Dies liegt daran, dass die Ergebnisse in unterschiedlicher

Reihenfolge aus der OCR ausgegeben werden können. Wenn zuerst immer verbunden wird, können so für gleiche Frames unterschiedliche Ergebnisse folgen. Deswegen wird zunächst jede Textregion um eine vom Benutzer definierte Prozentzahl in alle vier Richtungen vergrößert. Dann werden Kollisionen, also Regionen, die sich überlappen, für alle Regionen berechnet. So kann sichergestellt werden, dass die Ergebnisse immer die gleichen sind für dieselben Regionen. In einem zweiten Schritt werden die kollidierenden Regionen zusammengefasst. Dabei übernehmen die neuen Textregionen das kleinste Confidence-Level aller zusammengefügten Regionen und der Text wird durch einen Tab separiert verbunden. Abbildung 4.2 zeigt dabei wie sich das Merging verhält. Es sind oben zunächst drei erkannte Regionen zu sehen. Diese werden durch das Merging zusammengefasst, da diese sich überschneiden. Diese beiden Schritte können in mehreren Iterationen

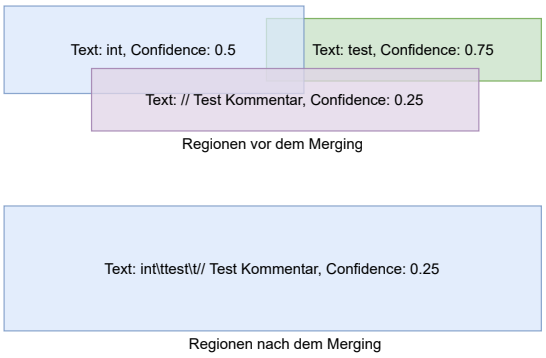

Abbildung 4.2: Beispiel für das Merging mit einer Iteration. Es entsteht eine neue Textregion, die alle 3 sich überschneidenden Regionen enthält.

wiederholt werden. Die Anzahl der Iterationen kann vom Benutzer angegeben werden. Falls zu wenige Iterationen durchgeführt werden, ist es allerdings möglich, dass noch Regionen übrig sind, die sich überschneiden. Abbildung 4.3 zeigt dabei welche Folgen zu wenige Iterationen haben können. Die zwei zusammengeführten Regionen erzeugen eine neue, große Region, die eine weitere Region überschneidet. Aus den zusammengefügten Regionen entstehen KeyFrames der .aois-Datei. Diese KeyFrames bestehen aus den Koordinaten der vier Randpunkte der Region, dem Zeitpunkt in Sekunden, zu welchem sich die Region an diesen Koordinaten befindet und ob diese auf der Bildschirmaufnahme sichtbar ist. Ein KeyFrame wird beispielsweise als nicht sichtbar markiert, falls auf der Bildschirmaufnahme das Fenster gewechselt wird, wohingegen Scrollen oder Bearbeiten meist nur veränderte Koordinaten zur Folge haben können. Diese zusammengefügten Regionen werden danach an das Tracking weitergegeben.

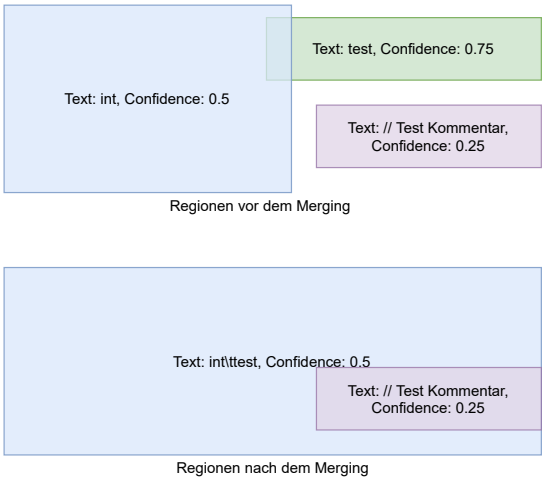

Abbildung 4.3: Beispiel für das Merging mit nur einer Iteration bei dem eine große Region entsteht, die eine weitere Region überschneidet

4.2.4 Tracking

Der letzte Schritt der einzelnen Frames und gefundenen Regionen ist das Tracking. Beim Tracking wird der Text mit bereits erkannten Texten abgeglichen. Dies muss geschehen, damit keine hohe Anzahl von gleichen AOIs mit unterschiedlichen KeyFrames entsteht. Das Abgleichen geschieht entweder durch Gleichheit der Texte oder durch Berechnung der wortweisen Levenshtein-Distanz. Die Levenshtein-Distanz zeigt dabei die Distanz zwischen zwei Zeichenketten an, die durch die Aktionen Ersetzen, Einfügen und Löschen entsteht. Bei Berechnung der Levenshtein-Distanz kann der Benutzer einen Grenzwert angeben, ab der ein Text noch zu einer bereits erkannten AOI zugeordnet wird. Die Levenshtein-Distanz wird benötigt, da die Ergebnisse der OCR teilweise nicht zuverlässig sind und da das Bearbeiten von Text unterstützt werden soll. Bei Erkennen einer Bearbeiten-Aktion wird der neue Text der AOI zugewiesen. Wird keine Übereinstimmung oder Levenshtein-Distanz im benutzerdefinierten Rahmen gefunden, entsteht aus dem Text und dem KeyFrame eine neue AOI. Andernfalls wird der KeyFrame einer bestehenden AOI zugeordnet. Außerdem muss beim Tracking darauf geachtet werden, dass AOIs nicht mehr sichtbar sein können. Es wird auf Grund dessen zunächst die Liste der AOIs kopiert. Aus dieser Kopie werden alle sichtbaren AOIs entfernt, sodass die unsichtbaren AOIs übrig bleiben. Für die unsichtbaren AOIs wird ein neuer KeyFrame an den Koordinaten des zuletzt sichtbaren KeyFrames erstellt, der die Eigenschaft $IsActive = False$ enthält. Sie ist somit als nicht mehr sichtbar markiert. Falls an dieser Stelle die AOI, für weniger Zeit als vom Benutzer angegeben, sichtbar war, wird diese als nicht wichtig markiert.

Diese Markierung wird durch ein erneutes Erscheinen auf der Aufnahme wieder entfernt. Abbildung 4.4 verdeutlicht den beschriebenen Prozess des Trackings als ein Flussdiagramm.

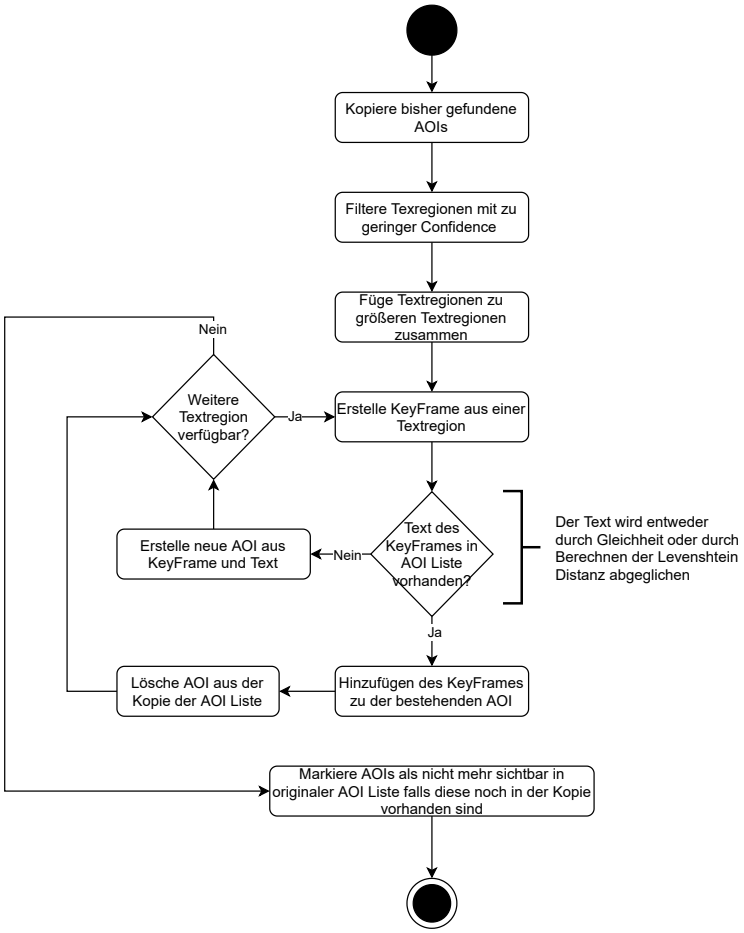

Abbildung 4.4: Flussdiagramm für das Tracking

4.2.5 Zuweisung der Daten

Sobald alle Frames verarbeitet sind, werden zunächst alle als nicht wichtig markierten AOIs wieder gelöscht. Diese AOIs waren entweder nur für zu kurze Zeit sichtbar, sodass sie kaum durch einen Menschen wahrgenommen werden können, oder sie sind durch eine fehlerhafte Erkennung der OCR entstanden. Nach entfernen dieser AOIs werden die aus den Textregionen gefundenen AOIs und Video-Metadaten im letzten Schritt in das JSON-Format konvertiert und als .aois-Datei gespeichert. Da das .aois-Format nicht den gefundenen Text enthält, wird eine weitere Datei erstellt. Diese Datei enthält neben dem Namen und der KeyFrames der AOI ebenfalls ihren Text.

Außerdem müssen die Eye Tracking Daten mit den AOIs abgeglichen werden. Dafür entsteht für jede AOI eine neue Spalte in der Tabelle mit den Eye Tracking Daten. Diese wird mit 4 unterschiedlichen Werten ausgefüllt:

1. **1**: Die AOI ist zu einem Zeitpunkt t auf dem Bildschirm sichtbar und die Region wird von einer Fixation getroffen
2. **0**: Die AOI ist zu einem Zeitpunkt t auf dem Bildschirm sichtbar, die Region wird allerdings nicht von der Fixation getroffen
3. **-1**: Die AOI ist zu einem Zeitpunkt t auf dem Bildschirm nicht sichtbar
4. **Zelle ist leer**: Die Eye Tracking Analyse befindet sich in der Initialisierung

Kapitel 5

Implementierung

Bei der Umsetzung dieser Arbeit wurden verschiedene Technologien und Architekturen verwendet. In diesem Kapitel werden diese näher erläutert. Der Abschnitt 5.1 Programmiersprachen und Bibliotheken geht auf die verwendeten Programmiersprachen und Bibliotheken ein. Die verwendete Architektur wird in 5.2 näher beschrieben. Dabei wird auch besonders auf das Testen der Software in 5.3 eingegangen.

5.1 Programmiersprachen und Bibliotheken

5.1.1 Programmiersprachen und Technologien

Der Großteil der Software wurde mit der Programmiersprache Python umgesetzt. Da das User Interface eine Webanwendung ist, wurde allerdings auch JavaScript verwendet, da eine interaktive Webanwendung sonst nicht umsetzbar wäre. Die genauen Technologien und ihre zugehörige Versionsnummer sind im folgenden aufgelistet:

1. **Python**: Für die Implementierung des Serverbackends und des VideoOCR Packages wird Python in der Version 3.7 verwendet. Eine höhere Version als 3.7 kann nicht verwendet werden, da OpenCV diese als maximale Abhängigkeit angibt.

2. **JavaScript**: Für die Implementierung der Funktionen im Frontend und der Kommunikation des Frontends mit der Backend API wird JavaScript im ECMAScript 2016 Standard verwendet. Diese Version von JavaScript verfügt über Funktionen wie Array.prototype.includes und wird von allen modernen Browsern unterstützt.

3. **Sass**: Die zusätzlichen CSS-Definitionen werden in Sass 1.35.1, einer Erweiterung von CSS mit Variablen und weiteren Features, geschrieben. Sass ist dabei ein sogenannter CSS-Präprozessor, dessen Quellcode

in Reinform nicht von Webbrowsern unterstützt wird. Deswegen wird der Sass-Code in CSS umgewandelt.

5.1.2 Bibliotheken

Neben den verwendeten Programmiersprachen wurden auch sieben Bibliotheken verwendet. Diese sind mit ihren Versionsnummern und Verwendungszwecken hier aufgeführt:

1. **EasyOCR** (1.4): Diese Bibliothek vereinbart das CRAFT und CRNN Paper und wird für die Optical Character Recognition verwendet. Die Versionsnummer ist 1.4 und war zum Zeitpunkt der Abgabe der Software für diese Arbeit die aktuellste Version der Bibliothek.

2. **OpenCV** (4.5.1): OpenCV ist eine Bibliothek mit Werkzeugen im Bereich der Bildverarbeitung und Computer Vision und wird in dieser Arbeit für das Auslesen der Frames und Videometadaten verwendet. Außerdem ist OpenCV eine Abhängigkeit von EasyOCR. Verwendet wird opencv-python 4.5.1. Dies ist die CPU Version des Packages. Eine GPU Unterstützung ist für die Verwendung von OpenCV innerhalb der Arbeit nicht notwendig, da nur Frames und Metadaten ausgelesen werden.

3. **NumPy** (1.20.2): Die Bibliothek NumPy ist eine Bibliothek für wissenschaftliche Berechnungen. Die ausgelesenen Frames aus OpenCV werden beispielsweise als 3D-NumPy-Array ausgelesen. Mit NumPy wird auch die absolute Differenz zwischen zwei Frames berechnet.

4. **pandas** (1.2.4): pandas ist eine Bibliothek für die Datenverarbeitung und Datenanalye von großen Datenmengen und wird in dieser Arbeit für das Verarbeiten der Eye Tracking Analyse Daten verwendet.

5. **Flask** (1.1.2): Flask ist ein Webframework, welches auf der Werkzeug Bibliothek und der Jinja Template Engine basiert. In der Arbeit wird Flask für die API und das Ausspielen des GUIs verwendet, welches als Webanwendung umgesetzt wurde.

6. **Bootstrap** (5.0.1): Boostrap ist eine Frontend-Bibliothek für das Web und wird in dieser Arbeit von der Frontend Version des GUIs verwendet. Dies dient der einheitlichen Darstellung in verschiedenen Browsern.

7. **Yappi** (1.3.2): Yappi ist ein Python Profiler, der im Gegensatz zu dem in Python enthaltenen cProfiler auch Multithreading unterstützt. Da die angefertigte Software sowohl Multithreading als auch mehrere Prozesse verwendet, wurde Yappi verwendet um Laufzeiten einzelner

Methoden zu messen und zu optimieren. In der ausgelieferten Software ist Yappi allerdings nicht mehr enthalten.

5.2 Architektur

Um der geforderten Modularität gerecht zu werden, gibt es mehrere Möglichkeiten die Software zu verwenden. Zum einen die Verwendung des VideoOCR Packages als Python Bibliothek, die Benutzung als Konsolenanwendung, die Verwendung der API oder die Benutzung mit dem User Interface.

Verwendung als Python Bibliothek oder Konsolenanwendung

Der Kern der Software besteht aus dem angefertigten Python Package. Dieses kann innerhalb anderer Anwendungen verwendet und als Bibliothek über den Python Paketmanager *pip* global in der Python Distribution installiert werden. Die Konsolenanwendung verwendet diese Bibliothek und gibt die Parameter mit Standardeinstellungen an den aufrufenden Benutzer weiter. Die Konsolenanwendung startet dabei die Pipeline, welche die drei Tasks *OCRTask*, *ProcessingTask* und *CleanupTask* ausführt. Der OCRTask führt dabei die Erkennung der AOIs innerhalb des gegeben Videos durch. Dabei wurde darauf geachtet, dass die Software sowohl auf schwachen Systemen verfügbar ist, als auch die Leistung von starken Systemen verwenden kann. Je nachdem welche Parameter der Benutzer angibt, werden mehrere Instanzen von EasyOCR Readern erstellt. Diese Reader laden die CNN (CRAFT [5]) und CRNN Modelle zu Start des OCR Tasks und sind zuständig für das Erkennen des Textes innerhalb einzelner Frames. Danach werden die Frames durch den Hauptthread ausgelesen. Diese Frames werden an die Reader weitergegeben, die in einem eigenen Python Thread arbeiten. Dabei sind Python Threads Fäden, also leichtgewichtige Ausführungskontexte. Sie haben keinen eigenen Speicherbereich und werden terminiert, falls der Hauptthread terminiert. Auf Grund dessen werden auch Semaphoren verwendet um die Lost Update-Anomalie zu vermeiden. Bei der Lost Update-Anomalie kommt es durch aufeinanderfolgendes Lesen des gleichen Speicherplatzes durch mehrere Threads und anschließenden Schreiben der beiden Threads zu Verlust von Daten, da das Lesen des einen Threads geschieht, bevor die Daten vom Vorgänger Thread verarbeitet und geschrieben werden. Dies muss verhindert werden, da sich die AOI Liste in einem geteilten Speicherbereich befindet und eine Lost Update-Anomalie zu Verlusten von KeyFrames oder neuen AOIs führen kann. Außerdem wartet der Hauptthread, sobald alle Reader belegt sind. Sobald ein Reader freigegeben wird, werden die nächsten Frames analysiert. Falls am Ende keine Frames mehr übrig sind, wird auf die letzten aktiven Threads gewartet, damit die Daten der letzten Frames nicht verloren gehen. Falls für das Tracking die Levenshtein-Distanz verwendet werden soll, wird diese über mehrere,

unabhängige Prozesse berechnet, da diese auf unterschiedlichen CPU Kernen ausgeführt werden können. Die Anzahl der Prozesse können vom Benutzer angegeben werden, jedoch können nie mehr als die maximale Anzahl von Kernen - 1 verwendet werden. Am Ende werden die gefundenen AOIs an die Daten zur Verarbeitung auf dem Haupthread weitergegeben. Diese wird durch den ProcessingTask ausgeführt. Das Programm terminiert nach dem Schreiben der Daten. Der CleanupTask wird von der Konsolenanwendung nicht verwendet. Dieser dient dazu, die Eingabedateien zu löschen. Dies ist bei der Verwendung der API sinnvoll, da sonst die Eingabedateien auf dem Server bleiben würden.

User Interface und die Verwendung als API

Das User Interface ist aufteilbar in Backend und Frontend. Das Backend ist als API aufgebaut und folgt dem *Representational State Transfer* Architekturmuster. Es kann ebenfalls ohne User Interface verwendet werden. Es existieren dabei vier Routen die über verschiedene HTTP-Methoden ansprechbar sind:

1. /**analyse** (POST): Die wichtigste Route ist die "/analyse"-Route. Hier werden die eingegeben Parameter zunächst auf ihre Korrektheit überprüft. Die benötigten Parameter sind dieselben, die auch für die Analyse als Konsolenanwendung verwendet werden. Sind die Parameter korrekt, wird eine Analyse des Videos mit den gegebenen Parameter in einem neuen Prozess gestartet. Im Kern ist dies die Nutzung des VideoOCR Packages innerhalb einer seperaten Anwendung. Die Prozess ID und Analyse ID werden als Antwort im JSON-Format zurückgegeben. Mit der Prozess ID kann der Prozess über die "abort"-Route wieder abgebrochen werden. Mit der Analyse ID können die Zwischenergebnisse der Analyse in Form von Bildern mit markierten AOIs oder das Endergebnis der Analyse in Form von Bildern und der fertigen .aois- und .tsv-Datei über die "/results/get"-Route abgefragt werden.

2. /**abort** (DELETE): Die Route für das Abbrechen des Analyse Prozesses ist "/abort". Diese Route ist nur über die HTTP-Methode DELETE ansprechbar. Es müssen Prozess ID und Analyse ID in den Daten angegeben werden. Falls die Kombination korrekt ist, wird der Prozess abgebrochen und alle bisherigen Daten, die dieser erzeugt hat, werden gelöscht. Auch die eingegebenen Video- und Eye Tracking Analyse-Daten werden gelöscht. Eine Erfolgs- oder Fehlermeldung wird im JSON-Format zurückgegeben.

3. /**delete** (DELETE): Die Route "/delete" löscht Ergebnisse einer Analyse falls die eingegebene Analyse ID existiert. Im Erfolgs- und

Fehlerfall wird jeweils eine entsprechende Antwort im JSON-Format zurückgegeben

4. **/results/get** (GET): Über die "/results/get"-Route werden die Ergebnisse der Analyse angefordert. Diese liegen in Form eines Arrays mit Links zu den Bild- und Analysedaten vor. Diese Route kann nur über die HTTP-Methode GET angesprochen werden. Die Daten müssen somit nicht über den Request-Body sondern die URL Parameter mitgegeben werden. Dabei muss die Analyse ID als Parameter mitgegeben werden.

Neben den REST Routen verfügt der Server noch über drei weitere Routen, die das User Interface in Form einer Webanwendung ausspielen. Zum einen die "/"-Route, welche die Startseite und Eingabemaske für die Analyse darstellt. Eine weitere Route ist die "/results"-Route. Sie benötigt die zwei GET-Parameter Analyse ID und Prozess ID. Die Prozess ID hat die Besonderheit, dass sie entweder 0 oder eine Prozess ID in Form einer UID ist, welche von dem Python Modul uuid generiert wird. Der Wert 0 weist darauf hin, dass die Analyse bereits abgeschlossen ist und nur die Ergebnisse geladen werden müssen. Ist der Wert anders als die 0, wird zusätzlich noch ein Button zum Abbrechen des Prozesses angezeigt. Die letzte verfügbare Route ist die "/recent"-Route. Sie gibt eine Seite aus, die alle Analyse Ergebnisse geordnet nach Datum anzeigt. Dazu gibt es zu jedem Eintrag die Möglichkeit sich die Parameter herunterzuladen, die Ergebnisse zu löschen oder erneut anzusehen.

5.3 Tests

Für diese Software wurden umfangreiche Tests durchgeführt. Die Tests werden dabei automatisch von Coverage.py ausgeführt. Coverage.py ist eine Bibliothek, welche die Überdeckung von Testfällen berechnet. Die Testfälle sind mit dem Python unittest-Framework geschrieben. Coverage.py greift dort auf die discover-Option von unittest zurück, um alle Testklassen und ihre zugehörigen Tests automatisiert auszuführen. Dabei können die Tests in zwei Kategorien eingeteilt werden. Zum einen das Testen des VideoOCR Packages. Dieses Package enthält den Code für die Analyse der Videos im Kontext von Eye Tracking. Hier befindet sich ein Großteil der eigentlichen Software. Die zweite Kategorie ist das Testen der API, welche für das Backend des User Interfaces zuständig ist. Für alle Bereiche zusammen wurde eine Anweisungsüberdeckung von 98% und eine Zweigüberdeckung von 91% durch Coverage.py errechnet.

Datei	Statements	Miss	Anweisungsüberdeckung
AOI.py	100	0	100%
AOIList.py	55	0	100%
AOIMerger.py	51	0	100%
Arguments.py	85	0	100%
CleanupTask.py	15	0	100%
Frame.py	110	0	100%
FrameCell.py	19	0	100%
KeyFrame.py	10	0	100%
LevenshteinPool.py	7	0	100%
LoggerFactory.py	14	0	100%
Media.py	9	0	100%
OCRTask.py	164	0	100%
Pipeline.py	23	0	100%
ProcessingTask	51	0	100%
Resultencoder.py	10	0	100%
Task.py	*10*	*2*	*80%*
Util.py	26	0	100%
Vertex.py	6	0	100%
VideoReader.py	68	19	72%
Gesamt	823 *(+10)*	19 *(+2)*	98%

Tabelle 5.1: Anweisungsüberdeckung des VideoOCR Packages

5.3.1 Testen des VideoOCR Packages

Das VideoOCR Package wurde durch Unittests und Integrationstests getestet. Dabei wurde darauf geachtet, dass für alle Methoden die Randfälle überprüft wurden. Insgesamt errechnet Coverage.py eine Anweisungsüberdeckung von 98%. In Tabelle 5.1 sind die jeweiligen Dateien mit der Anzahl der gewerteten Codeabschnitte (Statements), der Anzahl der nicht abgedeckten Codeabschnitte (Miss) und die Anweisungsüberdeckung zu finden. Task.py wird aus der Gesamtwertung ausgeschlossen, da Task eine abstrakte Klasse ist und keine Tests für Task existieren. Zusätzlich ist in C.1 VideoOCR Package eine Übersicht über die Zweigüberdeckung zu finden.

5.3.2 Testen des User Interfaces und der API

Auch das Backend des User Interfaces wurde durch Unit- und Integrationstests getestet. Teilweise existiert bei der API die Problematik, dass Coverage.py nicht korrekt die Prozesse der multiprocessing Bibliothek messen kann. Analysen werden in einem neuen Prozess gestartet. Dadurch ist es möglich mehrere Analysen parallel laufen zu lassen. Diese benutzen jeweils einen anderen CPU Core, falls verfügbar. Allerdings resultiert auch dies in dem

Datei	Statements	Miss	Anweisungsüberdeckung
app.py	137	4	97%

Tabelle 5.2: Anweisungsüberdeckung der API

Ergebnis, dass Coverage.py die Funktion ”ocr_process(arguments)” nicht mitwertet, da diese in einem externen Prozess ausgeführt wird. Auf Grund dessen wird nur eine 97% Anweisungsüberdeckung erreicht. Coverage.py unterstützt Bibliotheken wie multiprocessing mit der C-Extension. Allerdings hat sich dies in mehreren Versuchen nicht ergeben. Tabelle 5.2 zeigt ebenfalls die Datei für die API mit zugehöriger Anweisungsüberdeckung. Unter C.2 User Interface und API ist die Zweigüberdeckung für app.py zu finden.

Kapitel 6

Evaluation

Die Ergebnisse, die die angefertigte Software liefert, wurden einer Evaluation unterzogen. Dabei wurden zwei Bildschirmaufnahmen manuell und durch die Software ausgewertet. Die Ergebnisse werden im Abschnitt 6.1 Genauigkeit miteinander verglichen. Ebenso wird das Laufzeitverhalten insgesamt in Abschnitt 6.2 einer Evaluation unterzogen. Abschnitt 6.3 weißt auf die Limitierungen der Implementierung hin.

6.1 Genauigkeit

Die Genauigkeit ist die wichtigste Voraussetzung für einen erfolgreichen Einsatz der Software. In diesem Abschnitt werden die Ergebnisse, die diese produziert, evaluiert. Dabei wurde das Ergebnis der Software mit den Ergebnissen von zwei manuell durchgeführten Analysen verglichen. Die Analysen wurden auf insgesamt 22 Minuten Videomaterial durchgeführt. Diese 22 Minuten beinhalten 35295 verfügbare Videoframes. Für die automatische Analyse wurde auf Grund der Menge an Frames nur pro Sekunde des Videos zwei Frames verwendet. Ein durch die manuelle und automatische Analyse gefundener AOI *(True Positive)* zeichnet sich durch ähnliche Koordinaten und denselben Text aus. Wären exakt die gleichen Koordinaten gefordert, wäre die Genauigkeit sehr schlecht, da ein Mensch die Kästchen um eine AOI nie exakt gleich zieht und die Software raten müsste welchen Abstand der Mensch jedes mal von der Box zum Text lässt. Durch die Software gefundene AOIs, die allerdings zu klein oder deutlich zu groß sind, werden als *False Positive* bezeichnet. Dies kann zum Beispiel durch weniger oder mehr Text entstehen oder falls diese vom Menschen nicht als AOI vermerkt werden, weil sie zum Beispiel keinen Text enthalten. *False Negatives* sind durch den Menschen erkannte AOIs, die von der Software nicht erkannt werden. Insgesamt konnte so ein Precision Wert von 0.76, ein Recall Wert von 0.79 und somit ein F1-Wert von 0.77 errechnet werden. Eine Gesamtübersicht aller Ergebnisse zeigt Tabelle 6.1. Die Werte

Dokumentenart	True Positive	False Positive	False Negative
Code	200	134	101
Desktop	25	11	9
Diagramm	36	4	35
Dokument	451	76	44
Gesamt	712	225	189

Tabelle 6.1: Gesamtübersicht der Auswertung

können auf vier Arten von Dokumenten oder Szenen aufgeteilt werden. Unterschieden wurden dabei die Kategorien Code, Desktop, Diagramme und Dokumente. Als Dokumente wurden alle AOI innerhalb eines Dokuments, eines Buches oder einer Webseite mit Ausnahme von Diagrammen, gewertet. Diagramme sind auf Grund des deutlich niedrigeren Anteils von Text und dem häufig sehr kleinen Text aus Dokumenten ausgenommen. Als Code wird der Text innerhalb des Kontextes einer integrierten Entwicklungsumgebung *(integrated development environment, kurz IDE)* oder eines Texteditors mit Syntaxhervorhebung und Code-Dokumenten gewertet. Dazu zählen auch die Bedienungs- und Informationselemente der IDE. Die letzte Kategorie ist die Kategorie Desktop. Hier wurden hauptsächliche AOIs auf Windows oder macOS-Desktops aber auch dem Windows-Explorer gewertet. Da sich in den Auswertungen aber die wenigsten Szenen innerhalb dieser Umgebungen befinden, ist das Ergebnis nicht sehr aussagekräftig. Es ist festzuhalten, dass die Software teilweise andere Einstellungen für unterschiedliche Arten von Dokumenten benötigt. Die Einstellungen von Dokumenten mit viel Text, der visuell kaum getrennt ist, muss deutlich abweichen von den Einstellungen von Dokumenten mit größerem Text und klarer, visueller Trennung. Text innerhalb einer IDE ist ein gutes Beispiel dafür. Abbildung 6.1 zeigt dabei ein schlechtes Ergebnis auf Grund von suboptimalen Einstellungen. Grün markiert sind dabei die Ergebnisse der OCR. Die blauen Markierungen sind die gefundenen AOIs. Es ist ersichtlich, dass eine große AOI fast die gesamte IDE umspannt. Dabei wäre auch das Erkennen einzelner Methoden als AOI mit anderen Einstellungen möglich gewesen. Neben den unterschiedlichen Einstellungen kann es auch sein, dass die OCR für den selben Text unterschiedliche Ergebnisse liefert. Innerhalb der Tests kam dies zwar eher selten vor. Allerdings können so neue AOIs entstehen, die nur kurz sichtbar sind. Um dies zu vermeiden wurde das Markieren als wichtig und unwichtig und das spätere Löschen von unwichtigen AOIs implementiert. Positiv anzumerken ist auch, dass sogar Handschrift innerhalb von Abbildungen in einem Dokument erkannt wird. Abbildung 6.2 zeigt ein solches Beispiel. Die Farbgebung hat dort dieselbe Bedeutung wie im vorherigen Beispiel. Weitere Abbildungen mit Beispielen zu unterschiedlichen Szenen, wie beispielsweise der Google Suche oder einem normalen Dokument, sind im Anhang unter

Abbildung 6.1: Beispiel für ein schlechtes Ergebnis innerhalb einer IDE

Abschnitt D beigefügt.

6.2 Laufzeit

Die Laufzeit der Software ist von den Parametern abhängig. Maßgeblich beeinflusst wird die Laufzeit durch die Parameter:

1. –**gpu**: Analyse der Frames erfolgen bei Eingabe "True" auf einer GPU, falls diese verfügbar und PyTorch mit GPU-Unterstützung installiert wurde. Andernfalls wird die Analyse allein auf dem CPU durchgeführt. Falls ein GPU verfügbar ist, auf dem entweder Nvidia CUDA oder AMD ROCm Treiber installiert wurden, kann dieser genutzt werden um den Vorgang zu beschleunigen. So war es in Tests möglich die Analyse eines Frames mit denselben Einstellungen von 24.6 Sekunden (Intel i9-9880H Prozessor ohne GPU) auf 4.27 Sekunden (Intel i5-4570 Prozessor mit MSI GeForce Gtx 970 4GB) zu reduzieren.

2. –**skip_frames**: Definiert die Anzahl an Frames, die nach Analyse eines Frames mindestens ignoriert werden. Dieser Parameter kann ebenso großen Einfluss auf die Laufzeit der Software haben, da dieser die Zahl der berücksichtigten Frames für die Analyse deutlich verkleinern kann. Allerdings darf der Parameter nicht zu hoch eingestellt sein. Die Genaugkeit der Ergebnisse sinkt deutlich. Bereits das Ignorieren von 100 Frames hat zur Folge, dass bei 30 FPS 3.3 Sekunden des Videos

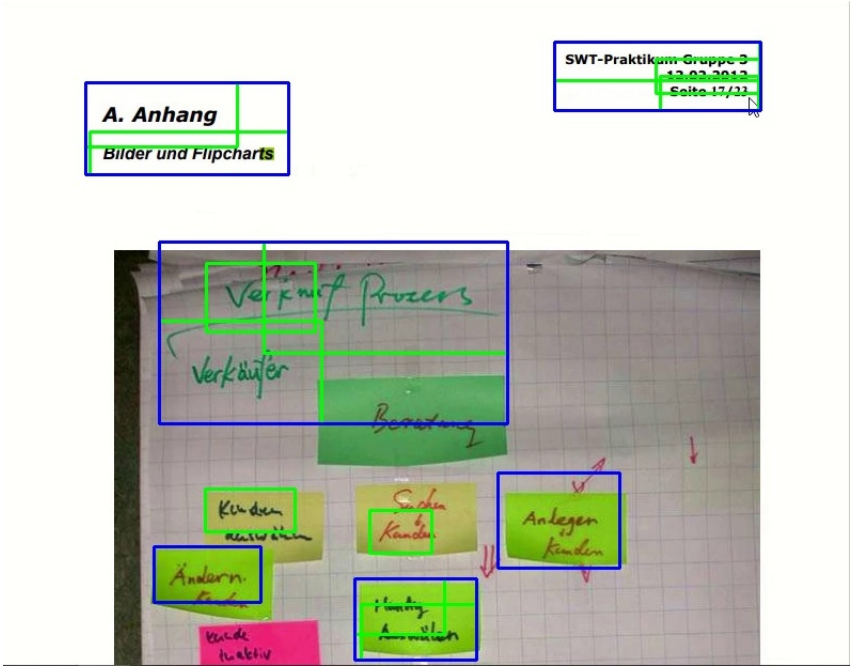

Abbildung 6.2: Beispiel für die Erkennung von Handschrift innerhalb eines Dokumentes

ignoriert werden. Für die besten Ergebnisse empfiehlt es sich diesen Parameter auf 0 zu stellen.

3. –**readers**: Definiert die Anzahl an Threads in denen EasyOCR Reader während des Prozess der Videoanalyse aktiv sind. Begrenzt wird die Anzahl nur durch das System, entweder durch die RAM oder GPU Leistung, da die Modelle zu Anfang in den RAM oder die GPU geladen werden. So hat sich im Schnitt eine 29% Verbesserung bei Verdopplung der Anzahl von zwei auf vier ergeben (ohne GPU Unterstützung). Bei der GPU Variante werden allerdings nicht mehrere Reader benötigt. Pytorch, die Bibliothek auf der EasyOCR basiert, verteilt die Last automatisch auf die GPUs, sodass diese während dem OCR-Vorgang optimal ausgelastet sind.

4. –**edit_distance_mode**: Definiert, ob die Levenshtein Distanz genutzt wird. Dieser Parameter kann bei zunehmender Anzahl von AOIs sehr viel Zeit in Anspruch nehmen. So muss im schlimmsten Fall für jede bereits gefundene AOI die Distanz berechnet werden. Dies geschieht für jedes neue OCR Ergebnis. Auch wenn die Berechnung der Levenshtein-Distanz über mehrere Prozesse ausgelagert wird, kostet das Erstellen von diesen Zeit. Auf Grund der teilweise unzuverlässigen Ergebnisse der OCR und der geforderten Unterstützung für das

Bearbeiten und Scrollen kann jedoch nicht auf die Levenshtein-Distanz verzichtet werden. Die Anzahl der AOIs würde ohne Verwendung der Levenshtein-Distanz deutlich ansteigen.

5. –**decoder**: Definiert, ob beim Decodieren der OCR Ergebnisse ein Greedy-Algorithmus, Beam-Search oder Word-Beam-Search verwendet wird. Standardmäßig wird der Greedy-Algorithmus verwendet. Die Ergebnisse der OCR verbessern sich jedoch durch die Beam- oder Word-Beam-Search. Allerdings hat dies direkte Auswirkungen auf die Laufzeit, da je nach Größe des Baumes, dies den Vorgang der OCR deutlich verlangsamt. Die Größe des Baumes kann dabei durch den Benutzer angegeben werden. Im Gegensatz zu der deutlichen Steigerung der Laufzeit wurden die Ergebnisse allerdings nicht deutlich besser. Dies kann sich für Text in natürlichen Szenen unterscheiden, für Text aus Bildschirmaufnahmen ist es nicht zu empfehlen Beam- oder Word-Beam-Search zu verwenden.

Getestet wurde die Software ebenfalls auf einer Windows-Maschine mit einem Intel i7 8700k, einer Nvidia GeForce RTX 3080 GPU und 32GB Arbeitsspeicher und einem Azure Linux vServer mit 64 vCores, vier Nvidia Tesla GPUs mit insgesamt 64 GB vRAM und ca. 500GB Arbeitsspeicher. Die Geschwindigkeit auf kurzen Videos (ein bis drei Minuten lang) unterschied sich kaum. Auf längeren Videos gibt es einen Unterschied. Getestet wurde dabei ein Video mit Länge von 19 Minuten:

1. **Windows-Maschine**: 122 Minuten und 30 Sekunden
2. **Azure vServer**: 87 Minuten und 55 Sekunden

Dieser Unterschied ergibt sich sowohl aus der Geschwindigkeit, mit der Text in einzelnen Frames erkannt werden kann, als auch aus der Zeit, die die Berechnung der Levenshtein-Distanz in Anspruch nimmt, wenn bereits sehr viele AOIs gefunden wurden.

6.3 Limitierungen der Implementierung

In diesem Abschnitt werden die Limitierungen der Implementierung erläutert. So gibt es Limitierungen im Bereich der OCR, des Mergings und der Laufzeit, die zu beachten sind.

OCR und Code-Dokumente

Die Erkennung von Text hat sich als sehr gut bewiesen. Die besten Ergebnisse werden bei Textdokumenten produziert. Dort funktioniert die OCR teils sogar zeilenweise gut. Meist auch unabhängig von der Schriftgröße. Problematisch wird die Texterkennung innerhalb von Code Dokumenten

mit Syntaxhervorhebung. Die Probleme treten auf, da Text in einer Zeile mehrere Farben haben kann. So kann ein Ausdruck wie ”const variable = 0;” drei verschiedene Farben haben. Ebenso wurde das CRNN mit Wörterbüchern trainiert. Dieses enthält keine für die Softwareentwicklung typischen Variablennamen, die aus mehreren Wörtern zusammengefügt im camelCase-Stil bestehen. Die Confidence sinkt und die Wörter werden nur noch einzeln erkannt. Weitere Probleme machen Sonderzeichen wie ”=” oder ”{” und ”}”. Gerade wenn nur eine Klammer in einer Zeile steht, kann Code als nicht mehr zusammenhängend erkannt werden und es entstehen mehrere AOIs pro Methode.

Merging und unterschiedliche Dokumentenarten

Aus der Erkennung einzelner Wörter entsteht ein weiteres Problem. Damit zum Beispiel in Dokumenten mit Syntaxhervorhebung noch sinnvolle AOIs erkannt werden können, werden bei dem Merging andere Einstellungen benötigt als für herkömmliche Textdokumente. Dazu zählt mitunter, dass die Anzahl der Iterationen, mit denen Textregionen zusammengefügt werden, erhöht werden muss. Auch muss das Margin um die Textregionen erhöht werden, da innerhalb von Code-Dokumenten teilweise sehr viel freier Platz ist. So ist innerhalb einer Methode zwischen zwei Anweisungen häufig eine Leerzeile zu finden. In Textdokumenten ist dies nicht häufig der Fall, falls der Text zusammengehörig ist. Meist haben diese beiden Dokumentenarten auch unterschiedliche Schriftgrößen. Diese wirken sich ebenso auf die Einstellungen des Merging aus. Wenn die Schriftgröße zwischen Dokumenten zu sehr schwankt und das Merging zu sehr auf eine Schriftgröße eingestellt wird, führt dies zu schlechten Ergebnissen bei anderen Schriftgrößen. Dies drückt sich entweder durch zu große AOIs (z.B. das ganze Textdokument) oder durch zu viele kleine AOIs (z.B. einzelne Wörter) aus. Im schlechtesten Fall kann das dazu führen, dass die Genauigkeit für eine Dokumentenart hoch ist, die Ergebnisse der anderen Dokumentenarten allerdings nicht verwertbar sind.

Laufzeit

Neben den Limitierungen in Hinblick auf die Genauigkeit kann die Laufzeit zu Problemen führen. Ohne GPU Unterstützung benötigt das Analysieren eines Frames 24.6 Sekunden auf einem Intel i9 Prozessor aus 2019. Ein älterer GPU aus 2014 beschleunigt den Vorgang auf 4.27 Sekunden. Ein Video der Länge 5 Minuten kann bei 30 FPS 9000 Frames und bei 60 FPS bereits 18000 Frames besitzen, die potentiell alle analysiert werden müssen. In dem 30 FPS Beispiel würde allein die OCR auf jedem Frame 640.5 Minuten benötigen falls ein GPU verwendet und jeder Frame analysiert wird. Aus diesem Grund muss parallelisiert werden. Je mehr Leistung zur Verfügung steht,

desto mehr Frames werden parallel verarbeitet. Hier empfiehlt es sich lange Videos in kleinere aufzuteilen. Dies führt nicht nur zu einer Verbesserung der Geschwindigkeit, es können für unterschiedliche Dokumentenarten und Szenen auch andere Einstellungen verwendet werden, sodass die Ergebnisse ebenfalls besser ausfallen.

Levenshtein-Distanz für das Tracking

Die Levenshtein-Distanz ist neben der OCR der größte Faktor, der die Laufzeit beinträchtigt. Auf die Levenshtein-Distanz kann allerdings nicht verzichtet werden, da die Ergebnisse der OCR nicht immer zuverlässig sind. Die Levenshtein-Distanz hat allerdings auch den Nachteil, dass sie bei kleiner Anzahl von Wörtern, typischerweise in Überschriften, schnell zu schlechten Zuweisungen führen kann. Ein Beispiel wäre die Überschrift "Laufzeit". Wenn nun eine maximale Levenshtein-Distanz von 2 angegeben wird, würde die Überschrift "Funktionale Anforderungen" dieser AOI zugewiesen werden. Bei einer maximalen Levenshtein-Distanz von 2 würde aber auch das Scrollen nicht mehr unterstützt werden. So kann es sein, dass beim Scrollen ein neuer Satzteil erscheint. Bei einer Levenshtein-Distanz von 2 würde nun eine neue AOI entstehen. Auf Grund dessen wurde eine anteilige Levenshtein-Distanz aufgeführt. Diese kann von dem Benutzer angegeben werden. Es wird dabei die Distanz durch die Anzahl der Wörter geteilt.

Kapitel 7

Zusammenfassung und Ausblick

In diesem Kapitel soll abschließend ein Überblick über die Inhalte und Ergebnisse dieser Arbeit gegeben werden. Abschnitt 7.1 Zusammenfassung gibt dabei den Überblick und fasst die Ergebnisse der Arbeit zusammen. In Abschnitt 7.2 Ausblick werden mögliche Erweiterungen der Software und zusätzliche Lösungen vorgeschlagen, um die Ergebnisse weiter zu verbessern.

7.1 Zusammenfassung

Trotz vieler bekannter Vorteile wird Requirement Traceability als Werkzeug des Requirements Engineerings viel zu häufig nicht oder nicht sinnvoll verwendet. Dies ist teilweise auf den hohen Aufwand zurückzuführen, der mit dem Erstellen, Aufrechterhalten und Wiederherstellen von Trace Links entsteht. Aus diesem Grund ist es wünschenswert diese Aufgaben zu automatisieren. Eine Möglichkeit dabei ist die Zuhilfenahme von Eye Tracking. Damit wird der Aufwand nur in ein anderes Gebiet verschoben. Aus diesem Grund soll die Auswertung der Eye Tracking Analysen ebenfalls automatisiert erfolgen. In dieser Arbeit wurde ein Ansatz für einen Schritt in Richtung einer solchen automatischen Auswertung präsentiert. Durch Erkennen von Textelementen in Bildschirmaufnahmen ist es möglich potenzielle AOIs zu generieren. Diese AOIs sind im besten Fall Anforderungen, Methoden einer Klasse oder sonstige Texte mit inhaltlichem Zusammenhang. Da der Text aus der OCR nur zeilen- oder wortweise zuverlässig erkannt wird, mussten die Ergebnisse zusammengefasst werden. Einzelne Wörter helfen nicht bei der automatischen Auswertung, ein Mensch müsste diese manuell verbinden. Der Algorithmus zum Zusammenfügen der Zeilen oder Wörter folgt der Heuristik ”nah beieinander stehende Objekte stehen in inhaltlichem Zusammenhang”. Neben dem Erkennen von Text wurden diese AOIs auch über die Länge der Aufnahme verfolgt. AOIs können sich verändern, zum Beispiel durch Scrollen

oder durch Bearbeiten. Da der Text in maschinenlesbarem Format vorliegt, geschieht das Verfolgen durch Gleichheit des Textes. Um das Editieren zu unterstützen, wird auch die Levenshtein-Distanz berechnet. Liegt die Distanz innerhalb eines vom Benutzer angegeben Schwellenwerts wird die erkannte AOI einer bereits bestehenden zugeordnet. Die Ergebnisse der angefertigten Software haben sich als zuverlässig bei Textdokumenten erwiesen. Da sich Textdokumente allerdings teilweise stark von Dokumenten anderer Art (z.B. Code) unterscheiden, kann dies zu Problemen führen. Wenn die Parameter zu sehr auf die eine Dokumentenart optimiert werden, ist es möglich, dass Ergebnisse für andere Dokumentenarten nicht mehr verwertbar sind.

7.2 Ausblick

Die Genauigkeit der Software im Bereich von Textelementen hat sich zwar als gut erwiesen, es entstehen allerdings teilweise zu viele AOIs. Die Levenshtein-Distanz ist dabei nicht das optimale Mittel für das Tracken der AOIs. Auch im Hinblick auf die Laufzeit, die die Levenshtein-Distanz benötigt, empfiehlt es sich für das Tracken weitere Methoden wie zum Beispiel die inverse Dokumentenhäufigkeit und Termfrequenz auszuprobieren. Auch die Koordinaten der AOIs können dabei berücksichtigt werden, sodass ein neues Gewichtungsschema denkbar ist. Dieses Schema kann dem von Ali et al.[3] präsentierten, ähneln. Eine Abhängigkeit einer Software wäre jedoch nicht nötig.
Um die Geschwindigkeit zu erhöhen, ist es auch möglich, das Video automatisiert in kleinere Teilvideos aufzuteilen und danach über mehrere Prozesse gleichzeitig analysieren zu lassen. Es müsste dafür eine Software, welche die Ergebnisse der Teilanalysen zusammenführt, erstellt werden. Gerade in der CPU Variante der Software würde dies zu einer erheblichen Beschleunigung führen.
Da der Text bereits in maschinenlesbarem Format vorliegt, würde es sich ebenfalls anbieten in Zukunft die Software um Methoden des Natural Language Processing zu erweitern. Durch das Verständnis des Textes könnten AOIs direkt mit einem Label versehen werden. So kann beispielsweise eine Anforderung ein entsprechendes Label erhalten, was die Verwendung innerhalb der Requirements Traceability weiter vereinfacht.
Neben dem Verständnis des Textes ist auch eine Erweiterung um das Unterstützen von Diagrammen denkbar. Diagramme werden, bis auf den enthaltenen Text, bisher nicht unterstützt. Da dieser häufig sehr klein ist, entstehen meist aus Diagrammen mehrere AOIs. Gosala et al.[10] stellen dabei beispielsweise eine Architektur eines CNNs vor, welches für die automatische Klassifikation von UML-Diagrammen verwendet werden kann. Losgelöst von der Requirements Traceability könnte die Software auch verwendet werden um bei der automatischen Auswertung von Eye Tracking

Analysen im Kontext von natürlichen Szenen zu helfen. Zusätzlich zu der Texterkennung wäre dort noch eine Objekterkennung sinnvoll, sodass auch Objekte als AOI erkannt werden können. DeepQuest AI stellt mit ImageAI[1] eine Python Bibliothek zur Verfügung, die Objekte in Bildern erkennen kann und diese Aufgabe übernehmen könnte.

[1] https://github.com/OlafenwaMoses/ImageAI

Literaturverzeichnis

[1] S. Albawi, T. A. Mohammed, and S. Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.

[2] N. Ali, Y.-G. Guéhéneuc, and G. Antoniol. *Factors Impacting the Inputs of Traceability Recovery Approaches*, pages 99–127. Springer London, London, 2012.

[3] N. Ali, Z. Sharafl, Y.-G. Guéhéneuc, and G. Antoniol. An empirical study on requirements traceability using eye-tracking. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 191–200, 2012.

[4] J. Arafat. Unterstützung der automatischen eye tracking datenanalyse mit interaktionsdaten. Bachelor's thesis, Gottfried Wilhelm Leibniz Universität Hannover, Aug 2020.

[5] Y. Baek, B. Lee, D. Han, S. Yun, and H. Lee. Character region awareness for text detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9365–9374, June 2019.

[6] C. Blake. *Eye-Tracking: Grundlagen und Anwendungsfelder*, pages 367–387. Springer Fachmedien Wiesbaden, Wiesbaden, 2013.

[7] L. Eikvil. Optical character recognition. *citeseer. ist. psu. edu/142042. html*, 26, 1993.

[8] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[9] P. M. Fitts, R. E. Jones, and J. L. Milton. Eye movements of aircraft pilots during instrument-landing approaches. *Aeronautical Engineering Review*, 1950.

[10] B. Gosala, S. R. Chowdhuri, J. Singh, M. Gupta, and A. Mishra. Automatic classification of uml class diagrams using deep learning technique: Convolutional neural network. *Applied Sciences*, 11(9), 2021.

[11] O. Gotel and C. Finkelstein. An analysis of the requirements traceability problem. In *Proceedings of IEEE International Conference on Requirements Engineering*, pages 94–101, 1994.

[12] N. Gupta. Artificial neural network. *Network and Complex Systems*, 3(1):24–28, 2013.

[13] P. Hejmady and N. H. Narayanan. Visual attention patterns during program debugging with an ide. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '12, page 197–200, New York, NY, USA, 2012. Association for Computing Machinery.

[14] M. Joos, M. Rötting, and B. M. Velichkovsky. *Spezielle Verfahren I: Bewegungen des menschlichen Auges: Fakten, Methoden und innovative Anwendungen*, pages 142–168. De Gruyter Mouton, 2008.

[15] A. Krenker, J. Bešter, and A. Kos. Introduction to the artificial neural networks. *Artificial Neural Networks: Methodological Advances and Biomedical Applications. InTech*, pages 1–18, 2011.

[16] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[17] A. O. Mohamed, M. P. Da Silva, and V. Courboulay. A history of eye gaze tracking. *hal-00215967*, 2007.

[18] K. Mohan, P. Xu, L. Cao, and B. Ramesh. Improving change management in software development: Integrating traceability and software configuration management. *Decision Support Systems*, 45(4):922–936, 2008. Information Technology and Systems in the Internet-Era.

[19] K. O'Shea and R. Nash. An introduction to convolutional neural networks. *CoRR*, abs/1511.08458, 2015.

[20] H. Partsch. *Requirements-Engineering systematisch.*, chapter 1, pages 6–8. eXamen.press. Springer, Berlin, Heidelberg, 2010. https://doi.org/10.1007/978-3-642-05358-0_1.

[21] G. C. Porras and Y.-G. Guéhéneuc. An empirical study on the efficiency of different design pattern representations in uml class diagrams. *Empirical Software Engineering*, 15(5):493–522, 2010.

[22] M. Rahimi and J. Cleland-Huang. Evolving software trace links between requirements and source code. *Empirical Software Engineering*, 23(4):2198–2231, Aug 2018.

[23] R. Richards. Representational state transfer (rest). In *Pro PHP XML and web services*, pages 633–672. Apress, Berkeley, CA, 2006.

[24] C. Rupp, M. Simon, and F. Hocker. Requirements engineering und management. *HMD Praxis der Wirtschaftsinformatik*, 46(3):94–103, 2009.

[25] F. Schneider and B. Berenbach. A literature survey on international standards for systems requirements engineering. *Procedia Computer Science*, 16:796–805, 2013. 2013 Conference on Systems Engineering Research.

[26] Z. Sharafi, A. Marchetto, A. Susi, G. Antoniol, and Y.-G. Guéhéneuc. An empirical study on the efficiency of graphical vs. textual representations in requirements comprehension. In *2013 21st International Conference on Program Comprehension (ICPC)*, pages 33–42, 2013.

[27] Z. Sharafi, Z. Soh, and Y.-G. Guéhéneuc. A systematic literature review on the usage of eye-tracking in software engineering. *Information and Software Technology*, 67:79–107, 2015.

[28] B. Sharif and H. Kagdi. On the use of eye tracking in software traceability. In *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*, TEFSE '11, page 67–70, New York, NY, USA, 2011. Association for Computing Machinery.

[29] B. Sharif and J. I. Maletic. An eye tracking study on camelcase and under_score identifier styles. In *2010 IEEE 18th International Conference on Program Comprehension*, pages 196–205, 2010.

[30] A. Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.

[31] B. Shi, X. Bai, and C. Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *CoRR*, abs/1507.05717, 2015.

[32] B. Shi, X. Bai, and C. Yao. An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(11):2298–2304, 2017.

[33] A. Veit, T. Matera, L. Neumann, J. Matas, and S. J. Belongie. Coco-text: Dataset and benchmark for text detection and recognition in natural images. *CoRR*, abs/1601.07140, 2016.

[34] B. Walters, M. Falcone, A. Shibble, and B. Sharif. Towards an eye-tracking enabled ide for software traceability tasks. In *2013 7th International Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 51–54, 2013.

[35] C. Wohlin et al. *Engineering and managing software requirements*, chapter 1, pages 1–3. Springer Science & Business Media, 2005.

[36] T. Yusufu, Y. Wang, and X. Fang. A video text detection and tracking system. In *2013 IEEE International Symposium on Multimedia*, pages 522–529, 2013.

[37] X. Zhou, C. Yao, H. Wen, Y. Wang, S. Zhou, W. He, and J. Liang. East: An efficient and accurate scene text detector. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

Anhang A

Flussdiagramm

A.1 Flussdiagramm OCRTask

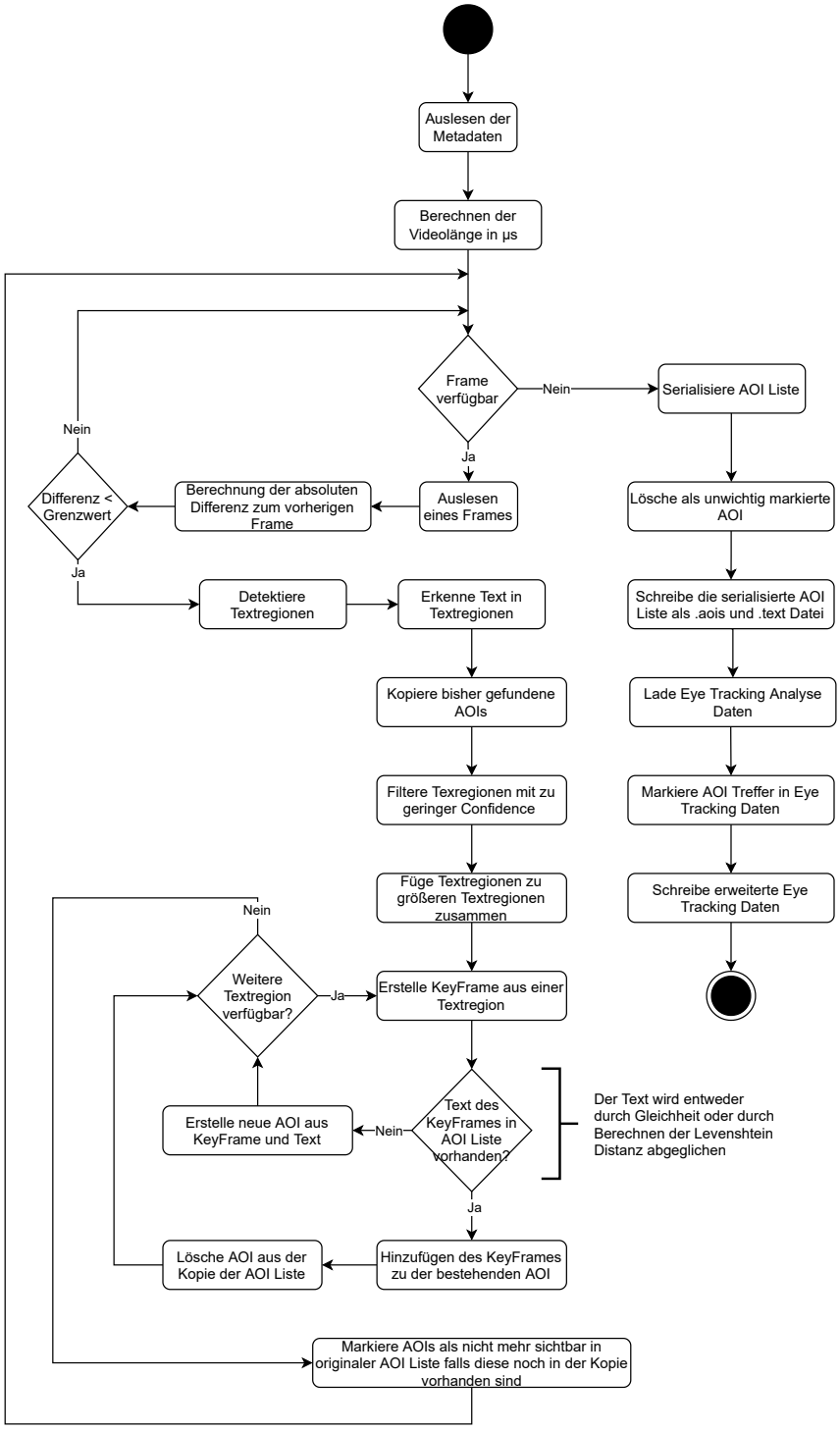

Abbildung A.1: Flussdiagramm für den OCRTask

Anhang B

.aois-Format

B.1 Beispielhafte .aois-Datei


```
{
  "Media": {
    "MediaType": 1,
    "Height": 1080,
    "Width": 1920,
    "MediaCount": 1,
    "DurationMicroseconds": 20700000
  },
  "Tags": [],
  "Version": 2,
  "Aois": [
    {
      "Tags": [],
      "Red": 0,
      "Green": 0,
      "Blue": 0,
      "KeyFrames": [
        {
          "Seconds": 0,
          "IsActive": false,
          "Vertices": [
            {
              "X": 335,
              "Y": 173
            },
            {
              "X": 431,
              "Y": 173
            },
            {
              "X": 431,
              "Y": 225
            },
            {
              "X": 335,
```

```
            "Y": 225
          }
        ]
      },
      {
        "Seconds": 0.04830917874396135,
        "IsActive": true,
        "Vertices": [
          {
            "X": 335,
            "Y": 173
          },
          {
            "X": 431,
            "Y": 173
          },
          {
            "X": 431,
            "Y": 225
          },
          {
            "X": 335,
            "Y": 225
          }
        ]
      },
      {
        "Seconds": 19.3719806763285,
        "IsActive": false,
        "Vertices": [
          {
            "X": 415,
            "Y": 273
          },
          {
            "X": 511,
            "Y": 273
          },
          {
            "X": 511,
            "Y": 325
          },
          {
            "X": 415,
            "Y": 325
          }
        ]
      }
    ],
    "Name": "Beispiel AOI"
  },
 ]
}
```

Anhang C

Zweigüberdeckung

C.1 VideoOCR Package

Datei	Stmts	Miss	Branch	BrPart	Zweigüberdeckung
AOI.py	100	0	42	0	100%
AOIList.py	55	0	20	0	100%
AOIMerger.py	51	0	26	0	100%
Arguments.py	85	0	2	0	100%
CleanupTask.py	15	0	0	0	100%
Frame.py	110	0	36	2	99%
FrameCell.py	19	0	0	0	100%
KeyFrame.py	10	0	2	0	100%
LevenshteinPool.py	7	0	2	1	89%
LoggerFactory.py	14	0	2	0	100%
Media.py	9	0	2	0	100%
OCRTask.py	164	0	70	15	94%
Pipeline.py	23	0	4	0	100%
ProcessingTask.py	51	0	8	0	100%
ResultEncoder.py	10	0	4	0	100%
Task.py	*10*	*2*	*0*	*0*	*80%*
Util.py	26	0	2	0	100%
Vertex.py	6	0	0	0	100%
VideoReader.py	68	19	22	5	67%
Gesamt	823 *(+10)*	19 *(+2)*	244	23	90%

C.2 User Interface und API

Datei	Stmts	Miss	Branch	BrPart	Zweigüberdeckung
app.py	137	4	44	4	96%

Anhang D

Ergebnisse der Software

D.1 Google Suche

Abbildung D.1: Beispiel für die Erkennung von AOIs innerhalb der Google-Suche

D.2 ER-Diagramm

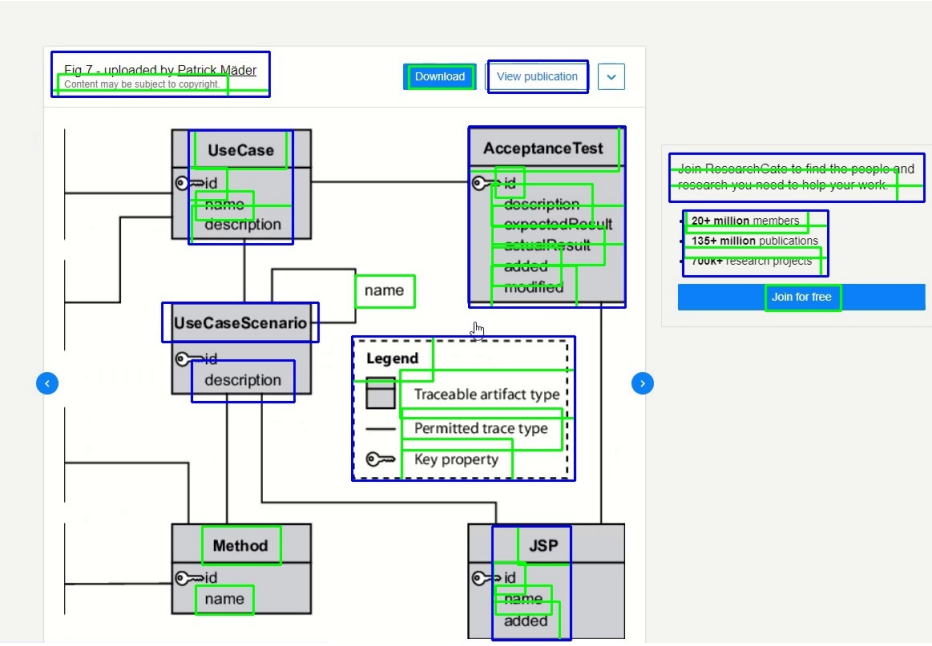

Abbildung D.2: Beispiel für die Erkennung von AOIs innerhalb einer Webseite mit einem ER-Diagramm

D.3 Dokument

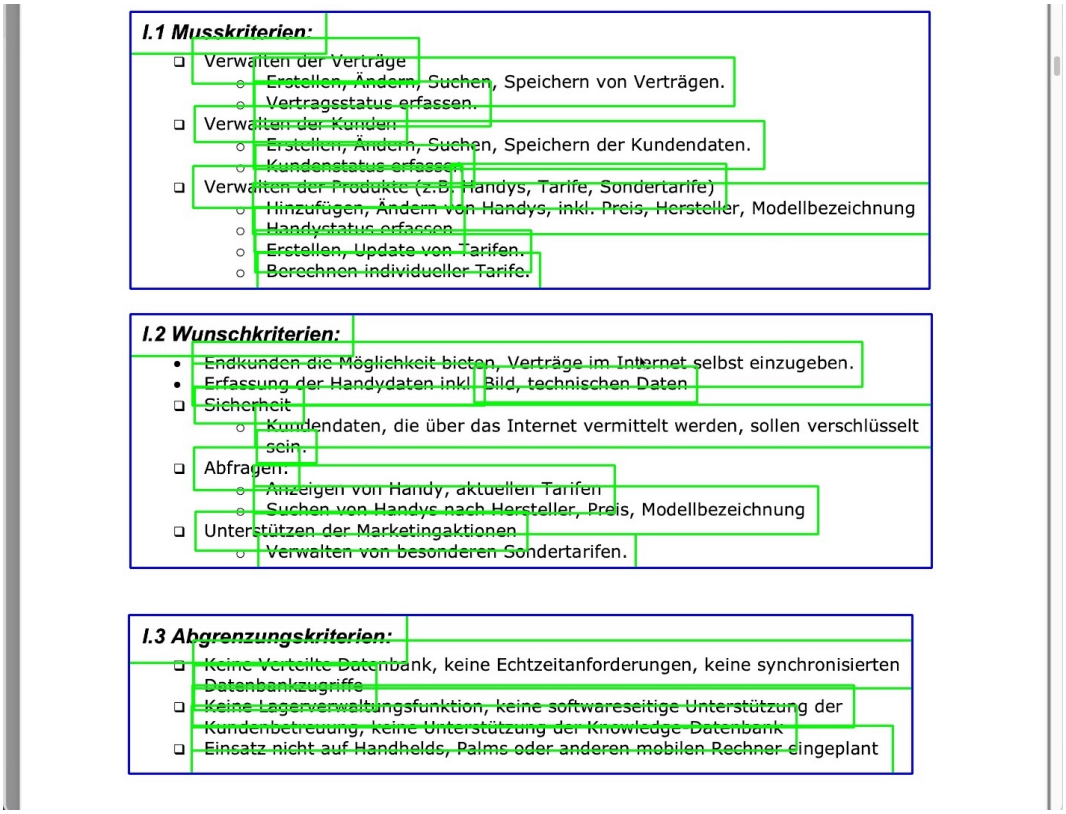

Abbildung D.3: Beispiel für die Erkennung von AOIs innerhalb eines Dokumentes)