HAW HAMBURG

Bachelorarbeit

Lisa Marie Ahlers

Konzeption und Implementierung einer mobilen Anwendung zur Übersetzung deutscher Schriftsprache in deutsche Gebärdensprache mit Hilfe von Texterkennung

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Lisa Marie Ahlers

Konzeption und Implementierung einer mobilen Anwendung zur Übersetzung deutscher Schriftsprache in deutsche Gebärdensprache mit Hilfe von Texterkennung

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stefan Sarstedt
Zweitgutachter: Prof. Dr.-Ing. Olaf Zukunft

Eingereicht am: 10.01.2020

Lisa Marie Ahlers

Thema der Arbeit

Konzeption und Implementierung einer mobilen Anwendung zur Übersetzung deutscher Schriftsprache in deutsche Gebärdensprache mit Hilfe von Texterkennung

Stichworte

Android, Gebärdenschrift, Texterkennung, Lemmatisierung

Kurzzusammenfassung

Für das Erlernen von deutscher Lautsprache kann Gebärdenschrift eine gute Unterstützung sein. Das Ziel dieser Arbeit ist, eine mobile Anwendung zu entwerfen und zu implementieren, welche Text aus Bildern extrahieren kann, um diesen dann zu übersetzen. Dazu wird eine Android Applikation mit Texterkennung und Sprachverarbeitung implementiert, welche Gebärdenschrift und -video Datenbanken verwendet.

Lisa Marie Ahlers

Title of Thesis

Design and implementation of a mobile application for the translation of German written language into German sign language using text recognition

Keywords

Android, Sign Writing, Textrecognition, Lemmatization

Abstract

Sign writing can be a good support for the learning of German spoken language. The goal of this work is to design and implement a mobile application that can extract text from images and translate it. An Android application with text recognition and language processing is implemented, which uses sign language and sign video databases.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Gebärdenschrift bietet die Möglichkeit, Gebärden schriftlich festzuhalten. Damit kann sie beispielsweise verwendet werden, um die Unterschiede in der Grammatik von DGS und deutscher Lautsprache darzustellen. Dies ist für das Erlernen der beiden Sprachen sehr hilfreich, also sowohl für Gehörlose, als auch für Hörende. Es gibt bereits Tools wie den delegs-Editor, die es möglich machen, Dokumente in Gebärdensprache anzulegen. Dies ist unter anderem für das Erstellen von Lehrmaterial hilfreich. Aber wenn ein geschriebener Text der deutschen Lautsprache beispielsweise gedruckt vorliegt, müssten die Worte erst abgetippt werden.
Hier kann eine App, welche in der Lage ist, über von der Kamera aufgenommene Bilder, Wörter von deutscher Lautsprache in Gebärdenschrift zu übersetzen, helfen. Durch Projekte wie das delegs-Projekt und SignPuddle gibt es bereits große Datenbanken mit Gebärdenwörterbüchern. Auf Basis dieser Datenbanken lässt sich die Anwendung erstellen, die auf ein großes Wörterbuch zurückgreifen kann und durch andere Plattformen wie „spread the sign“ und „signdict“ zusätzlich zu Gebärdenschrift auch Gebärdenvideos anbieten kann.
Die Herausforderung einer solchen App sind vor allem die Wörter, welche es nur in der deutschen Laut- und Schriftsprache gibt, die aber keine Entsprechung in Gebärdensprache haben (z.B. Artikel). Des Weiteren gibt es in deutscher Gebärdensprache keine flektierten Verben, es wird nur mit den Grundformen gearbeitet. Daher wird also ein wichtiger Teil der Anwendung sein, mit diesen Gegebenheiten umzugehen.

Diese Arbeit beschäftigt sich mit der Erarbeitung von Anforderungen an eine solche Applikation (Kapitel 3), welche dann als Basis für den Entwurf der Anwendung (Kapitel 4) und die anschließende Implementierung dienen (Kapitel 5). Die Grundlagen für die Arbeit werden im Kapitel 2 dargelegt.

2 Grundlagen

Im folgenden Kapitel werden die grundlegenden Konzepte für die weitere Bearbeitung der Arbeit erläutert. In 2.1 werden die Grundlagen der deutschen Gebärdendsprache erläutert. Daraufhin wird in Verbindung damit in 2.2 die Gebärdenschrift eingeführt. Anschließend werden die Grundlagen der Texterkennung in 2.3 und die Konzepte Stemming und Lemmatization in 2.4 beschrieben. Am Ende des Kapitels wird in 2.5 die mobile Plattform Android vorgestellt.

2.1 DGS

Gebärdensprachen sind genau wie Lautsprachen auch natürliche Sprachen, sie haben sich also ebenso über die Jahre entwickelt und wurden nicht erfunden. Die deutsche Gebärdensprache, abgekürzt DGS, ist keine visuelle Variante des Deutschen. Dies wären lautsprachbegleitende Gebärden, kurz LBG. In diesem Fall wird die Grammatik der Lautsprache übernommen und durch visuelle Darstellung unterstützt. LBG und DGS unterscheiden sich, da DGS eine eigene Grammatik hat und wie eine Fremdsprache erlernt werden muss. [Kleyboldt Thimo, 2016, S. 31 ff]
Einer der wichtigen grammatikalischen Unterschiede zeigt sich in der Verwendung von Verben. In DGS werden Verben, im Gegensatz zur Lautsprache, ohne Flexions-Endung genutzt. Außerdem gibt es die Voll- und Hilfsverben „sein“, „haben“ und „werden“ nicht. Durch diese fehlenden Verben muss ein DGS-Satz nicht zwingend ein Verb enthalten. Der Satz „Ich bin Lehrer“ wird so beispielsweise zu „Ich Lehrer“.
Auch der restliche Satzbau unterscheidet sich zur deutschen Lautsprache: In DGS stellt sich der einfache Satz wie folgt dar: Subjekt - Objekt - Verb, während es in der Lautsprache Subjekt - Prädikat - Objekt ist. [Kleyboldt Thimo, 2016, S. 120 f] Außerdem werden keine Artikel verwendet.
Ein Beispielsatz in Gebärdensprache und in Lautsprache:

Löwe wo? - Gebärdensprache
Wo ist der Löwe? - Lautsprache

2.2 Gebärdenschrift

Sign Writing, im Deutschen als Gebärdenschrift bekannt, ist ein Notationssystem welches 1974 von Valerie Sutton erfunden wurde. [Stephen E. Slevinski Jr., 2012]
Es gibt neben der Gebärdenschrift noch weitere Verschriftungssysteme wie das Stokoe-System und HamNoSys.
HamNoSys ist aber im Vergleich zur Gebärdenschrift nicht für den alltäglichen Gebrauch, sondern für Forschung im Bereich der Gebärdensprache entwickelt. Das System wurde an der Universität Hamburg entwickelt und erstmal 1989 publiziert. [Bentele]
Das Stokoe System begann als Notation benannt nach ihrem Erfinder William Stokoe. Erstmals veröffentlicht im Jahr 1960 wurden damit Linguisten auf die Gebärdensprachen aufmerksam. Inzwischen gibt es viele adaptierte Notationen womit die Stokoe Notation inzwischen mehr eine Familie von Notationen ist. [Martin] In Abbildung 2.1 sind Beispiele für die verschiedenen Notationssysteme zu finden.
Suttons Sign Writing entstand nicht wie die anderen beiden Systeme aus einer linguis-

ASL Sign	Sign Writing	Stokoe Notation	HamNoSys Notation
bears 			

Abbildung 2.1: Beispielwörter in den drei Notationssystemen. Angefertigt basierend auf [Writing, b].

tischen Perspektive. Da sie die Sprache selbst nicht kennt und nur ihre Bewegungen aufschreibt, kann Gebärdenschrift jede Gebärdensprache notieren. Die notierten Bewegungen sind generisch und nicht auf dem Vorwissen über eine bestimmte Sprache basiert. [Writing, a] Verschriftungssysteme wie die Gebärdenschrift sind noch nicht im Alltagsgebrauch, da die Gebärdensprache eine Sprache der direkten Kommunikation ist. Aber für

Gebärdensprachunterricht und Wörterbücher ist die Gebärdensprache ein gutes Mittel. [Kleyboldt Thimo, 2016, S. 47]
Initiativen wie das delegs-Projekt nutzen Gebärdensprache um Lehrmaterialien zu er-

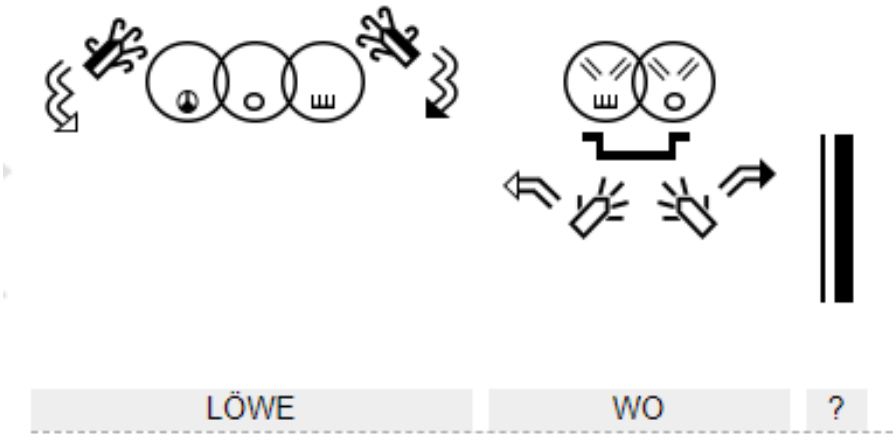


Abbildung 2.2: Beispielsatz in Gebärdenschrift.

stellen und im Unterricht DGS und die deutsche Lautsprache gegenüberzustellen. [delegs Team, 2019]
In Abbildung 2.2 ist ein Beispielsatz in Gebärdenschrift dargestellt. Dieser wurde mit dem delegs-Editor erstellt, welcher auch Wörterbücher für verschiedene Gebärdensprachen anbindet und bereitstellt.

2.3 Texterkennung

Als Texterkennung bezeichnet man den Prozess, Textbereiche in einem Bild zu lokalisieren und diese anschließend über eine Optical Character Recognition (kurz OCR) zu extrahieren. Der Ursprung dieser Systeme lässt sich auf das 19. Jahrhundert zurückführen und stammt aus der Entwicklung von Hilfsgeräten für Blinde.
Viele OCR Systeme sind für eine Sprache oder eine kleine Menge an Sprachen mit gleicher Schrift konstruiert. So weisen beispielsweise OCR Systeme, die ursprünglich für lateinische Schrift gebaut wurden, stark abfallende Genauigkeit für asiatische Schriften auf. Noch problematischer sind Dokumente mit gemischten Schriften. Inzwischen gibt es aber mehr und mehr Vorschläge wie OCR Systeme angepasst werden können, um sie multilingual zu machen. [Peng u. a., 2013]

Der Aufbau eines Texterkennungssystems sieht wie folgt aus: Es gibt vier Schritte, um aus dem Ausgangsbild den enthaltenen Text in digitaler Form zu erzeugen. Siehe Abbildung 2.3.

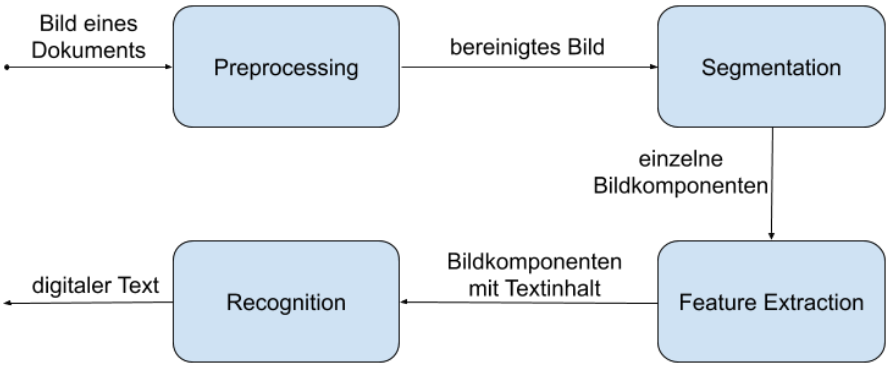


Abbildung 2.3: Verarbeitungsschritte eines Texterkennungssystems.

Preprocessing
Als erstes wird das Preprocessing durchgeführt. Dieser Schritt ist besonders wichtig, da die Effizienz eines Texterkennungssystems stark von der Qualität der Vorverarbeitung abhängt. Beim Preprocessing wird das Ausgangsbild so bearbeitet, dass Text besser zu erkennen ist. Dazu gehören Schritte wie Rauschunterdrückung, Schräglagenkorrektur und Entzerrung. [Nevetha und Baskar, 2015]

Segmentation
Der zweite Schritt ist die Segmentation. Dabei werden Kandidaten für Textregionen basierend auf textspezifischen Parametern gefunden. Die gröbste Form ist die Page Segmentation. Dafür gibt es im Wesentlichen zwei verschiedene Ansätze: top-down und bottom-up. Bei der top-down Methode wird das Bild rekursiv in kleinere Regionen unterteilt. Zwei typische Ansätze der top-down Methode sind der X-Y cut des Projektionsprofils und die background analysis. Wird bottom-up vorgegangen, werden am Anfang die lokal verbundenen Komponenten betrachtet und dann benachbarte Bereiche mit ähnlichen Texturstrukturen oder Features zusammengeführt. Dieses Verfahren ist ebenfalls rekursiv und bricht entsprechend ab, wenn ein bestimmtes Kriterium erfüllt ist, beispielsweise räumliche Beschränkungen. Zwei bottom-up Ansätze sind die Voronoi-Diagramm und die Docstrum basierte Methoden. Je nach System kann auch noch feingranularer segmentiert werden. Dazu gibt es “text line finding” und Word bzw. Character Segmentation. [Peng u. a., 2013]

Feature Extraction
Als Drittes wird die Feature Extraction durchgeführt. Sie findet wichtige Features die Textelemente von nicht textuellen Elementen wie Bildern, Grafiken, Logos und Ähnlichem unterscheiden. Am Ende dieses Schrittes sind also alle nicht textuellen Informationen entfernt, um dann nur noch die textuellen Informationen im Recognition Schritt zu verarbeiten.

Recognition
Der letzte Schritt ist die Recognition. Ein Klassifikationsalgorithmus identifiziert dabei textuelle Information als zugehörig zu bestimmten Zeichen, Symbolen oder Wörtern. Damit wird dann der Bildinhalt auf eine digitale Textrepräsentation abgebildet. Einige bekannte Algorithmen, die für die Erkennung einzelner Zeichen genutzt werden, sind beispielsweise SVM (Support Vector Machine), GMM (Gaussian Mixture Model), HMM (Hidden Markov Model) und Graph Matching. [Nevetha und Baskar, 2015]

2.4 Lemmatization und Stemming

Das Ziel von Stemming und Lemmatization ist grundsätzlich gleich: Die Flexionsformen eines Wortes sollen auf eine gemeinsame Grundform reduziert werden. Dabei nutzen die beiden Verfahren unterschiedliche Techniken, die zu verschiedenen Genauigkeiten führen.

Stemming
Stemming bezeichnet einen heuristischen Prozess, der die Wortendungen abschneidet, um so im Idealfall die Grundform des Wortes zu erhalten. Somit werden bei dieser Methode nicht nur die flektierten Formen eines Wortes zusammengefasst, sondern auch mit diesem verwandte Wörter. [Christopher D. Manning, 2009]
Es gibt dabei zwei Formen von Stemming: schwaches und starkes. Beim starken Stemming werden alle Suffixe und manchmal ebenfalls alle Prefixe entfernt. Das führt im Englischen beispielsweise dazu, dass „illness“ (Substantiv) auf „ill“ (Adjektiv) abgebildet wird. Beim schwachen Stemming sollen nur verschiedene Deklinationen eines Wortes zusammengeführt werden. Für die meisten Applikationen ist das schwache Stemming das geeignetere Verfahren, da Pre- und Suffixe oft eine lexikalische und semantische Bedeutung haben.
Die größten Probleme beim schwachen Stemming sind die folgenden:

- Erfassen der Grenze zwischen Wortstamm und Endung.
- Erkennen von regelmäßigen Veränderungen des Stamms (z.B. „Haus“ - „Häuser“).
- Erkennung von „angeklebten“ Buchstaben (z.B. „essen“ - „ge**g**essen“).
- Erkennen von unregelmäßigen Veränderungen des Wortstamms (z.B. „nehmen“ - „nahm“ - „genommen“).

Für Sprachen mit einer geringen Anzahl morphologischer Varianten, wie beispielsweise Englisch, ist ein Stemming Verfahren, welches den gängigsten Suffix entfernt, üblich. Dazu werden ein Suffix Wörterbuch, ein „longest match“-Algortihmus und einige einfache morphologische Regeln verwendet. Einer der bekanntesten Stemmer dieser Form ist der Porter Stemmer.
Für morphologisch komplexere Sprachen, wie zum Beispiel Deutsch, funktionieren solche Algorithmen aber deutlich schlechter. Dafür gibt es einige Gründe:

- Veränderungen des Wortstamms treten nicht am Ende sondern in der Mitte auf („Haus“ - „Häuser“).
- Alle Regeln der Deklination von Substantiven basieren auf dem Geschlecht und somit ist es, ohne eine tiefgehende Analyse oder ein Wörterbuch, unmöglich zu entscheiden, ob „er“ ein Suffix ist („Bilder“) oder ein Teil des Stamms („Leber“).
- Es gibt viele Ausnahmen, wann ein Vokal gegen einen Umlaut ausgetauscht wird, um eine Pluralform zu bilden („Hund“ - „Hunde“ aber „Mund“ - „Münder“).
- Im Deutschen werden viele zusammengesetzte Substantive verwendet (z.B. „Atomkraftwerk“ und „Atomkraftwerksdirektorenzimmer“).

Aber selbst wenn es keine solchen Gründe gibt, die ein bestimmtes Verfahren für eine Sprache schlechter machen, bleibt immer eine Fehlerrate zurück. Denn ein Stemming-Algorithmus, der nicht auf einer semantischen Textanalyse basiert, wird immer Fehler erzeugen. Das ist gut am Beispiel „nuts“ zu sehen: Dies kann einerseits der Plural von „nut“ sein, aber auch ein Synonym für „crazy“. Dem Stemming-Algorithmus ist es also ohne semantische Analysen in diesem Fall nicht möglich zu wissen, dass dies ein Synonym sein kann und kein Plural. Somit wird auch der beste Stemming-Algorithmus ohne eine semantische Analyse nie in der Lage sein alle Wörter korrekt abzuleiten. [Caumanns, 1999]

Lemmatization
Beim Lemmatization wird ein Vokabular und eine morphologische Analyse hinzugezogen, um die Basisform bzw. „Wörterbuchform“, des Wortes zu finden. Diese Basisform wird auch als Lemma bezeichnet. Für Lemmatizer ergeben sich 2 Herausforderungen:

- mehrdeutige Wortformen auf Basis des Satzkontexts eindeutig machen
- bis jetzt ungesehene Wörter lemmatisieren

Für Lemmatizer mit einem Wörterbuch als Basis ist die Herausforderung also, dieses Wörterbuch zu pflegen. Ein Lemmatizer auf Machine Learning Basis muss entsprechend trainiert werden, um auf ungesehene Wörter generalisieren zu können. Hierfür kann der Satzkontext wieder hilfreich sein. [Bergmanis und Goldwater, 2018]

Vergleich
Ein Stemming-Algorithmus benutzt zwar sprachspezifische Regeln, braucht aber weniger Wissen als ein Lemmatizer. Dieser benötigt im Vergleich zum Stemming-Algorithmus mehr Wissen in Form des Vokabulars und der Morphologie-Analyse. Durch den einfacheren Vorgang beim Stemming ist dieser Algorithmus der Lemmatization im Hinblick auf die Performance überlegen, dafür wird aber Genauigkeit geopfert, die der Lemmatizer bieten kann. Wie groß der Genauigkeitsunterschied ist, hängt aber stark von der Sprache ab. [Christopher D. Manning, 2009]

2.5 Android

Android ist ein Linux basierter Softwarestack welcher für verschiedene Geräte wie Smartphones, Smartwatches, Smart-TV's und Ähnliches genutzt werden kann. Die Android Plattform besteht aus 6 Komponenten (siehe Abbildung 2.4):

- System Apps
- Java API Framework
- Native C/C++ Libraries
- Android Runtime
- Hardware Abstraction Layer

- Linux Kernel

Zum Entwickeln von Android Applikationen wird auf der Java API und den System Apps aufgesetzt. Die Java API bietet die komplette Funktionalität des Android OS. Enthalten sind hier Komponenten wie das View System, Resource-, Notification- und Activitymanager und Content Provider zum Zugriff auf Daten anderer Applikationen. Für Standardfunktionen wie Kalender, Kontaktverwaltung, E-Mail und SMS Postfächer sowie einen Webbrowser gibt es Standardsytemapps auf deren Funktionalität bei der Entwicklung einer neuen Applikation zugegriffen werden kann.
Die C bzw. C++ Libraries sind teilweise ebenfalls durch die Java API verfügbar. Dadurch wird beispielsweise OpenGL für die Entwicklung von Applikation zugänglich. Die Android Runtime, kurz ART, führt ab Android 5.0 alle Applikationen in ihren eigenen Prozessen mit ihrer eigenen ART Instanz aus.
Android setzt auf einem Linux Kernel auf. Dieser erlaubt einerseits den Geräteherstellern Treiber für einen bekannten Kernel zu schreiben und andererseits profitiert Android von den Security-Features des Kernels. [Google, 2019c] Zu diesen Features gehört beispielsweise das userbasierte Berechtigungsmodell, die Prozessisolation und der erweiterbare Mechanismus für sichere Interprozesskommunikation. [Google, 2019d]

Nachdem die Grundlagen der Arbeit erläutert wurden, folgt als nächstes ein Kapitel zu den Anforderungen an die zu entwickelnde Applikation.

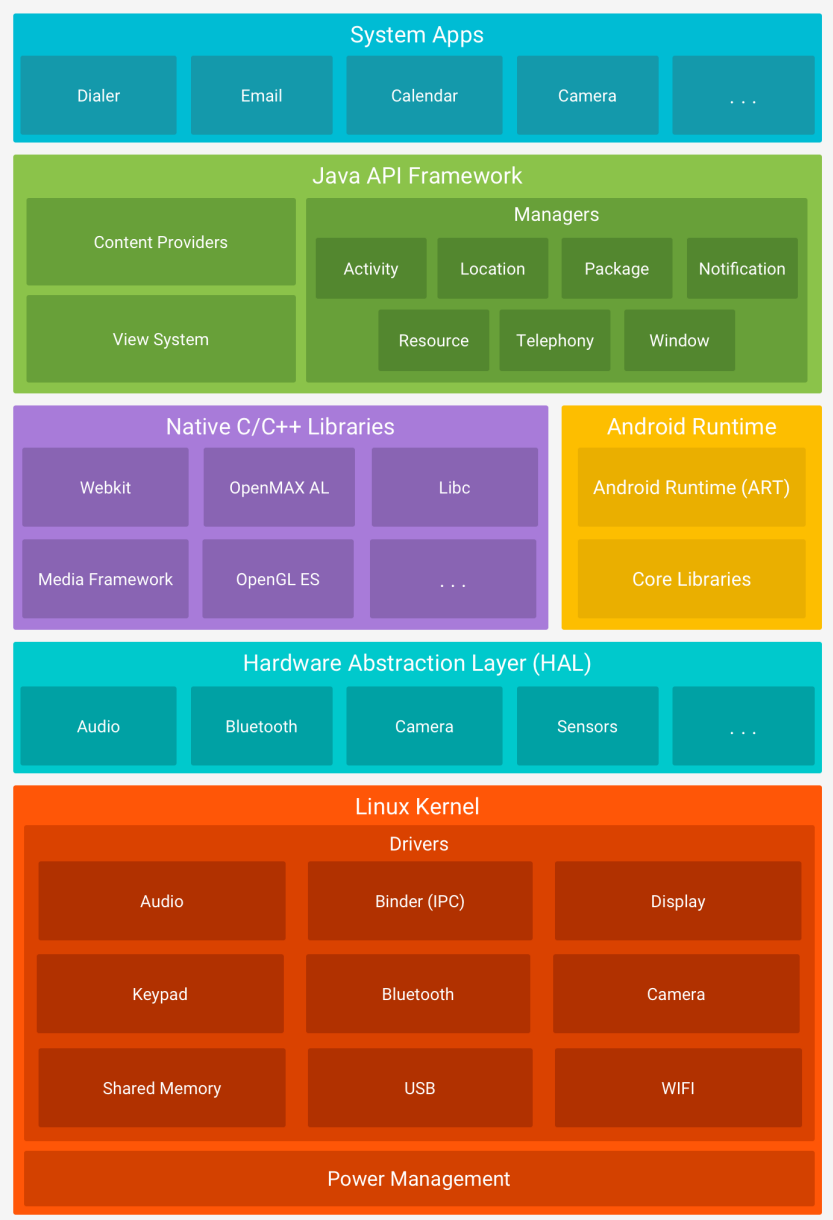


Abbildung 2.4: Der Software Stack der Android Plattform. [Google, 2019c]

3 Anforderungsanalyse

Der folgende Abschnitt arbeitet die Anforderungen an die zu konstruierende Anwendung heraus. Dazu werden die Stakeholder der Anwendung ermittelt und anschließend auf dieser Basis User Stories ausgearbeitet. Für diese User Stories werden dann Akzeptanzkriterien aufgestellt. Anschließend wird ein zentraler Use Case der Anwendung detaillierter beschrieben.

3.1 Stakeholder

Die wesentlichen Zielgruppen der Anwendung sind Gehörlose, welche die deutsche Lautsprache lernen und Hörende die DGS lernen wollen.
Dabei gibt es in diesen zwei Gruppen viele unterschiedliche Kenntnisstände bezüglich deutscher Lautsprache, DGS und Gebärdenschrift. Wichtige Unterscheidungen bilden hierbei vor allem die Kenntnisse in deutscher Laut- bzw. Schriftsprache. Es sollten zum Beispiel möglichst keine komplizierten Sätze in der App als Anweisungen zu finden sein, damit sie auch mit geringer Kenntnis der deutschen Lautsprache verwendet werden kann.

3.2 User Stories

Die folgenden User Stories sind aus einem Treffen mit dem delegs-Team entstanden. Das Team vereint gehörlose und hörende Entwickler, wobei die Hörenden im Prozess sind, DGS zu lernen. Damit bieten sie eine gute Basis, um zu bestimmen, welche Features die Zielgruppe der App benötigt.

1. Als Benutzer möchte ich die einzelnen Wörter in meinem Bild direkt anklicken können und somit danach suchen.

2. Als Benutzer möchte ich Wörter ohne DGS Äquivalent nicht zur Übersetzung auswählen können.
3. Als Benutzer möchte ich die Übersetzung des Wortes in Gebärdenschrift und einem Video sehen können.
4. Als Benutzer möchte ich zwischen alternativen Übersetzungen eines Wortes wechseln können.
5. Als Benutzer möchte ich auch manuell über Eintippen ein Wort suchen können.
6. Als Benutzer möchte ich auch bei flektierten Verben ein Ergebnis aus der Wörterbuchsuche erhalten.
7. Als Benutzer möchte ich sehen, wenn das Ergebnis nicht mit der flektierten Form gesucht wurde, sondern mit der Wörterbuchform.

Es gibt noch weitere Anforderungen, die das Team als sinnvoll erachtet, welche aber im Rahmen dieser Arbeit nicht umgesetzt werden:
Das delegs Wörterbuch existiert für mehrere Sprachen. Damit wäre es wünschenswert, dass die App ebenfalls diese Sprachen unterstützt. Da diese Sprachen aber zum Beispiel andere Einschränkungen haben, welche Wörter sich übersetzen lassen, wird in dieser Arbeit exemplarisch nur mit Deutsch und DGS gearbeitet. Die Ausweitung des Systems auf mehrere Sprachen ist aber für die Zukunft denkbar.
Des Weiteren wäre prinzipiell auch eine Übersetzung von Satzteilen oder ganzen Sätzen denkbar. Dies ist aber durch die veränderte Grammatik ein großer Aufwand und wird deshalb nicht im Rahmen dieser Arbeit behandelt. Es gibt für andere Sprachen wie beispielsweise Arabisch und Arabische Gebärdensprache bereits Bemühungen, solche Systeme umzusetzen [siehe Almasoud und Al-Khalifa, 2011].

3.3 Akzeptanzkriterien

Im Folgenden werden die Akzeptanzkriterien zu den einzelnen User Stories aufgeführt:

Kriterien zu Story 1:

- Nachdem ein Bild gemacht, wurde werden Worte im Bild markiert und sind klickbar.

- Beim Klick auf ein Wort wird dieses Wort in die Suche übernommen.

Kriterien zu Story 2:

- Artikel werden nicht im Bild markiert.
- Beim Klick auf einen Artikel wird dieser nicht in die Suche übernommen.

Kriterien zu Story 3:

- Nach einer erfolgreichen Suche wird eine Übersetzung in Gebärdenschrift und ein abspielbares Gebärdenvideo angezeigt.
- Wird ein Wort nur im Gebärdenschriftwörterbuch oder nur bei den Videos gefunden, wird das gefundene Gebärdenschriftbild bzw. Video angezeigt. Für das fehlende Element erscheint ein Hinweis, dass im entsprechenden Wörterbuch kein Eintrag vorhanden ist.
- Wird das Wort weder im Wörterbücher noch bei den Videos gefunden, erscheint ein entsprechender Hinweis.

Kriterien zu Story 4:

- Gibt es für ein Wort mehrere Übersetzungen, kann der Benutzer zwischen diesen hin und her wechseln. Ihm wird angezeigt, dass dies möglich ist.
- Gibt es nur eine Übersetzung, kann nicht gewechselt werden und es wird auch kein Hinweis darauf angezeigt.

Kriterien zu Story 5:

- Beim manuellen Eintippen eines Wortes wird das Wort für die Suche nicht verändert. Das Ergebnis der Suche wird wie in Story 3 angezeigt.

Kriterien zu Story 6:

- Wird eine flektierte Wortform gesucht (z.B. „geht“) wird im Wörterbuch nach der Basisform („gehen“) gesucht.
- Wird nach einer Basisform gesucht, wird das Wort unverändert im Wörterbuch gesucht.

Kriterien zu Story 7:

- Wird das gesuchte Wort für die Suche verändert, wird dies dem Nutzer angezeigt.
- Wird das Wort nicht verändert, wird kein Hinweis angezeigt.

3.4 Use Case: Suche eines Wortes über Erkennung im Bild

Akteur: Benutzer
Ziel: Der Benutzer erhält eine Übersetzung für ein Wort eines Textes

Auslöser
Der Benutzer möchte Wörter in einem Text übersetzen.
Vorbedingungen
Die Anwendung ist gestartet.
Nachbedingungen
Der Benutzer bekommt eine Übersetzung.

Erfolgsszenario

1. Der Benutzer macht ein Bild von einem Text und bestätigt.
2. Das System markiert Wörter im fotografierten Text. (siehe Abbildung 3.1)
3. Der Nutzer wählt eines der markierten Wörter aus. (siehe Abbildung 3.1)
4. Das System sucht nach einer Übersetzung. (siehe Abbildung 3.2)
5. Das System zeigt dem Nutzer eine Gebärdenschriftübersetzung und ein Gebärdenvideo an. (siehe Abbildung 3.2)

Fehlerfälle

2.a Das System ist nicht in der Lage, Text im Bild zu erkennen und entsprechend Wörter zu markieren. Der Benutzer kann wählen, ob er erneut ein Bild macht oder manuell den Suchbegriff eintippt.

4.a Falls keine Übersetzung gefunden wird, zeigt das System dem Benutzer eine entsprechende Meldung.

4.b Falls nur eine Übersetzung in Gebärdenschrift gefunden wird, zeigt das System diese an und eine Meldung, dass kein Video gefunden wurde.

4.c Falls nur eine Übersetzung in Form eines Videos gefunden wird, zeigt das System diese an und eine Meldung, dass keine Übersetzung im Gebärdenschriftwörterbuch gefunden wurde.

Zugehörige Anforderungen

Story 1, Story 3

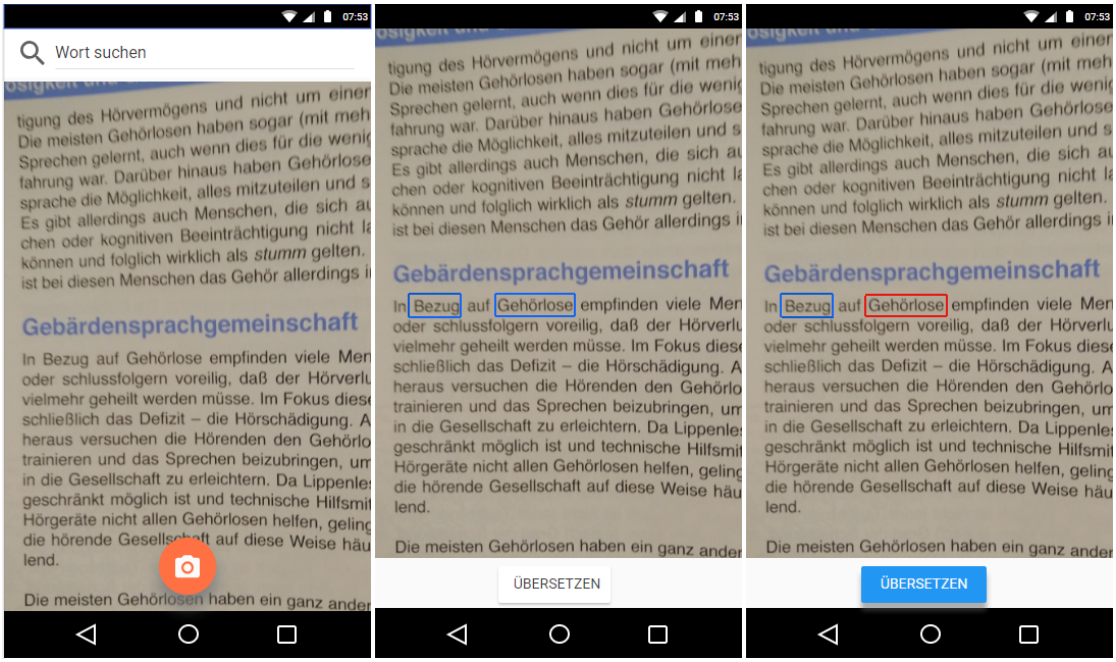


Abbildung 3.1: Mock der Texterkennung und Wortauswahl

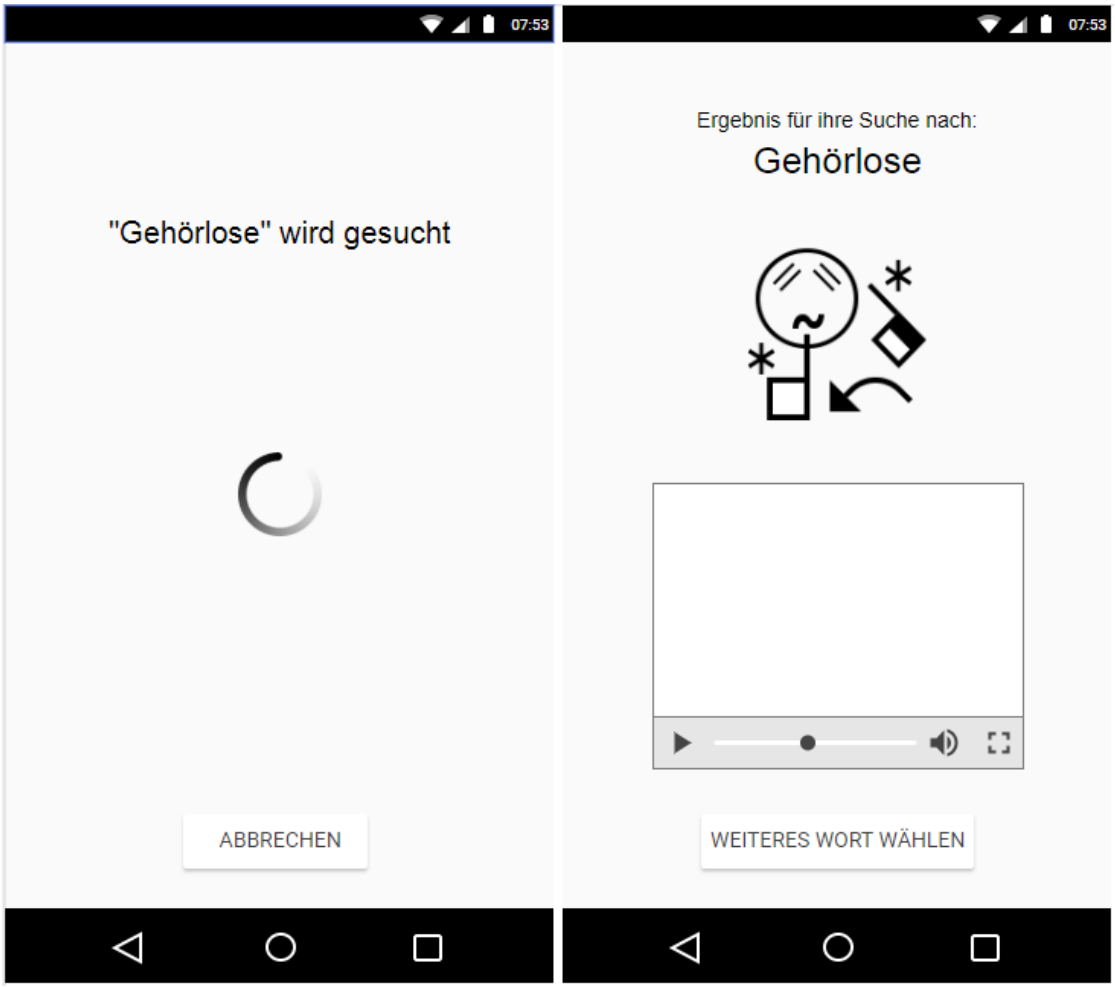


Abbildung 3.2: Mock der Suche und der Ergebnisanzeige

4 Entwurf

Im Folgenden wird das Design des Systems auf Basis der in Kapitel 3 beschriebenen Benutzeranforderungen ausgearbeitet. Dies beinhaltet die Aufteilung in Android Activities und Services, welche die Daten für die Activities laden und Eingaben des Nutzers weiterleiten und verarbeiten.

4.1 Benutzerführung innerhalb der Applikation

Um die Anforderungen aus dem vorherigen Kapitel umzusetzen, müssen die in den Mockups veranschaulichten Funktionen in unterschiedliche Android Komponenten aufgeteilt werden. Die vorgesehene Verteilung der Activities ist in Abbildung 4.1 veranschaulicht. Die Aufteilung der Anwendung in diese Activities soll die Funktionalitäten aufteilen und dafür sorgen, dass unterschiedliche Anwendungsfälle gut umgesetzt werden können.

Die **MainActivity** ist der Einstiegspunkt der Anwendung und bietet dem Nutzer die Möglichkeit, entweder direkt die Suche zu nutzen oder ein Bild zu machen, um seinen Text zu übersetzen.
Die **CameraActivity** kapselt die Interaktion mit der Kamera und das erhaltene Bild wird dann an die nächste Activity weitergegeben.
Die **WordSelectionActivity** zeigt dem Nutzer an, welche Bereiche des Textes erkannt wurden und unterscheidet Wörter die auswählbar sind von anderen. In dieser Activity wählt der Nutzer dann das zu suchende Wort über einen Klick aus.
Dieser Klick löst dann die Suche aus und die **SignSearchResultActivity** zeigt dem Nutzer sein Suchergebnis an. Diese Activity ist zusammen mit der WordSelectionActivity der wichtigste Teil der Anwendung bezogen auf die Anforderungen.
Alternativ kann der Nutzer über die **SignSearchActivity** ein Wort suchen und bekommt dann das Ergebnis ebenfalls in der SignSearchResultActivity angezeigt.

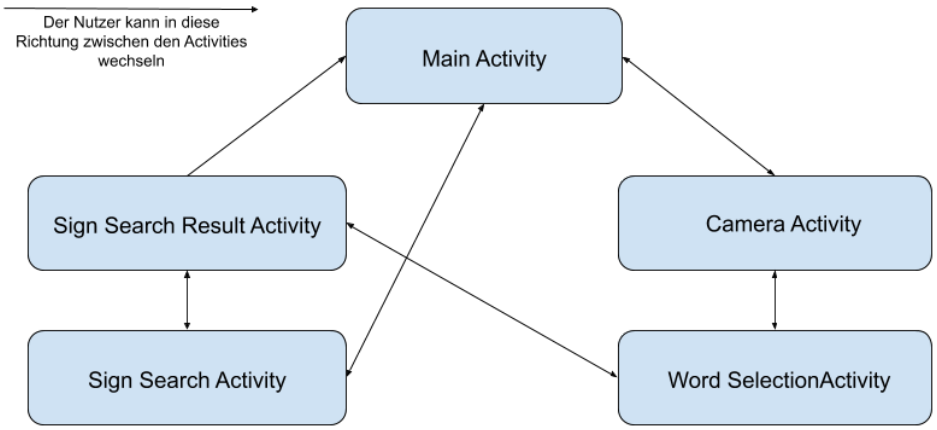


Abbildung 4.1: Benutzerführung innerhalb der Applikation

4.2 Entwurf des Gesamtsystems

Das Gesamtsystem besteht außer den schon angesprochenen Activities auch aus fachlichen Services und der Anbindung von Services aus Fremdsystemen.

Fremdsysteme
Die zwei Fremdsysteme SignDict und delegs erhalten jeweils einen eigenen Service in der Android App. Dieser übernimmt die direkte Kommunikation mit dem Fremdsystem, um eine stabile Schnittstelle für den Rest der Anwendung zu bieten. Die Services sollen über den SignSearchService gebündelt werden, dieser Service enthält die Suchlogik, um diese aus der entsprechenden Android Activity auszulagern. Außerdem wird für asynchrone Services ein ResultHandler verwendet. Dieser wird von der aufrufenden Activity erzeugt und in den Serviceaufruf hineingegeben. So wird der UI Code, der asynchron nach Erhalt des Anfrageergebnisses ausgeführt werden muss, gekapselt. Damit kann man jederzeit eine andere Benutzeroberflächen Technologie wählen, ohne Anwendungslogik aus den Activity-Klassen extrahieren zu müssen.

Fachliche Services
Ein wichtiger fachlicher Service ist der LanguageProcessingService. Dieser Service muss den Text so vorverarbeiten, dass nicht anwählbare Wörter herausgefiltert werden und ausgewählte Wörter durch die Lemmatisierung auf ihre Stammform zurückgeführt werden. Bei diesem Service ist zu beachten, dass er eventuell nicht rein auf dem Gerät laufen

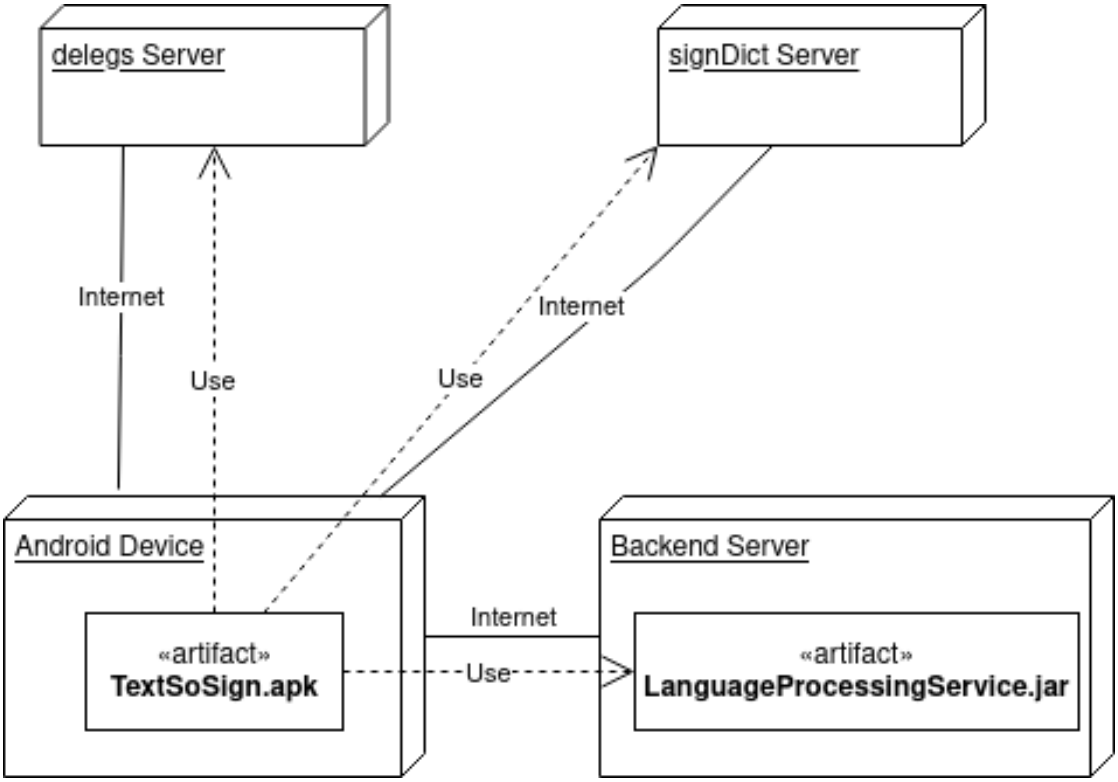


Abbildung 4.2: Verteilungssicht des Systems

kann, sondern auch auf einem Server laufen muss. Diese Trennung kann nötig sein, da Textverarbeitung potentiell sehr ressourcenaufwendig sein kann. Ob der Service komplett auf dem Android Gerät laufen kann, wird im Kapitel 5.1.2 geklärt.

In Abbildung 4.2 ist die Verteilungssicht des Systems gezeigt unter der Annahme, dass der LanguageProcessingService auf einen eigenen Server ausgelagert werden muss.
In Abbildung 4.3 ist eine Übersicht über die Service- und UI-Komponenten der Android App zu sehen; eine Persistenzschicht gibt es nicht, da die App keine Daten speichern muss.

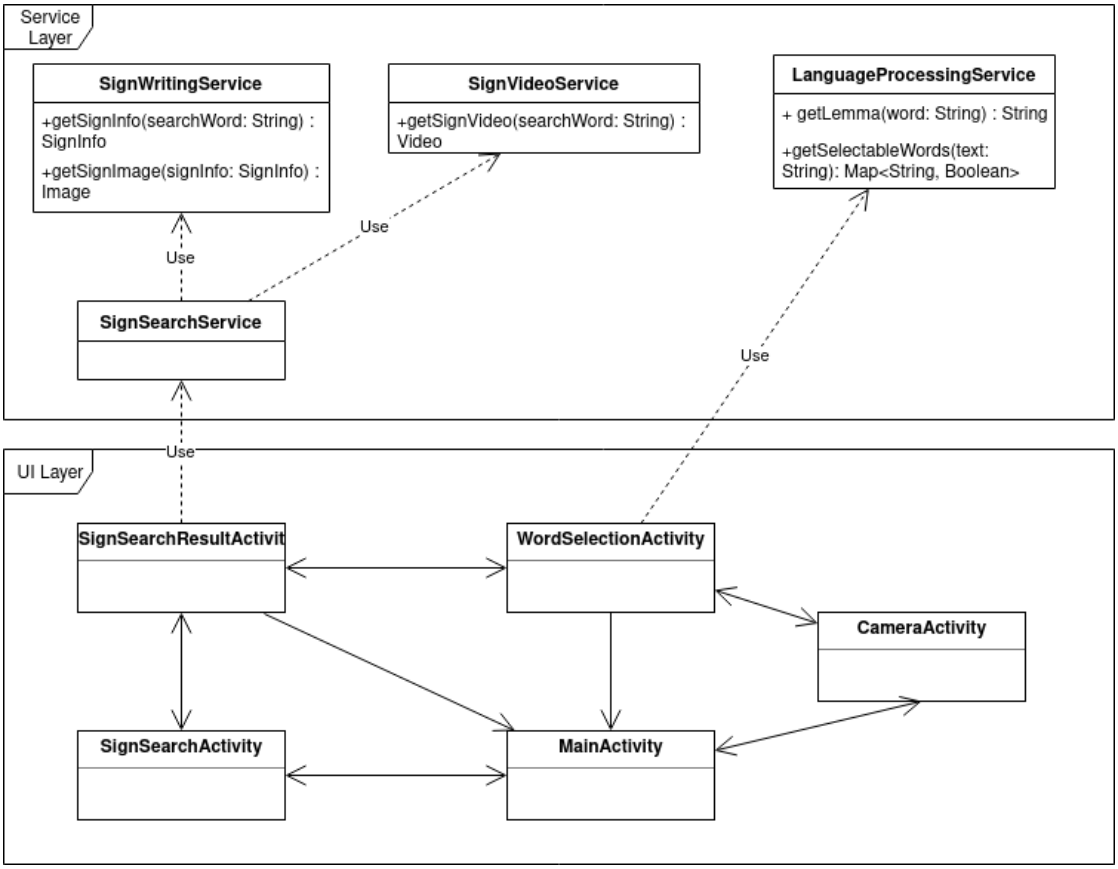


Abbildung 4.3: Android Applikation Systemübersicht

5 Implementierung

Im Folgenden werden unterschiedliche Facetten des Implementierungsprozesses beschrieben. Dazu gehören die Library Auswahl, die technische Anbindung der Fremdsysteme und interessante Aspekte und Probleme der Implementierung der Funktionalitäten der Anwendung.

5.1 Library Auswahl

Ein erster wichtiger Schritt ist die Auswahl der Libraries, die für verschiedene Aspekte der Anwendung genutzt werden. Wichtig sind für die Applikation die Auswahl geeigneter Texterkennungs- und Textverarbeitungs-Bibliotheken.

5.1.1 Texterkennung

Google Firebase ML Kit - Textrecognition
Das Google Firebase ML Kit befindet sich momentan (Stand Dezember 2019) noch in der Beta. Es bietet unter anderem Text Recognition[1] an. Es gibt dabei zwei Möglichkeiten, die Features zu nutzen. Als Offline Version und als Cloud Version.
Die Offline Version ist kostenlos, die Cloud Version ist für die ersten 1000 Aufrufe im Monat ebenfalls kostenlos. Um das Kit zu nutzen, muss das Android Projekt bei Google Firebase registriert werden. Wird die kostenlose Device Lösung verwendet, müssen die ML-Modelle heruntergeladen werden.
Um Dokumente oder Bilder mit viel Text zu verarbeiten, ist die kostenlose on Device Lösung laut Google nicht ausreichend; für diese Fälle wird empfohlen die Cloudvariante zu nutzen.

[1] https://firebase.google.com/docs/ml-kit/android/recognize-text

Es gibt eine detallierte Anleitung zur Einrichtung von Firebase und der Benutzung der Library. Diese lässt sich auch ohne Probleme umsetzten.

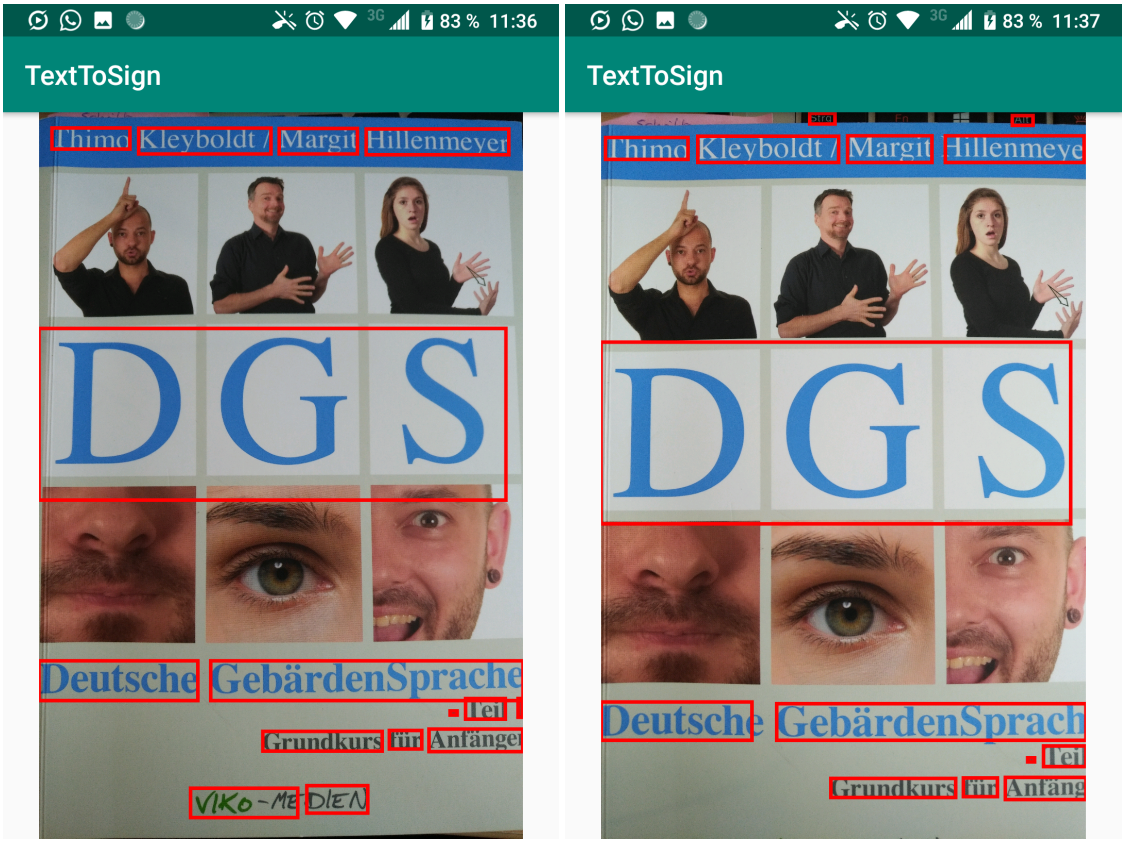
Google Mobile Vision Text Recognition

Bei Google Mobile Vision[2] handelt es sich um den Vorgänger des Google Firebase ML Kits. Die Texterkennung funktioniert für lateinbasierte Sprachen und unterteilt den Text in Blöcke, Zeilen und Wörter. Die Library bringt eine Dependency zu den GooglePlayServices mit sich. Es lässt sich sehr einfach benutzen, da zum Erhalten aller Informationen nur wenige Zeilen nötig sind. Es ist im Gegensatz zur Firebase Variante keine Registrierung nötig.

Vergleich zwischen GoogleFirebase ML und Google Mobile Vision

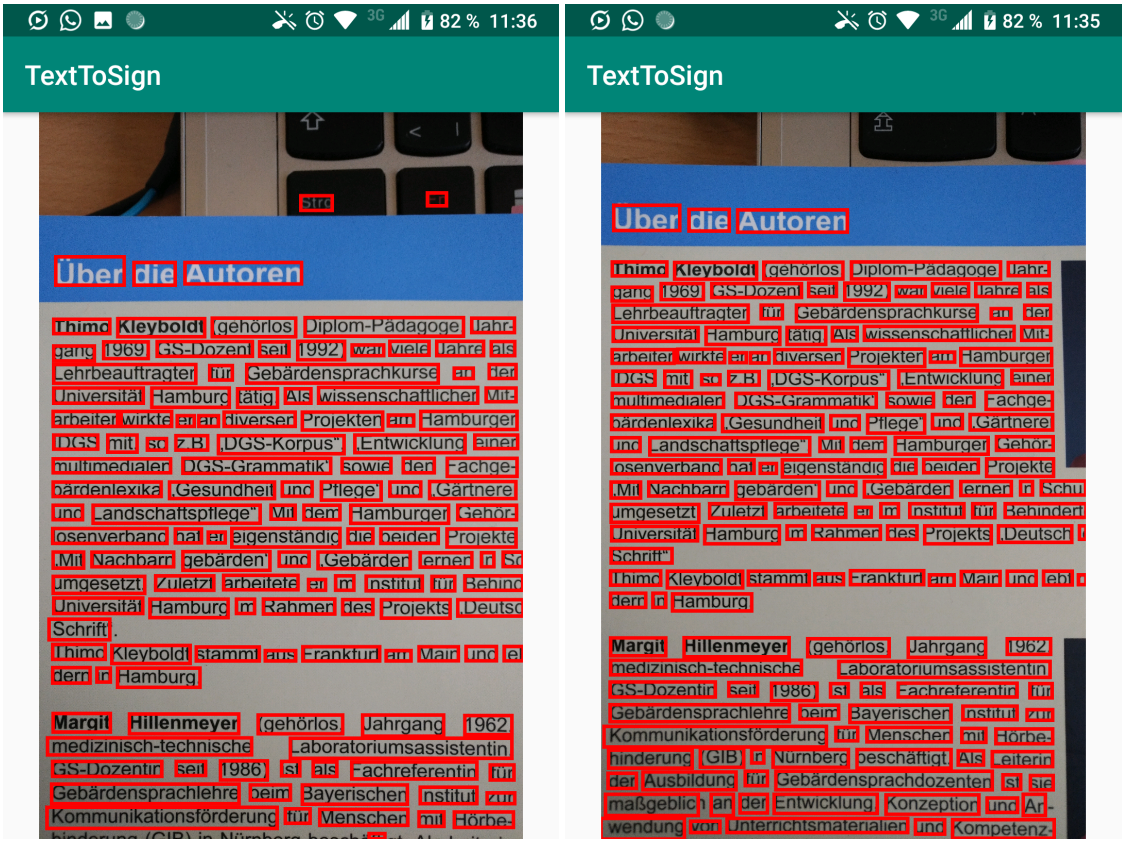
Im Folgenden sind einige Screenshots zu sehen, in denen sowohl gedruckter als auch handgeschriebener Text als Eingabe für die Texterkennung diente.

[2] https://developers.google.com/vision/android/text-overview



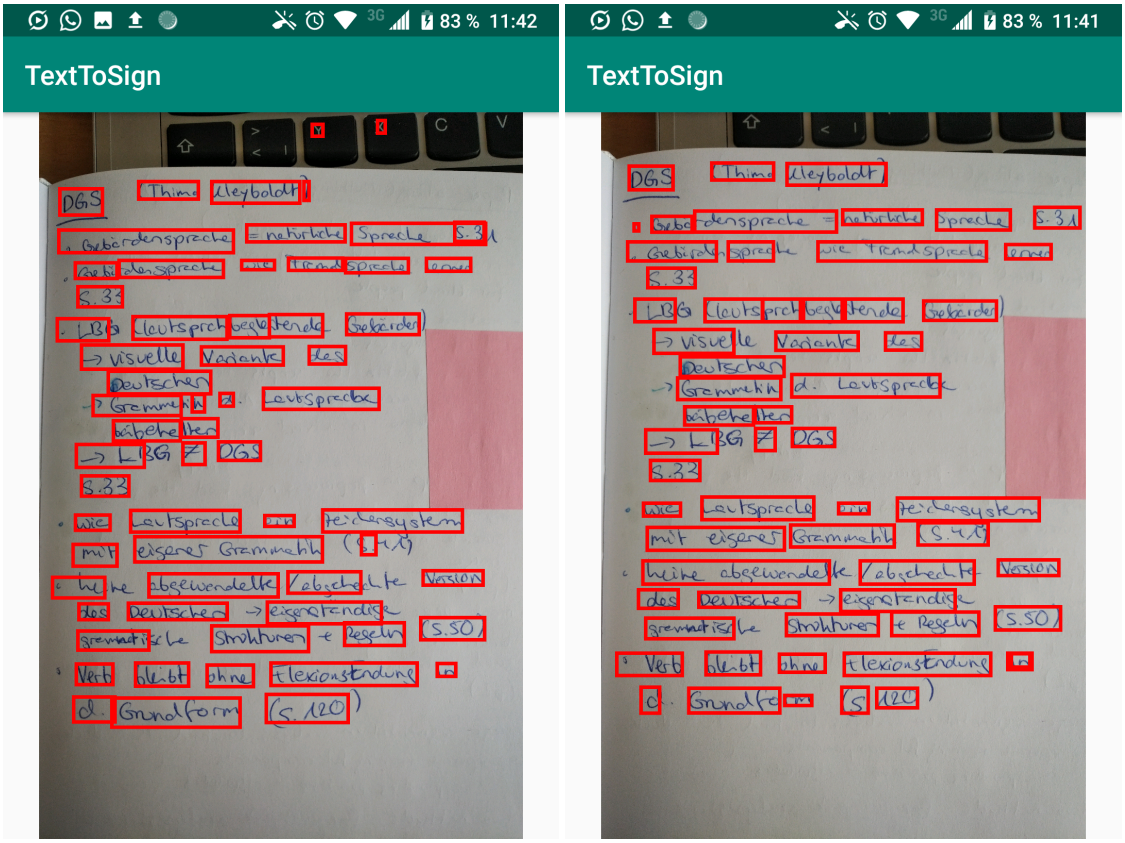
(a) Google Mobile Vision (b) Firebase

Abbildung 5.1: Vergleich bei einer Seite mit Bildern und Text



(a) Google Mobile Vision (b) Firebase

Abbildung 5.2: Vergleich bei einer Seite mit maschinengeschriebenem Text



(a) Google Mobile Vision (b) Firebase

Abbildung 5.3: Vergleich bei einer Seite mit handgeschriebenem Text

Wie in den Vergleichsbildern zu erkennen ist, sind die Unterschiede bei der Erkennung der Position der Wörter, auch beim handschriftlichen Text, minimal. Die korrekte Erkennung der Wörter funktioniert aber bei der Handschrift mit der Firebase Version besser. Für den Test wurde die Device Lösung des Firebase ML Kits benutzt.

Tess-Two (Tesseract)

Tesseract[3] ist eine Open Source OCR-Library, welche ursprünglich in C++ geschrieben ist. Es gibt aber unter anderem auch Varianten für Android (Tess-Two[4]), welche eine Java API über die ursprüngliche Library legen.

Im Gegensatz zu den Libraries von Google muss hier die Datei für die sprachspezifischen Daten selbst heruntergeladen und auf dem Gerät gespeichert werden. Die angebotene

[3]https://github.com/tesseract-ocr/tesseract
[4]https://github.com/alexcohn/tess-two

Schnittstelle ist ähnlich simpel wie bei den Google Libraries. Die gefundenen Worte und ihre Position lassen sich mit einem Funktionsaufruf finden.
Beim Initialisieren der TessBaseAPI Klasse stürzt die App aber ohne aussagekräftige Fehlermeldung ab. Beim Debugging stellt man dann fest, dass ein Fehler beim Aufruf der Methode auftritt, welche dass native Tesseract startet. Eine genauere Fehlermeldung lässt sich aber trotzdem nicht finden.

weitere OCR-Libraries
Eine weitere OCR-Library ist Asprise[5]. Diese Library ist aber für den gegebenen Anwendungsfall eher ungeeignet. Für die Markierung der Wörter auf dem Bild ist die Position der erkannten Worte wichtig. Diese muss bei Asprise aber aus einer xml-Datei ausgelesen werden, was im Vergleich zu den bereits genannten Libraries natürlich deutlich aufwändiger ist.

Es gibt auch einige sehr spezialisierte SDK's wie zum Beispiel das Anyline SDK[6] oder das Blink Input SDK[7]. Diese SDK's sind aber eher dafür ausgelegt spezifizierte Dokumente auszulesen. Dazu gehören beispielsweise Ausweise, KFZ-Kennzeichen, etc. Durch diese Ausrichtung auf festgelegte Dokumentstrukturen sind diese SDK's für den vorliegenden Anwendungsfall aufwändiger einzubinden als für z.B. das Auslesen eines Barcodes. Außerdem sind diese beiden SDK's nach einer zeitlich begrenzten Probenutzung kostenpflichtig.

Nach dem Test der verschiedenen Libraries scheinen die beiden Google Libraries für den gegebenen Fall die beste Wahl zu sein. In der zu bauenden Applikation wird die Device Lösung der Texterkennung des Firebase ML Kit genutzt. Sie liefert gute Ergebnisse bei gedrucktem Text und die etwas besseren Ergebnisse bei handschriftlichem Text sind ausschlaggebend für die Auswahl, da so die Nutzer auch handgeschriebene Texte einfach abfotografieren können.

[5] http://asprise.com/royalty-free-library/java-ocr-api-overview.html
[6] https://anyline.com/
[7] https://microblink.com/products/blinkinput

5.1.2 Textverarbeitung

Es gibt Java Libraries für die Lemmatisierung von Worten. Hierbei ist jedoch interessant, ob diese Libraries überhaupt in einer Android App verwendet werden können. Die Libraries sind oft durch die benötigten Daten groß, was ein Problem für Android sein kann. Es kann versucht werden, die Libraries zu verkleinern. Sollte diese Vorgehensweise keinen Erfolg bringen, kann die Textverarbeitung auf einen externen Server ausgelagert werden.

Eine Library für die Lemmatisierung ist die Stanford NLP Library[8]. Sie ist aber für den gewählten Anwendungsfall ungeeignet, da die Lemmatisierung nur für das englische Sprachpaket zur Verfügung steht. [NLP] Zu Testzwecken wurde die Library aber trotzdem einmal in Android eingebunden und hier zeigen sich sofort Probleme aufgrund der Größe der Library.

Eine Library die auch Lemmatisierung für deutsche Wörter bietet, ist Languagetool[9]. Auch hier ist der erste Versuch, die Library in Android einzubinden. Dabei müssen als Erstes Probleme mit duplizierten Dependencies gelöst werden, welche dann im gradle File über ein exclude entfernt werden. Nachdem dieses Problem gelöst wurde, konnte die Anwendung gebaut werden. Beim Installieren auf dem Gerät kommt es dann aber zu Problemen. Die Installation wird gestartet, scheint dann aber ohne ersichtlichen Grund stehen zu bleiben.

Da die LanguageTool Library auf dem Android Gerät selbst nicht zu funktionieren scheint wird ihre Funktionalität auf einen Server ausgelagert.

Die Benutzung der Languagetool Library ist hingegen sehr einfach, mit einer kleinen Methode bekommt man schon folgendes Ergebnis:
isst -> [essen/VER:2:SIN:PRÄ:NON, essen/VER:3:SIN:PRÄ:NON]
Aus diesem Ergebnis können dann die einzelnen Lemmata extrahiert werden und z.B. über ein Set zusammengefasst werden.

[8] https://stanfordnlp.github.io/CoreNLP/
[9] http://wiki.languagetool.org/java-api

5.2 Anbindung der Fremdsysteme

5.2.1 delegs

Das delegs Projekt bietet die Wörterbücher der Anwendung über eine API nach außen an. Dabei kann jede der von delegs unterstützten Sprachen abgefragt werden.
Die API besteht aus zwei Endpunkten. An einem werden Metadaten der Gebärden für ein gegebenes Suchwort zurückgegeben. Mit diesen Metadaten kann dann ein Bild der Gebärde am anderen Endpunkt abgefragt werden. Wichtig für diese Applikation ist, dass am ersten Endpunkt für das Suchwort Ergebnisse aus allen Sprachen zurückgegeben werden können. Bevor also mit den Ergebnissen der Gebärdenbild Endpunkt abgefragt wird, müssen die Metadaten nach Sprache gefiltert werden. So werden dem Benutzer am Ende nur deutsche Gebärden angezeigt.
Für die Http Anfragen wird Android Volley[10] verwendet, diese Bibliothek bietet eine sehr einfache Schnittstelle für Anfragen und enthält bereits Klassen für verschiedene erwartete Ergebnistypen. Beispielsweise für Bilder und JSON-Objekte.

5.2.2 SignDict

SignDict verwendet eine GraphQL API. Dabei werden Anfragen in Form eines JSON Objekts mit den benötigten Feldern an den Server geschickt. Als Response erhält man das gefüllte Objekt der Anfrage. Die GraphQL erlaubt auch sehr einfache Subselections ohne erneutes Anfragen, da dies über geschachtelte Objekte abgefragt werden kann. Außerdem kann jedes Feld mit eigenen Argumenten angefragt werden.
Für die Abfrage der GraphQL API kann in Android die Apollo Graph QL Android Library[11] genutzt werden.
Um die Anfragen im Code nutzen zu können, wird das Schema der GraphQL API als json-Datei heruntergeladen und in das Projekt eingefügt. Anschließend wird in einer Datei mit der GraphQL eine Query definiert, welche dann über einen gradle Task zu Code generiert wird. Dieser kann dann im Code der Anwendung verwendet werden.
Das Ergebnis der Anfrage an SignDict enthält die Video URL. Um das Video anzuzeigen wird ein ViedoView verwendet; diesen kann man auch per URL befüllen. Wichtig ist, dass das Befüllen des Video Views im UI Thread stattfindet. Um das Ergebnis der Abfrage in

[10] https://developer.android.com/training/volley/
[11] https://github.com/apollographql/apollo-android

der Activity zu verarbeiten, wird im SignVideoService ein ResultHandler verwendet. Eine Instanz dieser Klasse muss bei Anfragen mitgegeben werden und bietet eine Methode handleResult() an. In der Activity wird die handleResult Methode implementiert und kann somit auch auf den UI Thread der Activity zugreifen.

5.2.3 Ergebnis der Anbindung

In der Abbildung 5.4 ist ein Screenshot aus der Anwendung zu sehen. Im oberen Teil wird eines der Gebärdenschriftbilder aus der Abfrage von delegs gezeigt. Darunter ist das Video aus SignDict sichtbar.

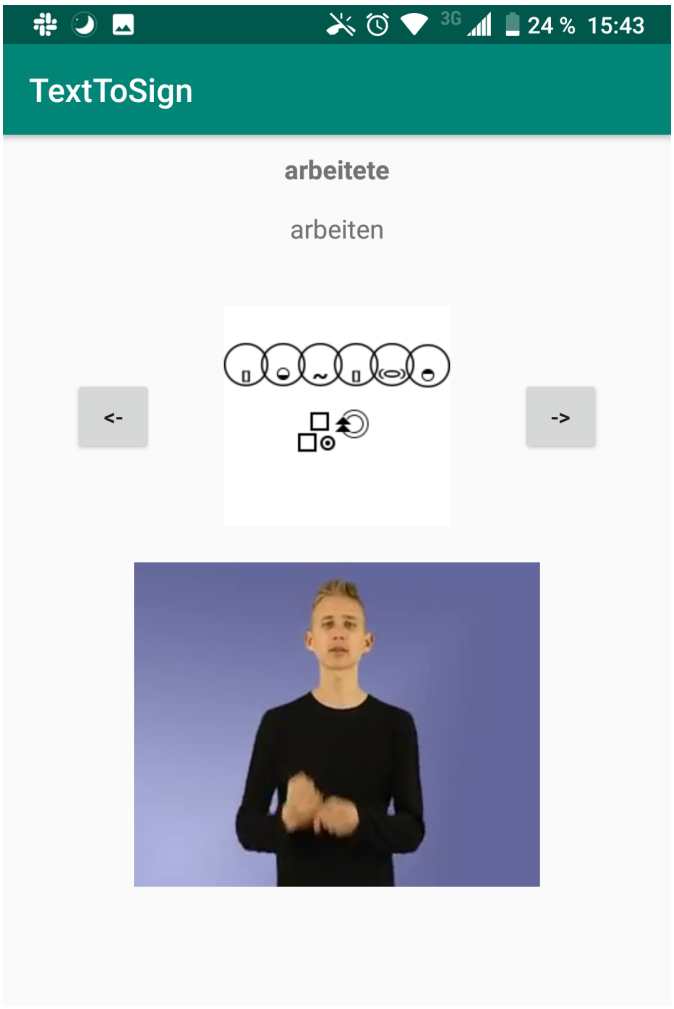


Abbildung 5.4: Ergebnis einer Suchanfrage

Beide Systeme funktionieren zuverlässig und liefern in vielen Fällen Ergebnisse. Was beim Ausprobieren verschiedener Wörter auffällt: delegs liefert für „Löwe“ kein Ergebnis, für „löwe“ gibt es hingegen mehrere. Daher müssen Wörter für die Anfrage an delegs im Code in Kleinbuchstaben transformiert werden. Für SignDict ist dies nicht nötig.

5.3 Textverarbeitung

Filtern der Wörter
Um zu entscheiden, ob ein Wort übersetzt werden kann, ist unter anderem auch die Wortart entscheidend. Artikel können beispielsweise nicht übersetzt werden. Um aber Informationen über die Wortart zu gewinnen, muss am besten der ganze Satz oder zumindest ein Teil übermittelt werden. Für die Languagetool Library ist die Zerlegung in Sätze sogar essentiell, da der Tagger auf Basis von einzelnen Sätzen agiert. Der Text wird also als Ganzes zum Server übermittelt, um dann dort zerlegt und getaggt zu werden. Das Ergebnis des Taggens kann dann interpretiert werden, um festzustellen, ob ein Wort in der Android Anwendung selektierbar sein soll oder nicht. Es kann also an dieser Stelle für Wörter, die den Tag Artikel erhalten haben, direkt die Übersetzbarkeit ausgeschlossen werden.

Lemmatisierung
Bei der Lemmatisierung müssen keine ganzen Sätze oder Textabschnitte übertragen werden. Sie kann also bei Auswahl des entsprechenden Wortes durchgeführt werden. Für die Lemmatisierung wird ebenfalls die oben genannte Library Languagetool verwendet. Da bei der Lemmatisierung oft mehrere Ergebnisse geliefert werden, die aber oft gleich sind, werden die Ergebnisse in einem Set zusammengefasst, um Duplikate zu entfernen.

5.4 Probleme

Unterteilung eines Textes in einzelne Sätze
Zum Zerlegen eines Textes in einzelne Sätze kann der java.text BreakIterator verwendet werden. Er wird mit einem entsprechenden Locale initialisiert und soll dann auch mit Abkürzungen wie „z.B.“ umgehen können. In der Realität funktioniert dies aber nur teilweise und zusätzliche Eigenarbeit ist notwendig um alle Abkürzungen zu unterstützen.

So funktionieren beispielsweise Abkürzungen wie „etc.“ und „z.B.“ ohne weitere Anpassungen. Aber Abkürzungen wie „Dr.“ werden nicht erkannt. Solche Fälle können über zusätzlichen Code natürlich auch erkannt werden, aber es ist nicht offensichtlich welche Abkürzungen vom BreakIterator erkannt werden und welche nicht. Dadurch wird es natürlich Fälle geben, bei denen die Auftrennung in einzelne Sätze nicht richtig funktioniert.

Auswahl eines Wortes auf Basis eines Klicks auf das Wort

Das Auswählen der markierten Wörter soll über einen Klick auf das gezeigte Bild erfolgen. In Android kann an Views ein OnTouchListener angemeldet werden. Dies wird an dem ImageView, welcher das Bild mit gezeichneten Rechtecken um die erkannten Wörter anzeigt, getan. Wenn sie nicht weiter verarbeitet werden, passen die vom MotionEvent des TouchListeners gelieferten Koordinaten aber nicht zum Koordinatensystem, in dem sich die Rechtecke befinden.

Dies liegt daran, dass der ImageView andere Dimensionen als die Bitmap, auf der die Rechtecke berechnet werden, haben kann. Um diesen Fehler zu korrigieren, kann anhand von Breite und Höhe der Bitmap und des ImageViews jeweils die Ratio errechnet werden, um die x und y Koordinate des Events dann damit zu multiplizieren. Interessanterweise ist danach die Verschiebung in x-Richtung behoben, aber es gibt immer noch einen Offset in y-Richtung. Dieser Offset ist auch nicht konstant, sondern nimmt zu, je weiter unten im Bild geklickt wird. Das Gerät, auf dem getestet wurde, hat eine sehr ungewöhnliche Bildschirmgröße, daher war eine Vermutung, dass es durch diese zur beobachteten Verzerrung kam. Beim Versuch auf einem anderem Gerät zeigte sich aber, dass der Fehler unabhängig vom Gerät ist.

Nach einiger Recherche zeigte sich, dass Drawable Objekte in einem ImageView anhand der dort gespeicherten ImageMatrix transformiert werden. So auch die Bitmap des Bildes und der Rechtecke. Das Problem lässt sich lösen, indem die invertierte ImageMatrix auf die Punkte des TouchEvents angewendet wird. Diese Lösung funktioniert, da die Koordinaten des TouchEvents immer im Koordinatensystem des Image Views angegeben sind. Will man nun diese mit Koordinaten der Bitmap vergleichen müssen diese Koordinaten zuerst in das Bitmap-Koordinatensystem überführt werden.

$x_{IV} * IM^{-1} = x_{BM}$

Diese Rechnung lässt sich darüber erklären, dass die Koordinaten mit der ImageMatrix IM in das ImageView Koordinatensystem überführt wurden.

$x_{IV} = x_{BM} * IM$

Um dies aufzuheben kann nun also die inverse der ImageMatrix verwendet werden.

$x_{BM} * IM * IM^{-1} = x_{BM} * I = x_{BM}$

I beschreibt die Identitätsmatrix. Da die inverse Matrix definiert ist als die Matrix M^{-1} für die gilt $M * M^{-1} = I = M^{-1} * M$ kann man also durch Multiplikation eines Punktes mit sowohl M als auch M^{-1} den Effekt der einzelnen Matrizen aufheben.

Damit funktioniert die Selektion der Rechtecke und die Wörter können über einen Klick darauf ausgewählt werden.

5.5 Qualitätssicherung

Die Erfüllung der Anforderungen an die Anwendung wird einerseits mit automatisierten Tests und andererseits auch mit manuellen Akzeptanztests geprüft.

5.5.1 Automatisierte Tests

In Android können zwei Arten von Tests geschrieben werden: Unit Tests und Instrumented Tests. Die Instrumented Tests können verwendet werden, um Integrationstests und automatisierte funktionale UI Tests zu schreiben. [Google, 2019e]

Bei Unit Tests in Android ist zu beachten, dass beim Mocken der Dependencies je nach Art dieser, unterschiedliche Frameworks empfohlen werden. Für Tests mit Dependencies zum Android Framework wird empfohlen, Robolectric zu nutzen um Frameworks zu mocken. Hat die Klasse keine Dependencies zu Android oder nur wenige, nicht komplexe, kann auch ein Framework wie Mockito genutzt werden, um einzelne Objekte zu mocken. [Google, 2019a]

Neben der Möglichkeit die UI über Instrumented Tests zu prüfen, gibt es auch noch das Espresso Test Recorder Tool. Bei diesem Tool können UI Tests erstellt werden, ohne Code zu schreiben. [Google, 2019b]
Die UI Tests wurden aber als Instrumented Tests mit dem Espresso Testframework geschrieben, also mit SourceCode. Diese Tests werden dann am Ende entweder auf einem Android Emulator oder einem angeschlossenen Android Gerät ausgeführt.

Für die Anwendung müssen vor allem die Textverarbeitungsmethoden auf dem Backend

Server getestet werden. In der Android Applikation muss vor allem sichergestellt werden, dass die Wortmarkierungen korrekt angezeigt werden bzw. nicht angezeigt, werden wenn es sich um ein herausgefiltertes Wort handelt. Außerdem kann in UI Tests geprüft werden, ob beispielsweise die Akzeptanzkriterien für die Story 3 erfüllt sind.

5.5.2 Manuelle Akzeptanztests

Um zu verifizieren, dass die in Kapitel 3 beschriebenen Anforderungen erfüllt werden wird die Anwendung auch manuell getestet. Dabei dienen die im Abschnitt 3.3 beschriebenen Akzeptanzkriterien als Richtlinie. Außerdem kann der in Abschnitt 3.4 als Basis genutzt werden.
Für die Durchführung der manuellen Tests werden die im Folgenden gezeigten Anwendungsszenarien durchgespielt. Es wird jeweils die Nutzeraktion und das erwartete Systemverhalten aufgeführt.

Aktion	**Erwartetes Ergebnis**
Klicke auf den Button „Starte Kamera“	Die Kamera öffnet sich
Fotografiere einen gedruckten Text, der Artikel enthält	Das Bild wird gezeigt und nach einer Verarbeitungszeit werden die Wörter mit Kästen umgeben. Die Artikel sind von blauen Kästen umgeben.
Klicke auf einen blauen Kasten	Nichts passiert
Klicke auf einen roten Kasten	Eine neue Ansicht wird geöffnet in der die Ergebnisse der Suche nach dem gewählten Wort angezeigt werden. Wird keine passende Übersetzung gefunden wird ein Hinweis angezeigt, dass keine Übersetzung gefunden wurde.

Tabelle 5.1: Anwendungsszenario: Wort über Texterkennung suchen

Aktion	**Erwartetes Ergebnis**
Klicke auf den Button „Suchen“	Es wird eine Eingabemaske mit einem Suchfeld und einem „Suchen“-Button angezeigt
Gebe „Löwe“ in das Suchfeld ein und klicke auf „Suchen“	Eine neue Ansicht wird geöffnet in der die Ergebnisse der Suche angezeigt werden. Es wird eine Übersetzung in Gebärdenschrift und eine als Gebärdenvideo angezeigt. Auf beiden Seiten des Gebärdenschriftbildes wird jeweils ein Button mit einem Pfeil angezeigt.
Klicke auf einen der Pfeile	Es wird ein anderes Gebärdenschriftbild gezeigt
Klicke auf den anderen Pfeil	Es wird wieder das erste Bild gezeigt
Navigiere über den Zurückbutton des Smartphones zur Suche	Dieselbe Suchansicht wie zuvor wird geöffnet
Suche nach „Test“	Es wird ein Gebärdenschriftbild angezeigt. An der Stelle des Videos wird der Hinweis angezeigt, dass kein Gebärdenvideo gefunden werden konnte.

Tabelle 5.2: Anwendungsszenario: Wort über suchen

6 Fazit

Das Ziel dieser Arbeit war, ein mobiles System zur Übersetzung von abfotografierten Texten in Gebärdensprache zu entwickeln. Die hierfür im Kapitel 3.2 festgelegten User Stories konnten im Rahmen der Arbeit alle umgesetzt werden. Weitere Anforderungen, welche die Anwendung noch erweitern, werden im Ausblick in Abschnitt 6.1 beschrieben.

Durch die vorhandenen Libraries kann eine Texterkennung relativ schnell in eine Android Applikation eingebaut werden. Nach Evaluation verschiedener Bibliotheken wurde für die Anwendung die Texterkennung des Google Firebase ML Kits genutzt. Um die Ergebnisse der Texterkennung mit dieser Library noch zu verbessern, kann die momentan verwendete Device Lösung der Library durch ihre kostenpflichtige Cloudvariante ersetzt werden.

Die Herausforderungen liegen vor allem bei der Textverarbeitung. Der Wunsch war, die Textverarbeitungsbibliothek direkt in der Android App zu nutzen. Da die Einbindung der Textverarbeitungslibrary in die Android Applikation aber so problematisch war, wurde diese letztendlich in eine Serveranwendung verschoben. Unabhängig davon müssen dann andere Herausforderungen gelöst werden, um den Text in eine Form zu bringen, mit der die Bibliothek arbeiten kann. Hier war vor allem die Aufteilung des Textes in die einzelnen Sätze ein wichtiger Punkt. Diese gelingt aufgrund der Verwendung von Punkten in Abkürzungen nicht in allen Fällen, aber kann durch das Pflegen einer entsprechenden Liste verbessert werden.

Durch die genutzten Fremdsysteme delegs und SignDict kann die Anwendung, ohne eine eigene Datenbank zu pflegen, von Anfang an auf eine Vielzahl von Gebärden in Form von Gebärdenschrift und -videos zugreifen. Damit kann sie direkt von Nutzen sein, ohne dass über die Nutzer oder die Extraktion aus anderen Datenbanken neue Einträge generiert werden müssen.

6.1 Ausblick

Die in dieser Arbeit erstellte Anwendung ist ein erster Schritt, um Dokumente mit Hilfe einer App in Gebärdenschrift zu übersetzen. Um dieses Ziel zu erreichen, gibt es noch einige Erweiterungen, die vorgenommen werden können. Dabei liegen die Herausforderungen vor allem darin, ganze Sätze grammatikalisch korrekt zu übersetzen. Wie in Kapitel 3 bereits erwähnt, gibt es bereits Versuche eine solche Übersetzung für Arabisch und arabische Gebärdensprache bereitzustellen [siehe Almasoud und Al-Khalifa, 2011].

Es gibt aber auch noch weitere Erweiterungen, die möglich sind.
Wird beispielsweise kein Wort im Wörterbuch gefunden wie z.B. bei Namen, ist es denkbar, eine Übersetzung mit Hilfe des Fingeralphabetes der Gebärdensprache anzuzeigen.
Außerdem ist die in Kapitel 3.2 beschriebene Ausweitung auf weitere Sprachen natürlich ein wichtiger Punkt. Hierbei sind die Herausforderungen vor allem die Anpassung auf die jeweils andere Grammatik und Übersetzbarkeit von Wörtern.

Literaturverzeichnis

[Almasoud und Al-Khalifa 2011] Almasoud, Ameera M. ; Al-Khalifa, Hend S.: A Proposed Semantic Machine Translation System for Translating Arabic Text to Arabic Sign Language. In: *Proceedings of the Second Kuwait Conference on e-Services and e-Systems*. New York, NY, USA : ACM, 2011 (KCESS '11), S. 23:1–23:6. – URL http://doi.acm.org/10.1145/2107556.2107579. – ISBN 978-1-4503-0793-2

[Bentele] Bentele, Susanne: *HamNoSys*. – URL http://www.signwriting.org/forums/linguistics/ling007.html. – Zugriffsdatum: 06.05.2019

[Bergmanis und Goldwater 2018] Bergmanis, Toms ; Goldwater, Sharon: Context Sensitive Neural Lemmatization with Lematus. In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana : Association for Computational Linguistics, jun 2018, S. 1391–1400. – URL https://www.aclweb.org/anthology/N18-1126

[Caumanns 1999] Caumanns, Jörg: *A Fast and Simple Stemming Algorithm for German Words*. Center für Digitale Systeme - Freie Universität Berlin, 1999. – URL http://www.inf.fu-berlin.de/lehre/WS98/digBib/projekt/_stemming.html

[Christopher D. Manning 2009] Christopher D. Manning, Hinrich S.: *An Introduction to Information Retrieval*. Cambridge University Press, 2009. – URL http://www.informationretrieval.org/. – S. 32 ff

[delegs Team 2019] delegs Team: *Das Projekt*. 2019. – URL https://delegs.de/das-projekt/. – Zugriffsdatum: 16.04.2019

[Google 2019a] Google: *Build local unit tests*. 2019. – URL https://developer.android.com/training/testing/unit-testing/local-unit-tests. – Zugriffsdatum: 02.01.2020

[Google 2019b] Google: *Create UI tests with Espresso Test Recorder*. 2019. – URL https://developer.android.com/studio/test/espresso-test-recorder. – Zugriffsdatum: 02.01.2020

[Google 2019c] Google: *Platform Architecture*. 2019. – URL https://developer.android.com/guide/platform. – Zugriffsdatum: 17.04.2019

[Google 2019d] Google: *System and kernel security*. 2019. – URL https://source.android.com/security/overview/kernel-security.html. – Zugriffsdatum: 06.05.2019

[Google 2019e] Google: *Teyt your App*. 2019. – URL https://developer.android.com/studio/test/. – Zugriffsdatum: 02.01.2020

[Kleyboldt Thimo 2016] Kleyboldt Thimo, Hillenmeyer M.: *DGS Deutsche Gebärdensprache Teil 1*. Hamburg : Viko-Medien, 2016

[Martin] Martin, Joe: *Stokoe Notation*. – URL http://www.signwriting.org/forums/linguistics/ling006.html. – Zugriffsdatum: 06.05.2019

[Nevetha und Baskar 2015] Nevetha, M. P. ; Baskar, A.: Applications of Text Detection and Its Challenges: A Review. In: *Proceedings of the Third International Symposium on Women in Computing and Informatics*. New York, NY, USA : ACM, 2015 (WCI '15), S. 712–721. – URL http://doi.acm.org/10.1145/2791405.2791555. – ISBN 978-1-4503-3361-0

[NLP] NLP, Stanford: *Using Stanford CoreNLP on other human languages*. – URL https://stanfordnlp.github.io/CoreNLP/human-languages.html. – Zugriffsdatum: 11.12.2019

[Peng u. a. 2013] Peng, Xujun ; Cao, Huaigu ; Setlur, Srirangaraj ; Govindaraju, Venu ; Natarajan, Prem: Multilingual OCR Research and Applications: An Overview. In: *Proceedings of the 4th International Workshop on Multilingual OCR*. New York, NY, USA : ACM, 2013 (MOCR '13), S. 1:1–1:8. – URL http://doi.acm.org/10.1145/2505377.2509977. – ISBN 978-1-4503-2114-3

[Stephen E. Slevinski Jr. 2012] Stephen E. Slevinski Jr.: *Modern Sign Writing*. 2012. – URL https://github.com/Slevinski/msw/blob/master/MSW.pdf. – Zugriffsdatum: 15.04.2019. – S. 7

[Writing a] Writing, Center For Sutton M.: *Sign Writing Printing.* – URL http://www.signwriting.org/forums/linguistics/ling004.html. – Zugriffsdatum: 06.05.2019

[Writing b] Writing, Center For Sutton M.: *Writing the Same Signs in Different Transcription Systems.* – URL http://www.signwriting.org/forums/linguistics/ling001.html. – Zugriffsdatum: 06.05.2019

A Anhang

A.1 Codebeispiele

Der gesamte Source Code der Arbeit ist auf der beiliegenden CD zu finden. Im Anhang folgen nun einige Codebeispiele und die Dokumentation der delegs API.

```
fun search(searchWord: String) {
    signSearchService.getSignSearchResult(searchWord, object : ResultHandler<List<SignInfo>> {
        override fun handleResult(result: List<SignInfo>) {
            signInfos = result
            getSignWritingImage()

            if (signInfos.size <= 1) {
                btn_previous_sign.visibility = View.INVISIBLE
                btn_next_sign.visibility = View.INVISIBLE
            } else {
                btn_previous_sign.visibility = View.VISIBLE
                btn_next_sign.visibility = View.VISIBLE
            }

            if (signInfos.isEmpty()) {
                signImageView.visibility = View.INVISIBLE
                image_not_found_text.visibility = View.VISIBLE
            }
        }
    }, object : ResultHandler<String?> {
        override fun handleResult(result: String?) {
            if (result != null) {
                this@SignSearchResultActivity.runOnUiThread {
                    signVideoView.setVideoURI(Uri.parse(result))
                    signVideoView.setOnPreparedListener { mp ->
                        mp.isLooping = true
                    }
                    signVideoView.start()
                }
            } else {
                this@SignSearchResultActivity.runOnUiThread {
                    signVideoView.visibility = View.INVISIBLE
                    video_not_found_text.visibility = View.VISIBLE
                }
            }
        }
    })
}
```

Abbildung A.1: ResultHandler für den SignSearchService

```
val firebaseImage = FirebaseVisionImage.fromBitmap(bitmap)

val detector = FirebaseVision.getInstance().onDeviceTextRecognizer

val rectangles: MutableMap<Rect?, String> = mutableMapOf()
detector.processImage(firebaseImage)
    .addOnSuccessListener { firebaseVisionText ->
        val rectPaint = Paint()
        rectPaint.style = Paint.Style.STROKE

        val tempBitmap =
            Bitmap.createBitmap(bitmap!!.width, bitmap.height, Bitmap.Config.RGB_565)
        val canvas = Canvas(tempBitmap);
        canvas.drawBitmap(bitmap,  left: 0f,  top: 0f,  paint: null);

        languageProcessingService.getSelectableMapping(
            firebaseVisionText.text.replace(
                 oldValue: "\n",
                 newValue: " "
            ),
            object : ResultHandler<Map<String, Boolean>> {
                override fun handleResult(result: Map<String, Boolean>) {
                    for (block in firebaseVisionText.textBlocks) {
                        for (line in block.lines) {
                            for (element in line.elements) {
                                val rect = RectF(element.boundingBox)
                                rectPaint.color = Color.BLUE
                                rectPaint.strokeWidth = 4.0f

                                val isSelectable = result.get(element.text)
                                if (isSelectable ?: true) {
                                    //Finally Draw Rectangle/boundingBox around word
                                    rectPaint.color = Color.RED
                                    rectPaint.strokeWidth = 20.0f
                                    rectangles.put(element.boundingBox, element.text)
                                }
                                canvas.drawRect(rect, rectPaint)
                            }
                        }
                    }
                    //Set image to the `View`
                    imageViewGN.setImageDrawable(
                        BitmapDrawable(
                            applicationContext.resources,
                            tempBitmap
                        )
                    )
                }
            }
        )
    }
    .addOnFailureListener { it: Exception
        // Task failed with an exception
        Toast.makeText( context: this,  text: "Fehler bei der Texterkennung", Toast.LENGTH_LONG)
    }
```

Abbildung A.2: Firebase Text Recognition Anbindung

```
val items = textRecognizer.detect(frame)
var blocks: List<TextBlock> = listOf()
var text = ""

var myItem: TextBlock?
for (i in 0 until items.size() step 1) {
    myItem = items[i]
    text += myItem.components.map { line -> line.components.map { word -> word.value } }

    //Add All TextBlocks to the `blocks` List
    blocks = blocks.plus(myItem)

}
//END OF DETECTING TEXT

val rectPaint = Paint()
rectPaint.color = Color.WHITE
rectPaint.style = Paint.Style.STROKE
rectPaint.strokeWidth = 4.0f

val tempBitmap = Bitmap.createBitmap(bitmap!!.width, bitmap.height, Bitmap.Config.RGB_565)
val canvas = Canvas(tempBitmap)
canvas.drawBitmap(bitmap,  left: 0f,  top: 0f,  paint: null)


val rectangles: MutableMap<Rect, String> = mutableMapOf()
//Loop through each `Block`
for (textBlock in blocks) {
    val textLines = textBlock.components

    //loop Through each `Line`
    for (currentLine in textLines) {
        val words = currentLine.components

        //Loop through each `Word`
        for (currentword in words) {
            //Get the Rectangle/boundingBox of the word
            val rect = RectF(currentword.boundingBox)
            rectPaint.color = Color.RED
            rectPaint.strokeWidth = 20.0f

            //Finally Draw Rectangle/boundingBox around word
            canvas.drawRect(rect, rectPaint)
            rectangles.put(currentword.boundingBox, currentword.value)

            //Set image to the `View`
            imageViewGN.setImageDrawable(
                BitmapDrawable(
                    applicationContext.resources,
                    tempBitmap
                )
            )
        }

    }
}
```

Abbildung A.3: Beispielhafte Google Mobile Vision Einbindung

```
fun getSignInfo(searchWord: String, resultHandler: ResultHandler<List<SignInfo>>) {
    val requestJson = "{searchTokens : [$searchWord]}"

    val jsonObjectRequest = JsonObjectRequest(
        Request.Method.POST,
        baseUrlSignInfo,
        JSONObject(requestJson),
        Response.Listener { response ->
            val signInfos = mutableListOf<SignInfo>()
            if (response.has(searchWord)) {
                val result = response.getJSONArray(searchWord) as JSONArray

                for (i in 0 until result.length()) {
                    val idObject = (result[i] as JSONObject).get("id") as JSONObject
                    val signUpperId = idObject.getString( name: "upperId")
                    val signLowerId = idObject.getString( name: "lowerId")
                    val signLocale = idObject.getString( name: "language")
                    signInfos.add(SignInfo(searchWord, signUpperId, signLowerId, signLocale))
                }
            }
            resultHandler.handleResult(signInfos)
        },
        Response.ErrorListener { error ->
            Log.e(
                SignWritingService::class.java.simpleName,
                error.toString()
            )
        })

    queue.add(jsonObjectRequest)
}

fun getSignImage(signInfo: SignInfo, resultHandler: ResultHandler<Bitmap>) {
    val params =
        "?upperId=${signInfo.upperId}&lowerId=${signInfo.lowerId}&signlocale=${signInfo.signLocale}"
    val jsonObjectRequest = ImageRequest(
         url: baseUrlSignImage + params,
        Response.Listener<Bitmap> { response ->
            resultHandler.handleResult(response)
        }, maxWidth: 0, maxHeight: 0, ImageView.ScaleType.CENTER, Bitmap.Config.RGB_565,
        Response.ErrorListener { error ->
            Log.e(
                SignWritingService::class.java.simpleName,
                error.toString()
            )
        })

    queue.add(jsonObjectRequest)
}
```

Value captured in a closure

Abbildung A.4: delegs API Abfrage

A.2 delegs API Dokumentation

ToSign Service Documentation

URL: delegs.de/delegseditor/signwritingeditor/tosign

Supported REST Methods: POST

Description:

The ToSign is used to find the available signs corresponding to one or multiple searchwords. These searchwords are handed to the service as a JSON object, representing a set of strings, in a “POST” Request.

Input:

JSON Object containing the key “searchTokens” mapped to a set strings, which represent the searchwords.

Response:

JSON Object denoting a mapping from each searchword to the available sign data structure. Each sign contains an id (source, upperid, lowerid, language. Additionally the width and revision are included in the sign.

Example Request:

Request Header:

```
POST /delegseditor/signwritingeditor/toSign HTTP/1.1
Host: delegs.de
Accept: */*
Content-Type: text/html
Content-Length: 51
```

Request Body:

```
{"searchTokens":["Hallo","Test"]}
```

Response:

```
HTTP/1.1 200 OK
Date: Wed, 21 Dec 2016 14:09:32 GMT
Server: Apache/2.4.10 (Debian)
Transfer-Encoding: chunked
```


```
{"test":[{"id":{"source":"IMPORTED_BUT_OVERWRITTEN_IN_DELEGS","upperId":"1272","lowerId":"Test","language":"DGS"},"revision":"96146","width":"123"},{"id":{"source":"IMPORTED_BUT_OVERWRITTEN_IN_DELEGS","upperId":"3048","lowerId":"Test","language":"DGS"},"revision":"127297","width":"123"}],"hallo":[{"id":{"source":"IMPORTED","upperId":"388","lowerId":"Hallo","language":"DGS"},"revision":"62786","width":"78"},{"id":{"source":"IMPORTED","upperId":"3892","lowerId":"hallo","language":"DGS"},"revision":"8937","width":"103"},{"id":{"source":"IMPORTED","upperId":"11971","lowerId":"hallo","language":"DGS"},"revision":"18787","width":"104"}]}
```

SignImage Service Documentation

URL: delegs.de/delegseditor/signwritingeditor/signimages

Supported REST Methods: GET

Description:

The SignImage service is used to load the image of a specific sign. The information which sign image to load is given either by specifying a sign dictionary entry by its id or by handing over a description of the sign image itself. This description is an encoded list of symbols and their positions.

Input:

Sign description, either by specifying each symbol and its position encoded in a string format using the following parameter:

signdata=[encoded_sign_data]

For example s20600x42y83z6r000000wFFFFFF: This string contains the identifier for the symbol (s20600), the position of the symbol (x42y83z6), the rotation (r000000) and the fill (wFFFFFF).

```
GET /delegseditor/signwritingeditor/signimages?signdata=s20600x42y83z6r000000wFFFFFFs14c02x49y100z5r000000wFFFFFF HTTP/1.1
Host: delegs.de
Accept: */*
Content-Type: text/html
Content-Length: 0
```

Or by specifying the id of a sign saved in the delegs dictionary using the following parameters:

upperId=[sign_upper_id]&lowerId=[sign_lower_id]&signlocale=[sign_locale]

The following sign locales are supported:

- ASL (American Sign Language)
- BSL (British Sign Language)
- DGS (Deutsche Gebärdensprache)
- LIBRAS (Língua Brasileira de Sinais)
- LSE (Lengua de Signos Españoles)
- LSF (Langue des Signes Française)
- LSFB (Langue des Signes de Belgique Francophone)
- LSM (Lengua de Señas Mexicana)
- LSQ (Langue des Signes Québécoise)
- PJM (Polski Język Migowy)
- SZJ (Slovenski Znakovni Jezik)

```
GET /delegseditor/signwritingeditor/signimages?upperId=1272&lowerId=Test&signlocale=DGS HTTP/1.1
Host: delegs.de
Accept: */*
Content-Type: text/html
Content-Length: 0
```

Optional parameter:

The scale parameter must be a value greater than 0.0 and lesser than or equal to 4.0.

scale=[scale_factor_of_the_image]

```
GET /delegseditor/signwritingeditor/signimages?upperId=1272&lowerId=Test&signlocale=DGS&scale=3.5 HTTP/1.1
Host: delegs.de
Accept: */*
Content-Type: text/html
```

source=[source_string]

The parameter source can be set to the following values:

- IMPORTED
- DELEGS
- IMPORTED_BUT_OVERWRITTEN_IN_DELEGS
- UNKNOWN
- SYSTEM
- DELEGS_LOCAL

All new users should add the source parameter to the request, otherwise only signs which do exist in signpuddle can be loaded.

```
GET /delegseditor/signwritingeditor/signimages?upperId=1272&lowerId=Test&signlocale=DGS&source=IMPORTED HTTP/1.1
Host: delegs.de
Accept: */*
Content-Type: text/html
Content-Length: 0
```

Response:

The image of the sign or, if the parameters were incorrect or incomplete, a 404 error.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Konzeption und Implementierung einer mobilen Anwendung zur Übersetzung deutscher Schriftsprache in deutsche Gebärdensprache mit Hilfe von Texterkennung

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original