UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Entwicklung einer Android-App zur Erkennung und Übersetzung von Worten in Kamerabildern

Bachelorarbeit

zur Erlangung des Grades eines Bachelor of Science (B.Sc.)
im Studiengang Computervisualistik

vorgelegt von
Sergej Neumann

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	Dipl.-Inf. Diana Röttger (Institut für Computervisualistik, AG Computergraphik)

Koblenz, im Dezember 2012

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

.....
(Ort, Datum) (Unterschrift)

Danksagung

Ich möchte mich an dieser Stelle bei Prof. Dr. Müller bedanken für die Betreuung und Unterstützung dieser Bachelorarbeit.
Der Arbeitsgruppe Computergraphik möchte ich danken für die Bereitstellung eines Android Smartphones.
Vielen Dank auch an die Teilnehmer der Evaluation und die Korrekturleser, die mir eine große Hilfe waren.
Besonderer Dank geht an Lisa Rennwanz, die so gut wie jeden Tag mit mir in der Bibliothek saß und das gleiche Schicksal teilte.

Zusammenfassung

Diese Bachelorarbeit beschreibt den Entwurf und die Implementierung eines Übersetzungsprogramms für die Android Plattform. Die Besonderheit der Anwendung ist die selbstständige Texterkennung mit Hilfe des Kamerabildes. Diese Methode soll den Übersetzungsvorgang in bestimmten Situationen erleichtern und beschleunigen.
Nach einer Einführung in die Texterkennung, die ihr zugrundeliegenden Technologien und das Betriebssystem Android, werden sinnvolle Anwendungsmöglichkeiten vorgestellt. Anschließend wird ein Entwurf der Anwendung erstellt und die Implementierung erläutert. Zum Schluss wird eine Evaluation durchgeführt, dessen Absicht das Aufzeigen von Stärken und Schwächen der Anwendung ist.

Abstract

This bachelor's thesis describes the conception and implementation of a translation software for the Android platform. The specific feature of the software is the independent text recognition based on the view of the camera. This approach aims to enhance and accelerate the process of translation in certain situations.
After an introduction into text recognition, the underlying technologies and the operation system Android useful applications are described. Then the concept of the software is created and the implementation examined. Finally an evaluation is conducted to identify strengths and weaknesses of the software.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Was vor etwa zehn Jahren nur gut ausgerüstete Desktop-PCs leisten konnten vollbringen heutzutage mobile Geräte, die in jede Hosentasche passen. Laut einer Statistik von Bitkom [Bit12] war im April 2012 jeder dritte Deutsche im Besitz eines Smartphones. Im Zusammenspiel von Hardware, die immer kleiner wird und der Betriebssysteme, die immer effizienter arbeiten, bieten die mobilen Geräte ungeahnte Möglichkeiten und Anwendungsgebiete. Vor allem die Integration von zusätzlicher Hardware wie Kameras und Sensoren gibt Entwicklern die Chance neue Ideen umzusetzen, die mit stationären Rechnern keine oder nur schwer Anwendung finden würden. Diese Art der Programme sind mit dem Aufkommen von Apple iPhones und Android-Geräten besonders populär geworden, da sie oftmals sehr nützlich sind und eine noch nie dagewesene Portabilität anbieten. Im deutschen Sprachgebrauch wird diese Art der Anwendungen auch als App[1] bezeichnet. Eine weit verbreitete Sorte von Apps sind Übersetzer. Diese arbeiten meistens online und haben ihre Daseinsberechtigung dadurch, dass sie dem Benutzer das Nachschlagen in Wörterbüchern oder anderen Medien, die man im Gegensatz zu seinem Smartphone selten dabei hat, ersparen.
Eine Weiterführung dieser Idee ist die Beschleunigung der Übersetzung durch das Einsetzen der integrierten Kamera mit anschließender automatischer Texterkennung. Eine App, die dem Benutzer das Lesen und Eintippen des zu übersetzenden Wortes oder sogar Satzes abnimmt, könnte im Vergleich zu einer herkömmlichen Übersetzungs-App Zeit und Mühe sparen. Gerade bei Auslandsaufenthalten könnte eine solche App nützlich sein.

1.2 Ziel

Das Ziel dieser Bachelorarbeit ist die Auseinandersetzung mit den entsprechenden Thematiken und die anschließende prototypische Entwicklung einer Android App die den Benutzer beim Übersetzen unbekannter Wörter sinnvoll unterstützt. Dabei soll in erster Linie ein Grundverständnis für das Prinzip der Texterkennung und die nötige Kompetenz in der Android-Programmierung angeeignet werden. Für die im Programm benötigte Texterkennung und Übersetzung sollen anschließend vorhandene Bibliotheken recherchiert und analysiert werden. Desweiteren soll ein Konzept und mögliche Anwendungsgebiete erarbeitet werden. Das Zusammenführen der verschiedenen Komponenten zu einer benutzerfreundliche App, das

[1] zusätzliche Applikation, die auf bestimmte Mobiltelefone heruntergeladen werden kann (http://www.duden.de/rechtschreibung/App)

Erkennen von Grenzen und Problemen, sowie das Skizzieren von Ansätzen zur Optimierung stehen bei dieser Arbeit im Vordergrund.

2 Grundlagen

Dieses Kapitel soll in der ersten Hälfte eine Einführung in die Texterkennung geben und die Entscheidung für Tesseract als Texterkennungsbibliothek begründen, welche danach genauer betrachtet wird. Hierbei erfolgt die Erklärung für die Entscheidung der Plattform und eine Einführung in Android. Zum Schluss werden fünf verwandte Projekte beschrieben, die im Laufe der Bachelorarbeitrecherche genauer betrachtet wurden.

2.1 Texterkennung

2.1.1 Geschichte

Die Entwicklung der Texterkennung, häufig auch OCR (Optical Character Recognition) genannt, reicht bis in das Jahr 1929 zurück, in dem Gustav Tauschek eine Maschine zum Patent anmeldete, die mit Hilfe von Schablonen und Fotodetektoren die zehn arabischen Ziffern erkennen konnte [Ona11]. Wie in Abbildung 1 zu sehen ist, beleuchtete eine Lampe die zu erkennende Ziffer, die auf eine Linse durch eine rotierende Trommel mit ei-

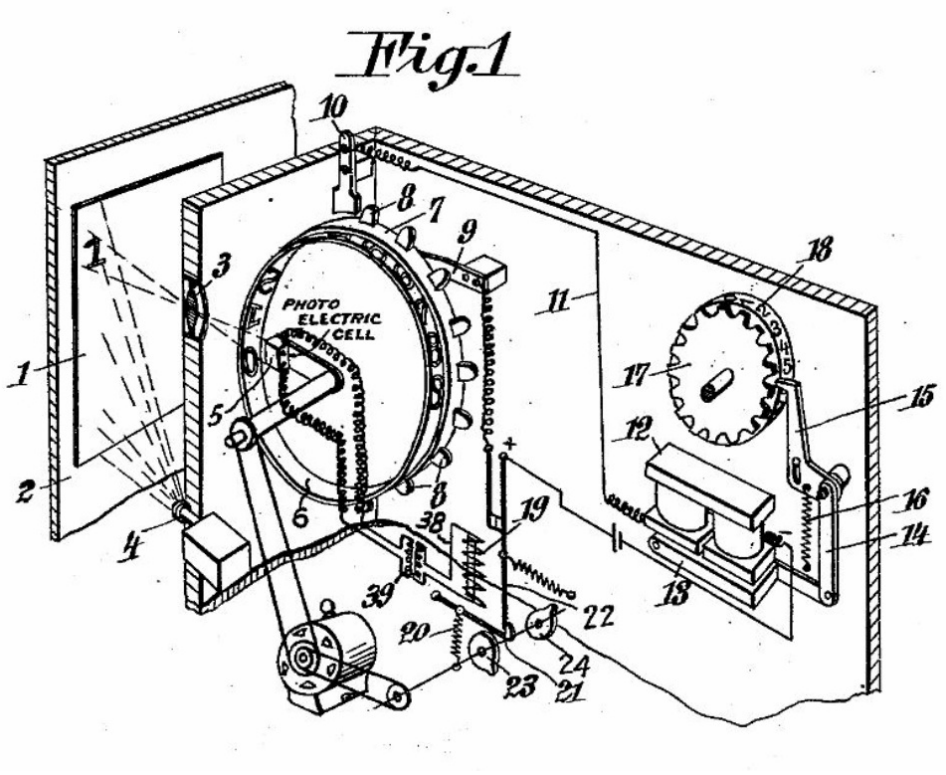

Abbildung 1: U.S. Patent von Gustav Tauschek aus dem Jahre 1929 [Bos12]

nem Negativ der zehn Ziffern abgebildet wurde. Durch eine Fotodiode im Inneren der Trommel konnte der Unterschied zwischen der Zahl mit dem vorgegebenen Muster ermittelt werden. Erreichte dieser ein Minimum, war die Zahl identifiziert. Obwohl diese Maschine nur wenig mit der heutigen OCR-Technologie gemeinsam hat, kann man sie dennoch als den Wegbereiter der Texterkennung ansehen. Die moderne Texterkennung kam das erste Mal in den 40er Jahren mit der Entwicklung des Computers zum Vorschein [Che07]. Schnell entwickelte sich der Wunsch nach einer idealen, selbstständigen Lesemaschine, obwohl man sich der Komplexität der Problemstellung bewusst war. Anfang der 60er Jahre wurden die ersten kommerziellen Computer eingeführt, die gedruckte Ziffern auf Belegen identifizieren konnten. Mit der Zeit gewann OCR für Bereiche wie Einzelhandel, Banken, Krankenhäuser, Post und Zeitungsverlage an Bedeutung. Im Laufe der 70er Jahre wurden genormte OCR-A und OCR-B Schriftarten eingeführt (vgl. Abb. 2), die sich dadurch auszeichneten, dass ihre Buchstaben von einem Rechner besonders leicht zu unterscheiden waren. Diese haben sich aber nie wirklich durchgesetzt. 1978 kam eine Maschine von Kurzweil [Cha12] auf den Markt, die Druckschrift in Blindenschrift oder Ton ausgeben konnte. Solche Geräte konnten damals für Preise im sechsstelligen Bereich erworben werden und waren für den Privatanwender somit kaum finanzierbar. Ende der 70er wurden die ersten Versuche in Richtung Handschrifterkennung unternommen. Erst ab Mitte der 80er, in denen die rasante Entwicklung und die damit verbundene Preissenkung der Computerhardware und Software stattfand, konnten die zuvor erforschten Ansätze umgesetzt und in für Privatanwender bezahlbaren Dimensionen realisiert werden. Die Forschung unterschied sich im weiteren Verlauf insofern, als dass sich auf der einen Seite die kommerziellen Systeme, die auf Effizienz getrimmt waren, und auf der anderen Seite die Laborsysteme, die möglichst fehlerfrei arbeiten sollten, entwickelten. Neben der Effizienz und der Fehlerfreiheit wurde auch intensiv an der Erkennung von multiplen Schriftarten und Zeichen geforscht, wobei das chinesische Alphabet mit etwa 3000 verschiedenen Zeichen zu einer der großen Herausforderungen für die Zeichenerkennung geworden ist. Zusätzlich zur reinen Zeichenerkennung wurden im Laufe der Zeit Methoden entwickelt, welche die Fehlerrate durch kontextbasierte Analysen minimieren sollen.

Abbildung 2: Genormte OCR-A Schriftart [GJo12]

2.1.2 Voraussetzungen der OCR-Bibliothek

Die für diese Bachelorarbeit interessanten Bibliotheken sind diejenigen, die kostenlos und frei zugänglich sind. Sie sollten in jedem Umfeld verwendet, modifiziert und verteilt werden dürfen. Entscheidend ist auch die Programmiersprache, in der die Bibliothek implementiert wurde. Da Android-Anwendungen nur in Java und teilweise in C++ programmiert werden können, kommen auch nur Bibliotheken in die engere Wahl, die in diesen Programmiersprachen geschrieben sind. Die Texterkennung soll offline, also auf dem Gerät selbst geschehen. Eine onlinebasierte Texterkennung ist auszuschließen, da die Up- und Downloadzeiten für ein echtzeitfähiges System zu lange wären. Von großer Bedeutung ist natürlich auch die Qualität der Resultate, die die Bibliothek liefert.

2.1.3 Wahl der OCR-Bibliothek

Neben einigen kommerziellen Angeboten sind im Laufe der Recherche unter anderem folgende Bibliotheken gefunden worden: GOCR[1], Ocrad[2], Kognition[3], Ocre[4], Puma.NET[5] und Tesseract[6]. Eine Evaluation innerhalb einer Dissertation [MW12] an der Universität Erlangen ergab, dass Tesseract im Vergleich zu vielen anderen frei erhältlichen OCR-Anwendungen am Besten abschnitt. Sogar ein Vergleich [Hei09] zwischen Tesseract und drei kommerziellen Produkten ergab, dass das Open-Source-Projekt die besten Ergebnisse erzielt. Abbildung 3 zeigt die Ergebnisse eines Vergleichs [Goh12] verschiedener Texterkennungsprogramme. Abbyyocr[7], eine kommerzielle Variante weist eine fehlerfrei Erkennung auf. Dicht gefolgt, und mit großem Abstand zu den anderen frei Programmen ist in diesem Fall Tesseract. Die im vorangegangen Kapitel aufgezählten Auschlusskriterien und die Ergebnisse der hier genannten Evaluationen führen dazu, dass die Wahl auf Tesseract fällt. Auffällig ist auch der Rückgang neuer Releases[8] der Alternativen zu Tesseract, die vor der Veröffentlichung von Tesseract 2.03 im Sommer 2008 deutlich höhere Veröffentlichungsfrequenzen zeigten. Diese Umstände haben die Entscheidungsfrage hinsichtlich der Texterkennung wesentlich vereinfacht. Infolgedessen waren eigene Tests verschiedener Bibliotheken nicht nötig.

[1] http://jocr.sourceforge.net
[2] http://ftp.gnu.org/gnu/ocrad
[3] http://sourceforge.net/projects/kognition
[4] http://freecode.com/projects/ocre
[5] http://pumanet.codeplex.com
[6] http://code.google.com/p/tesseract-ocr
[7] http://www.abbyy.de/ocr-sdk-windows
[8] Veröffentlichung, besonders einer neuen oder überarbeiteten Software

	abbyyocr	cuneiform	gocr	ocrad	tesseract
License	Proprietary	BSD	GPL2	GPL3	Apache 2.0
Version	8.0	0.9.0	0.48	0.19	SVN r402
Input-Format	PNG	PNM	PNM	PNM	TIF
courier/black	100% (2.92s)	61% (1.11s)	67% (0.09s)	21% (0.02s)	81% (0.63s)
courier/gray	100% (2.85s)	✖	67% (0.09s)	21% (0.03s)	81% (0.63s)
justy/black	11% (3.62s)	3% (1.14s)	31% (0.11s)	1% (0.02s)	15% (0.61s)
justy/gray	14% (3.45s)	✖	31% (0.10s)	1% (0.02s)	15% (0.60s)
times/black	100% (2.80s)	96% (1.07s)	76% (0.16s)	82% (0.03s)	92% (0.74s)
times/gray	100% (2.87s)	✖	76% (0.16s)	82% (0.03s)	92% (0.74s)

Abbildung 3: Ergebnisse einer Evaluierung verschiedener Texterkennungsprogramme [Goh12]

2.2 Tesseract

2.2.1 Entwicklung

Tesseract ist eine Open Source Software, die anfangs zur reinen Zeichenerkennung diente und später auch eine integrierte Analyse der Seitengestaltung besaß. Sie ist in C++ programmiert und beinhaltet keine grafische Benutzeroberfläche. Ihre Anfänge reichen bis in das Jahr 1984 zurück, indem Tesseract als Forschungsprojekt bei der Firma Hewlett-Packard in Bristol begann [R.07]. Von da an wurde sie zehn Jahre lang weiterentwickelt bis sie schließlich 1995 bei einer jährlichen OCR-Prüfung mit anderen kommerziellen Systemen mithalten konnte. Nachdem HP aus dem OCR-Markt ausstieg, lag das Projekt bis 2005 brach. Mit der Hilfe des Information Science Research Institute der University of Nevada, Las Vegas konnte HP den alten Code als Open Source Projekt über Google herausbringen. Einige Fehler, die seit 1995 im Code lagen, mussten beseitigt werden, bis das Projekt dann stabil genug war, um wiederveröffentlicht zu werden [Vin12]. Zu diesem Zeitpunkt war Tesseract nur auf lateinische Schriftzeichen und die englisch Sprache spezialisiert. Mittlerweile existieren Sprachpakete, so-

genannte "traineddata-files" für mehr als 30 Sprachen[1] und ihre dazugehörigen Schriftzeichen. Seit Version 2 lassen sich diese im Trainingsmodus selbst erstellen. Hierfür werden Tesseract eine Reihe von Zeichen-Samples vorgegeben, aus denen es sich Dateien generiert, die beim Abgleichen der Buchstaben benötigt werden. Weiterhin können die Trainingsdateien Wörterlisten aus derjenigen Sprache beinhalten, für die sie erstellt wurde.
In den letzten fünf Jahren hat es nun zwei große Versionssprünge gegeben. Obwohl es ein Open Source Projekt ist, findet die Weiterentwicklung nicht öffentlich statt, sodass Details zu weiteren Funktionen und der aktuellen Arbeit im Verborgenen bleiben. Fertiggestellte Versionen bringen jedoch eine ausreichende Dokumentation und eine Liste der Neuerungen mit sich, sodass sich Tesseract in andere Projekte als Texterkennungs-Engine einbinden lässt.

2.2.2 Funktionsweise

Als Eingabe Tesseract ein Farb- oder Graustufenbild in *tiff* oder *bmp* Format übergeben (vgl. Abb. 4), welches zunächst in Layoutelemente aufgeteilt wird. Anschließend wird das Bild einer Binarisierung unterzogen, bei der das Schwellwertverfahren von Otsu angewendet wird. Der Algorithmus setzt voraus, dass eine dominante Vorder- und eine Hintergrundfarbe existiert und berechnet einen optimalen Schwellwert, der das Bild in zwei Pixelklassen mit minimaler Varianz aufteilt. Danach wird ein sogenanntes Connected Component Labeling durchgeführt, welches zusammenhängende Segmente als Blobs, also als potentielle Buchstaben, markiert. Da der Text selten exakt waagerecht im Bild vorzufinden ist, muss die Verdrehung ermittelt werden. Dazu wird berechnet, welche Blobs benachbarte Buchstaben oder Worte sein könnten. Aus den Richtungen der Verbindunglinien zwischen den Blobs kann dann die Verdrehung bestimmt werden. Da eine Korrektur mittels Rotation einen Qualitätsverlust nach sich ziehen könnte, wird darauf verzichtet. Nachdem aus den Layoutelementen die Textzeilen gefunden sind, werden präzisere Grundlinien mittels quadratischer Splines ermittelt. Dies ermöglicht den besseren Umgang mit leicht gekrümmten Zeilen, welche in der Realität oft vorkommen können. Anschließend wird geprüft, ob es sich um eine proportionale Schrift handelt oder nicht. Danach werden die Textzeilen in Worte zerteilt, um sie danach in Buchstaben zu separieren. Schriften, die eine feste Breite aufweisen sind in diesem Fall leichter zu behandeln, kommen aber in der Realität seltener vor. Wenn alle möglichen Blobs gefunden und in ihre kleinsten Bestandteile zerteilt sind, erfolgt die tatsächliche Zeichenerkennung. Auf dieser Ebene geht es um Mustererkennung, die Zeichen für Zeichen abläuft. Auf Wortebene finden zwei Durchläufe statt. Beim Ersten werden erfolgreiche Worte, al-

[1] http://code.google.com/p/tesseract-ocr/downloads/list

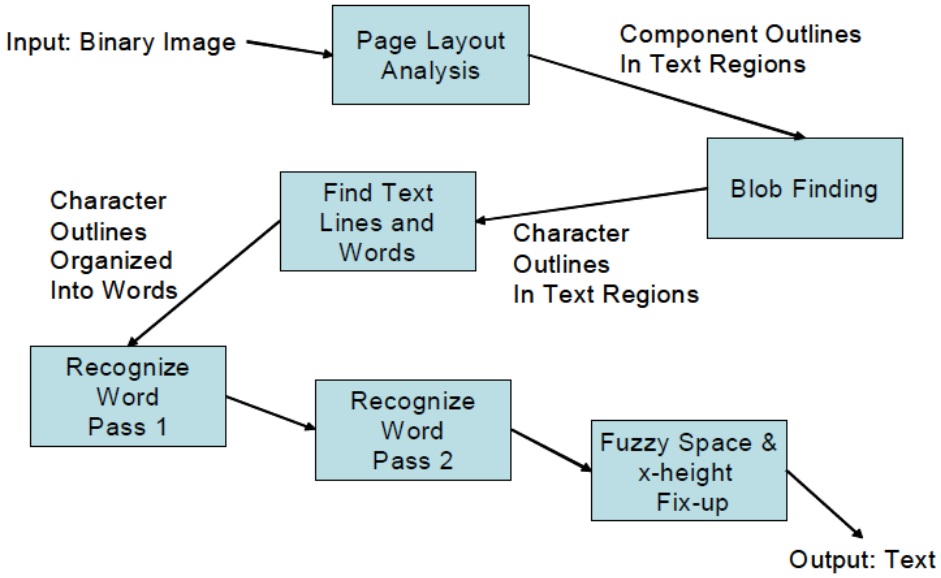

Abbildung 4: Ablaufdiagramm von Tesseract [Smi09]

so die, die im Wörterbuch zu finden oder nicht mehrdeutig sind, an den Klassifizierer zum Training weitergegeben. Wenn der Klassifizierer genügend Samples hat, kann er im zweiten Durchlauf schriftartbezogenes, antrainiertes Wissen einbringen. Am Ende durchlaufen die gefundenen Worte eine kontextbasierte und syntaktische Analyse. Diese Aneinanderreihung von Methoden ermöglicht es, auch nicht idealistische Schriftbilder mit einer hohen Präzision zu erkennen.

2.3 Android

2.3.1 Gründe für die Plattformentscheidung

Bei der Wahl des Betriebssystem kamen nur zwei Kandidaten, die aktuell und auch in Zukunft eine größere Rolle spielen werden, in Frage (vgl. Abb. 5) Android von Google und iOS von Apple. Zum einen waren für Android erforderliche Java-Kenntnisse vorhanden. Im Gegensatz dazu lagen keine Erfahrungen für Objectiv C, die Programmiersprache von iOS, vor. Zum anderen ist das Testen von eigenen Android-Anwendungen auf einem Endgerät vorteilhafter, da keine Gebühren für Entwicklerlizenzen zu entrichten sind. Faktoren wie die Verbreitung von Android-Geräten und die Sympathie für das OpenSource-Projekt haben die Entscheidung ebenfalls beeinflusst. Auch die Tatsache, dass kein Apple Rechner, der für die Entwicklung von iOS-Anwendungen nötig wäre, zur Verfügung stand sprach für Android.

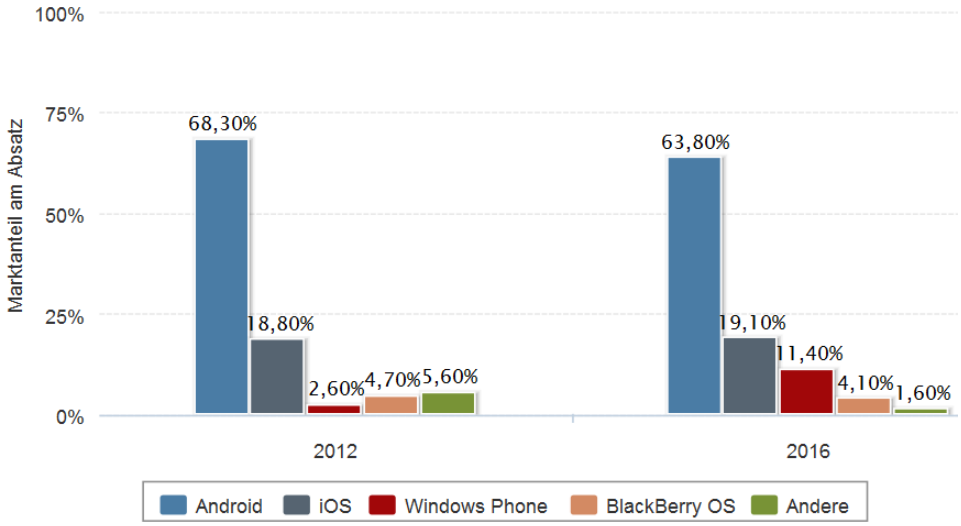

Abbildung 5: Prognose zu den Marktanteilen der Betriebssysteme am Absatz von Smartphones weltweit in den Jahren 2012 und 2016 [Sta12]

2.3.2 Allgemein

Android ist ein quelloffenes und freies Betriebsystem, das von Google und der Open Handset Alliance speziell für mobile Geräte entwickelt wurde und sich seit der Veröffentlichung im Oktober 2008 wie kein anderes verbreitet hat. Derzeit kommen täglich 1,3 Millionen neue Geräteaktivierungen zu den bereits 500 Millionen vorhandenen dazu [Wir12]. Das Ziel der Offenlegung der Plattform war eine gemeinsame Anpassung und Verbesserung durch verschiedene App-Entwickler und nicht durch eine zentrale Firma [goa12]. Der Linux Kernel verleiht dem Betriebssystem die notwendige Flexibilität zur Übertragung auf verschiedene Hardwareplattformen.

2.3.3 Architektur

Die Systemarchitektur [Bra10] von Android ist in vier Schichten unterteilt. Jede dieser Schichten abstrahiert die darunter liegende Ebene. Die Basis für Android ist ein angepasster Linux-Kernel der Version 2.6. Der Kernel bildet die Schicht zwischen der Hardware und der restlichen Architektur und enthält unter anderem Hardwaretreiber, Prozess- und Speicherverwaltung sowie Sicherheitsverwaltung. Android beinhaltet eine Reihe von Bibliotheken für einige Kernkomponenten, die aus Performancegründen in C und C++ programmiert sind. Sie stellen die meisten Funktionen zur Verfügung, wie z.B. Medienwiedergabe, den Surface Manager, 3D Bibliotheken und die Standard C Bibliotheken.

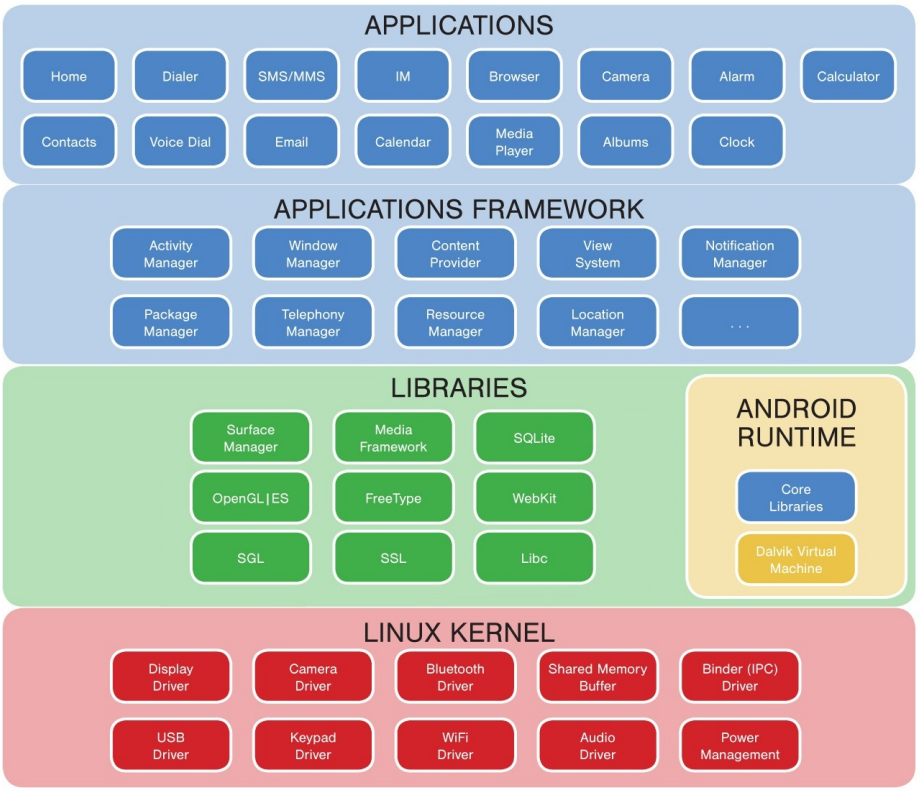

Abbildung 6: Android Architektur [Sig12]

Jede Android-Anwendung läuft als eigener Prozess mit ihrer eigenen Instanz der Dalvik Virtual Machine. Dieser Umstand ist vorallem für die Anwendungsentwicklung und die Sicherheit von Bedeutung ist. Dalvik wurde speziell dafür entwickelt mehrere virtuelle Maschinen effizient parallel ausführen zu können. Es führt Dateien im *dex*-Format aus, die für einen minimalen Speicherverbrauch optimiert sind. Diese Dateien werden, nachdem sie von Java kompiliert wurden, mit dem *dx*-Werkzeug in das nötige Format umgewandelt. Dadurch das Dalvik register- und nicht stapelbasiert ist, benötigt die VM weniger Zwischenschritte um den Bytecode auf dem Prozessor auszuführen. Dalvik nutzt die Low-Level-Speicherverwaltung und das Threading des optimierten Kernels aus.
Das Anwendungsframework ist die Schicht, die für die Entwicklung von Android Apps unmittelbar von Bedeutung ist und stellt den Rahmen für eine einheitliche Anwendungsarchitektur bereit. Ziel ist es, Anwendungen nach einheitlichen Richtlinien zu entwickeln und somit die Integration und Wiederverwendung von Anwendungen unter Android zu vereinfachen. Es erlaubt den Entwicklern die Benutzung von bereits vorhandenen Ele-

menten zum Erstellen von GUIs oder die Nutzung von Ressourcen. Das Laufenlassen von Hintergrunddiensten, der Nachrichtenaustausch mit anderen Applikationen und das Nutzen von Hardware, wie Sensoren oder Kamera, wird hier ebenfalls ermöglicht. Um unerlaubte Zugriffe zu vermeiden, werden dem Benutzer bei der Installation von Apps die Rechte die dem Programm erteilt werden, veröffentlicht. Diesen muss er zustimmen, damit das Programm genutzt werden kann.
Im Werkzustand besitzt Android grundlegende Apps wie E-Mail-, SMS-, Kalender-, Karten-, Kontakte- und Browseranwendungen. Alle Anwendungen wurden mit der Programmiersprache Java implementiert.

2.3.4 Aufbau einer Android-Anwendung

Eine Android-App besteht im Wesentlichen aus Activities [Pan10]. Eine Activity ist ein Fenster mit dem der Benutzer mittels Tasten- oder Touchscreenberührungen interagiert. Das Aussehen der Benutzerfenster wird in Layouts definiert. Es kann immer nur eine Activity gleichzeitig aktiv sein, bei einem Wechsel wird die zuvor laufende pausiert. Das Aufrufen erfolgt über Intents. Intents sind Objekte, die Informationen über eine Operation beinhalten und das Bindeglied zwischen mehreren Komponenten von Android bilden. Ein weiterer essentieller Baustein ist das Android Manifest. Hierbei handelt es sich um eine *xml*-Datei, die dem System alle nötigen Informationen zum Starten der App liefert. Unter anderem wird hier angegeben, welche Activities die App umfasst, mit welcher sie starten soll und welche Befugnisse ihr erteilt werden.

2.3.5 Lebenszyklus einer Android-Anwendung

Anders als Plattformen, bei denen man Anwendungen nur starten oder beenden kann, hat Android mehrere Status [Mei12], die eine Anwendung in ihrer Laufzeit durchläuft (vgl. Abbildung 7). Android nutzt eine Vielzahl von Callback-Methoden zur Verwaltung der Lebenszyklen von Activities. Das Aufrufen dieser Methoden wird bis auf die *onCreate*-Methode vom System übernommen. Beim Starten einer Anwendung und somit der ersten Activity, werden gleich drei Methoden ausgeführt: *onCreate*, *onStart* und *onResume*. Startet man eine weitere Activity, wird die vorherige mit *onPause* pausiert. Dadurch können Ressourcen freigegeben werden, wodurch das System weniger gebremst wird. Sobald die Activity wieder in den Vordergrund rückt, wird *onResume* ausgeführt. Wurde die Activity mit *onStop* beendet, kann sie mit *onRestart* neugestartet werden. Hierbei ist anzumerken, dass Activities auch automatisch beendet werden wenn sie lange nicht mehr sichtbar waren und Arbeitspeicher benötigt wird. Die Methode *onDestroy* beendet den Lebenszyklus einer Activity endgültig.

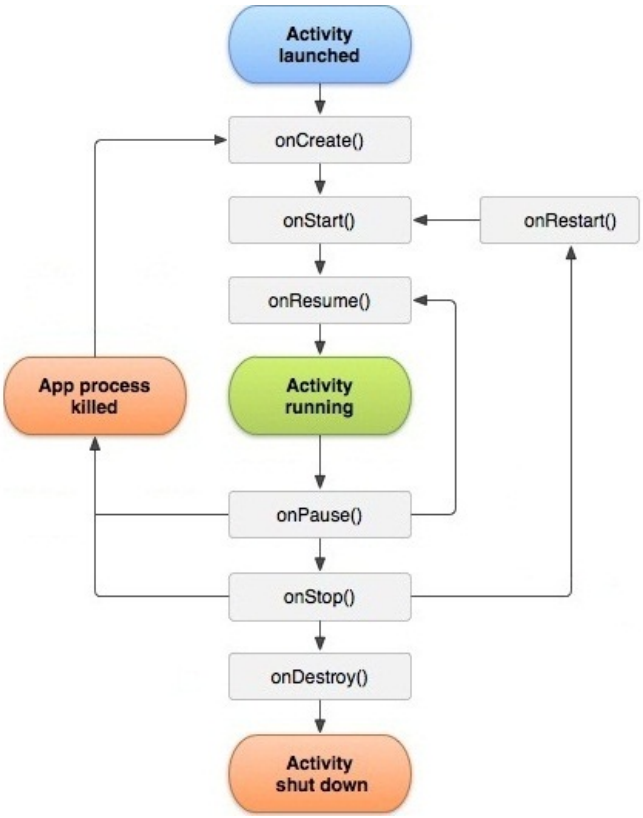

Abbildung 7: Zustandsdiagramm einer Android Activity [Goo12]

2.4 Verwandte Projekte und Apps

2.4.1 Voice Enabled Digital Camera and Language Translator

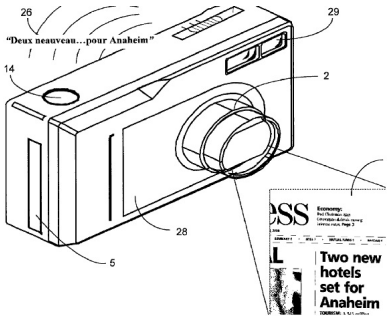

Abbildung 8: Zeichnung aus dem Patentdokument [Pie01]

Ein eingetragenes Patent von Thomas und Allison Piehn aus dem Jahre 2001 beschreibt ein Gerät, das optisch einer Digitalkamera ähnelt, aber zusätzliche Funktionen bietet (vgl. Abb. 8). Nach dem Fotografieren eines Textes soll dieser erkannt und in einer andere Sprache übersetzt werden. Die übersetzten Worte werden anschließend mit Hilfe von eingebauten Lautsprechern und einem Display akustisch und visuell wiedergegeben. Weitere Informationen zur Entwicklung eines Prototypen oder weiterführenden Forschungen der Erfinder sind nicht bekannt.

2.4.2 PDA Translator

2005 wurde an den NTT Cyper Space Laboratories in Yokosuka ein OCR Übersetzer für PDAs entwickelt [HN05]. Das System ist in der Lage japanische Schrift zu erkennen und sie ins Englische zu übersetzen. Das PDA ist über WLAN mit einem Server verbunden (vgl. Abb. 9), der die Bilder der eingebauten Kamera des mobilen Geräts empfängt und verarbeitet. Nachdem die Zeichen erkannt und zu möglichen Worten zusammengesetzt wurden , werden sie mit dem am selben Institut entwickelten ALT-J/E Translation System [SB96] übersetzt. Neben dem Eingangsbild bekommt der Server auch die Koordinaten des Anfangs- und Endbuchstaben übermittelt, die der Benutzer mit einem Stift auf dem Display festlegt. Um die Zeichenerkennung für natürliche Bilder zu optimieren, wurde ein robuster Zeichenklassifizierer verwendet. Dieser arbeitet mit einem unpräzisen Zeichenerkenner, der sich dadurch auszeichnet, dass er eine hohe Trefferquote bei der Bestimmung von Zeichenkandidaten erzielt. Wahrscheinliche Kandi-

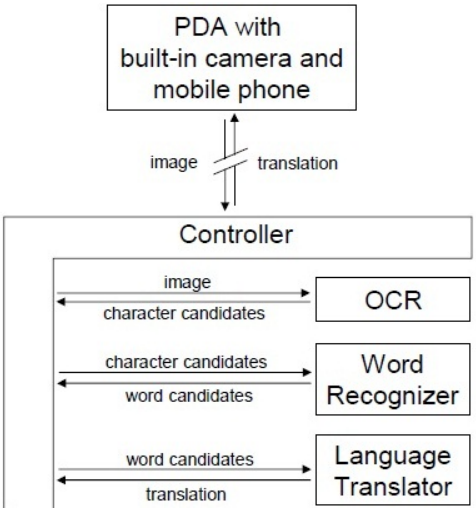

Abbildung 9: System Architektur des PDA-Übersetzers [HN05]

daten sind dann erst diejenigen, die sich an Orten mit hoher Trefferanzahl befinden.

2.4.3 TranslatAR

Abbildung 10: TranslatAR auf Nokia N900 [Tur11]

Das Programm TranslatAR enstand bei ein Projekt der University of California und wurde beim IEEE Workshop on Applications of Computer Vision im Jahre 2011 vorgestellt [Tur11]. Es handelt sich um einen Augmented Reality Wort-für-Wort Übersetzer, der für ein Nokia N900 konzipiert wurde. Augmented Reality im Sinne der Computervisualistik bedeutet die Erweiterung der Realität durch weitere Informationen. Nach dem Antippen eines Wortes wird seine Position und Orientierung ermittelt. Vereinfacht gesagt erfolgt die Lokalisierung des Wortes mittels Gradienten. Vertikal und horizontal werden die Grenzen des Wortes abgetastet und eine Bounding BoxI[1] herum gelegt. Danach wird der Winkel des Wortes mit Hilfe von Hough Transformation innerhalb der Box bestimmt. Um den Realitätsgrad der Überblendung zu erhöhen werden die Vorder- und Hintergrundfarben des Bildes geschätzt. Unter Annahme einer einfarbigen Schrift- und einer gleichmäßigen Hintergrundfarbe kann man zwei Bündelungen im Farbraum erwarten. Mittels Tesseract und der Google Translate API wird das Wort erkannt und übersetzt.

2.4.4 Word Lense

Word Lense [Vis12b] ist eine kostenpflichtige Augmented Reality Übersetzungs App von Quest Visual für Apple iOS, die seit Ende 2010 verfügbar ist. Im Spätsommer diesen Jahres erschien auch eine Android Version dieser App. Anders als im ersten Beispiel muss der Benutzer keine Angabe über den Ort des Wortes machen, den er übersetzt haben möchte, da der Text des gesamten Kamerabildes behandelt wird. Darüber hinaus benötigt die

Abbildung 11: Ausschnitt aus dem Word Lense Werbeclip [Vis12a]

[1] Eine Geometrie die einen Körper umschließt.

App keine Internetverbindung, weil die
Übersetzung über ein internes Wörterbuch abläuft. Anfangs nur für die Übersetzung zwischen der englischen und spanischen Sprache ausgelegt, sind mittlerweile auch Italienisch und Französisch integriert.

2.4.5 Google Translate for Android

Die App von Google [Inc12c], die ursprünglich keine integrierte Texterkennung besaß wurde Anfang 2010 veröffentlicht und verfügte über grundsätzliche Funktionen, wie das Übersetzen und Vorlesen nach Eintippen eines oder mehrerer Wörter in 65 Sprachen. Nach und nach wurden weitere Features wie Handschrifteingabe, die phonetische Umschrift nichtlateinischer Schrift oder der Konversationsmodus, bei der das Gesprochene für den Gesprächspartner direkt übersetzt wird, eingebaut. Anfang August diesen Jahres hat Google seine Übersetzungs-App um eine OCR-Funktion erweitert. Der zu übersetzende Text wird wie in einem üblichen Kameramodus abfotografiert und nachträglich, wie auf Abbildung 12 zu sehen, mit dem Finger eingegrenzt.

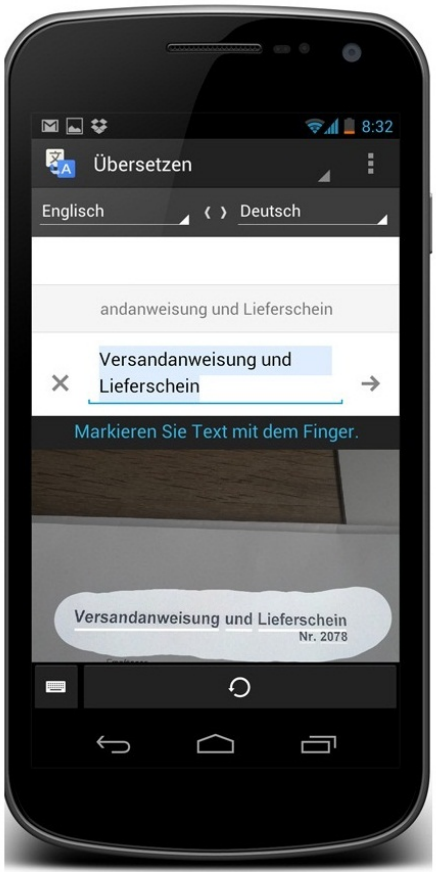

Abbildung 12: OCR Funktion von Google Translate [Inc12c]

3 Konzept

Dieses Kapitel umfasst die Konzeption des zu entwickelnden Programms. Neben der genaueren Ideenbeschreibung wird der Aufbau skizziert und mögliche Anwendungsbeispiele aufgezählt.

3.1 Idee

Die Idee des Programms, das im Rahmen dieser Bachelorarbeit geschrieben wird, war die eines effektiven Übersetzers auf einem mobilen Gerät. Mit Druckbuchstaben geschriebene Worte sollen auf dem schnellsten und effizientesten Wege erkannt und übersetzt werden. Dabei geht es in erster Linie um einzelne Worte oder kürzere Sätze, weniger um die Übersetzung von größeren Textpassagen in Büchern oder Zeitungsartikeln. In diesen Fällen hat ein Scanner deutliche Vorteile gegenüber einer eingebauten Kamera in einem Smartphone.
Der Grundgedanke ist eine Anwendung, die das mit der internen Kamera aufgenommene Bild auf dem Display wiedergibt, als würde man durch das Gerät hindurchschauen. Ist im Kamerabild nun ein Wort zu sehen, soll dieses erkannt und an den Übersetzer weitergegeben werden, der dann das Wort in der gewünschten Sprache zurückliefert. Eine weitere Idee, die optional behandelt wird, ist eine Augmented Reality Einblendung über dem tatsächlichen Wort. Im Falle der Implementierung einer solchen Erweiterung ist eine anschließende Benutzermeinung hinsichtlich des Mehrwerts interessant.

3.2 Aufbau

Das Programm sollte so konzipiert sein, dass sich nach dem Starten ein Hauptmenü öffnet, welches den Benutzer begrüßt, kurz in das Programm einweist und ihn dazu auffordert die Sprachen zu wählen. Die Sprachauswahl soll mit Hilfe einer Dropdownliste realisiert werden. Sind In- und Output gewählt, sollen diese durch einen Button bestätigt werden, woraufhin sich der Kamervorschau-Modus öffnen soll. Dieser stellt das aktuelle Kamerabild dar. Am Rand liegende Buttons sollen dann die Erkennung und Übersetzung steuern. Eine Bestimmung des zu erkennden Textbereichs soll per Fingerberührung gelöst werden. Weitere notwendige Werkzeuge sollen im Laufe der Implementierung erprobt und daraufhin entsprechend programmiert werden.

3.3 Anwendungsszenarien

Eine mögliche Anwendung wäre die Benutzung der App im Ausland. Im Urlaub findet man oft Schilder oder andere Hinweise, die nicht selbsterklä-

rend sind und den Urlauber zu einer selbstständigen Übersetzung zwingen. Meistens wird hierzu ein Wörterbuch oder ein digitaler Übersetzer eingesetzt, der voraussetzt, dass die zu übersetzenden Worte erst abgetippt werden müssen. Oftmals ist in diesen Momenten die Hürde zu groß, woraufhin die Übersetzung ausgelassen und auf das Verständis verzichtet wird. Ein selbstlesender Übersetzer dieser Art könnte den Menschen diese Situationen erleichtern.
Vor allen Dingen könnte diese Art der Übersetzung Eingangsprachen behandeln deren Schriftzeichen nicht bekannt sind. Die Übersetzung kyrillischer, chinesischer oder arabischer Zeichen sind mit konventionellen Methoden nur sehr schwer durchzuführen, wenn das Alphabet nicht bekannt ist und die Tastatur nur lateinische Tasten hat.
Ein weiteres Szenario könnte in der Kompetenz- und Wissensvermittlung stattfinden. Man könnte diese Applikation beim Vokabelnlernen in der Schule unterstützend einsetzen. Beispielsweise wären auf diese App angepasste Vokabelkärtchen im Fremdsprachenunterricht denkbar. Der spielerische Aspekt dieses Lernens könnte den Schüler motivieren und hätte somit einen positiven Nebeneffekt.

4 Implementierung

In diesem Kapitel werden zum einen wichtige Bestandteile der Programmierumgebung beschrieben und zum anderen die einzelnen Komponenten der Anwendung anhand von Codebeispielen erklärt.

4.1 Plattform

Dadurch, dass die Wahl der Plattform auf Android gefallen ist, werden der Umsetzung des Programms viele Möglichkeiten und eine gut dokumentierte API geboten. Die Wahl der API[1] Version richtete sich nach der Verbreitung aller Android Versionen[2]. In Abbildung 13 ist leicht zu erken-

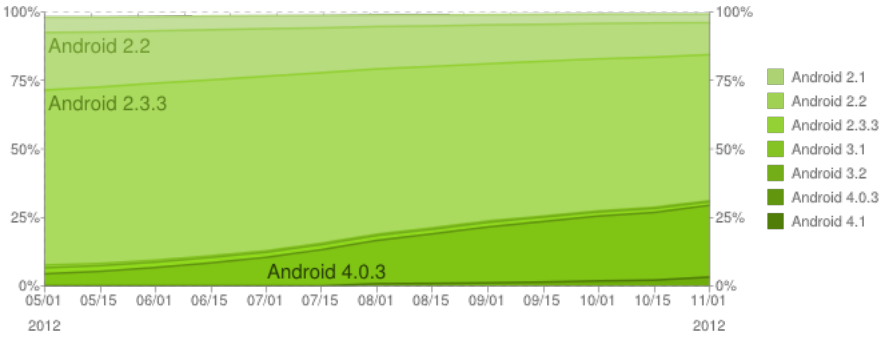

Abbildung 13: Darstellung der Android-Versionen die in den letzten 6 Monaten auf Google Play zugegriffen haben

nen, dass Version 2.3.3, verfügbar seit dem 23. Februar 2011, immernoch die weitverbeiteste Androidversion ist. Aus diesem Grund wurde die Anwendung auf dem API Level 10 programmiert, was versichtert, dass sie auf allen Geräten funktioniert, die Version 2.3.3 oder höher installiert haben. Ein niedrigeres Level zu wählen würde zwar zusätzlich versichern, dass die restlichen knapp 20 Prozent die App auch nutzen könnten, es würde allerdings auch die Möglichkeiten bei der Programmierung einschränken.

4.2 Entwicklungsumgebung

4.2.1 Eclipse

Android-Apps werden hauptsächlich in Java geschrieben. Eclipse bietet einige Erweiterungen an, die es möglich machen für Android zu entwickeln weshalb die Entscheidung bei der Wahl der Entwicklungsumgebung auf Eclipse 4.2 in Verbindung mit dem Android Development Tool (ADT) als

[1] Schnittstelle die dem Entwickler erlaubt Anwendungen zu schreiben und dabei bereits vorhandene, standardisierte Bibliotheksroutinen zu nutzen.

[2] http://developer.android.com/resources/dashboard/platform-versions.html

Plugin fiel. Diese Kombination ermöglicht es, Apps auf eine komfortable Art und Weise für Android auf dem PC zu erstellen und auf einem (virtuellen) Android-Gerät zu testen.

4.2.2 Software Development Kit

Zusätzlich zu dem ADT benötigt man auch ein Android Software Development Kit (SDK) [Inc12b], das über eine umfangreiche Auswahl an Entwicklungswerkzeugen verfügt. Der Lieferumfang umfasst ein Fehleranalyseprogramm, etliche Bibliotheken, ein Telefonemulator, Dokumentationen, Beispielcodes und Übungsbeispiele.

4.2.3 Native Development Kit

Das Native Development Kit [Inc12a] beinhaltet eine komplette Toolchain[1] , die dem Entwickler die Möglichkeit bietet in C++ geschriebene Programmteile für ein Android Projekt zu builden[2] und zu kompilieren[3]. Der Hauptgrund für die Verwendung von C++ Code liegt in der Beschleunigung. Man kann nicht sagen, dass ein nativer Code immer eine Beschleunigung des Programms bezweckt, da ungünstig geschriebene C++ Teile im Vergleich zu einer gut geschriebenen Java Anwendung sogar verlangsamend wirken können. Spielt Performance und Schnelligkeit eine Rolle ist C++ allerdings immer dann überlegen, wenn prozessor- und speicherlastige Berechnungen in den Vordergrund rücken. Besonders bei bildverarbeitenden Prozessen, wie die der Texterkennung, macht sich das bemerkbar [Lee10]. Üblicherweise befinden sich die nativen Bibliotheken im Library-Pfad, wo das System nach ihnen sucht. Der auf einem Android-Gerät bereitgestellte Pfad hierfür ist nur lesen und somit unbrauchbar für unsere Bibliothek. Das NDK hilft hier, indem es die Bibliothek in die Application Package (APK) Datei packt. Wird die App nun auf dem Gerät installiert, erstellt das System einen *Lib*-Ordner, der dieser Anwendung vollen Zugriff bietet, für andere aber blockt [Gag11].

4.2.4 Java Native Interface

Das Java Native Interface (JNI) [Oft12] ist eine standardisierte Schnittstelle, die dem Entwickler die bidirektionale Möglichkeit bietet auf native Funktionen und Bibliotheken, die in C++ geschrieben sind, von Java aus zu zugreifen. Für die Implementierung muss im Java-Programm in einem statischen Initialisierungsblock die gewünschte native Bibliothek geladen werden.

[1] Eine systematische Sammlung von Werkzeug-Programmen
[2] Der Prozess in dem Quellcode zu Objektcode konvertiert wird
[3] Das Übersetzen von Code einer höheren Sprache in Maschinensprache

4.3 Komponenten

4.3.1 Startmenü

Nach dem Starten des Programms erscheint die Startmenü-Activity. Diese besteht aus einer Begrüßung und dem entworfenen App-Logo. Der untere Teil der Startseite besteht aus zwei Dropdown Listen, bei Android "Spinner" genannt, und einem Start-Button. Die beiden Spinner bieten dem Benutzer die Möglichkeit die Eingangs- und Ausgangssprache festzulegen. Einerseits sind diese Angaben für die Übersetzung nötig, andererseits spielt die Angabe der Eingangssprache bei der Texterkennung eine wichtige Rolle. Je nachdem welche Sprache gewählt ist sucht sich die Tesseract Bibliothek das passende Sprachmodul auf dem internen Speicher des Smartphones heraus, um die Texterkennung dementsprechend anzupassen. Beim Betätigen des Buttons wird ein Intent verschickt, das die aktuelle Activity pausiert und die OCR-Activity startet.

Abbildung 14: Startmenü Screenshoot

4.3.2 Kameravorschau

Die Grundlage der OCR-Activity ist eine Kameravorschau, die durch die *Preview*-Klasse repräsentiert wird. Diese wird von der *View* abgeleitet und zeichnet auf dem sichtbaren Bereich der Activity das Kamerabild. Ein Überblick der wichtigsten Methoden zeigt Abbildung 15. Die Basisklasse einer Kamervorschau ist die *SurfaceView*, eine Ansicht, über die der Programmierer volle Kontrolle hat. Zusätzlich benötigt man ein *SurfaceHolder*-Objekt das mit der *View* über eine Schnittstelle kommuniziert. In der *surfaceCreated*-Methode wird auf die Kamera zugegriffen und festgelegt wo das Kamerabild gezeichnet werden soll. Analog dazu beendet *surfaceDestroyed* den Kamerazugriff und beendet die Vorschau. Die letzte notwendige Methode ist *surfaceChanged*, die immer dann aufgerufen wird, wenn die Größe des sichtbaren Bereichs geändert wird. In diesem Fall wird sie allerdings nur einmal ausgeführt, da die Größe des Vorschaubildes konstant bleiben soll. Damit die Kameravorschau an die Bildschirmauflösung angepasst ist und auf allen gängigen mobilen Geräten die bestmögliche Darstellung geboten wird, wird in der *surfacechanged*-Methode die höchstmögliche Auflösung

```
class Preview extends SurfaceView implements SurfaceHolder.Callback {
    public Preview(Context context) {
            super(context);
            init();
    }
    public void init() {
            mHolder = getHolder();
            mHolder.addCallback(this);
            mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }
    public void surfaceCreated(SurfaceHolder holder) {
            mCamera = Camera.open();
            mCamera.setPreviewDisplay(holder);
            mCamera.addCallbackBuffer(mBuffer);
            mCamera.setPreviewCallbackWithBuffer(new PreviewCallback() {
        }
    }
    public void surfaceDestroyed(SurfaceHolder holder) {
            mCamera.stopPreview();
            mCamera.release();
            mCamera = null;
    }
    public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
            myParameters = myCamera.getParameters();
            List<Size> sizes = myParameters.getSupportedPreviewSizes();
            myParameters.setPreviewSize(sizes.get(3).width, sizes.get(3).height);
            myCamera.setParameters(myParameters);
            mCamera.startPreview();
    }
```

Abbildung 15: Vereinfachter Ausschnitt der Kameravorschau-Klasse

mit dem Befehl *getSupportedPreviewSizes* abgefragt und in den Kameraparametern gesetzt.

4.3.3 Fokussierung

Besonders wichtig für die Texterkennung ist neben der Beleuchtung und der Vermeidung von starken Verzerrungen auch die Bildschärfe. Verschwommene Buchstabenkanten führen zu schlechteren Ergebnissen und können durch Setzen der richtigen Brennweite vermieden werden. Hierfür verfügen die meisten Kameras über eine Autofokusfunktion, die gezielt eingesetzt werden muss. Ein permanentes Autofokussieren führt dazu, dass die Brennweite ständig springt. Ein zusätzlicher Autofokus-Button wäre zwar eine Lösung, würde aber die Kette der Button Befehle unnötig erweitern. Die Autofokussierung unmittelbar vor der Texterkennung zu programmieren würde bedeuten, dass die Texterkennung auch dann gestartet werden würde, wenn der Fokus falsch gesetzt ist. Deswegen wurde eine Lösung entwickelt, um die Autofokussierung ohne Tastendruck zu aktivieren. Hierfür wurde die SensorEventListener-Schnittstelle implementiert

und eine Methode geschrieben (vgl. Abb. 16), die genau dann den Autofokus auslöst, wenn das Gerät in der Z-Achse um einen bestimmten Wert beschleunigt wird. Die Z-Achse ist diejenige Achse, die orthogonal zum Bildschirm verläuft.

```
public void onSensorChanged(SensorEvent event) {
    float z = event.values[2];
    if (!myInitialized) {
        LastZ = z;
        myInitialized = true;
    }
    float deltaZ = Math.abs(LastZ - z);
    if (deltaZ > 0.5 && myAutoFocus) {
        myAutoFocus = false;
        myPreview.setCameraFocus(myAutoFocusCallback);
    }
    LastZ = z;
}
```

Abbildung 16: *SensorListener* für Autofokus

4.3.4 Tesseract für Android

Der nächste wichtige Schritt zur Erreichung des Ziels war die Einbindung von Tesseract in das Android Projekt. Auf Grund dessen, dass Tesseract in C++ geschrieben ist, wäre es zunächst notwendig diesen Code mittels JNI und dem NDK an Java anzupassen. Da diese Umschreibung ein großer Aufwand wäre und sie von einem Google Mitarbeiter schon einmal durchgeführt wurde[1], konnte sie übernommen werden. Nachdem die Programmierumgebung mit dem nötigen NDK aufgesetzt und einige wichtige Pfade und Variablen richtig gesetzt wurden [Rat12] , lag die Aufgabe nun darin, das Tesseract Projekt zu builden, es in Eclipse zu importieren und als Library zu deklarieren, damit es im zukünftigen App-Projekt als solche benutzt werden kann. Nachdem das TessBaseAPI-Objekt initialisiert ist, wird ein Pfad angegeben, der zu einer Sprachdatei führt, die den Zweck hat, die Texterkennung auf die Schriftzeichen und das Vokabular derjenigen Sprache anzupassen, die als Eingangssprache eingestellt wurde. Der erkannte Text wird in UTF8 codiert und anschließend von unbrauchbaren Sonderzeichen befreit. Zum Schluss werden noch Leerzeichen am Anfang und am Ende des erkannten Textes abgeschnitten.

[1] http://code.google.com/p/tesseract-android-tools/source/list

```
// Tesseract Methode
public void tess(Bitmap bm) {
    TessBaseAPI baseApi = new TessBaseAPI();
    baseApi.setDebug(true);
    // Sprachdateipfad und Sprache werden initialisiert
    baseApi.init(DATA_PATH, lang);
    // Bild wird gesetzt
    baseApi.setImage(bm);
    // Text wird erkannt und in UTF8 codiert
    recognizedText = baseApi.getUTF8Text();
    baseApi.end();
    // Textoptimierung
    recognizedText = recognizedText.replaceAll("[^a-zA-Z0-9]+", " ");
    recognizedText = recognizedText.trim();
```

Abbildung 17: Texterkennungs-Methode

4.3.5 Selektierwerkzeug

Bedingt dadurch, dass das menschliche Auge und das Gehirn anders lesen als eine Texterkennungssoftware ergeben sich Störfaktoren, die für den Menschen zwar unbedeutend sind, der Software aber große Probleme bereiten. Bei den ersten Tests wurde deutlich, dass zuviele Bildinformationen um den Textbereich herum das Ergebnis verfälschen und den Erkennungsalgorithmus teilweise überlasteten. Das Foto eines Buchdeckels mit einer klar geschriebenen und leicht zu erkennenden Schrift ist zwar kein Problem für Tesseract, hält man aber die Kamera so, dass man noch das Bücherregal und den gemusterten Teppichboden mit Muster im Hintergrund sieht, versucht die Software aus detektierten Merkmalen Zeichen zu matchen, die dort gar nicht vorhanden sind. Somit galt es einen Weg zu finden, den zu untersuchenden Bereich einzugrenzen.
Im Kapitel 2.4 wurden verwandte Projekte gezeigt, in denen ein Antippen des Wortes benutzt wurde. Da bei dieser App eine Übersetzung von mehreren Worten gleichzeitig möglich sein sollte, wäre diese Variante unbrauchbar. Eine Markierung der Umrandung des Textes durch Entlangfahren des Fingers auf dem Touchscreen wäre ebenso denkbar wie das Platzieren eines Rechtecks über der Textfläche. An dieser Stelle wurde sich für die Eingrenzung des Bildausschnitts durch Aufspannen eines Rechtecks entschieden. Dieses soll mit Hilfe von vier beweglichen Punkten realisiert werden. Mittels Touchpad-Berührung und Fingerbewegung sollen die Koordinaten der Ecken verändert und das Auswahlfenster skaliert werden.
Hierfür wurde die TouchRect-Klasse geschrieben. Diese wird von der View-Klasse abgeleitet, was die Möglichkeit bietet auf dem Bildschirm zu zeichnen. Die vier beweglichen Ecken sind schwarze Rechtecke, die am Anfang an festen Orten gezeichnet werden und miteinander verbunden sind. Be-

rührt der Nutzer nun den Bildschirm, wird die Touchposition gespeichert und mit bestimmten Werten verglichen. Zunächst werden alle Berührungen innerhalb des Rechtecks ignoriert. Erfolgt eine Berührung ausserhalb des Rechtecks werden die kürzestesten Abstände des Berührungspunktes zu den Koordinaten der vier Ecken berechnet. Die Berechnung erfolgt über den euklidischen Abstand. Dieser ergibt sich aus der Summe der absoluten Differenzen der Einzelkoordinaten.

$$\sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{1}$$

Liegt dieser unter einem Schwellwert wurde eine Ecke berührt. Bewegt sich nun der Finger anschließend vom Punkt weg während er noch das Display berührt, wird die Fingerposition verfolgt und die Eckpunktkoordinaten aktualisiert. Dabei wird darauf geachtet, dass sie nicht weiter bewegt werden dürfen als die beiden anliegenden Ecken. Beim Loslassen des Bildschirms werden die boolschen Variablen, die Auskunft darüber geben, welche Ecke bewegt wird, wieder zurückgesetzt.

4.3.6 Übersetzung

Nachdem der Antrag zur Teilnahme am University Research Program for Google Translate[1] auf Grund von zu hoher Nachfrage nicht bearbeitet werden konnte, fiel die Option die Google Translate API zu Forschungszwecken kostenlos nutzen zu können weg. Die Einbindung einer vollständigen Wörterbuchdatenbank war aufgrund des Aufwands nicht geplant. Deswegen wurde auf die Alternative Microsoft Bing[1] zurückgegriffen. Es handelt sich um eine kostenlose API für angemeldete Benutzer, mit der man die Möglichkeit hat bis zu 1.000.000 Zeichen pro Monat übersetzen zu lassen. Die Implementierung der Übersetzungsmethode ist in Abbildung 18 zu sehen. Für die Identifizierung ist eine Kunden ID und ein Schlüssel nötig, die bei der Initialisierung des Übersetzungsobjektes gesetzt werden. Das Angebot umfasst eine bidirektionale Übersetzung in 37 Sprachen, sowie eine automatische Sprachenerkennung für den Fall, dass die Eingangssprache unbekannt ist. Eine Veröffentlichung dieser App würde einen Windows Azure Account[1] und die zusätzliche Einbindung eines Eingabefeldes für die Zugangsdaten im Startmenü dieser App voraussetzen.

4.3.7 Buttons

Um mit der Anwendung zu interagieren wurden einige Buttons eingebaut, die wichtige Funktionen besitzen. In der *onCreate*-Methode werden

[1] http://research.google.com/university/translate/proposal.html
[1] http://www.microsofttranslator.com/dev
[1] https://www.datamarket.azure.com

```
public class Translation {

    public static String translateWord() {
        Translate.setClientId("b2c28974-94a1-40c1-a28c-911cbc5c319b");
        Translate.setClientSecret("KdgxbqXwF+XGY6a19qwTyB2ANkesEGuw+TvYMxd7kX8=");
        String word = (String) Ocr.getWord();
        String translatedText = Translate.execute(word, Language.in, Language.out);
        return translatedText;
    }

}
```

Abbildung 18: Übersetzungs-Methode

die *ImageViews*[2] der Buttons geladen und positioniert, in der *onIntercept*-TouchEvent-Methode die Bildschirmberührungen an deren Position abgefangen und in den *onClick*-Methoden die Funktionalität zugewiesen. Als Erstes wurde der OCR-Button integriert. Wird dieser ausgelöst, speichert das Gerät das aufgespannte Rechteck, das in der Vorschau zu sehen ist, als Bild ab. Dieser Speichervorgang wird in einem seperaten Thread durchgeführt, damit die Zeichenmethoden der Oberflächen keine Latenzen erfahren. Das in einem Puffer fester Größe abgelegte Vorschaubild wird zu-

Abbildung 19: App im Betrieb

[2]Bezeichnung für die Instanz eines Bildobjekts innerhalb eines Android-Layouts.

nächst aus den *NV21*[1]-byte-arrays in ein *YUV*[2]-Format abgespeichert. Anschließend wird dieses in das *JPG*-Format komprimiert, um es danach in das für Tesseract passende *ARGB8888*-Bitmap-Format[3] zu konvertieren (vgl. Abb. 20). Eine vorimplementierte Möglichkeit für die direkte Konvertierung des Android Kamera-Vorschau-Formats in das Bitmap-Format gibt es nicht.

```
public Bitmap getPicturefromRect(int x, int y, int width, int height) {
    Bitmap b = null;
    Size s = myParameters.getPreviewSize();
    YuvImage yuvi = new YuvImage(myBuffer, ImageFormat.NV21, s.width, s.height, null);
    ByteArrayOutputStream oStream = new ByteArrayOutputStream();
    yuvi.compressToJpeg(new Rect(x, y, width, height), 100, oStream);
    b = BitmapFactory.decodeByteArray(otStream.toByteArray(), 0, oStream.size());
    b = b.copy(Bitmap.Config.ARGB_8888, true);
    yuvi = null;
    oStream = null;
    return b;
}
```

Abbildung 20: Framespeicherung und Konvertierung

Desweiteren wurde ein Button zur Aktivierung des Licht implementiert, der bei Bedarf und bei Existenz einer LED-Lampe an der Kamera betätigt werden kann. Zusätzliche Beleuchtung ist vorallem bei schlechten Lichtverhältnissen notwendig. Selbst bei relativ guten und für das menschliche Auge ausreichenden Bedingungen kann die Lampe das Ergebnis verbessern. Infolge dessen, dass mobile Geräte wie Smartphones Kameras mit sehr kleinen Objektiven haben und dadurch relativ lichtschwach sind, kommt es oft dazu, dass die Lichtempfindlichkeit des Kamerasensors verstärkt werden muss. Eine höhere ISO-Zahl zieht Bildrauschen mit sich, wodurch die Texterkennung negativ beeinflusst wird.
Ein dritter Button übergibt den erkannten Text an die Übersetzungsmethode, die schließlich die Worte in der gewünschten Sprache zurückgibt und auf dem Bildschirm darstellen lässt.
Der letzte Button wechselt zwischen dem Standard- und dem AR-Modus.
Abbildung 21 zeigt die App im Betrieb.

[1]Eine abgewandelte Version des YUV-Formats. Es ist das Standardformat der Kamervorschaubilder bei Android.

[2]Ein Bildformat das zur Darstellung der Farbinformation Luminanz und Chrominanz verwendet.

[3]Ein Bildformat mit RGB und Alpha-Kanal für Transparenz. Jedes Pixel wird in 4 bytes gespeichert, jeder Kanal in 8 bit.

4.3.8 AR Modus

Zusätzlich zu der normalen Vorgehensweise, bei der ein Bildbereich abgespeichert, an die Texterkennung weitergegeben und nach der Übersetzung am oberen Displayrand dargestellt wird, gibt es einen Augmented Reality (AR) Modus. Während diesem Modus erfolgt eine permanente Texterkennung und Übersetzung der gefundenen Worte des ausgewählten Bereichs. Außerdem werden die übersetzten Worte nicht in der oberen Leiste platziert, sondern unmittelbar über das Wort gelegt. In einer seperaten Methode werden die Koordinaten der Wörter abgefragt damit ein Textfeld aufgespannt wird, das die selben Abmessungen hat wie der reale Text. Entsprechend wird auch die Schriftgröße durch die Ermittlung der Buchstabenhöhe angepasst.
Dieser Modus kann von Vorteil sein wenn mehrere leicht erkennbare Worte nacheinader übersetzt werden sollen, da weniger Buttons gedrückt werden müssen. Handelt es sich um mittelschwer bis schwer erkennbare Worte empfiehlt sich der Standard Modus, weil falsch erkannte Zeichen dann nicht so irritierend wirken.

Abbildung 21: Verfolgung über Wortkoordinaten

4.3.9 Manifest

Das Android Manifest, ohne das die App nicht lauffähig ist, beinhaltet einige grundlegende Informationen, die zum Abschluss erläutert werden sollten. Unter anderem wird der App die Erlaubnis erteilt auf das Internet und auf die Kamera zuzugreifen, bei der zusätzlich noch vom Autofokus-Mechanismus Gebrauch gemacht wird. Damit ausgeschlossen wird, dass

die App nicht auf älteren Versionen als 2.3.3 benutzt werden kann, wird der minimale SDK Level, der hier 10 ist, angegeben. Da Text meist eine größere Ausdehnung in der Breite hat und die Buttons seitlich liegen ist das Handling im Querformat angenehmer. Deswegen wurde im Manifest festegelegt, dass die OCR-Activity in der Bildschirmorientierung den Wert *Landscape* hat und eine Portraitansicht dadurch ausgeschlossen ist. Ausserdem wurde die Startmenü-Activity als Ausgangspunkt vermerkt und ein Verweis auf das App-Logo gesetzt, das im Android Hauptmenü erscheinen soll und die App im Gesamtbild abrundet.

5 Evaluation

Um eine nachvollziehbare Bewertung der App abzugeben, wurde eine Evaluation durchgeführt, deren Ablauf und Ergebnisse in diesem Kapitel wiedergegeben werden, um anschließend ein Fazit ziehen zu können.

5.1 Testgerät

Das Testen der App über den Eclipse-Emulator lief sehr langsam ab. Der Grund dafür liegt darin, dass die CPU Instruktionen zwischen den x86 Prozessoren und den in Android Smartphones verbauten ARM-Prozessoren übersetzt werden müssen. Die Verwendung einer USB-Kamera in Verbindung mit dem Emulator ist nach einer Vielzahl von Versuchen ebenfalls gescheitert, weswegen für die Phase der Entwicklung und der Evaluation das Smartphone Galaxy S2 von Samsung zur Verfügung gestellt wurde. Dieses bietet einen 1,2GHz Dual-Core Prozessorer, 837 MB RAM und eine 8 Megapixel Kamera. Darüberhinaus hat es eine Bildschirmauflösung von 800*480 Pixel und verfügt über WLAN. Somit hat es optimale Vorraussetzungen für die gegebene App.

5.2 Vorgehen

Der Ablauf des Benutzertests bestand aus 3 Abschnitten. Zuerst wurden der Proband, seine aktuelle Motivation und seine Erfahrungen mit Smartphones und Apps mit Hilfe eines Fragebogens eingeschätzt. Im zweiten Teil sollte sich der Proband mit der App vertraut machen und verschiedene Übersetzungsszenarien durchspielen. Zum Schluss sollte der Proband die zweite Hälfte des Fragebogens ausfüllen und Beurteilungen und Anmerkungen in Bezug auf Intuitivität, Nützlichkeit und Mehrwert der App abgeben.

5.3 Szenarien

Um zu prüfen, mit welchen Gegebenheiten die App und vorallem der Benutzer gut auskommt und mit welchen nicht, wurden selbstgedruckte Schilder in verschiedenen Sprachen erstellt und dem Probanden vorgehalten.Um reale Bedingungen zu schaffen und nicht nur idealistisch, einfache Szenarien zu bieten wurden einige Herausforderungen im Test eingebaut. Aus diesem Grund haben sich die Schilder im Schwierigkeitsgrad hinsichtlich der Texterkennung und dem Handling mit der App in folgenden Punkten unterschieden:

- unterschiedlicher Text-Hintergrund-Kontrast
- verschiedene Schriftart
- Worte deren Sprache der Benutzer nicht zuordnen kann
- störende Bildelemente zwischen den Worten
- leichte Verzerrungen
- Reflexionen
- am Text nah angrenzende Objekte

Anschließend folgte die Umschaltung in den AR-Modus. Dem Benutzer sollte das Prinzip dieses Modus verdeutlicht werden, damit er ein Gefühl dafür bekommt und in der Lage ist die letzte Aufgabe zu lösen. In der letzten Aufgabe ging es darum, dass das im Kapitel 3.3 erwähnte Anwendungsszenario, bei dem die App als Vokabellernuntersützung dienen soll, durchzuspielen. Dabei wurden im laufenden AR-Modus, bei dem Texterkennung und Übersetzung permanent durchlaufen, Vokabelkärtchen gezeigt und dem Benutzer durchgehend übersetzt.

Abbildung 22: Testaufbau

5.4 Ergebnisse

Die Evaluation wurde mit einer Anzahl von acht Teilnehmern durchgeführt, bei denen die Altersspanne zwischen 19 und 25 Jahren lag. Unter ihnen waren drei weibliche und fünf männliche Probanden. Eine größere Anzahl von Testpersonen wurde nicht in Betracht gezogen, da laut J. Nielsen [Nie94] die Anzahl der gefundenen Probleme nur unwesentlich steigt, wenn mehr als 8 Personen teilnehmen. Sie zeigten alle Interesse an der Thematik und führten die Tests sorgfältig durch. Schon nach wenigen Tests wurden einige Schwachstellen der Anwendung deutlich, die bis zu diesem Zeitpunkt nur erahnt werden konnten. Das Hintergrundwissen und die im Laufe der Entwicklung angeeignete Routine für den optimalen Umgang mit der App führten bei Experten-Tests zwar zu positiven Ergebnissen, dürfen bei Neubenutzern aber nicht vorrausgesetzt werden. Das Gefühl für die optimale Ausleuchtung und den Blick für die geeignete Schärfe sind wichtige Faktoren im Zusammenhang mit der sensiblen Texterkennungs-Engine. Ein Großteil der Probanden hatte verständlicherweise in dieser Hinsicht Probleme. Die Tipps, die in der Einführung des Evaluationsbogens gegeben wurden, wurden zwar gelesen, doch oftmals erst nach erneutem Hinweisen befolgt. So wurde die Lichtfunktion in Situationen, in denen sie nützlich gewesen wäre, von manchen auch nach mehreren fehlgeschlagenen Versuchen nicht genutzt. Das Scharfstellen machte manchmal Probleme, wenn die Probanden versuchten zu nah an den Text zu gehen, ohne zu merken, dass das Bild nicht mehr 100 prozentig scharf ist. Auch das Einrahmen eines Wortes bei nah angrenzenden Bildmerkmalen sorgte stellenweise für Schwierigkeiten. Den Testkandidaten war zwar bewusst, dass das Verschieben der Vier Eckpunkte um das Wort herum nötig ist, doch haben viele nicht das Maß an Bedeutung in diesem Punkt erkannt und ungenau gearbeitet. Diese Tatsache ist nachvollziehbar, wenn die Auseinandersetzung mit dem Texterkennungsalgorithmus nicht vorhanden ist.
Das Experimentieren mit der AR-Funktion empfanden viele als aufregend. Besonders positiv ist die letzte Aufgabe ausgefallen. Dadurch, dass die Texterkennung auf den Vokabelkarten reibungslos ablief und die sie im aktivierten AR-Modus nur durch das Halten der Kamera auf das Wort übersetzt wurden hatten die Probanden hierbei am meisten Spaß. Obwohl die Erkennung der chinesischen Zeichen nicht bei jedem Wort funktionierte, war auch hier die Resonanz durchweg positiv.
Zusammenhänge im Umgang mit der App und den im Fragebogen angegegeben Daten vor dem Test konnten nur bedingt hergestellt werden. Lediglich Testpersonen die kein Smartphone besaßen wiesen weitere Verhaltensauffälligkeiten auf, indem sie beispielsweise die “Zurück”-Taste, die betätigt werden sollte, um ins Startmenü zurückzukehren, nicht kannten. Das Alter, das Geschlecht oder die Einstellung zu neuen Apps hatte keinen

signifikanten Einfluss.
Die abschließende Umfrage ergab ein durchwachsenes Endergebnis. Die Bedienung wurde von fast allen als intuitiv eingeschätzt. Auch die Übersetzungen konnten, bis auf einige Ausreißer, überzeugen. Die Formulierung der Übersetzungen von längeren Sätzen wurde hin und wieder als eigenartig angesehen, jedoch konnte die Botschaft des Satzes immer übermittelt werden. Die Texterkennung wurde, wie die Beobachtungen deutlich gemacht haben, von allen als nur teilweise zufriedenstellend oder schlechter eingeordnet. Dabei schnitt die Erkennung englischer Texte am besten ab, was mit der Qualität der Sprachpakete zusammenhängt. Die Mehrheit der Probanden fanden eine solche App sinnvoll, würden sie allerdings erst nach einer Optimierung der Texterkennung auf ihrem eigenen Smartphone installieren.

6 Fazit und Ausblick

Das Ergebnis der Evaluation hat eindeutig gezeigt, dass weitere Überlegungen und Verbesserungen nötig sind, damit diese Anwendung effizient eingesetzt werden kann. Die wesentliche Erkenntnis dieser Bachelorarbeit ist die Tatsache, dass eine gewöhnliche Texterkennungsbibliothek für diese Art der Bilder nicht ausreicht und Tesseract in einer App dieser Art einen entscheidenden Schwachpunkt darstellt. Die gängigen OCR-Programme sind darauf ausgelegt Texte aus eingescannten Dokumenten zu extrahieren. Dokumente bestehen meist aus schwarzem Text mit einheitlicher deutlicher Schrift auf weißem Hintergrund ohne störende Artefakte, ohne signifikanten Helligkeitsunterschieden und ohne Rauschen. Bekommt die Anwendung Bilder mit diesen idealen Bedingungen, funktioniert sie tadellos. Texte in natürlichen Bilder weisen andere Eigenschaften auf, deswegen muss eine Lösung gefunden werden, die mit diesen erschwerten Bedingungen umgehen kann. Das Selektierwerkzeug erfüllt zwar seinen Zweck, indem es in den meisten Fällen die irrelevanten Teile des Bildes wegschneidet, jedoch liegen störende Elemente meistens zu nah am Text oder gar zwischen den Worten, wodurch das Werkzeug kaum einsetzbar ist und die Erkennung fehlschlägt.

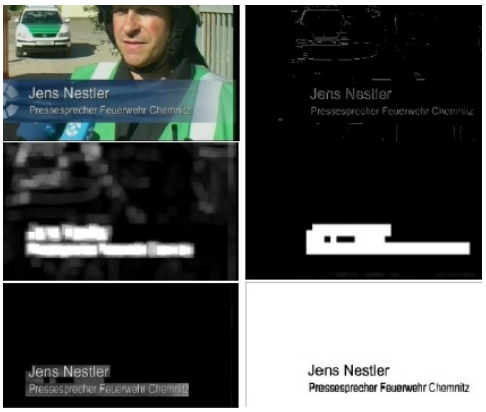

Abbildung 23: Textmaskierung natürlicher Bilder [Hei09]

Eine Verbesserungsmöglichkeit wäre eine zusätzliche Bildverarbeitung vor der Übergabe an Tesseract. Ein Ansatz dazu wurde in [Hei09] gefunden. Bei diesem fünfschrittigen Verfahren, das dafür gedacht ist Texte aus Videos zu extrahieren, wird zuerst geprüft, ob ein Bild über Text verfügt. Enthält ein Pixel in seiner Umgebung eine bestimmte Anzahl an Kantenpixel, ist er ein Textkandidat. Anschließend erfolgt eine diskrete Cosinus-Transformation auf 16 mal 16 Pixel großen Fenstern, die mögliche Textregionen im mittleren Frequenzbereich findet. Die Wahrscheinlichkeit eines Pixelblocks ein Textkandidat zu sein wird letztlich als Energie ausgedrückt, berechnet und normalisiert. Zusammenhängende Textkandidaten mit geringer Größe oder ungeeignetem Breiten- Höhenverhältnis werden verworfen. Die entstandene Maske wird dann aus dem Originalbild herausgeschnitten und letztendlich binarisiert. Das Verfahren wird anhand eines Beispielbildes in Abbildung 23 demonstriert. Mit dieser Vorverarbeitung würde die Geschwindigkeit der Erkennung zwar gebremst werden,

jedoch könnten die Ergebnisse von Tesseract dadurch deutlich verbessert werden, da jegliche Bildinformationen verschwinden würden, die keinen Text enthalten. Es wäre interessant zu sehen, wie sich eine solche Erweiterung auswirken würde und wo weitere Grenzen liegen.
Abgesehen von der Texterkennungsschwäche, hat die Umsetzung der Übersetzungs-App gezeigt wie prädestiniert Android, in Verbindung mit einem leistungsfähigen Smartphone, für diese Art von Anwendungen ist. Aufgrund der jüngsten Veröffentlichungen ist zu erwarten, dass die Verbindung der zwei Technologien, OCR und Übersetzung, eine reifende Entwicklung zeigen wird. Man kann prognostizieren, dass mobile Apps jeglicher Anwendungsgebiete eine immer größer werdende Rolle in unseren Alltag spielen werden. Der Fantasie sind keine Grenzen gesetzt, es bleibt abzuwarten, wann die Technologie auf ihre stößt.

7 Literaturverzeichnis

Literatur

[Bit12] Bitkom. Jeder dritte hat ein smartphone. http://www.bitkom.org/de/presse/8477_71854.aspx, Stand: 14.12.2012.

[Bos12] Rebecca Boswell. U.s. patent 2,026,329. http://prezi.com/csawuhdiismd/ocr-cs4753/, Stand: 14.12.2012.

[Bra10] S Brahler. Analysis of the android architecture. 2010.

[Cha12] Pranob K Charles. A review on the various techniques used for optical character recognition, 2012.

[Che07] Mohamed Cheriet. *Character Recognition Systems - A Guide for Students and Practioners*. John Wiley & Sons, 2007.

[Gag11] Marko Gagenta. *Learning Android*. O'Reilly, 2011.

[GJo12] GJo. Ocr-a schriftart. http://upload.wikimedia.org/wikipedia/commons/thumb/9/93/OCR-A_SP.svg/220px-OCR-A_SP.svg.png, Stand: 14.12.2012.

[goa12] Philosophy and goals. http://source.android.com/about/philosophy.html, Stand: 14.12.2012.

[Goh12] Andreas Gohr. Linux ocr software comparison. http://www.splitbrain.org/blog/2010-06/15-linux_ocr_software_comparison, Stand: 14.12.2012.

[Goo12] Google. Activitiy lifecycle. http://developer.android.com/images/activity_lifecycle.png, Stand: 14.12.2012.

[Hei09] Stephan Heinich. Inhaltsbasierte suche in audiovisuellen medien auf basis von texterkennung. 2009.

[HN05] Masaaki Nagata Kuniko Saito Hideharu Nakajima, Yoshihiro Matsuo. Portable translator capable of recognizing characters on signboard and menu captured by built-in camera. 2005.

[Inc12a] Google Inc. Android ndk. http://developer.android.com/tools/sdk/ndk/index.html, Stand: 14.12.2012.

[Inc12b] Google Inc. Get the android sdk. http://developer.android.com/sdk/index.html, Stand: 14.12.2012.

[Inc12c] Google Inc. Google translate for android. https://play.google.com/store/apps/details?id=com.google.android.apps.translate&hl=de, Stand: 14.12.2012.

[Lee10] Sangchul Lee. Evaluating performance of android platform using native c for embedded systems. 2010.

[Mei12] Reto Meier. *Professional Android 4 application development*. John Wiley & Sons, 2012.

[MW12] Klaus Meyer-Wegener. Erweiterbare objekterkennungsbasierte automatische annotation von bildern, 2012.

[Nie94] Jacob Nielsen. *Usability Methods*. John Wiley & Sons, 1994.

[Oft12] Adrian Ofterdinger. Java native interface ab j2se 1.4. http://www.sigs.de/publications/js/2005/05/ofterdinger_JS_05_05.pdf, Stand: 14.12.2012.

[Ona11] Ö. N. Onak. Comparison of ocr algorithms using fourier and wavelet based feature extraction, 2011.

[Pan10] Arno Becker; Marcus Pant. *Android 2*. dpunkt.verlag, 2010.

[Pie01] Thomas Barry Piehn. Voice enabled digital camera and language translator. http://www.google.com/patents/US20010056342, 2001.

[R.07] Smith R. An overview of the tesseract ocr engine. 2007.

[Rat12] Sylvain Ratabouil. *Android NDK - Beginner's Guide*. Packt, 2012.

[SB96] Satoshi Shirai and Francis Bond. Approaches to disambiguation in alt-j/e. 1996.

[Sig12] Sigma. Android architektur. http://www.sigmadesigns.com/uploads/library/android.jpg, Stand: 14.12.2012.

[Smi09] Ray Smith. Adapting the tesseract open source ocr engine for multilingual ocr. 2009.

[Sta12] Statista. Prognose zu den marktanteilen der betriebssysteme am absatz vom smartphones weltweit in den jahren 2012 und 2016. http://de.statista.com/statistik/daten/studie/182363/umfrage/prognostizierte-marktanteile-bei-smartphone-betriebssystemen, Stand: 14.12.2012.

[Tur11] Victor Fragoso; Steffen Gauglitz; Shane Zamora; Jim Kleban; Matthew Turk. Translatar: A mobile augmented reality translator. 2011.

[Vin12] Luc Vincent. Announcing tesseract ocr. http://googlecode.blogspot.de/2006/08/announcing-tesseract-ocr.html, Stand: 14.12.2012.

[Vis12a] Quest Visual. Introducing word lens. http://www.youtube.com/watch?v=h2OfQdYrHRs, Stand: 14.12.2012.

[Vis12b] Quest Visual. Word lens. http://questvisual.com/us/, Stand: 14.12.2012.

[Wir12] Jörg Wirtgen. Google: 500 millionen android-geräte aktiviert. http://www.heise.de/mobil/meldung/Google-500-Millionen-Android-Geraete-aktiviert-1705107.html, Stand: 14.12.2012.

8 Anhang

Fragebogen des Benutzertests

Benutzertest der Übersetzungs-App

Worum geht es bei dieser App?

Wie der Titel schon sagt, handelt es sich bei diesem Programm um einen Übersetzer. Allerdings müssen Sie hierbei das Gesuchte Wort nicht eintippen oder nachschlagen, sondern die Kamera „lesen“ lassen. Geben Sie zunächst die gewünschte Ein- und Ausgabesprache ein und bestätigen Sie. Nun können Sie ihren Text mit Hilfe des Rechtecks und ihren Fingern eingrenzen, mit dem OCR-Button (Optical Character Recognition) erkennen und mit dem Translate-Button übersetzen lassen.

Worauf muss ich achten?

Versuchen Sie so gut es geht den Text einzugrenzen, damit das Programm nicht versucht am Rand liegende Zeichen zu erkennen. Schauen Sie mit der Kamera möglichst senkrecht auf den Text, damit keine großen Verzerrungen entstehen. Achten Sie darauf, dass der Fokus der Kamera richtig gesetzt ist. Sollte das Bild etwas unscharf sein, drehen sie das Gerät um einige Grad vom Bild weg und dann wieder zurück. Dasselbe gilt für die Helligkeit. Benutzen Sie das Licht, wenn sie merken, dass es zu dunkel ist.

Werden meine Daten veröffentlicht?

Nein, diese Evaluation ist völlig anonym. Antworten Sie bitte dennoch ehrlich und im Kontext der gestellten Fragen. Viel Spaß bei diesem Test!

Allgemeine Angaben:

Alter:_____

Geschlecht:_____

	Trifft nicht zu		Trifft teilweise zu		Trifft zu	Keine Angabe
Ich besitze ein Smartphone.	☐	☐	☐	☐	☐	☐
Ich nutze regelmäßige ein Smartphone.	☐	☐	☐	☐	☐	☐
Ich lade oft neue Apps herunter und probiere sie aus.	☐	☐	☐	☐	☐	☐
Ich habe eine Übersetzungs-App auf meinem Smartphone.	☐	☐	☐	☐	☐	☐
Ich benutze die Übersetzungs-App regelmäßig.	☐	☐	☐	☐	☐	☐
Ich bin neugierig und möchte herausfinden wie gut der Übersetzer funktioniert.	☐	☐	☐	☐	☐	☐

Aufgabe 1:

Versuchen Sie die Ihnen gezeigten Testbilder mit der App zu übersetzen.

Aufgabe 2:

Wechseln Sie in den AR-Modus und probieren Sie es an dem nächsten Bild aus. Bewegen Sie die Kamera ein wenig und schauen Sie was passiert.

Aufgabe 3:

Bleiben Sie im AR-Modus und gehen Sie die Vokabelkärtchen durch.

	Trifft nicht zu		Trifft teilweise zu		Trifft zu	Keine Angabe
Die Bedienung der App ist intuitiv.	☐	☐	☐	☐	☐	☐
Die Übersetzungen waren hilfreich.	☐	☐	☐	☐	☐	☐
Die Texterkennung konnte überzeugen.	☐	☐	☐	☐	☐	☐
Ich finde den AR-Modus nützlich.	☐	☐	☐	☐	☐	☐
Ich würde eine App mit eigenständigem Eintippen der Wörter bevorzugen.	☐	☐	☐	☐	☐	☐
Ich finde die App sinnvoll.	☐	☐	☐	☐	☐	☐
Ich würde diese App auf meinem Smartphone installieren.	☐	☐	☐	☐	☐	☐

Welche Testszenarien konnten gut gelöst werden, welche nicht?

_____

_____

_____

Ich würde mir bei diesem Programm wünschen …

_____

_____

_____

Besondern gefallen hat mir an diesem Programm …

_____

_____

_____

Außerdem möchte ich noch anmerken …

_____

_____

_____

Vielen Dank für die Teilnahme!